

Università degli Studi di Salerno

Dipartimento di Informatica

Dottorato di Ricerca in Informatica
Curriculum Internet of Things and Smart Technologies
XXXVIII Ciclo

Tesi di Dottorato / Ph.D. Thesis

Leveraging LLMs for Software Engineering and Healthcare: Adaptation Strategies and Empirical Evaluation

Gabriele DE VITO

Supervisor: **Prof. Filomena FERRUCCI**

PhD Program Director: **Prof. Andrea DE LUCIA**

A.A 2024/2025

Gabriele De Vito: *Leveraging LLMs for Software Engineering and Healthcare: Adaptation Strategies and Empirical Evaluation* © October 2025

Software Engineering Lab, University of Salerno
https://sesalabunisa.github.io/

This research was done under the supervision of Prof. Filomena Ferrucci, from November 1st, 2022, to October 31st, 2025.

The final version of the thesis has been revised based on the reviews made by Prof. Helena Holmström Olsson and Prof. Panagiota Chatzipetrou.

The doctoral committee was composed by:

Prof. Carmine Gravino	University of Salerno, Italy
Prof. Panagiota Chatzipetrou	Örebro University School of Business, Sweden
Prof. Marc Oriol	Universitat Politècnica de Catalunya, Spain

First Release on October 2025. Revised on November 2025.

In loving memory of my father
To my mother and my family
for their unwavering support.

Programs must be written for people to read,
and only incidentally for machines to execute.

— Harold Abelson [2]

ACKNOWLEDGMENTS

After almost thirty years in IT companies, followed by a period spent convincing teenagers at my Technical Institute that computer science is actually interesting, I thought my capacity for intellectual surprise had been completely exhausted. I was wrong, completely wrong. I started my PhD with the curiosity of a child discovering a new world, ready to marvel at every new piece of knowledge. But just when I was beginning to grow and become familiar with this new world, I found myself here writing my thesis. These three years have been much more shorter than expected, and I still can't believe that my PhD has finished! Anyway, it has been a fantastic journey, and I would like to thank the many people who helped and accompanied me in this incredible adventure.

First of all, **Filomena Ferrucci** deserves recognition as the person who made me fall in love with research. She saw in me what I couldn't see: the curiosity to discover and the perseverance to pursue answers beyond comfortable assumptions. She believed in me when I was still skeptical about my potential as a researcher, providing me with the confidence necessary to begin this journey. Yet her guidance proved equally valuable when my boundless curiosity threatened to make me meet Icarus's fate – her ability to redirect my enthusiasm toward what is productive ensured that I didn't fly too close to the sun with wax wings and fall into the sea.

Thanks, Filomena, I have no words to express how much I've appreciated your guidance and patience :-).

I also thank **Fabio Palomba**. In my opinion, Fabio possesses the rare gift of intellectual honesty, and his questioning – persistent, precise, occasionally uncomfortable – forced me to abandon the habit I had, coming from the industrial world, of accepting correlation as explanation. His insights made me understand the difference between knowing how to solve problems and understanding why solutions work, and I think for an old stubborn-head like me, that's a great step forward.

Again, thanks, thanks, and thanks.

I want to thank **Dario Di Nucci** and **Carmine Gravino**, who believed in my abilities and allowed me to act as a potential link between academic research and industrial practice, giving me the opportunity to contribute to a few research projects at both European and national levels.

Once again, thank you.

I must thank **Andrea De Lucia**, who orchestrated my (but not only mine) entire PhD journey with the wisdom of an elder brother. Sometimes the most profound guidance operates so smoothly that it becomes almost invisible.

What can I say... other than infinite thanks.

Many thanks also to my colleagues **Valeria Pontillo**, **Stefano Lambiase**, **Giulia Sellitto**, **Gilberto Recupito**, **Dario Di Dario**, **Vincenzo De Martino**, **Giammaria Giordano**, **Antonio Della Porta**, **Viviana Pentangelo**, **Alexandra Sheikina**, and **Giusy Annunziata** who treated me as a friend not a Grandpa :-) and were always available.

Last but not least, my heartfelt thanks to **Athanasios Angelakis** for having made my period abroad so easy and comfortable. He trusted and drove me in my research, and it was not trivial!

Obviously, **my family** deserves special recognition for their patience with my transformation from someone who solved problems (or at least tried to) into someone who obsesses over trying to explain why problems exist in the first place. Their support made this adventure feel like a natural extension of who I am rather than an academic exercise. Their tolerance for my research enthusiasms – and their gentle teasing when those enthusiasms became excessive – kept me grounded (thank goodness for that).

Thanks from the bottom of my heart.

I apologize to anyone I've forgotten to mention; the omission reflects the limits of my memory (considering my age) rather than the boundaries of my gratitude. I hope that the people I have collaborated with have been satisfied with the work I have done and have appreciated my enthusiasm. I am certainly proud of my achievements, even though I know that is just a small step on a much challenging ladder.

ABSTRACT

Large Language Models (LLMs) are reshaping the development of artificial intelligence applications, clearly displaying unexpected abilities in natural language understanding and generation tasks. As these systems evolve, their deployment in specific domains like software engineering and healthcare presents exciting opportunities but also new challenges. The textual nature of software artifacts and unstructured healthcare data makes these domains prime candidates for processing by language models, opening up previously unexplored possibilities for automating tasks that have traditionally been performed by humans. On the other hand, LLM-based systems in these domains must meet stringent requirements for accuracy, consistency, and safety, and must integrate smoothly into existing professional workflows.

Even though the interest from the research community is growing and researchers have probed diverse LLM architectures in order to balance their capabilities with deployment considerations, and investigated different methods for enhancing task performance (i.e., adaptation approaches), there are still limitations and challenges that threaten the practicality of LLM-based applications in real-world software engineering and healthcare scenarios. In particular, there is a fragmented evidence regarding (i) different adaptation strategies across specialized domains and complex tasks and how well they perform, (ii) different model architectures and the kinds of tasks they are good at (or not), with (iii) insufficient account taken of the need for reliable, consistent performance in LLM applications destined for high-stakes environments.

In this thesis, we aim to address these issues by developing novel LLM adaptation and deployment approaches specifically designed for software engineering and healthcare domains. We present a set of contributions, ranging from confidence-based frameworks to modular architectures and domain-specific adaptation methods, which we evaluate across diverse applications, including use case quality improvements, automated testing generation,

functional size estimates, clinical decision support, and drug interaction predictions. Through an extensive analysis, we investigate (i) the performance of various adaptation strategies in domain-specific tasks, (ii) the factors influencing the stability and dependability of LLM performance, (iii) the features of the model architecture that affect performance in specialized settings, and (iv) the prerequisites for a successful integration into current workflows.

This dissertation reports findings that challenge several common assumptions about the optimal deployment strategies for LLMs. First of all, we find that model size does not correlate directly with performance effectiveness. Small models achieve results comparable to systems orders of magnitude larger after appropriate domain adaptation. Local deployment using open-weight models can match the performance of cloud-based, proprietary alternatives while providing enhanced data sovereignty and operational independence. Second, our research uncovers method-model interaction complexities that do not lend themselves to universal solutions by demonstrating that using the identical adaptation strategies can have dramatically different outcomes across LLMs. Third, we observe that confidence-based frameworks offer safety-first design capabilities while creating better protection and operational efficacy, contrary to the expectation that conflicts exist between safety and effectiveness. We also show that user acceptance is contingent on the transparency of the decision-making process and users' autonomy, not just on raw system performance.

Lastly, we outline a range of open issues that we believe need to be taken up by the research community, including the development of predictive model-method compatibility frameworks, dynamic confidence calibration across domains, systematic bias detection and mitigation in ensemble failures, and longitudinal performance stability mechanisms to ensure consistent performance over time.

CONTENTS

1 Introduction 1
1.1 Context and Motivation 1
1.2 Research Goals 2
1.3 Research Contributions 6
1.4 Structure of the Thesis 9

Part I Background
2 Background 13
2.1 Large Language Models Fundamentals 13
2.1.1 Architecture and Training 14
2.1.2 Model Operation and Inference 15
2.2 Adaptation Systems 16
2.2.1 Prompt Engineering 16
2.2.2 In-Context Learning 18
2.2.3 Retrieval-Augmented Generation 19
2.2.4 Fine-tuning Approaches 20
2.2.5 Agentic Systems 22
2.3 Challenges and Limitations 23
2.3.1 Technical Limitations 23
2.3.2 Ethical and Social Concerns 24
2.3.3 Domain-Specific Challenges 25

Part II Leveraging LLMs for Software Engineering
3 Enhancing Use Case Quality 29
3.1 Introduction 29
3.2 Related Work and Motivation 31
3.2.1 Quality of Use Case Descriptions 31
3.2.2 NLP Approaches for Requirements Analysis 32
3.2.3 Motivation of Our Work 34
3.3 The ECHO approach 35
3.3.1 Co-Prompting Method Engineering 35
3.3.2 Iterative process to enhance the quality of Use Cases 36
3.4 Experimental method 39

3.4.1 Experiment definition and context 39
3.4.2 Variable selection 40
3.4.3 Experiment setting 41
3.4.4 Data analysis 42
3.5 Results and discussion 43
3.6 Threats to validity 45
3.7 Conclusion 46
4 Generating Acceptance Test Cases from Use Cases 49
4.1 Introduction 49
4.2 Related Work and Motivation 50
4.2.1 Approaches for Generating UAT 50
4.2.2 Motivation of our Work 51
4.3 The AGORA Method 52
4.3.1 Phase 1: Listing test cases 53
4.3.2 Phase 2: Writing the test case details 55
4.3.3 Optimization of Determinism 56
4.3.4 AGORA Tool 59
4.3.5 Replication Package 59
4.4 Empirical Experiment 59
4.4.1 Research question and hypothesis 59
4.4.2 Context of the Experiment 60
4.4.3 Variable Selection 60
4.4.4 Experiment Design 61
4.4.5 Analysis Plan 62
4.5 Analysis of the Results 63
4.5.1 Quality of Acceptance Test Cases (Hypothesis H1) 63
4.5.2 Acceptance Test Case Production Time (Hypothesis H2) 65
4.5.3 User Feedback on the Use of AGORA 66
4.6 Discussion, Implications, and Future work 66
4.6.1 Discussion 67
4.6.2 Threats to Validity 68
4.7 Conclusion 69
5 Automation of COSMIC Functional Size Measurement 71
5.1 Introduction 71
5.2 Background and Motivation 74
5.2.1 COSMIC 74
5.2.2 Motivation of our work 76

5.3 Related Work 80
5.3.1 COSMIC measurement using NLP 80
5.3.2 Automatic COSMIC measurement 83
5.4 The CosMet Approach 86
5.4.1 Prompt Engineering 87
5.4.2 Examples Preparation 88
5.4.3 Validation Process 95
5.4.4 Hyperparameters tuning 97
5.4.5 The CosMet Tool 99
5.4.6 CosMet Setup 101
5.5 Empirical Assessment of CosMet 102
5.5.1 Research Goals and Questions 102
5.5.2 Dataset Creation and Ground Truth 103
5.5.3 Evaluation Metrics 105
5.5.4 Research method for $RQ_{5.1}$ 108
5.5.5 Research method for $RQ_{5.2}$ 109
5.5.6 Research method for $RQ_{5.3}$ 111
5.6 Experimental Results 112
5.6.1 $RQ_{5.1}$ - How effective is CosMet in measuring use cases? 112
5.6.2 $RQ_{5.2}$— How effective is CosMet in mapping FURs to the COSMIC Generic Software Model? 116
5.6.3 $RQ_{5.3}$— How efficient is CosMet in reducing measurement time? 120
5.7 Discussion, Implications, and Limitations 122
5.7.1 Implications and Future Work 123
5.7.2 Threats to validity 124
5.8 Conclusion 127
6 Conversational Software Engineering Assistant 129
6.1 Introduction 129
6.2 Related Work 130
6.3 The C4SE System 131
6.3.1 Motivating Scenarios 131
6.3.2 Tool Architecture and Functionalities 133
6.3.3 Novel Characteristics 138
6.4 C4SE Evaluation 139
6.5 Conclusion 140
7 Generation of Solidity Smart Contracts 141

7.1 Introduction 141
7.2 Background 142
7.2.1 Smart Contracts and No-Code Platforms 143
7.2.2 Astrakode Blockchain 143
7.3 Related Work 144
7.3.1 Code Generation 144
7.3.2 Research Gaps 146
7.4 The Proposed Methods 147
7.4.1 Prompt Engineering Best Practices 147
7.4.2 Method 1: Naive Generation 148
7.4.3 Method 2: Augmented Generation 150
7.4.4 Method 3: Enhanced Generation 152
7.4.5 LLM Choice 156
7.4.6 Integration with AstraKode Blockchain 156
7.5 Design of the Empirical Evaluation 157
7.5.1 Dataset 158
7.5.2 $RQ_{7.1}$: Effectiveness of the Methods 158
7.5.3 $RQ_{7.2}$: Efficiency and Performance Trade-offs of the Methods 160
7.6 Results and Discussion 160
7.6.1 $RQ_{7.1}$: Effectiveness of the Methods 161
7.6.2 $RQ_{7.2}$: Efficiency and Performance Trade-offs of the Methods 163
7.7 Threats to Validity 165
7.8 Conclusions and Future Work 166
8 Issue Report Classification 169
8.1 Introduction 169
8.2 Related Work 173
8.3 Proposed Methods 175
8.3.1 Prompt Engineering Guidelines and Patterns 175
8.3.2 Handling Lengthy Issue Descriptions 176
8.3.3 Method 1: Intent Extraction and Classification 179
8.3.4 Method 2: Ensemble Classification 181
8.3.5 Method 3: Explained Few-Shot Learning 182
8.4 Empirical Evaluation 187
8.4.1 Dataset Selection 187
8.4.2 LLMs Selection 191
8.4.3 Baselines 192

8.4.4 Experimental Procedure and Metrics 193
8.5 Analysis of the Results 202
8.5.1 RQ8.1: Method Effectiveness and Operational Characteristics 202
8.5.2 RQ8.2: Classification Consistency 214
8.5.3 RQ8.3: Error Patterns and Classification Dynamics 218
8.5.4 Lessons Learned 234
8.6 Threats to Validity 236
8.7 Conclusions 238

Part III Leveraging LLMs for Healthcare

9 Adverse Drug Reaction Management 243
9.1 Introduction 243
9.2 State of Art and Motivation 245
9.2.1 CDSSs Challenges and Limitations 245
9.2.2 LLMs Opportunities 246
9.2.3 Motivation of our Work 246
9.3 HELIOT Framework 247
9.3.1 CDSS Design and Approach 248
9.3.2 HELIOT Prototype 257
9.4 Evaluation Methodology 262
9.4.1 Research Goals 262
9.4.2 Synthetic Patient Dataset Creation 262
9.4.3 Real-World Validation Case Study 267
9.4.4 Model Selection and Setup 268
9.4.5 Experimental Design 268
9.5 Results 273
9.5.1 Evaluation Using the Synthetic Dataset 273
9.5.2 Evaluation Using the Real-World Dataset 279
9.5.3 Clinical Usability Evaluation Results 284
9.5.4 Safety Analysis and Risk Assessment 286
9.5.5 Confidence Analysis 287
9.5.6 Research Question Assessment 289
9.6 Implications, Limitations and Future Work 291
9.6.1 Practical Implications and Future Work 291
9.6.2 Limitations 293
9.6.3 Deployment Challenges and Mitigation Strategies 294

9.6.4 Ethical Considerations and Regulatory Compliance 295
9.7 Conclusion 296
10 Drug-Drug Interaction Prediction 299
10.1 Introduction 299
10.2 Related Work 301
10.2.1 Machine learning-based prediction methods 301
10.2.2 LLMs in Drug Discovery 303
10.2.3 Research Gap 304
10.3 Datasets and Data Processing 305
10.3.1 Datasets 305
10.3.2 Data Processing 306
10.4 Methods 307
10.4.1 Selected LLMs 309
10.4.2 Zero-shot Approach 310
10.4.3 Few-shot Approach 311
10.4.4 Fine-tuning Approach 313
10.5 Empirical Evaluation 315
10.5.1 Research Questions 315
10.5.2 Metrics 316
10.5.3 Research methods for $RQ_{10.1}$ 316
10.5.4 Research methods for $RQ_{10.2}$ 316
10.5.5 Research methods for $RQ_{10.3}$ 317
10.5.6 Research methods for $RQ_{10.4}$ 317
10.6 Results and Discussion 318
10.6.1 RQ10.1: Zero-shot Learning Analysis 319
10.6.2 RQ10.2: Few-shot Learning Analysis 321
10.6.3 RQ10.3: Impact of Fine-tuning 323
10.6.4 RQ10.4: Comparative Analysis 324
10.7 Implications, Limitations and Future Work 328
10.7.1 Practical Implications 328
10.7.2 Future Directions 329
10.7.3 Limitations 330
10.8 Conclusion 331

Part IV Conclusion and Further Research Directions
11 Lessons Learned and Open Issues 335
11.1 Lessons Learned 335

11.2 Cross-Context Generalizability and Boundary Conditions 344
11.3 Open Issues 345
12 Further Research Directions 349
12.1 Architectural Intelligence and Predictive Compatibility 349
12.2 Contextual Intelligence and Dynamic Information Architecture 350
12.3 Adaptive Learning and Continuous Evolution 351
12.4 Confidence Architecture and Explainable Intelligence 351
12.5 Ensemble Intelligence and Collaborative Architectures 352

Part V Appendix
A Appendix Heliot 357
A.1 Usability Questionnaire 357
B Appendix LLMDDI 367
B.1 Experiments with Full Fine-Tuning 367
B.2 Additional results 367

Bibliography 377

LIST OF FIGURES

Figure 1.1 Adaptation Framework Pipeline. 4
Figure 3.1 Unoptimized instructions provided to ChatGPT. 35
Figure 3.2 Prompt P1: Starting Use Case Analysis. 37
Figure 3.3 ChatGPT analysis of a use case in response to P1 38
Figure 3.4 Prompt P2: Feedback to ChatGPT 38
Figure 3.5 Prompt P3: Asking for the Enhanced Use Case version 39
Figure 3.6 Enhancing use case quality activity diagram. 39
Figure 3.7 Example of a use case improved by the two groups. 44
Figure 4.1 A simple prompt to produce UATs. 52
Figure 4.2 LLM response for the prompt provided in Figure 4.1. 53
Figure 4.3 Second LLM response for the prompt provided in Figure 4.1. 54
Figure 4.4 AGORA Process. 54
Figure 4.5 Test list prompt. 56
Figure 4.6 Result of the test list prompt. 57
Figure 4.7 Prompt for test case details. 57
Figure 4.8 Result for producing the UAT details. 58
Figure 4.9 Form filled out by the oracle for each use case. 62
Figure 4.10 Normal distribution of the U test for hypothesis H1. 64
Figure 4.11 Normal distribution of the U test for hypothesis H2. 65
Figure 5.1 Use Case: Insert a New Customer 75
Figure 5.2 GPT-4 COSMIC Knowledge. 77
Figure 5.3 Prompt to analyze and measure a use case. 77
Figure 5.4 GPT-3.5 COSMIC analysis. 78
Figure 5.5 GPT-4 COSMIC analysis. 79
Figure 5.6 CosMet components 86

Figure 5.7 Prompt templates for the Sentence Splitter. 88
Figure 5.8 Prompt templates for the COSMIC Analyzer. 89
Figure 5.9 GPT-4 Data Movements Identification without splitting. 90
Figure 5.10 GPT-4 Data Movements Identification with splitting. 91
Figure 5.11 GPT-4 response for a typical "split sentences" request 94
Figure 5.12 GPT-4 response for the "Functional process measurement" request 96
Figure 5.13 CosMet user interface: use case 100
Figure 5.14 CosMet user interface: COSMIC measurement analysis 101
Figure 5.15 Time reduction using CosMet according to the measurers. 122
Figure 6.1 The C4SE's architecture. 133
Figure 6.2 C4SE Augmented Prompt Template 135
Figure 6.3 C4SE functional behavior. 136
Figure 6.4 C4SE Classification Prompt 136
Figure 6.5 C4SE Relevant Information Extraction 137
Figure 6.6 C4SE Augmented Prompt 137
Figure 6.7 C4SE Final Response Example 138
Figure 7.1 Naive Generation Method. 148
Figure 7.2 Naive Prompt Template. 149
Figure 7.3 Correct Code Prompt Template. 149
Figure 7.4 Augmented Generation Method. 150
Figure 7.5 Incomplete Code Detection Prompt Template. 151
Figure 7.6 Complete Function Code Prompt Template. 151
Figure 7.7 Enhanced Generation Method. 153
Figure 7.8 Contract Elements Generation Prompt Template. 154
Figure 7.9 Library and Inheritance Analysis Prompt Template. 154
Figure 7.10 Detailed Function Specifications Prompt Template. 155
Figure 7.11 Chatbot Integration in AKB. 157

Figure 8.1 Overall Methodology used in the study. 171
Figure 8.2 Approach for Dealing with Long Text. 177
Figure 8.3 Reduce Prompt. 178
Figure 8.4 Intent Extraction and Classification: Components. 179
Figure 8.5 Extract Prompt for Method 1. 180
Figure 8.6 Prompt for classification in Method 1. 181
Figure 8.7 Ensemble Classification: Components. 182
Figure 8.8 Explained Few-Shot Learning: Components. 183
Figure 8.9 Extract Prompt for Method 3. 184
Figure 8.10 Few-Shot Learning with Explanation Prompt. 185
Figure 8.11 Distribution of documentation features across different issue report types in our dataset. 189
Figure 8.12 Distribution of token lengths (title + body) in the primary and long issue test sets. The x-axis is log-scaled to account for the strong skewness in token counts. 191
Figure 8.13 Differences in weighted average F1 scores between each proposed method–LLM combination and the RoBERTa and SetFit baselines, across three evaluation settings: Primary Test Set, Longer Issues Test Set (MapReduce Approach), and Longer Issues Test Set (Truncation Approach). Positive values indicate performance improvements over the baseline. 208
Figure 8.14 Confusion Matrices of Issue Misclassifications 218
Figure 8.15 Sankey Diagrams of Misclassification Flows Across LLMs and Methods. 220
Figure 8.16 Combined analysis of classification confusions and misclassification causes across methods and LLMs. 226
Figure 9.1 Decision Support Prompt: System Prompt. 249
Figure 9.2 Decision Support Prompt: User Prompt. 249
Figure 9.3 Ingredient Translation Prompt. 251
Figure 9.4 HELIOT CDSS Architecture. 252
Figure 9.5 HELIOT CDSS Data Flow. 253

Figure 9.6 Overview of the Pharmaceutical Data Pipeline Process. 257
Figure 9.7 Class Distribution in the Patients Dataset. 264
Figure 9.8 Experiment Design and Setup. 269
Figure 9.9 HELIOT Safety Performance Dashboard. 287
Figure 10.1 Data preprocessing pipeline. 306
Figure 10.2 Methods overview showing the three-tier LLM approach for DDI prediction: zero-shot, few-shot, and fine-tuning workflows with drug pair input processing and classification output. 308
Figure 10.3 Zero-shot prompt template 310
Figure 10.4 Few-shot prompt template 312
Figure 10.5 Zero-shot performance of the top 10 LLMs on DDI prediction: Validation Dataset. 319
Figure 10.6 Random Few-shot performance of the top 10 LLMs on DDI prediction: Validation Dataset. 321
Figure 10.7 Similarity-based Few-shot performance of the top 10 LLMs on DDI prediction: Validation Dataset. 322
Figure 10.8 Fine-tuning results of different LLMs on DDI prediction: Validation Dataset. 324
Figure 10.9 Sensitivity comparison across external datasets. 325
Figure 10.10 Accuracy comparison across external datasets. 326
Figure 10.11 F1 comparison across external datasets. 326

LIST OF TABLES

Table 1.1 Summary of the Thesis Contribution 7
Table 1.2 Research Goals and Contributions relationship. 9
Table 3.1 Criteria and aspects for evaluating the quality of use cases 41
Table 3.2 Questionnaire No. 1 – Quality assessment of use cases executed by the control group 42

Table 3.3 Questionnaire No. 1 – Quality assessment of use cases executed by the Experimental group 42
Table 3.4 Questionnaire No. 3 – Experimental group's experience with using ECHO 43
Table 3.5 Questionnaire No. 2 – Quality assessment of the enhanced use cases executed by the Oracle 43
Table 4.1 Summary table of related works 51
Table 4.2 Quality evaluation. 61
Table 4.3 Questionnaire on the experience with AGORA 61
Table 4.4 Experimental group UAT results 64
Table 4.5 Control group UAT results 64
Table 4.6 Feedback on the experience with AGORA 66
Table 5.1 COSMIC analysis of the "Insert new customer" use case 76
Table 5.2 Comparison with Related Work on COSMIC Measurement using NLP 82
Table 5.3 Questionnaire no. 1 provided to the measurers. 111
Table 5.4 Questionnaire no. 2 provided to the measurers. 112
Table 5.5 Quantitave Evaluation Results. 113
Table 5.6 Evaluation Results by COSMIC Concept. 116
Table 5.7 Participants's answers to the questionnaire no 1. 120
Table 5.8 Measurement durations recorded by the participants for FID-TCT. 122
Table 7.1 Descriptive Statistics for Code Quality Metrics and Methods 160
Table 7.2 Statistical Test Summary for Code Quality Metrics 162
Table 7.3 Descriptive Statistics for Efficiency Metrics and Methods 164
Table 7.4 Efficiency Metrics for Contracts Using User-Defined Libraries 164
Table 7.5 Statistical Test Summary for Efficiency Metrics 165

Table 8.1 Distribution of labels in the manually-verified dataset 189
Table 8.2 Weighted-averaged F1-scores achieved by the proposed methods across different temperature (T) settings. 203
Table 8.3 Classification results in terms of F1-measure for each class, plus weighted-averaged values (best F1-measure values are highlighted in bold). In the last column, the table also reports the weighted average F1 variation (Δ) against the best-performing baseline (SetFit), with positive Δ values indicating performance gains. 204
Table 8.4 Results of the ablation study on the intent extraction component. 206
Table 8.5 Weighted average F1 scores for issue classification methods across different test sets and input processing approaches. The highest F1 scores in each column are highlighted in bold. 207
Table 8.6 Weighted average F1 scores achieved by GPT-4o for longer issue classification using three input processing strategies: MapReduce-style decomposition, truncation, and direct ingestion of full issue descriptions. 208
Table 8.7 Average cost (in USD) to classify a single issue report using different LLMs, methods, and processing strategies. 209
Table 8.8 Results of the statistical tests investigating differences in perplexity metrics between correctly and incorrectly classified issue reports. 211
Table 8.9 Method 2 classification performance with alternative, perplexity-based voting mechanisms. 211
Table 8.10 Consistency Results 214
Table 8.11 Consistency Results for the proposed methods, across different temperature settings. 215

Table 8.12 Distribution of Misclassification Causes Across Methods and LLMs. For the sake of conciseness, Method 1, Method 2, and Method 3 are abbreviated as M1, M2, and M3, respectively. 225
Table 8.13 Interpretive framework of method-specific insights across RQ8.1–RQ8.3. 236
Table 9.1 Case Classification, Reaction Type, and Alert Type. 250
Table 9.2 Drug Dataset Structure. 258
Table 9.3 Drug Distribution in Dataset. 259
Table 9.4 Processed Leaflet Dataset Structure. 260
Table 9.5 Ingredient Synonyms Dataset Structure. 260
Table 9.6 Clinical Cases Classifications. 263
Table 9.7 Synthetic Dataset Structure. 265
Table 9.8 Case Distribution by Classification, Reaction Type, and Alert Type. 266
Table 9.9 Classification Results per Category (averaged over five runs). 273
Table 9.10 Classification Results per Reaction Type (averaged over five runs). 274
Table 9.11 Classification Results per Alert Type (averaged over five runs). 275
Table 9.12 Gemma3: Classification Results per Category. 276
Table 9.13 Gemma 3: Classification Results per Reaction Type. 276
Table 9.14 Gemma3: Classification Results per Alert Type. 277
Table 9.15 Claude Sonnet 4.0: Classification Results per Category. 278
Table 9.16 Claude Sonnet 4.0: Classification Results per Reaction Type. 278
Table 9.17 Claude Sonnet 4.0: Classification Results per Alert Type. 279
Table 9.18 GPT-4o Classification Results per Category on Real-World Dataset. 279
Table 9.19 Classification Results per Reaction Type for GPT-4o on Real-World Dataset. 280

Table 9.20 Classification Results per Alert Type for GPT-4o on Real-World Dataset. 280
Table 9.21 Classification Results per Category - Gemma 3 Real-World Evaluation. 281
Table 9.22 Classification Results per Reaction Type - Gemma 3 Real-World Evaluation. 282
Table 9.23 Classification Results per Alert Type - Gemma 3 Real-World Evaluation. 282
Table 9.24 Classification Results per Category - Claude Sonnet Real-World Evaluation. 283
Table 9.25 Classification Results per Reaction Type - Claude Sonnet Real-World Evaluation. 284
Table 9.26 Classification Results per Alert Type - Claude Sonnet Real-World Evaluation. 284
Table 9.27 Clinical Usability Evaluation Results Summary. 285
Table 9.28 Confidence Analysis Results Across LLMs and Datasets. 288
Table 9.29 Risk Threshold Framework for Clinical Decision Support. 290
Table 10.1 Drugs and Initial DDI For Datasets. 305
Table 10.2 Datasets with Positive and Negatives Drug Pairs. 308
Table 10.3 Zero-shot performance of reasoning-enhanced LLMs on DDI prediction. 320
Table 10.4 Optimal LoRA parameters for fine-tuned models obtained through hyperparameter search. 323
Table 11.1 Summary of Lessons Learned and their relationship with the Research Goals indicated in the Chapter 1. 343
Table B.1 Full fine-tuning results for Phi-3.5 2.7B with half trainable parameters. 367
Table B.2 Detailed classification metrics for Phi-3.5 2.7B full fine-tuning with half trainable parameters. 367
Table B.3 Full fine-tuning results for Phi-3.5 2.7B with 4 trainable layers. 367

Table B.4 Detailed classification metrics for Phi-3.5 2.7B full fine-tuning with 4 trainable layers. 368
Table B.5 Zero-shot performance of different LLMs on DDI prediction. 368
Table B.6 Random Few-shot performance of different LLMs on DDI prediction. 369
Table B.7 Similarity-based Few-shot performance of different LLMs on DDI prediction. 370
Table B.8 Fine-tuning results of different LLMs on DDI prediction. 370
Table B.9 Sensitivity comparison across external datasets. 371
Table B.10 Accuracy comparison across external datasets. 371
Table B.11 F1 comparison across external datasets. 372

1 INTRODUCTION

1.1 CONTEXT AND MOTIVATION

The recent success of Large Language Models (LLMs) in research and commercial applications is a groundbreaking development in artificial intelligence that is opening up previously unheard-of possibilities for automating knowledge-intensive, complex tasks that have previously only been performed by humans. Software engineering emerges as a particularly fertile domain for LLM integration, where the discipline's inherent dependence on structured yet flexible textual representations creates natural synergies with the capabilities of language models. The textual nature of code, documentation, requirements specifications, and other software artifacts makes them prime candidates for processing by language models. That is why the commercial success of GitHub Copilot [1] exemplifies the paradigm shift that is happening. GitHub Copilot is all about natural language prompts leading to the automatic generation of code that remains under the supervision and responsibility of the human developer [271]. However, the transformative potential of LLMs extends well beyond just code generation. The software lifecycle is an array of activities – e.g., requirement analysis, test case generation, quality assessment, and metric prediction – that are performed by natural language processing, analytical reasoning, and knowledge synthesis, domains where LLMs have shown exceptional proficiency [104]. This convergence creates an enormous opportunity to automatically perform software engineering tasks that we traditionally associate with a human expert.

Another domain where integration of LLMs might lead to an impactful change is healthcare. The LLM application landscape in this sector is characterized by a combination of complex medical knowledge and the unstructured data healthcare systems con-

1 GitHub Copilot. https://github.com/features/copilot

tinuously generate, creating a unique environment ideally suited for advanced AI intervention [286]. Timely access to both structured and unstructured information is necessary to allow intelligent support of clinical decision-making. To that end, the LLMs' capability of extracting meaningful content from a large volume of data could lead to better patient outcomes and more informed clinical decisions, alleviating burdens on the healthcare system [21].

The response from the research community has been multifaceted, probing various model architectures that range from huge, proprietary ones (e.g., GPT-4o [219], Anthropic Sonnet [17]) to efficient, compact alternatives (e.g., Phi-3.5 2.7B [1], Qwen2.5 3B [326]) that balance capability with practical deployment considerations. This diversification of model architectures, which aims to optimize models' working capability-practicality equilibrium while addressing the domain-specific requirements, presents a vibrant landscape of opportunities and challenges. Such diversity makes model selection and specialization for specific tasks increasingly complex, requiring organizations to carefully assess which architectures and task adaptation approach best fit their specific requirements in terms of performance, cost, and deployment constraints [340]. As LLMs mature, an even more urgent need for rigorous analysis emerges, given that LLM adoption decisions increasingly affect how well an organization performs and how competitively it positions itself.

1.2 RESEARCH GOALS

Even though LLMs have shown promise in natural language processing tasks and initial success in specialized domains, they still have several critical problems that limit their effective deployment in specific software engineering and healthcare contexts. Our investigation into the current state of LLM applications in these domains found some key aspects that still need a systematic deep dive:

1. **Fragmented empirical evidence on comparative effectiveness of adaptation strategies across specialized domains.** Prompt engineering has emerged as a primary mechanism

for adapting LLMs to specific tasks [314], yet the investigation of different adaptation approaches (e.g., zero-shot learning, few-shot learning, Retrieval-Augmented Generation, prompt engineering patterns, and fine-tuning) remains fragmented. Current research often focuses on proof-of-concept applications without comparing these adaptation strategies across different tasks. This limitation is particularly critical in domains where accuracy and reliability are paramount, as the choice of adaptation strategy can significantly impact system performance and user acceptance.

2. **Inadequate handling of complex and lengthy inputs in domain-specific applications.** LLMs have a limited context window (i.e., the number of words they can handle in input and output), making it difficult for them to work with long or complex inputs [346]. When evaluating performance with long requirement documents or elaborate system specifications in software engineering, it is often common to see a significant performance change [13, 302], even when models have very large context windows (e.g., GPT-4o [219]). The same difficulty is also encountered in healthcare applications that require the analysis of long patient records and vast amounts of medical literature. However, the problem remains unsolved.

3. **Insufficient attention to consistency and reliability in mission-critical applications.** Applications requiring consistent and reliable outputs encounter challenges with using LLMs if they are the key technological source due to their non-deterministic nature [82]. To make matters worse, the variability of LLM outputs may cause possible serious safety issues (e.g., contradictory clinical recommendations), or inconsistency (e.g., nonsensical code in software engineering). Existing methods have often not adequately addressed these reliability issues.

4. **Limited integration with existing workflows and domain-specific requirements.** LLM-based applications are often developed as standalone solutions without adequately considering their integration into established practices and exist-

ing technological ecosystems. This issue is particularly critical in healthcare environments, where stringent requirements such as regulatory compliance, patient data privacy protection, and integration with electronic health record systems are mandatory [286]. Similarly, in software engineering domain, LLM-based tools have shown significant difficulties in effectively integrating with existing development environments and methodologies.

5. **Lack of systematic evaluation across diverse model architectures and deployment scenarios.** Current research efforts are focused on the largest, most capable models, ignoring that many LLMs exist in different sizes and architectures that may perform well for particular specialized tasks [340]. This is an important limitation since many operational environments have computational limitations, and smaller, more efficient models can deliver acceptable performance for specialized tasks [82].

These aspects result in five recurring *failure modes* for LLMs, specifically lengthy Input, domain gap, non-determinism, reliability and transparency, and integration issues.

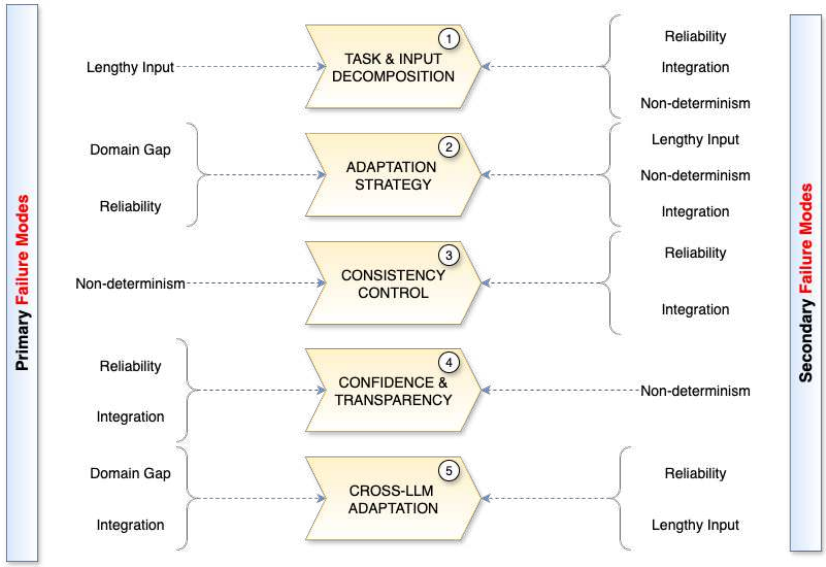


Figure 1.1: Adaptation Framework Pipeline.

This thesis seeks to address the above-mentioned gaps by proposing and evaluating innovative adaptation approaches to improve the effectiveness and reliability of LLM-based applications, with particular attention to practical deployment aspects and domain-specific constraints.

From a methodological perspective, we adopt a systematic framework based on five adaptation levers to address the identified failure modes:

- **Task and Input Decomposition.** Break complex tasks and long inputs into structured, manageable units. This lever preserves salient context, reduces context-window pressure, and makes processing auditable and integrable with existing workflows.
- **Adaptation Strategy.** Select and tune the most suitable technique for the task and domain (e.g., structured prompting, few-shot with curated exemplars, retrieval-augmented generation, fine-tuning). This lever closes domain-knowledge gaps, enforces output structure, and improves accuracy-cost trade-offs.
- **Consistency Control.** Reduce output variance via hyperparameter settings (temperature, seeds), multi-run protocols, agreement checks (e.g., constrained decoding, lightweight ensembles), and canonical output schemas. This lever ensures repeatability across runs and robustness to small input changes.
- **Confidence and Transparency.** Quantify and expose uncertainty (e.g., logprob/perplexity surrogates, calibration with ECE/Brier), provide explicit rationales and traceable evidence, and apply risk thresholds for human hand-off. This lever enables safe use, auditability, and user trust in high-stakes contexts.
- **Cross-LLM Adaptation.** Design prompts, tools, and pipelines to be portable across models and sizes; compare architectures under identical conditions; and implement routing/-fallback policies based on task, cost, latency, and privacy. This lever mitigates vendor lock-in, surfaces model–method interactions, and supports cost-effective deployment.

Each lever is designed to primarily address one or more failure modes (see Figure 1.1) and has secondary effects on other critical issues. The application can be modular (not all levers are always necessary) and iterative until the accuracy, consistency, cost, and integrability requirements specified by the application context are met.

More specifically, the high-level research goals that were considered in this thesis can be summarized as follows:

RG1: *Investigate how different adaptation mechanisms can be effectively performed in specific software engineering and healthcare domains.*

RG2: *Develop methods to ensure consistent and reliable LLM performance in specific software engineering and healthcare domains.*

RG3: *Explore LLM integration with existing workflows in specific software engineering and healthcare domains.*

RG4: *Perform systematic evaluations of the effects of model size and architecture for specialized tasks in specific software engineering and healthcare domains.*

1.3 RESEARCH CONTRIBUTIONS

To pursue the research goals, we present a set of contributions that fall into two main domains: software engineering and healthcare applications. Each contribution is discussed in a dedicated chapter, based on a research paper that has been submitted or published as illustrated in Table 1.1.

In particular, in the software engineering domain, we present the following studies:

- Chapter 3 presents ECHO, an approach that enhances the quality of use cases exploiting LLMs [83]. Here, we introduce a novel co-prompt engineering approach – a technique that utilizes the reflective capacity of LLMs to optimize prompts and generate synthetic data – that can facilitate the design of prompts and improve requirements specification.

- Chapter 4 describes AGORA, an approach for automating the user acceptance test generation process from use cases

Table 1.1: Summary of the Thesis Contribution

Domain	Chapter	Paper
Software Engineering Part II	3	De Vito Gabriele, Palomba Fabio, Gravino Carmine, Di Martino, Sergio, and Ferrucci Filomena, "Echo: An approach to enhance use case quality exploiting large language models." Proc. 49th Euromicro Conf. SEAA, IEEE, 2023. [83]
	4	De Vito Gabriele, Vassallo Gabriele, Palomba Fabio, and Ferrucci Filomena, "AGORA: An Approach for Generating Acceptance Test Cases from Use Cases." Proc. 50th Euromicro Conf. SEAA, IEEE, 2024. [87]
	5	De Vito Gabriele, Di Martino Sergio, Ferrucci Filomena, Gravino Carmine, and Palomba Fabio, "LLM-Based Automation of COSMIC Functional Size Measurement From Use Cases." IEEE Trans. Softw. Eng., 2025. [75]
	6	De Vito Gabriele, Lambiase Stefano, Palomba Fabio, and Ferrucci Filomena, "Meet c4se: Your new collaborator for software engineering tasks." Proc. 49th Euromicro Conf. SEAA, IEEE, 2023. [81]
	7	De Vito Gabriele, D'Amici Damiano, Izzo Fabiano, Ferrucci Filmena, and Di Nucci Dario, "LLM-Based Generation of Solidity Smart Contracts from System Requirements in Natural Language: The AstraKode Case." Proc. Int. Conf. IEEE SANER, IEEE, 2025. [74]
	8	De Vito Gabriele, Starace Luigi, Palomba Fabio, Di Martino Sergio, and Ferrucci Filomena, "Advancing LLM-Based Issue Report Classification with Explained Few-Shot Learning, Intent Extraction, Ensemble, and Summarization." TOSEM (2025). Under Review (II round). [86]
Healthcare Part III	9	De Vito Gabriele, Filomena Ferrucci, and Athanasios Angelakis. "HELIOT: LLM-Based CDSS for Adverse Drug Reaction Management." Knowledge-Based Systems (2025): 114184. [77]
	10	De Vito Gabriele, Filomena Ferrucci, and Athanasios Angelakis. "HELIOT: LLM-Based CDSS for Adverse Drug Reaction Management." Knowledge-Based Systems (2025). Under Review (II round). [79]

[87]. This approach employs an adaptation process, that features task decomposition and prompt engineering patterns to address LLM context limitations and non-determinism.

- Chapter 5 introduces CosMet, an approach to automate the COSMIC functional size measurements from the use case documents [75]. This chapter considers few-shot learning and hyperparameter optimization to improve accuracy and measurement consistency and transparency.

- Chapter 6 presents an agentic conversational software engineering assistant called C4SE, which organizes multiple specialized 'agent' models, a dynamic knowledge base, and a conversational memory, to build a unified platform for software development tasks, which are usually executed through separate tools [81]. C4SE shows how to manage the context loss and the tool fragmentation challenges.

- Chapter 7 introduces an approach for the generation of smart contracts from natural language requirements [74]. It employs several adaptation techniques, which includes prompt engineering patterns, zero-shot, chain-of-thought reasoning, and iterative refinement strategies for blockchain development.

- Chapter 8 introduces novel issue classification methods that use several adaptation strategies, including: (i) MapReduce-inspired approaches for handling long texts, (ii) ensemble voting mechanisms to tackle non-determinism, and (iii) co-prompt engineering design of few-shot learning prompts using explicit reasoning [86]. The chapter also presents a cross-LLM analysis of classification performance and a comprehensive exploration of error patterns and their underlying causes.

In the healthcare domain, we propose the following contributions:

- Chapter 9 introduces HELIOT, a clinical decision support system that uses Retrieval-Augmented Generation, structured prompt engineering, confidence scores, and flexibly integrates with Electronic Health Record (EHR) systems to contextualize adverse drug reaction alerts based on patient-specific histories, thereby reducing alert fatigue [77].

- Chapter 10 presents a new text-based approach that uses carefully designed prompts to process multiple drug attributes simultaneously (i.e., target organisms, gene interactions, and SMILES notation), without requiring sophisticated feature engineering or structured chemical representations [79]. This study systematically compares zero-shot, few-shot, and fine-tuning adaptation techniques across 18 LLMs of differing sizes and architectures.

Each study helps to address one or more research goals, as reported in Table 1.2.

Apart from the contributions specified above, two further common factors characterize this work.

Table 1.2: Research Goals and Contributions relationship.

RG	Paper
RG1	[83] [87] [75] [81] [74] [86] [77] [79]
RG2	[83] [87] [81] [75] [74] [86] [77] [79]
RG3	[75] [74] [86] [77]
RG4	[86] [77] [79]

- **Empirical Validation.** All studies utilized public datasets, and practitioners from commercial and industry sectors were involved with every study. The engagement of professionals varied depending on the domain's characteristics and requirements, ranging from participation focused on validation to collaborative involvement throughout the entire research process (see Chapter 11 for a detailed discussion).

- **Open-source Tools and Replication Packages.** Every contribution was made available with code, prompts, hyperparameters, and evaluation scripts, as a way to enable reuse and development in this field further.

1.4 STRUCTURE OF THE THESIS

This dissertation is structured in four parts. Part I outlines the essential background in order to understand the principles of LLMs, adaptation, and their potential and pitfalls. Part II and Part III represent the two core areas of the work – one for each group of contributions outlined in Table 1.1 –, that develop the chapters that further describe the individual contributions. Given the task-specific and domain-dependent nature of LLM-based applications, related work is discussed within each dedicated chapter rather than in a centralized literature review section.

Specifically, in Part II – comprising Chapters from 3 to 8 – we illustrate LLM-based approaches for different contexts in software engineering, such as use case quality improvement, functional size measurement automation, and issue report classifications.

In Part III, we present applications in healthcare for drug reaction management (Chapter 9) and drug-drug interaction prediction (and Chapter 10).

Finally, Part IV is intended to connect the dots among the studies presented in this thesis. Chapter 11 reflects on some of the lessons learned across all applications and discusses patterns and challenges associated with the use and deployment of LLMs to specialized domains. In Chapter 12, we provide some potential future research directions based on the empirical work as well as emerging opportunities within the area.

Part I

BACKGROUND

2

BACKGROUND

This chapter provides the necessary background to understand the research contributions presented in this thesis. We begin by exploring the fundamental concepts underlying LLMs, including their architecture, training methodologies, and operational principles. We then examine the various techniques available for adapting these models to specific domains and applications. Finally, we discuss the current challenges that limit LLM deployment, providing a balanced perspective on both the potential and constraints of this technology.

2.1 LARGE LANGUAGE MODELS FUNDAMENTALS

LLMs represent a fundamental shift in artificial intelligence and offer a completely new way for machines to process and generate human language. Traditional natural language processing (NLP) approaches are built on handcrafted rules or task-specific architectures. LLMs learn language patterns and knowledge directly from vast amounts of text data through neural network training.

Indeed, as traditional NLP relied on separate models for different applications, (i.e, one for translation, one for summarization, and one for question answering, [195]), LLMs have the ability to perform several language tasks through a single architecture, adapting their behavior based on textual instructions called prompts. This versatility can be attributed to the principle of language modeling, namely predicting the next word (or token, that is the basic processing unit of text) in a text sequence based on the previous context. This objective might appear simple, but when done at scale with billions to trillions of parameters and massive datasets, it provides LLMs with the potential to perform complex tasks that may require logical reasoning, mathematical problem-solving, and creative text generation [16].

2.1.1 *Architecture and Training*

The journey toward modern LLMs began with a key question: how can a machine simultaneously understand individual words and their relationships across long passages? Earlier forms of neural networks processed text sequentially, word by word, which made it challenging to represent relationships among ideas that are distantly related. The introduction of the Transformer architecture by Vaswani et al. [300] allowed networks to, in a sense, "think" about all the words "at once" while also being able to reason about relationships between them. In other words, the attention mechanism allows the network to create a web where any word can relate to any other directly; distance does not impact the nature of the relationship. Let us consider the sentence *"The cat, which had been sleeping peacefully in the garden all morning, suddenly jumped."* When we read *"jumped,"* we instinctively connect it back to *"cat"* despite the intervening phrase. The same type of behavior is available to a neural network through the use of attention, and is realized through an elegant mathematical framework that involves queries, keys, and values. As the model processes any given word, it generates a query that describes information it is looking for, and the other words will be keys to describe what information is being made available. The model matches queries with keys, generating attention weights, and combines the values associated with the keys to generate its final representation of meaning. Multi-head attention is even better, because it enables multiple "parallel" attention processes, for instance, one to track grammatical relationships, another for semantic reasoning, and a third to track the overall narrative. Because attention alone cannot convey word order (e.g., *"the cat chased the dog"* vs *"the dog chased the cat!"*), Transformers utilize positional encodings and residual connections to retain positional information in the sequence and to ensure continuous flow of states through different layers in the network. The overall architecture is composed of two main blocks, an **encoder** to interpret input text and a **decoder** to produce output text, and combinations of these blocks produce up to three distinct classes of models [111, 169]. Encoder-only models (e.g., *BERT* [90]) are trained to understand text in both directions (bidirectionally), that is, to process the text from both sides

to allow models to recreate rich contextual representations eventually. Decoder-only models like *GPT* [36, 219], *Anthropic Claude* [17], or open-weight alternatives, such as *LLama* [290] and *Google Gemma* [284], produce output text through autoregressive processing, generating one word at a time using all previous words as context. Lastly, encoder-decoder models (i.e., *FLAN-T5* [55], *BART* [175]) use a hybrid approach by trying to understand the input first fully and only then produce an appropriate output.

The true power of these architectures emerges through pre-training on huge text corpora comprising hundreds of billions (to trillions) of tokens taken from books, articles, websites, and reference materials [36]. During this process, models learn to predict patterns in human language, whether predicting the next word in a sequence or reconstructing masked portions of text [222]. This learning objective requires models to internalize grammar rules, factual knowledge, reasoning patterns, and cultural understanding [36, 222]. The typical process has two stages: general exposure to many text sources followed by a refinement training on higher-quality materials. Training at this scale requires managing thousands of specialized processors and GPUs across multiple data centers and employing complicated techniques for distributing the computation [36, 222]. The result is a strange paradox of artificial intelligence: models learn not through logic or rules, but through accumulating linguistic experience.

2.1.2 *Model Operation and Inference*

Once trained, the models transition from the learning phase to application, where we can use and manage their linguistic experience and generation potential through specific parameters called hyperparameters. These hyperparameters control the sampling strategies for selecting the next token to output at inference time, balancing coherence and creativity.

Temperature acts as a primary control for output randomness [132]. At a temperature of 0, the model becomes more deterministic, as it tends constantly to select the highest probability token, which is valuable for tasks with an expected consistent output but creates monotonous, repetitive texts. At higher temperatures, the probability distributions flatten, which brings out more creativ-

ity and typically, sense, but also less coherence. The relationship is not linear: at a temperature of 0.7, one may yield natural responses relevant to a conversational exchange, whereas at a temperature of 1.5, one will likely produce surreal, nonsensical text.

TOP-K and TOP-P sampling provide alternative control methods for generation diversity. The key concept of top-k sampling is to limit consideration at each step to the k most probable token, thereby avoiding low-probability tokens which may interfere with coherent generation [106]. Top-p, also referred to as nucleus sampling, adjusts this restriction dynamically by including tokens until their cumulative probability reaches a threshold p [132].

A model with high temperature, coupled with low top-k, might have enough variability to generate creative outputs within a constrained space. In contrast, a low temperature and a high top-p sampling may limit creative options.

As we can notice, the effects of these hyperparameters on the generation process can be complex and, therefore, their setting requires careful balancing of creativity and coherence for the intended application.

2.2 ADAPTATION SYSTEMS

Like any other type of AI, LLMs should have different settings or adaptation techniques to help align their performance with the actual user needs, the domains in which they will be applied, and the tasks that need to be performed. As discussed in the sections below, these techniques can involve simple adjustments to the prompt without any further training, or more complicated retraining techniques that require changing model parameters.

2.2.1 *Prompt Engineering*

The most direct and resource-efficient method for adapting LLMs is PROMPT ENGINEERING, which consists of carefully designing prompts to obtain the desired results and requires no training. All this means that we can leverage the model's already existing capabilities and immediately use them across different tasks and domains.

Prompt engineering is a multifaceted discipline that combines multiple, complementary techniques to optimize LLM performance, where each technique benefits from an iterative cycle of experimentation and refinement.

The first technique is ZERO-SHOT PROMPTING, where the model is instructed to reach the intended result without examples through clear and simple instructions [36, 165]. In this approach, the aim is to design explicit instructions and expectations (i.e., prompts) that are similar to those the model was trained on. An example of an unambiguous prompt that most LLMs should generally be able to follow is *"Analyze the sentiment of the following text, and classify it as positive, negative, or neutral:"*, where the instruction (*"Analyze the sentiment of the following text"*) and the nature of the task (*"classify it as positive, negative, or neutral"*) are well-defined and clear.

When a task becomes more complicated (e.g., mathematical problem solving, logical reasoning, or any analysis that occurs in multiple steps), zero-shot instructions often are not enough. In these scenarios, researchers have discovered that CHAIN-OF-THOUGHT prompting is more useful and effective [308]. Rather than just asking for answers, prompting the model with *"Let's think step by step"* or *"Show me your reasoning process"* encourages the LLM to break down complex problems into more manageable parts and "think" through them step-by-step [308].

With regard to patterns – namely, structured, reusable prompt design frameworks – several have been proposed in the literature, in particular the PERSONA, OUTPUT FORMATTING, and TEMPLATE-BASED patterns [314].

PERSONA PATTERNS change the behavior of the LLM by defining a particular role or point of view. When beginning the interaction with instructions such as *"You are a software architect with 15 years of experience"* or *"Answer as a teacher who is patiently explaining things to a beginner,"* the model adopts a specific form of communication, a level of knowledge, and a certain method of conveying information. This pattern format is highly effective for domain-specific applications where the level of expertise or tone matters.

OUTPUT FORMATTING PATTERNS provide structure and consistency to LLM's responses. A prompt can request a specific output format such as *"Provide your response in JSON format with 'summary',*

'key_points', and 'recommendations' fields" or *"Structure your analysis using the following sections: Problem, Analysis, Solution, Implementation."* These patterns create outputs that are *"parsable"*, that is they can be easily processed programmatically, or simply generate standardized responses as per application requirements.

TEMPLATE-BASED APPROACHES provide a more systematic technique to prompt engineering, as templates can create reusable frameworks for common tasks. Templates may include placeholders for variable information, like specific instruction patterns. For instance, a code review template may include sections for functionality, style, and possible improvements, with relevant code snippets inserted as needed.

Prompt almost always involves iteratively refining instructions to achieve the desired performance based on model outputs. The nature of this process results in interesting, and sometimes counterintuitive, insights – a shorter prompt may perform better than a longer one, or that slight changes in wording can drastically improve response accuracy [189]. While many expert practitioners develop systematic testing, version control, and performance measurement approaches to prompt development, much of the success of prompt engineering relies on the practitioner's creativity and knowledge of the particular model and training patterns. A prompt that works well for one model may fail to produce better than random for another model, and practitioners have to experiment and adapt their approach.

2.2.2 *In-Context Learning*

In-context learning is one of the most intriguing features of LLMs, allowing them to learn new tasks purely from examples in the input context [36]. Unlike traditional machine learning approaches requiring explicit training, LLMs can often infer patterns from a few examples and then apply them to new instances. How LLMs generally accomplish this capacity remains somewhat of a mystery, but studies indicate that they seem to develop abilities that resemble meta-learning by encountering innumerable instances of patterns in different contexts over the course of pre-training [317].

Few-shot learning, in particular, encapsulates this ability, allowing to teach the LLM with respect to a new task (such as custom text classification or formatting instruction) through as few as 2-5 examples of input-output pairs [36]. Thanks to this technique, the model learns specific mappings and the underlying transformation rules, and can generalize from concrete examples to abstract patterns. For maximizing effectiveness, it is essential to select the right number and "type" of examples, as diverse and representative examples tend to outperform redundant ones. Additionally, recent studies [191, 342] demonstrated that LLMs may exhibit recency bias, giving greater weight to the last examples provided, and this means that the ordering of examples is important as well. The trade-off between providing a sufficient number of examples and maintaining enough context space for the prompt requires special care, which often means prioritizing quality over quantity. The phenomenon highlights interesting features of LLM cognition – models can extract abstract rules from concrete inputs, generalize to different surface forms, and even combine different learned patterns in a single context [188, 323]. However, after a certain number of examples, performance typically plateaus; complex tasks may require more demonstrations than the context window allows, and learned patterns do not persist across conversations and require new examples each time.

2.2.3 *Retrieval-Augmented Generation*

Retrieval-Augmented Generation (RAG) specifically addresses LLMs' core shortcoming: a LLM's knowledge base remains frozen at training time, creating substantial gaps in both its current information and domain-specific knowledge [176]. RAG combines LLM generative capabilities with information retrieval, so that models can access external knowledge bases during generation.

At a high level, the technique operates in two steps: retrieval and generation. Given a query, the system first searches external knowledge databases (documents, databases, or repositories) to find relevant information. The retrieved information is then added to the LLM's context. To the extent that the retrieved content is correct and accurate, it serves as factual grounding for the LLM in future generations. The model generates responses based

on both its pre-trained knowledge and more current, specific information. Knowledge base construction is the most important step of a successful RAG system. The documents are chunked into "pieces" that are more manageable, encoded into dense vector representations by using embedding models, and then stored in vector databases optimized for similarity searches. The quality and structure of the knowledge base directly determine the performance of the system: the better structured, complete, and current the information is, the more accurate the LLM response will be.

Retrieval strategies vary in sophistication. Simple similarity search compares query embeddings with document embeddings, returning the most similar chunks. More advanced types of retrieval employ query expansion, in which the original queries can be expanded with other similar terms, or can be restructured to attempt to get better coverage of the original information contained in the user queries. Hybrid retrieval uses both dense vector similarity search and standard keyword-based searches to capture semantic similarity and exact matches. However, RAG introduces new challenges. The quality of the retrieval is directly correlated to the quality of the generation itself: if the retrieval returns incomplete or old information, it may mislead the model or create blind spots. Some advanced retrieval systems address this issue through a multi-hop retrieval for complex queries, exploiting a refined real-time ranking system based on relevance or confidence scoring to help models determine the reliability of retrieved information.

2.2.4 *Fine-tuning Approaches*

Fine-tuning is the most computationally intensive adaptation approach, since it involves further training on task-specific datasets to modify model parameters and behavior. Unlike prompt engineering or RAG, the model's current parameters get modified when using the fine-tuning approach, enabling specialization for particular domains or applications.

Full fine-tuning allows for the continuation of the training process with a pre-trained model on target domain-specific data. This means that all parameters can be modified, in order to adapt the

model to completely new tasks, writing styles, or knowledge domains. The process requires a large amount of computational resources and careful hyperparameter tuning to avoid the issues of catastrophic forgetting, where the model loses previously learned capabilities while acquiring new ones. Fine-tuning is a supervised learning method that uses task-specific input and output pairs. For example, a model fine-tuned for general medical applications can be trained on clinical notes and diagnostic summaries, while a code generation model can be fine-tuned on some programming problems and their solutions. The performance and robustness of the resulting model depend on the quality and diversity of the training data.

Parameter-efficient fine-tuning methods have emerged as a practical substitute to full fine-tuning, allowing for a high-quality model with far fewer computational requirements [89, 137, 196]. One of the most commonly used parameter-efficient methods is Low-Rank Adaptation (LoRA) and works by appending smaller trainable matrices that capture task-specific adaptations without modifying the original model weights [137]. During inference, these adaptation matrices are combined with the frozen base model, achieving performance comparable to full fine-tuning with a fraction of the computational cost. Adapters are another parameter-efficient way to fine-tune models; these methods add small neural network modules between transformer layers. These modules learn similar task-specific transformations, but they keep the base model frozen; furthermore, many adapters can exist for different tasks, and users can swap between them at inference time. Instruction tuning is a variant of fine-tuning that is developing in popularity, which allows models to be trained to follow natural language instructions across diverse tasks. Rather than fine-tuning for specific applications, instruction-tuned models learn to interpret and execute different commands, improving their ability to understand user intent and generate appropriate responses. Basically, this approach bridges the gap between general-purpose models and task-specific fine-tuning.

Fine-tuning is complex, and many factors have to be managed, including (1) the quality and quantity of the training data, (2) the fine-tuning approach, and (3) metrics for evaluation. In particular, small datasets may lead to overfitting, which creates models

that perform well on training examples but poorly on real-world inputs. We can rely on techniques including regularization, data augmentation, and validation to mitigate those issues and produce even stronger models. The selection of a fine-tuning method varies based on resources, data limitations, and performance requirements. Full fine-tuning offers the most flexibility, but takes a significant amount of computational resources and investments. Parameter-efficient methods provide practical tradeoffs, while instruction tuning offers a more generalized and broad adaptability across many tasks. Each fine-tuning approach is tailored to meet different needs within the available LLM adaptation strategies.

2.2.5 *Agentic Systems*

Agentic systems represent an adaptation approach that extends LLM capabilities beyond simple text generation through the implementation of architectures that support autonomous planning, tool usage, and multi-step reasoning [322]. Similar to some of the adaptation techniques previously discussed, agentic approaches do not directly modify model parameters or input context, but rather orchestrate and leverage LLMs within more complex computational frameworks. Three main components are integrated in single-agent architectures. The first are planning modules, which use decomposition algorithms to break down complex goals into a series of simple actions and adjust resolution strategies dynamically based on intermediate outcomes [328]. Persistent memory systems represent the second component category, and their purpose is to preserve information and results (e.g., task state representations) obtained during execution sessions, enabling the accumulation of experiential knowledge useful not only for solving the current problem but also for similar problems [228]. Integration with external tools is the last component category, and it enables programmatic access to APIs, databases, and computational resources, which significantly expands LLM capabilities – as well as reduces hallucinations – and the available solution space [256]. Multi-agent systems implement coordination protocols among specialized LLMs, based on diverse topologies ranging from hierarchical structures with coordinator and executor agents, to peer-to-peer networks for distributed expertise sharing,

to frameworks that employ critical debate to reach consensus on a given problem [96, 183]. Adaptation in agentic systems operates at two levels. At the first level, there is an individual adaptation, where each agent is specialized through the techniques discussed within this section (e.g., fine-tuning, prompt engineering, or RAG). The second level relates to systemic coordination, which is implemented through communication protocols that define standard formats for information exchange and rules for activity synchronization [133, 320]. This coordination – and its systemic complexity – introduces specific failure modes, such as error propagation through dependency graphs, coordination deadlocks, and unstable feedback cycles [305], which need to be addressed for a successful implementation. Notwithstanding these difficulties, agentic systems are proving to be more effective than single LLMs, especially in specific domains such as multi-step mathematical problem solving, software code generation and debugging, and research tasks, making them a viable paradigm for applications needing sophisticated autonomy in decision-making.

2.3 CHALLENGES AND LIMITATIONS

Despite their significant potential, LLMs have limitations that hinder their utility and create challenges in practice. These limitations should be considered when deploying LLM-based solutions because they have practical, ethics, and technical implications.

2.3.1 *Technical Limitations*

From a technical perspective, there are multiple limitations.

The most relevant limitation is hallucination, which is when models produce a plausible output that is simply incorrect [152]. According to Ji et al, hallucination occurs because LLMs predict the next tokens or continuation of prompts [152] using probability distributions. In other words, when models are exposed to new topics or edge cases, their generating mechanism may produce answers that appear to be verifiable but actually contain fabricated information, references, or even made-up narratives. All this is problematic in mission-critical applications where inaccu-

racies could result in potentially dangerous or life-threatening situations.

Knowledge cutoffs and hallucinations are closely related, because models have temporal limits beyond which they lack knowledge or information. LLMs cannot update their knowledge after the training phase, make adjustments, or identify any subsequent developments that took place after the dataset was gathered. [91, 233]. This restriction causes a widening gap between model knowledge and current reality and is especially troublesome in fields that are changing quickly.

Another point of focus is related to attention mechanisms. Zhuang et al. argue that despite the fact that these mechanisms theoretically enable LLMs to account for all of their previous tokens, memory and computation constraints make this impractical [346]. In lengthy conversations or documents, the fixed context length (i.e., the total number of input and output tokens) produces a sliding window effect that may cause LLMs to lose track of prior information, leading to inconsistent responses or missing details [161]. Furthermore, because attention mechanisms are quadratic in complexity, lengthening the context raises memory and computational costs dramatically, causing serious scalability issues that restrict opportunities for practical implementation [333].

Finally, the non-deterministic nature of LLM generation poses additional challenges because it directly impacts the reliability of the generated output. Models sample from probability distributions rather than deterministic rules, so identical input can generate different output. Although this variability in responses might be useful for creative purposes, it is problematic for systems requiring reliable, consistent, and reproducible outputs [82].

2.3.2 *Ethical and Social Concerns*

LLMs also introduce ethical and social questions on the human-generated content used to train LLMs and their encroaching presence in society [82].

The first ethical concern is bias. As identified by Navigli et al. [207] and Kotek et al. [166], LLMs reproduce and propagate bias inherent in the data on which they are trained, and the results may also reinforce biases and stereotypes related to gender, race,

religion, and other protected demographic characteristics that may not always be easy to identify. In scenarios such as hiring recommendations, education assessments, and healthcare their outputs may systematically harm specific groups, despite appearing objective and neutral. The lack of transparency into LLMs' decision-making process compounds these concerns [292, 338]. In particular, Turpin et al. [292] highlight that, unlike traditional software that relies on traceable logic, LLMs' use of dense neural networks makes it difficult to understand how they reach their answers, limiting the ability to identify errors, correct problematic behavior, or provide explanations to affected users. The ease with which LLMs can produce deceptive content makes these problems even more concerning from a societal perspective. The capacities of LLMs to produce human-like content, containing convincing fake news, disturbing phishing content, and complex propaganda, indicate that LLMs can be utilized to generate misinformation unintentionally or intentionally at scale and speed the world has never seen, especially when paired with automated distribution systems [152].

Last, there are privacy concerns that may arise from several sources, such as the potential leakage of personal information from training data and user interaction with LLMs. The social issues also extend to employment displacement, student cheating, and changes in how human beings engage in their own creative practices with LLM assistance. These ethical and social concerns need engagement from developers, policymakers, and society at large to take responsible action and adopt new models of governance that keep pace with the quickly evolving capabilities of LLMs.

2.3.3 *Domain-Specific Challenges*

LLM deployment discloses a disconnect between statistical coherence and operational validity in specific contexts. The evaluation approach is the root cause of this issue, given that existing metrics, such as BLEU or ROUGE, can confirm the text's surface similarity, but they are unable to evaluate the semantic and functional correctness that confirms domain-specific proficiency, as discussed below.

2.3.3.1 *Software Engineering Domain*

Software system correctness exceeds mere syntax checking because it encompasses many other aspects that cannot be assessed by surface-level analysis. There is little middle ground that conventional NLP metrics can capture [104], and code generation is an example of this complexity. LLMs can produce "compilable" code, but they struggle with the unwritten rules of a codebase (e.g., its architectural patterns and performance constraints). But software engineering also means adherence to established practices and standards that demand domain knowledge and skills, rather than merely identifying linguistic patterns [271]. The variability of LLM output does not fit well with these practices, and, consequently, there is a need for new approaches to integrating AI assistance into these proven methodologies.

2.3.3.2 *Healthcare Domain*

In healthcare applications there are similar issues, but there are also life-critical outcomes that change the entire risk-benefit equation for LLM deployment [21, 280]. The margin for error is virtually nonexistent, and the consequences of failure may potentially lead to catastrophic results, creating limits that are different from conventional AI deployments. Healthcare practitioners need recommendations that are clinically correct and transparent, in order to verify, question, and incorporate them into their practice. Patient safety in clinical practice adds additional complexity that necessitates sophisticated reasoning about options, treatments, pharmacological interactions, patient-specific characteristics, etc. The opaque nature of the LLM and the fact that statistical correlation may not capture causal relationships critical for patient safety in this field, do not align with these expectations.

Perhaps more importantly, the healthcare regulatory environment drastically restricts deployment options, resulting in extensive chains of validation requirements and audit trails that LLM-based applications currently do not meet. All this creates a dichotomy where the domains most likely to benefit from AI assistance are also those most resistant to current LLM capabilities due to their safety and reliability requirements.

Part II

LEVERAGING LLMS FOR SOFTWARE ENGINEERING

This part presents our contributions to applying *LLMs* across the software development lifecycle, from requirements engineering to code generation and issue management. We introduce novel approaches that leverage LLMs' natural language processing capabilities to automate traditional software engineering tasks and enhance practitioner productivity. The chapters provide empirical evidence that the proposed approaches make LLM effective in improving software quality while reducing manual effort in complex engineering processes.

3

ECHO: AN APPROACH TO ENHANCE USE CASE QUALITY EXPLOITING LARGE LANGUAGE MODELS

3.1 INTRODUCTION

UML use cases are widely used in software engineering to express the functional requirements of a system[146]. The main advantage of use case descriptions is that they are written in natural language, and thus can be easily understood also by stakeholders, to verify if the described functional requirements correspond to their needs [58]. Nevertheless, this key advantage can also result in their main drawback, as the absence of precise rules can lead to poorly-written use cases. Let us observe that the quality of use cases is critical for the quality of all the other software artifacts deriving from them, such as more detailed UML models, including sequence diagrams and analysis object models [14, 32, 68, 107, 234, 241, 263], and then of the final product. Accurate use cases are also critical for deriving better software functional size measures, such as Use Case Points (UCP) [159] employed for effort/cost estimation, benchmarking, productivity assessment, etc. [23]. Producing high-quality use cases is time-consuming due also to the absence of adequate tool support.

Given the importance of the quality of use cases, many attempts have been made to apply Natural Language Processing (NLP) techniques to reduce the risks of ambiguity, incorrectness, and incompleteness [20, 26, 45, 105, 110, 230, 329]. However, as highlighted by Zhao et al. [339], these approaches have found limited adoptions by practitioners, mostly due to technological limitations of past NLP approaches. As discussed in 2, recent advances in LLMs present new opportunities for automating natural language-based software engineering tasks.

To the best of our knowledge, LLMs have not yet been applied for supporting requirements engineers in tasks involving natural language and in particular for improving the quality of use cases.

To verify whether LLMs can be effective also in this domain, we propose ECHO (**E**ffective **C**ommunication with llms to en**H**ance the quality **O**f use cases), a novel approach based on LLMs for supporting software engineers in enhancing the quality of UML use cases. ECHO consists of two main elements: (1) a novel co-prompt engineering approach, which lets the LLM learn how to react to users' inputs and produce the expected output. This technique allows us to design a suitable prompt for the LLM to obtain the qualitative analysis of use cases under our identified criteria (prompt P1 in the sequel); and (2) an iterative process to enhance the quality of use cases. This process allows software engineers to interact with the LLM which, starting from prompt P1, can give accurate suggestions according to the expected quality criteria, consider the practitioners' feedback on the suggested improvements (prompt P2), and formulate the enhanced version of the use cases based on the agreed suggestions (prompt P3).

To validate our approach we have performed a study aiming to address the following research question:

RQ3.1: *Can the ECHO approach support software engineers to improve the quality of UML use cases?*

We instantiate ECHO using ChatGPT Web interface. The study involved seven software engineering professionals. One of them acted as the oracle, and the other six were randomly assigned to the experimental group (i.e., they used ECHO to improve use cases) or to the control group (i.e., they manually improved use cases). Both groups were given the same use cases and asked to complete questionnaires. The oracle assessed the quality of the enhanced use cases at the final stage, unaware of the group that improved them.

The outcome of this research can contribute to designing effective prompts to interact with language models and to develop approaches for enhancing the quality of software requirements and estimating software development efforts, leading to improved software development project outcomes.

3.2 RELATED WORK AND MOTIVATION

This section provides an overview of the existing research on use case quality assessment and NLP approaches for requirements analysis. We then motivate our work by highlighting the opportunities for enhancing use case quality through systematic prompt engineering and LLMs.

3.2.1 *Quality of Use Case Descriptions*

UML use cases are a software engineering technique that specifies system requirements in natural language. However, creating high-quality use case descriptions is challenging also since no precise rules are available. To be considered high-quality, use case descriptions must meet various criteria, including correctness, completeness, and clarity [287, 295]. Correctness requires the use case to describe the system's behavior accurately (i.e., the steps must be in the correct order, and should be atomic). Completeness demands that all possible scenarios, including alternative and error scenarios, be covered. Clarity mandates that all stakeholders can easily understand the use case without ambiguity. In addition, the use case must follow a logical sequence of events, provide the same level of detail for each step, and use consistent terminology.

The quality of use cases is particularly critical for deriving accurate functional size measures such as UCP [159]. UCP estimation relies on identifying and counting transactions within use case scenarios - a process that is directly affected by use case quality. Incomplete use cases may lead to missing transactions and underestimated project size. Incorrect use cases can result in wrongly identified or merged system actions, impacting transaction count accuracy. Unclear use cases introduce ambiguity in transaction identification, leading to inconsistent size estimates across different estimators.

However, manual assessment of these quality criteria is time-consuming and subjective [116], highlighting the need for automated approaches that can systematically evaluate use case quality while maintaining consistency across different evaluators and contexts.

3.2.2 *NLP Approaches for Requirements Analysis*

The early 2000s marked the beginning of systematic efforts to automate requirements quality assessment through computational approaches. Tools such as ARM (Automated Requirements Measurement), originally developed by NASA and later reconstructed by Carlson and Laplante [45], established the foundation for pattern-based requirements analysis. ARM introduced quality indicators including imperatives, weak phrases, and optional terms to identify potential ambiguities.

The pattern-matching approach was further refined with tools like RETA (Requirements Template Analyzer) [20], which automated conformance checking against established templates such as EARS (Easy Approach to Requirements Syntax). These early approaches established the principle that linguistic patterns could be systematically identified and used to assess requirements quality. However, Bajceta et al.'s comparative study [26] revealed significant limitations in these traditional approaches. While tools could achieve high recall rates, with some reaching up to 98%, precision remained problematic, often generating numerous false positives that limited practical adoption. The study found that domain-specific tools like RCM, developed specifically for the railway industry, achieved better precision at 68% compared to generic tools, highlighting the critical importance of domain knowledge in requirements analysis.

The limitations of purely rule-based systems led to the exploration of machine learning approaches. Parra et al. [230] demonstrated that machine learning classifiers could effectively combine multiple quality metrics to achieve more nuanced quality assessments compared to the rigid linear combinations used in traditional approaches. Their work showed that supervised learning could better capture the complex relationships between different quality factors, though it required substantial training data and domain expertise to achieve good performance.

The evolution toward more sophisticated analysis continued with the development of specialized tools targeting specific types of ambiguity. TAPHSIR [105] represents a notable advancement in addressing anaphoric ambiguity, combining feature-based machine learning for detection with BERT-based language models

for anaphora resolution. The tool achieved perfect recall in detecting ambiguous cases, including unacknowledged ambiguity where different individuals interpret the same requirement differently without realizing the ambiguity exists. However, TAPHSIR's precision of approximately 60% still required significant manual effort to filter false positives, and its effectiveness was limited to a specific type of ambiguity.

Recent industrial applications have demonstrated the potential of multi-modal approaches. Ye et al.'s Rebot [329] introduced a neural network-based system that combines both textual and visual features from requirements documents, achieving 81.3% accuracy in classifying requirements quality into three levels. Their work highlighted the importance of considering not just textual content but also visual layout and presentation quality. However, Rebot's evaluation was limited to a single industrial context, and users found the generated suggestions too simplistic to effectively guide revision work.

The requirements engineering community has also invested significant effort in developing comprehensive frameworks for systematizing quality factors. Frattini et al. [117] proposed an ontology of quality factors that provides a structured approach to understanding and categorizing the various dimensions of requirements quality. Their systematic analysis of 105 primary studies identified over 200 unique quality factors, revealing both the richness and fragmentation of the field. However, this proliferation of quality factors has led to an overload problem, with many factors lacking empirical validation and practical relevance.

Domain-specific applications have shown promise but also revealed the challenges of generalization. Ferrari et al. [110] developed NLP patterns specifically for the railway domain, demonstrating that incorporating domain knowledge and context-specific patterns leads to more effective ambiguity detection compared to generic approaches. However, such domain-specific solutions require substantial customization and expertise transfer to be applicable in other contexts.

Despite decades of research and development, several fundamental limitations persist across existing NLP approaches for requirements analysis. Traditional pattern-based systems struggle with context-dependent ambiguities and generate excessive false

positives that limit practical adoption. Machine learning approaches require substantial training data and domain expertise, while their black-box nature makes it difficult for practitioners to understand and trust their recommendations. Domain-specific tools show better performance but lack generalizability, and most existing tools operate in isolation rather than as part of integrated development workflows.

The evidence from comparative studies suggests that no single approach effectively addresses all quality issues in requirements, and the field has yet to achieve the level of automation and accuracy that would enable widespread industrial adoption. As noted by Zhao et al. [339], many existing tools have found limited adoption by practitioners, primarily due to technological limitations and usability issues.

3.2.3 *Motivation of Our Work*

The situation described in the previous section creates a clear opportunity for more sophisticated approaches that can leverage the contextual understanding and reasoning capabilities of modern LLMs to address the longstanding challenges in automated requirements analysis (as discussed in Chapter 2). Nonetheless, LLMs' effectiveness is closely tied to prompt quality. Indeed, LLMs can provide different answers to essentially identical questions when formulated differently, making prompt engineering crucial for achieving reliable results. To better understand the importance of prompt engineering, see the example in Fig. 3.1, where we asked ChatGPT to produce a specific response to improve use case quality. Without providing ChatGPT with the right prompt, the tool generates an incomplete and undetailed response, which does not consider preconditions and does not comprise details for all the criteria and aspects we need.

As we can note, the default *"behavior"* of ChatGPT is inadequate to achieve the desired result. Indeed, ChatGPT does not consider the post-conditions and the error scenarios in its analysis and suggestions. Therefore, we need to find the correct prompts and rules to instruct an LLM for improving use cases quality, which motivated our approach, as discussed in detail in the next section.

Authors

I want you to analyze the following use case in terms of completeness, correctness, and clarity and provide suggestions to improve the basic flow, alternative flows, actor descriptions, preconditions and postconditions, and error handling.

ID: CPM:US2.002
Title: Add a new alert
Actors: Doctor
Preconditions:
- The main scenario of CPM:US2.001 has been executed successfully

Main scenario:
1. The user clicks "Add alert".
2. The system shows a form to add a new alert.
3. The user fills in the alert fields.
4. The user presses "Save"
5. The system saves the changes.
6. The system shows a confirmation message, stating "The new alert has been saved".
The system returns to the results list of configured alerts.

Alternative flow no. 01:
4. At step 4 of the main scenario, the user presses "Cancel".

ChatGPT

Completeness:
The use case seems complete, but it lacks some essential details, such as the list of alert fields, the format of data, and the input validation requirements.

Correctness:
The use case seems correct, but it would be better to specify the user's access level to add an alert, as this information is essential for security and compliance purposes.

Clarity:
The use case is reasonably clear, but some details are missing, and it would benefit from additional clarification and details.

Suggestions for improvement:
- Specify the list of alert fields and their format, including input validation requirements, to ensure that the user provides the necessary and accurate information.
- Specify the user's access level required to add an alert to comply with security and compliance policies.
- Clarify what happens if the user provides invalid or incomplete data, and how the system handles such errors.
- Consider adding alternative flows that handle exceptional situations, such as network errors or system failures, to prevent data loss and ensure the user's safety.
- Consider adding additional details to the actor description, such as their role, responsibilities, and skills, to improve the use case's clarity and consistency.

Figure 3.1: Unoptimized instructions provided to ChatGPT.

3.3 THE ECHO APPROACH

This section illustrates ECHO which comprises two primary phases: 1) Co-Prompting Method Engineering, 2) Feedback-Based process to enhance the quality of use cases.

3.3.1 *Co-Prompting Method Engineering*

Prompt engineering is a creative fine-tuning approach with multiple promptings until we get the desired responses from the model. Some prompt engineering patterns exist. Among them, the "Persona pattern," gives the LLM a persona or role to play when generating output [314]. This pattern intends to give the LLM a "role" that helps it to select what types of output to generate and what details to focus on. Based on this pattern, we set a role for the LLM,

i.e., starting our prompt with the words *"Act as a ..."* (e.g., *"Act as a software engineer..."*) and then add as many details as possible. The answer will be more pertinent to the role indicated.

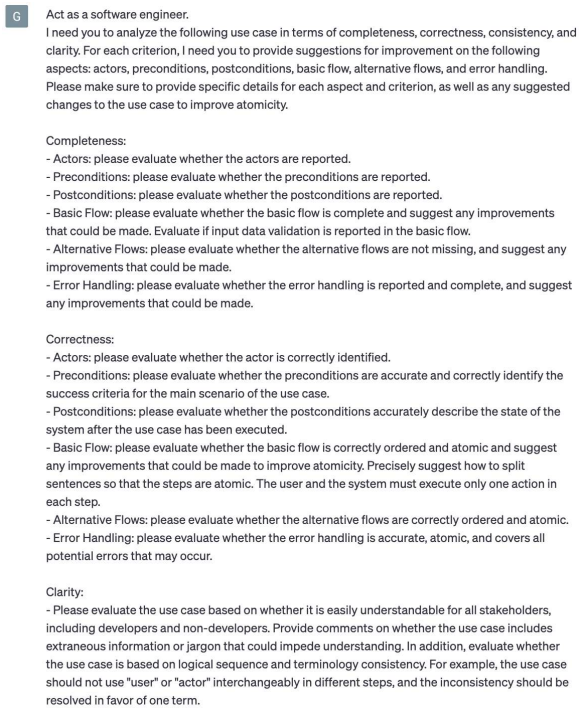
However, this approach still makes a detailed analysis of the quality of use cases hard to obtain. This issue concerns the high number of quality criteria and aspects the LMM must consider simultaneously, acting in that particular role. Therefore, it is necessary to proceed with subsequent feedback to allow the LLM to formulate a more precise analysis, such as: *"You did not consider the preconditions. Moreover, I need details for each criterion regarding the following aspects: basic flow, alternative flows, actor descriptions, preconditions, postconditions, and error handling."*

This process can continue with constant feedback and tuning until we obtain the desired result. Taking into account the above considerations, the question is: having the desired analysis, how can we design the right prompt to instruct the LLM to produce exactly that output? Aiming to find an answer, we decided to use ChatGPT as a prompt engineering companion, asking it how to provide input to enable it to generate the desired results, as follows: *"Act as a prompt engineer. Write the prompt I should have written so that you understood how to provide an analysis similar to the last one."* This led ChatGPT to suggest the correct instructions to get that particular response. The final version of the prompt (**P1**) to send to ChatGPT in order to provide a preliminary use case analysis is shown in Fig. 3.2, while the ChatGPT's response to P1 is shown in Fig. 3.3.

As we can see, the response produced by the tool is now complete and detailed because it comprises all the criteria and, for each criterion, the aspects we requested to improve the use case quality.

3.3.2 *Iterative process to enhance the quality of Use Cases*

The second stage of our approach is related to the design of an effective process that, starting with the initial qualitative analysis of a use case using the prompt P1, (i) allows us to tune the analysis, considering the practitioners' feedback iteratively, and (ii) produces the enhanced version of the use case.



Act as a software engineer.
I need you to analyze the following use case in terms of completeness, correctness, consistency, and clarity. For each criterion, I need you to provide suggestions for improvement on the following aspects: actors, preconditions, postconditions, basic flow, alternative flows, and error handling. Please make sure to provide specific details for each aspect and criterion, as well as any suggested changes to the use case to improve atomicity.

Completeness:
- Actors: please evaluate whether the actors are reported.
- Preconditions: please evaluate whether the preconditions are reported.
- Postconditions: please evaluate whether the postconditions are reported.
- Basic Flow: please evaluate whether the basic flow is complete and suggest any improvements that could be made. Evaluate if input data validation is reported in the basic flow.
- Alternative Flows: please evaluate whether the alternative flows are not missing, and suggest any improvements that could be made.
- Error Handling: please evaluate whether the error handling is reported and complete, and suggest any improvements that could be made.

Correctness:
- Actors: please evaluate whether the actor is correctly identified.
- Preconditions: please evaluate whether the preconditions are accurate and correctly identify the success criteria for the main scenario of the use case.
- Postconditions: please evaluate whether the postconditions accurately describe the state of the system after the use case has been executed.
- Basic Flow: please evaluate whether the basic flow is correctly ordered and atomic and suggest any improvements that could be made to improve atomicity. Precisely suggest how to split sentences so that the steps are atomic. The user and the system must execute only one action in each step.
- Alternative Flows: please evaluate whether the alternative flows are correctly ordered and atomic.
- Error Handling: please evaluate whether the error handling is accurate, atomic, and covers all potential errors that may occur.

Clarity:
- Please evaluate the use case based on whether it is easily understandable for all stakeholders, including developers and non-developers. Provide comments on whether the use case includes extraneous information or jargon that could impede understanding. In addition, evaluate whether the use case is based on logical sequence and terminology consistency. For example, the use case should not use "user" or "actor" interchangeably in different steps, and the inconsistency should be resolved in favor of one term.

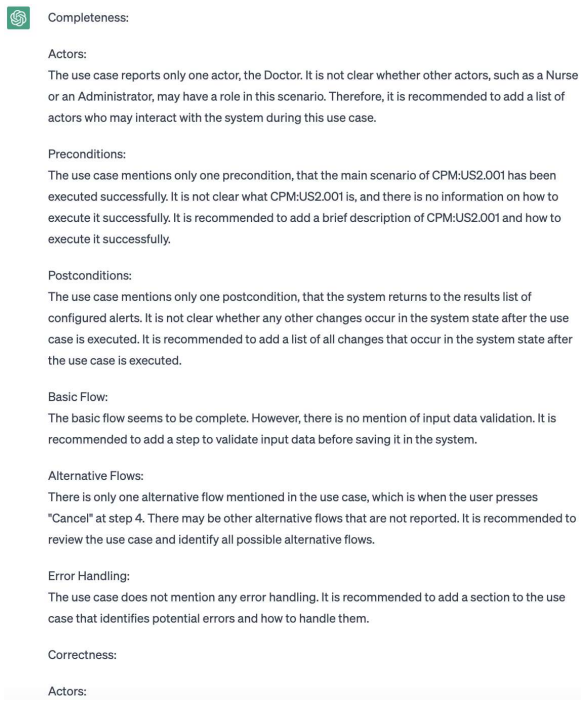
Figure 3.2: Prompt P1: Starting Use Case Analysis.

Feedback is critical because we want ChatGPT able to remove suggestions from the qualitative analysis or simply need more details on a specific criterion. For example, let us suppose that ChatGPT provides the following suggestion to improve the use case:

> *"The use of examples or screenshots could also enhance clarity."*

While the suggestion is generally valid, adding examples of values for input (or output) fields does not necessarily make the use case clearer. In this case, we can ask ChatGPT to ignore the suggestion in its analysis using the prompt (**P2**) shown in Fig. 3.4, where the *"SUGGESTIONS"* placeholder represents the list of analysis suggestions we want the tool to ignore.

On the other hand, to ask ChatGPT to provide the improved version of a use case, the prompt (**P3**) to employ is shown in Fig. 3.5.



Completeness:

Actors:
The use case reports only one actor, the Doctor. It is not clear whether other actors, such as a Nurse or an Administrator, may have a role in this scenario. Therefore, it is recommended to add a list of actors who may interact with the system during this use case.

Preconditions:
The use case mentions only one precondition, that the main scenario of CPM:US2.001 has been executed successfully. It is not clear what CPM:US2.001 is, and there is no information on how to execute it successfully. It is recommended to add a brief description of CPM:US2.001 and how to execute it successfully.

Postconditions:
The use case mentions only one postcondition, that the system returns to the results list of configured alerts. It is not clear whether any other changes occur in the system state after the use case is executed. It is recommended to add a list of all changes that occur in the system state after the use case is executed.

Basic Flow:
The basic flow seems to be complete. However, there is no mention of input data validation. It is recommended to add a step to validate input data before saving it in the system.

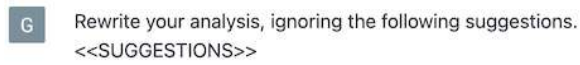
Alternative Flows:
There is only one alternative flow mentioned in the use case, which is when the user presses "Cancel" at step 4. There may be other alternative flows that are not reported. It is recommended to review the use case and identify all possible alternative flows.

Error Handling:
The use case does not mention any error handling. It is recommended to add a section to the use case that identifies potential errors and how to handle them.

Correctness:

Actors:

Figure 3.3: ChatGPT analysis of a use case in response to P1



G Rewrite your analysis, ignoring the following suggestions.
<<SUGGESTIONS>>

Figure 3.4: Prompt P2: Feedback to ChatGPT

Indeed, we can ask ChatGPT to reanalyze the improved version of the analyzed use case, and the process can continue until we are satisfied with the quality of the use case version provided by the tool.

With these building blocks to instruct the LLM, we can design the overall iterative ECHO process shown in Fig. 3.6.

Considering the limitations of the ChatGPT conversational context (maximum number of tokens 4096, including input and output tokens), i.e., the content of the dialogue between the user and the chatbot, it is necessary to repeat prompt P1 (and following P2 and P3) for each new use case to analyze.

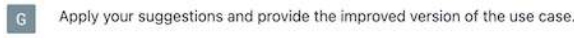


Figure 3.5: Prompt P3: Asking for the Enhanced Use Case version

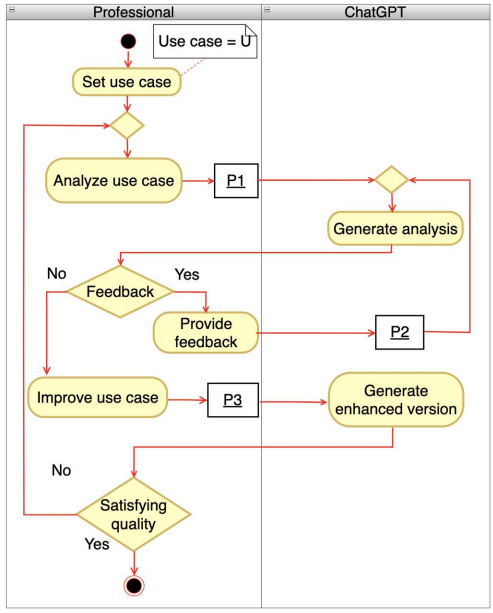


Figure 3.6: Enhancing use case quality activity diagram.

The transcripts of the conversations with ChatGPT for our prompt engineering approach are available as supplemental material on GitHub at `https://github.com/gadevito/FGG`.

3.4 EXPERIMENTAL METHOD

This section describes the design of our experimentation while the results are presented in the next section.

3.4.1 *Experiment definition and context*

To answer our research question and assess the proposed approach, we conducted an experiment involving 6 professionals from an Italian IT company and a consultant. The company was founded in 2002 and has over 50 employees. It uses UML regularly. The involved professionals had the same experience level. All partic-

ipants have worked with UML use cases for 5-7 years. They had the same distribution of roles, with all participants being business analysts or product owners. Most of them had never used LLMs. The consultant was an experienced professional with over 25 years of experience in software engineering and more than 15 years of experience in software effort/size estimation. For our experiment, we used 5 use cases, written in English, from a use case model provided by an Italian IT Company that works in the Telemedicine field. The use case model contained the requirement specifications of a microservice application based on the IoT paradigm. The use cases had different complexity, quality, and linguistic styles.

3.4.2 *Variable selection*

We identified the following variables for our experiment:

- **Quality**: high-quality use cases must include elements, such as the basic flow, alternative flows, actor descriptions, preconditions, post condition, and error handling, and must satisfy the following criteria:
  - *Correctness*: use cases must accurately describe the system's behavior regarding the steps in all flows, actors, and pre and post-conditions.
  - *Completeness*: the steps in all flows must be present and in the correct order, all the actors must be accurately described, and pre and post-conditions must be present and complete.
  - *Clarity*: use cases must be clear, without ambiguities, and easily understandable by stakeholders. They must not have discrepancies, follow a logical sequence of events, provide the same level of detail for each step, and use consistent terminology.

  The above quality criteria are evaluated according to Table 3.1, where the total score (sum of all the scores of the aspects analyzed) for each criterion is calculated as follows: Poor (5-10), Fair (11-15), Sufficient (16-20), Good (21-24), Excellent (25-25).

- **Unadjusted UCP**: as a sort of sanity check, we used the number of Unadjusted Use Case Points (UUCP) of the analyzed use cases to quantitatively assess the impact of the improvement. In particular, we selected UUCP method because it is less subjective and less dependent on weights [159].

Table 3.1: Criteria and aspects for evaluating the quality of use cases

Quality Criteria	Basic Flow	Alternative flows	Actor descriptions	Preconditions & postconditions	Error handling
Completeness	*No. of missing steps*	*No. of missing or incomplete flows*	*No. of missing or inaccurately described actors*	*No. of missing or incomplete conditions*	*No. of missing or incomplete scenarios*
1. Poor	> 3	> 3	> 3	> 3	> 3
2. Fair	2-3	2-3	2-3	2-3	2-3
3. Sufficient	1	1	1	1	1
4. Good	All essential steps are present	All essential flows are present and complete	All actors are present	All essential conditions are present	All essential scenarios are present
5. Excellent	All steps are present	All alternative flows are present and complete	All actors are present and accurately described	All preconditions and postconditions are present and complete	All error scenarios are present and ensure the system's reliability
Correctness	*% of incorrect steps*	*% of incorrect flows*	*% of incorrect descriptions*	*% of incorrect conditions*	*% of incorrect scenarios*
1. Poor	> 50%	> 50%	> 50%	> 50%	> 50%
2. Fair	30 – 50%	30 – 50%	30 – 50%	30 – 50%	30 – 50%
3. Sufficient	10 – 30%	10 – 30%	10 – 30%	10 – 30%	10 – 30%
4. Good	< 10%	< 10%	< 10%	< 10%	< 10%
5. Excellent	All the steps are correct	All the alternative flows are correct	All the actor descriptions are correct	All the conditions are correct	All the error scenarios are correct
Clarity	*No. of unclear or ambiguous steps*	*No. of unclear or ambiguous flows*	*No. of unclear or ambiguous descriptions*	*No. of unclear or ambiguous conditions*	*No. of unclear or ambiguous scenarios*
1. Poor	> 3	> 3	> 3	> 3	> 3
2. Fair	2-3	2-3	2-3	2-3	2-3
3. Sufficient	1	1	1	1	1
4. Good	All essential steps are clear and unambiguous	All flows are clear and unambiguous	All actors are clear and unambiguous	All conditions are clear and unambiguous	All scenarios are clear and unambiguous
5. Excellent	All steps are unambiguous and easy to follow	All flows are unambiguous, and add value to the use case	All actors are clearly and accurately described, and their interactions are clear	All conditions are clear, unambiguous, and accurately reflect the system state	All scenarios are clear and unambiguous, and help ensure system's reliability

3.4.3 *Experiment setting*

We chose the consultant as the oracle and randomly assigned the other software engineers to the experimental and control groups. The members of the two groups did not know the group they belonged to. We asked both groups to improve the use cases provided. In particular, the experimental group was asked to use ECHO, while the control group was asked to proceed manually. As a preliminary step, we trained the experimental group's mem-

bers in an hour-long session on using ECHO. In particular, we taught them how to use the prompts we designed (described in Section IV) to ask ChatGPT to analyze the quality of use cases and get suggestions (**P1**), ask ChatGPT to consider their feedback (**P2**) and provide the new, improved version of the use case (**P3**). We also asked the experimental group to execute at least two iterations of the ECHO process to improve the use cases. We asked each group to keep track of the changes made to the initial use cases regarding the criteria and aspects listed in Table 3.1 and to fill in the questionnaire n. 1 (see Tables 3.2 and 3.3), reporting the scores for each criterion, and the time taken to improve each use case. Afterward, each group worked independently while members of each group performed the required tasks together.

Table 3.2: Questionnaire No. 1 – Quality assessment of use cases executed by the control group

Use Case	Completeness (Score: 1-5)	Correctness (Score: 1-5)	Clarity (Score: 1-5)	Number of changes	Effort (Minutes)
1	3	3	2	13	13
2	2	3	2	17	16
3	2	3	3	8	9
4	2	3	2	13	17
5	2	3	2	10	12

Table 3.3: Questionnaire No. 1 – Quality assessment of use cases executed by the Experimental group

Use Case	Completeness (Score: 1-5)	Correctness (Score: 1-5)	Clarity (Score: 1-5)	Number of changes	Number of discarded suggestions	Effort (Minutes)
1	2	3	2	16	0	5
2	2	3	2	14	0	6
3	2	3	2	14	0	3
4	2	3	2	14	0	6
5	2	2	2	20	1	6

3.4.4 *Data analysis*

The use cases improved by the two groups were subjected anonymously to the evaluation of the oracle, based on the criteria indicated in Table 3.1, who proceeded to fill in questionnaire no. 2 (see Table 3.5), reporting the scores of the various criteria and cal-

culating the number of UUCPs of the improved use cases. At the end of the experiment, we asked the experimental group to fill in questionnaire no. 3 (see Table 3.4), which included questions to rate the level of satisfaction with using ECHO.

Table 3.4: Questionnaire No. 3 – Experimental group's experience with using ECHO

No.	Questions	Answers
1	How easy was it to use ECHO to enhance the quality of the use cases? (1 Very difficult – 5 Very easy)	4.3 (Mean)
2	How useful do you think ECHO was in enhancing the quality of the use case scenarios? (1 Not useful at all - 5 Extremely useful)	4.7 (Mean)
3	How satisfied are you with ECHO in improving use cases? (1 Not satisfied at all - 5 Extremely satisfied)	4.7 (Mean)
4	Would you use ECHO to improve the quality of use cases? (1 Not probable at all - 5 Very probable)	4.3 (Mean)
5	Do you plan to experiment with ChatGPT for other software engineering tasks? (1 Not at all - 5 Sure)	4.3 (Mean)
6	If yes, which? (Requirements Engineering - System Design - Software Development - Testing - Other)	Requirements Engineering: 100% Testing: 66%

Table 3.5: Questionnaire No. 2 – Quality assessment of the enhanced use cases executed by the Oracle

	Completeness (Score: 1-5)		Correctness (Score: 1-5)		Clarity (Score: 1-5)		UUCP		
Use Case	Control Group	ECHO Group	Control Group	ECHO Group	Control Group	ECHO Group	Initial	Control Group	ECHO Group
1	4	5	4	5	5	4	23	28	33
2	4	4	5	5	5	4	30	30	30
3	4	5	5	5	4	5	20	30	30
4	4	4	4	5	4	4	15	20	20
5	4	5	4	5	4	5	13	18	18

The initial use cases, the use cases improved by the two groups, the logs of the changes made, and the ChatGPT responses in the different phases of the experiment are available as supplemental material on GitHub at: https://github.com/gadevito/FGG.

3.5 RESULTS AND DISCUSSION

Tables 3.2 and 3.3 present the outcomes of questionnaire no. 1, obtained after the two groups improved the provided use cases in

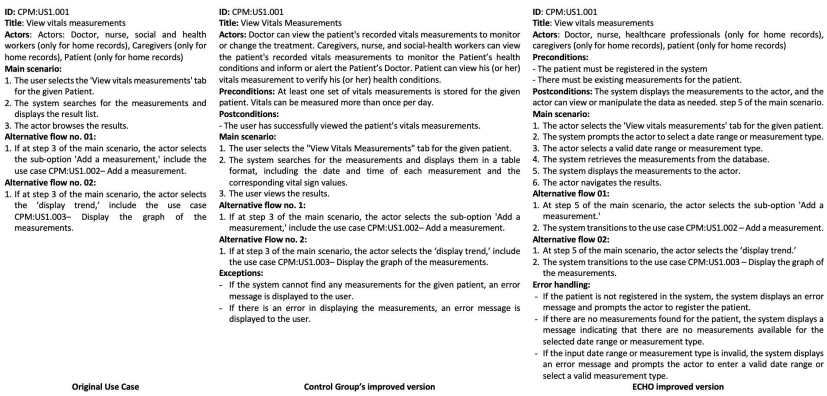


Figure 3.7: Example of a use case improved by the two groups.

the initial phase of our experiment. Fig. 3.7 illustrates an instance of a use case enhanced by the two groups.

The control group spent from 9 to 17 minutes making 12.2 changes on average per use case, while the experimental group spent from 3 to 6 minutes making 15.6 changes on average and discarding only one suggestion of those provided by ChatGPT, indicating that the use of ECHO can reduce the effort required to improve use cases.

Then, as planned, the oracle assessed the use cases improved by the two groups, calculated their UUCP values, and answered questionnaire no. 2 (to specify the quality of the enhanced use cases) whose results are shown in Table 3.5. We can observe that the oracle's assessment of the use cases improved by the control and experimental groups was similar in clarity. However, the experimental group's improved versions were slightly better in correctness and completeness, indicating that ECHO effectively supports professionals in enhancing the quality of UML use cases. As for the use case size measurement, the original versions were characterized by 101 UUCPs in total, as measured by the oracle. Differently, 126 UUCPs were obtained for the use cases enhanced by the control group and 131 UCCPs for the use case versions enhanced by the experimental group. Analyzing the number of UUCPs obtained and use cases, the Oracle found that the control group merged some system actions, leading to an error in determining the total number of transactions. This incorrectness

impacted the obtained UUPCs value, which relies on an accurate transaction count.

Finally, we evaluated satisfaction levels with using ECHO for software engineering tasks based on the experimental group's responses to questionnaire no. 3. The results are shown in Table 3.4 and indicate that users found ECHO to be easy to use, helpful, and satisfying in enhancing the quality of use cases. In particular, the average score for how easy it was to use ECHO was 4.3 out of 5, suggesting that most users found it straightforward to work with the tool. The average score for how valuable ECHO was in enhancing the quality of use case scenarios was 4.7 out of 5, indicating that users found the approach very effective in improving use cases. Additionally, the average score for satisfaction with ECHO in improving use cases was 4.7 out of 5, which implies that users were generally pleased with the results they obtained from using the tool. Regarding whether users would use ECHO to improve the quality of use cases, the average score was also 4.3 out of 5, indicating that the majority of users were likely to continue using our approach. Furthermore, the results show that participants plan to experiment with ChatGPT for other software engineering tasks, with a special interest for Requirements Engineering and Testing tasks.

> **Answer to RQ3.1:**
>
> ECHO can effectively support software engineers in improving the quality of UML use cases in terms of completeness, correctness, and clarity. Moreover, it can help to identify missing details or unclear requirements, obtain more complete transactions in the use case scenarios and, therefore, improve the accuracy of their size estimation in terms of UCPs.

3.6 THREATS TO VALIDITY

In the following we outline potential issues that may have affected our findings and how we addressed them.

Possible threats to internal validity include participant bias and the Hawthorne effect. Participants could tailor their responses to please researchers or be influenced by the knowledge of participating in an experiment. To reduce these threats, we kept the

hypothesis hidden, ensured anonymity, and divided participants into two groups operating unseen to curb behavioral alterations. The study duration was also reduced to minimize such interference. A further concern is the likelihood of placing more experienced individuals in the treatment group. To prevent this threat, we instructed the IT company to provide participants with similar experiences with UML. In addition, the oracle's bias is a conceivable risk. Nevertheless, this was mitigated by choosing a consultant from our network who boasts significant experience and carries the necessary certification. Future research could further assess this potential risk by enlisting two experts to serve as oracles and comparing their evaluations of every use case refinement iteration with ECHO. Finally, the inability to adjust hyperparameters like "temperature" using the ChatGPT Web interface presents issues due to the inherent randomness of its responses. This issue may be handled in future studies by utilizing the Completion API of ChatGPT, which facilitates hyperparameter adjustment to regulate potential randomness.

One potential threat to external validity is the limited generalizability of results due to small participant samples confined to business analysts or product owners experienced in UML use cases. Thus, it might not apply to other groups like software developers or designers. Additionally, the experiment was confined to a few use case domains. Future research should involve a more diverse participant pool and broader use case domains for better generalizability. Further, the novelty effect may manipulate results as most participants were first-time LLM users. Therefore, future studies should allow more time for participants to acquaint themselves with LLMs before experimentation.

3.7 CONCLUSION

This chapter provides initial evidence that ECHO can be effective in enhancing the quality of UML use cases, also reducing the effort with respect to a manual verification. Our study also found that starting from the use cases improved by the experimental group a more accurate size estimation in terms of UUCPs can be derived w.r.t. the ones based on the use cases enhanced by the control group, indicating that ECHO can be effective in improv-

ing the accuracy of UCP estimation, given the better quality of the use cases.

Moreover, the study revealed that our approach could help identify missing or unclear requirements, improving overall system quality. The satisfaction survey results show that the users found ECHO easy to use, helpful, and satisfying in enhancing the quality of use cases. Additionally, most users reported continuing to use ECHO to improve the quality of UML use cases. Many expressed interest in using ChatGPT for other software engineering tasks.

Future research could investigate the effectiveness of LLMs in enhancing the quality of other types of software artifacts, such as requirements documents, design specifications, and test cases. In addition, studies could explore the impact of varying the different LLM model hyperparameters.

4

AGORA: AN APPROACH FOR GENERATING ACCEPTANCE TEST CASES FROM USE CASES

4.1 INTRODUCTION

User Acceptance Testing (UAT) is a critical phase in the software development lifecycle [37], as it verifies that the final product adheres to the requirements and expectations of stakeholders. Use cases describing the interactions between the user and the system are commonly used to produce UATs. However, manually creating comprehensive UATs from use cases is time-consuming and error-prone.

This chapter presents an LLM-based approach for UAT, named AGORA, which aims to overcome these challenges through prompt engineering and hyperparameter tuning. AGORA consists of two main phases: 1) Listing test cases and 2) Writing the UAT details. Both phases leverage LLMs to analyze use cases and produce UATs. The first phase focuses on identifying test cases, while the second phase is dedicated to the detailed generation of each test case. We developed a Web Tool, implementing AGORA, and used it to validate the approach conducting a controlled experiment with an Italian IT company. The study involved seven professionals. One served as an oracle, while the other six were randomly divided into two groups to produce the UATs: an experimental group using AGORA and a control group generating UATs manually. Both groups were assigned the same use cases. The oracle assessed the quality of the produced UATs at the final stage, unaware of the group that produced them. The results highlighted that AGORA produces UATs as effectively as manual techniques but much faster. User feedback also reflects great satisfaction with AGORA. All this underscores the potential of the approach to enhance software testing efficiency and overcome the hurdles of manual UAT creation.

4.2 RELATED WORK AND MOTIVATION

4.2.1 *Approaches for Generating UAT*

Automatically producing UAT is challenging, with various proposed approaches (see Table 4.1), often leveraging NLP but limited to specific domains or formats. Nebut et al. [208] propose generating test scenarios from UML use cases but face scalability issues in complex systems. Carvalho et al. [46] develop NAT2TEST to create test cases from Controlled Natural Language requirements, suggesting future performance optimizations. Yue et al. [332] introduce a tool to convert natural language test cases into executable tests, but their method has restrictions and lacks broad generalizability. Goffi et al. [123] use Javadoc comments and NLP to generate test oracles for Java, though their prototype struggles with complex conditions. Silva et al. [262] employ Colored Petri Nets in NAT2TEST for test case generation but do not fully address completeness and consistency. Allala et al. [12] combine Model-Driven Engineering with NLP to convert user requirements into test cases, though their research is still in the early stages. Fischbach et al. [112] automate 56% of test cases from agile acceptance criteria using NLP in SPECMATE, but their method relies on language patterns and is not domain-independent.

Despite these advances, current approaches share several common limitations that hinder their widespread adoption. Most existing methods require requirements to be written in specific formats or controlled languages, creating barriers for practitioners who must adapt their natural documentation practices. The rigid writing patterns imposed by these tools increase the cognitive load on engineers and testers, who must learn new specification languages or modify their existing workflows. Furthermore, many solutions lack domain independence, limiting their applicability across different software development contexts. These constraints often result in poor integration with existing development processes, requiring significant changes to established practices and potentially disrupting team productivity.

Study	Main Features	Limitations
[208]	Formalizes use case descriptions and generates test cases.	Requires preconditions and postconditions in a specific format.
[46]	Uses Data-Flow Reactive Systems (DFRS) requirements.	Limited to requirements in controlled natural language.
[332]	Generates test cases from an RTCM language.	Limited to the writing style of test cases imposed by RTCM.
[123]	Uses Javadoc comments to complete test cases.	Comments must follow a specific pattern.
[262]	Explores test case generation using Petri Net simulation.	Interpretation of Colored Petri Nets can vary.
[12]	Generates test cases from use cases or user stories.	Requires a specified format for use cases or user stories.
[112]	Uses recursive dependency matching to formulate test cases.	Requires advanced knowledge of dependency matching.
[303]	Requirements compatible with RUCM specifications.	Limited to the use of RUCM specifications.

Table 4.1: Summary table of related works

4.2.2 *Motivation of our Work*

While LLMs show promise for software engineering tasks, their application to User Acceptance Test (UAT) generation from use cases presents specific challenges. LLMs exhibit non-deterministic behavior, are sensitive to input formulation, and have token limitations that can truncate outputs for complex use cases [29, 192].

To illustrate these challenges, consider the example in Figure 4.1, where a simple prompt generates incomplete UATs (Figure 4.2) lacking essential elements such as test IDs, preconditions, and proper flow coverage. Re-executing the same prompt yields the output in Figure 4.3, which includes "Title" and "Objective" and captures the alternative flow but not the error flow, highlighting the approach's non-determinism and lack of unique test format or predictable flow coverage.

Furthermore, in verbose scenarios, such as a 12,460-character use case with 19 event flows, the output is truncated when the LLM's token limit is reached. Complete examples are available in our online repository `https://github.com/GabrieleVassallo/AutomaticUAT`. These limitations motivate the need for a systematic approach to guide LLMs in generating comprehensive and consistent UATs from use cases.

Prompt no. 1: Generating UATs from a Use Case

Provide the user acceptance tests for the following use case:
US-1.02: Add a Measurement
Description: The user adds a spectral index measurement to historical agricultural land data.
Actors: Farmer
Main Scenario

1. *The user clicks on "New Measurement"*
2. *The system displays a form to select the spectral index*
3. *The user selects the desired spectral index*
4. *The system displays the parameters related to the given index*
5. *The user fills out the form*
6. *The user presses the "Save" button*
7. *The system adds the new measurement for the selected index*
8. *The system displays the message "Save successful"*
9. *The system displays the updated list of measurements*

Alternative Flows Alternative Flow No. 01

4a At step 4, the user presses the "Cancel" button
5a The system closes the form

Exceptions Exception Flow No. 01

7a At step 7, the save fails
8a The system displays the message "Save failed error 564"

Figure 4.1: A simple prompt to produce UATs.

4.3 THE AGORA METHOD

AGORA is an approach that unfolds in two primary phases (see Figure 4.4): 1) Listing test cases and 2) Writing the test case details. Both phases leverage LLMs to analyze use cases and produce UAT.

In the first phase, AGORA employs an LLM to examine use cases, pinpointing testable flows through sophisticated semantic analysis to grasp context and intent. It outputs a JSON-formatted structured list of test cases detailing identifiers, descriptions, flow types (main, alternative, exception), inter-use case relationships, and explicit presence within the use case. The second phase of AGORA takes the list of identified test cases. It proceeds with the detailed generation of each test case, including the precondi-

LLM output for prompt no. 1

Title: Verify Successful Addition of a Spectral Index Measurement
Test Steps:

1. *Click on the "New Measurement" button.*
2. *Verify that the system displays a form to select the spectral index.*
3. *Select a desired spectral index from the available options.*
4. *Verify that the system displays the parameters related to the given index.*
5. *Fill out the form with the necessary measurement details.*
6. *Click the "Save" button.*
7. *Verify that the system saves the new measurement for the selected index.*
8. *Verify that the system displays the message "Save successful".*
9. *Verify that the system displays the updated list of measurements, including the newly added measurement.*

Figure 4.2: LLM response for the prompt provided in Figure 4.1.

tions, involved actors, and the sequence of test steps with specific inputs and expected results. This phase is crucial to ensure the test cases are complete, accurate, and ready to be executed. We utilized GPT-4 to implement our approach, although AGORA is compatible with other models. Additionally, we created a user-friendly web application to streamline test case generation and facilitate AGORA's empirical evaluation, catering to users without deep expertise in language models or software testing.

4.3.1 *Phase 1: Listing test cases*

The first phase of the AGORA approach is crucial for outlining the functional perimeter of the UAT. This phase directly addresses the challenges posed by the context limits and the non-determinism of GPT-4. The context limitation implies that GPT-4 can process a limited number of tokens (8,192, with a maximum of 4,096 for the output)[1]. In the presence of long or complex inputs, exceeding this limit would lead to incomplete and insufficiently detailed responses. Moreover, the model's non-determinism can lead to variable responses even with the same input. All this could fail to ensure systematic and complete functional coverage. Finally, there

1 OpenAI Documentation. https://platform.openai.com/docs/api-reference

Second LLM output for prompt no. 1

Title: Add a New Spectral Index Measurement
Objective: Verify that the user can successfully add a spectral index measurement to the historical measurements of agricultural land.
Steps:
1. *Navigate to the "New Measurement" section.*
2. *Select a spectral index from the provided options.*
3. *Fill out the form with the required parameters.*
4. *Click the "Save" button.*

Test Case 2: Cancel Adding Measurement
Objective: Verify that the user can cancel the process of adding a new measurement.
Steps:

1. *Navigate to the "New Measurement" section.*
2. *Select a spectral index and start filling out the form.*
3. *Click the "Cancel" button.*

Figure 4.3: Second LLM response for the prompt provided in Figure 4.1.

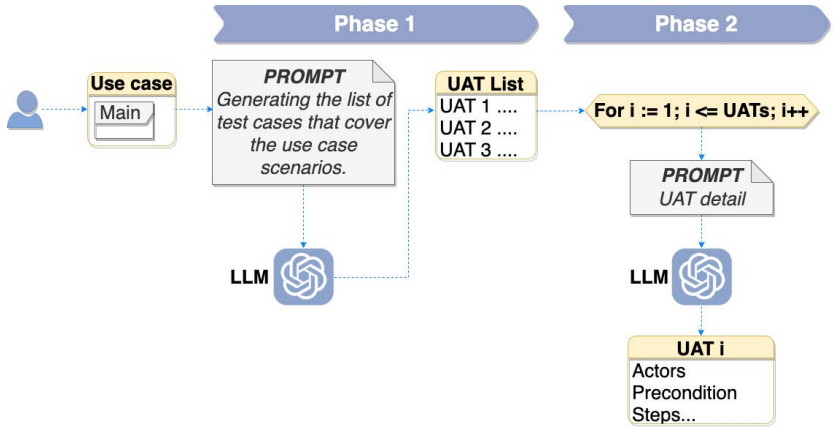


Figure 4.4: AGORA Process.

is a risk that the model may generate test cases that do not match the anticipated scenarios, wasting resources and time. In Section 5.2.2, some examples show the issues previously described.

A well-designed prompt has mitigated the described issues. We exploited the following guidelines:

1. Generate a test case for the main flow and each alternative or exception flow in the use case.

2. Structure test cases in JSON format, following a given example.
3. Exclude inexplicitly stated scenarios in the use case.
4. Avoid additional information or formatting not requested.

These guidelines optimize GPT-4's capabilities for effective test case generation. The prompt standardizes GPT-4's output into a JSON format, listing test cases with unique identifiers, descriptions, flow types, UAT indicators, and explicit presence in the original use case. This format is easily interpretable, aiding subsequent processing and integration. Figure 4.5 shows the prompt used in the first phase, and Figure 4.6 displays an example output.

The iterative prompt strategy involved evaluating LLM answers at each step to ensure they met the original requirements, refining them with subsequent prompts to fine-tune the results. This process ensures quality and accuracy before moving to the next phase.

4.3.2 *Phase 2: Writing the test case details*

The second phase of the AGORA generates detailed UAT, ensuring each test case is complete, correct, clear, and ready for execution. It aims to create unambiguous test cases that are aligned with use case scenarios and structured in a JSON format for easy understanding and implementation. The format includes:

- Precondition: Specifies the initial conditions under which the test should be executed, ensuring that the context is appropriate for the test flow.
- Actors: Lists the involved actors, ensuring that all roles necessary for the test are identified and understood.
- Steps: Details the test steps, including inputs and expected results, to provide step-by-step guidance for test execution.

Using specific prompts, GPT-4 completes test case details for each acceptance test ID, ensuring test steps are accurate and reflect the use case. This phase can be parallelized, as each test case is independent, thus optimizing efficiency and speed.

Prompt List Tests

Act as an experienced software engineer in test engineering. Produce an acceptance test using the provided use case, following these guidelines:
*** Guidelines ***:*

1. *Generate a single test case for the main flow of the provided use case.*
2. *Generate a single test case for each alternative flow or exception explicitly reported in the use case.*
3. *Provide the test case in JSON format, following the structure of the provided example.*
4. *If alternative or error scenarios are not explicitly specified in the provided use case, provide only the test case for the main flow.*
5. *Do not include additional information or markdown formatting in your response JSON.*

Here is an example of how to structure the JSON:
{"Tests": [{"Id": "Acceptance Test ID, in the format TA-three-digit progression", "Description": "brief but explanatory description of the test case, for example Personal Data Entry", "SC": "P if main flow, FA if alternative flow, FE if exception flow", "SS": "S if the flow simply includes another use case and does not add further specific steps, N otherwise", "ES": "S if the flow is explicitly present in the use case or if it is the main scenario, N otherwise", "UC": "use case ID"}]}
Answer only with the requested JSON without markdown and without adding other information.
{Use_case}

Figure 4.5: Test list prompt.

An iterative prompt engineering approach, similar to Phase 1, was used with progressive refinements. The final Phase 2 prompt is shown in Figure 4.7, and an example output is in Figure 4.8.

4.3.3 *Optimization of Determinism*

Determinism in the context of this work refers to the ability of the LLM to provide consistent and predictable results in response to similar or identical inputs. This aspect is crucial for UAT. For optimizing determinism, it is necessary to operate in multiple di-

Example result for the test list prompt.

{"Tests": [{"Id": "TA-001", "Description": "Access the 'Settings' section in the 'Configuration' interface", "SC": "P", "SS": "N", "ES": "S", "UC": "US0.001"}, {"Id": "TA-002", "Description": "Initiate 'Configure the parameters of the Settings section' after filling in the blank fields", "SC": "FA", "SS": "S", "ES": "N", "UC": "US0.001"}}}

Figure 4.6: Result of the test list prompt.

Prompt List Tests

I am providing you with a use case and the identified acceptance test IDs. Act as a software engineer experienced in test engineering to complete the acceptance test related to the use case. Respond with a well-formed JSON as in the following example.
Example:
{"Precondition": "if SC='P' precondition of the use case, otherwise insert the steps of the main flow to be executed", "Actors": "list of actors of the use case", "Test": [{"Step": "increasing numbering of the step performed", "Input": "input", "Result": "expected result of the step"}]}
Answer only with the JSON I requested, without adding any other information.
***** Start Use Case*
{use_case}
***** End Use Case*
{uat_list_from_phase1}
Provide the user acceptance test with ID = {ID_UAT}

Figure 4.7: Prompt for test case details.

rections. First and foremost, it is essential to carefully select and configure the hyperparameters of GPT-4 that influence the generation of responses, in particular:

- **Temperature:** It regulates the model's predictability and originality, with values from 0 to 1. A temperature near 1 increases randomness and diversity, while a near 0 makes the output more deterministic and predictable.

UAT detail example

{"Precondition": "At least one measurement exists in the database", "Actors": ["GP", "Relative"], "Test": [
{"Step": 1, "Input": "The actor accesses the 'Measurements' section", "Result": "The system displays an interface where it is possible to view for each parameter the measurements related to the last update"}, {"Step": 2, "Input": "The actor uses the graph viewing feature related to one of the detected parameters", "Result": "The system opens a popup with the available graphs for the selected parameter"}, {"Step": 3, "Input": "The actor navigates the interface", "Result": "The system allows the navigation of the graphs and the display of the measurements in graphic form"}] }

Figure 4.8: Result for producing the UAT details.

- **Top_p:** It refines text generation in LLMs by selecting words with high cumulative probability, balancing creativity and consistency.
- **Best_of:** It generates multiple server-side completions, selecting the 'best' response based on the lowest logarithmic probability per token.
- **Frequency_penalty:** It controls the model's tendency to repeat predictions by decreasing the likelihood of previously generated words based on their frequency in the prediction.
- **Presence_penalty:** It promotes originality by reducing word repetition, unlike the frequency penalty, which ignores prior term usage.

Therefore, according to OpenAI documentation, we set the temperature, presence penalty, and frequency penalty to 0, while best and top p to 1, making the model's responses more deterministic. Secondly, prompts must be carefully designed with clear, detailed instructions to guide GPT-4 in generating specific and relevant outputs, reducing ambiguity and increasing consistency. Examples of desired outputs and structured templates were provided to stabilize responses, following the "show, do not tell" approach [99]. Finally, after each output generation, results were

analyzed for consistency and adherence to requirements, identifying non-determinism patterns and adjusting prompts as needed.

4.3.4 *AGORA Tool*

We developed a Web tool using the Streamlit Python framework [2] and GPT-4 API [3]. The tool analyzes GPT-4 responses and displays UATs on a web page. The tool features an intuitive interface for entering use case scenarios and generating corresponding UATs, manipulating the JSON format produced by GPT-4 into a user-friendly display.

4.3.5 *Replication Package*

The replication package of our study is publicly available in the online GitHub repository https://github.com/GabrieleVassallo/AutomaticUAT and includes the following: i) the source code of the AGORA tool, ii) the use case model used for the empirical experiment, iii) the UATs produced by the control and experimental groups, and iv) the results of the experiment. The use of the AGORA tool is licensed under an International Creative Commons Attribution 4.0 license.

4.4 EMPIRICAL EXPERIMENT

This section details our empirical experiment to evaluate the proposed approach, covering the research question, hypotheses, experiment context, variables, design, and analysis plan.

4.4.1 *Research question and hypothesis*

AGORA aims to help test engineers produce high-quality UATs faster than the manual approach. Thus, we propose to answer the following research question:

2 Streamlit Framework. https://streamlit.io/
3 OpenAI Documentation. https://platform.openai.com/docs/api-reference

RQ4.1: *Can AGORA support software engineers by automating the production of UATs from use cases?*

The research question has been outlined through the formulation of the following hypotheses:

- **H1**: AGORA produces qualitatively better UATs than the manual approach.
- **H2**: AGORA speeds up UAT production from use cases.

4.4.2 *Context of the Experiment*

To address the research question and assess the proposed approach, we conducted a controlled experiment with six professionals from an Italian IT company, Kiranet, and a professional consultant. Kiranet, established in 2002 with over 50 employees, regularly uses UML in its software development process. The company's project manager selected test engineers with a good understanding of UML and UAT and similar experience levels. The consultant has over 25 years of software engineering experience. The experiment used ten use cases (available online in the repository [299]) from a telemedicine microservice, FIDTCT, provided by Kiranet, varying in complexity, quality, and linguistic styles.

4.4.3 *Variable Selection*

The following dependent variables have been identified for the experiment:
Quality: High-quality UATs must meet the following criteria:

- *Completeness*: It ensures UATs cover all the use case flows, with a test case for every non-trivial flow.
- *Clarity and Understandability*: It ensures test cases are clear and unambiguous and the steps to be executed are correct.
- *Correctness*: It ensures test cases are formally and semantically correct, verifying that actors, preconditions, postconditions, inputs, and outputs are accurate and consistent with the use case requirements.

TABLE 4.2 report the criteria, elements and evaluation formulas for *Quality* variable.

Table 4.2: Quality evaluation.

Criterion	Elements to assess	Evaluation
Completeness (COM)	Number of tested flows (FT) and Number of use case flows (NF)	$COM = FT/NF$
Clarity and Understandability (CeC)	Language simplicity, Instructions clarity, Consistency of terms, Ambiguity avoidance	$CeC = \sum_{EC=CC1}^{CC4} EC/4$
Correctness (COR)	Actors identification, Pre/postconditions, Input/output validation, Requirements compliance, Scenarios testing	$COR = \sum_{EC=CO1}^{CO7} EC/7$
Quality Value	Mean of all criteria	**Mean(COM, CeC, COR)**

Time: This variable measures the time required to produce the UATs for each use case.
The independent variable is the **approach used**, which refers to using AGORA or the manual approach.

4.4.4 *Experiment Design*

We chose the consultant as the oracle and randomly assigned test engineers to the experimental or control groups, keeping their assignments confidential. Both groups were tasked with producing UATs for given use cases; the experimental group used AGORA, while the control group did it manually. Participants were informed about data confidentiality and their right to leave the experiment at any time. The experimental group received one-hour training on using AGORA to generate UATs from use cases. Both groups were given a Word template for writing UATs and shown how to track execution times before working independently.

Table 4.3: Questionnaire on the experience with AGORA

Question	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
AGORA is easy to use					
AGORA is useful for writing UATs					
I would use AGORA for writing UATs					
I would like to try similar tools for other software engineering tasks.					

The test cases produced by the two groups were anonymously evaluated by the oracle, based on the criteria indicated in Section 4.4.3, using the form shown in Figure 4.9 to record the scores for the various criteria.

Use Case no. X																		
		No of SCENARIOS	Criteria															
			Completeness	Clarity and Understandability					Correctness								Time (seconds)	MEAN
		No. of UAT		CC1	CC2	CC3	CC4	Value	CO1	CO2	CO3	CO4	CO5	CO6	CO7	Value		
Group no. 1	Participant 1																	
	Participant 2																	
	Participant 3																	
	MEAN																	
Group no. 2	Participant 4																	
	Participant 5																	
	Participant 6																	
	MEAN																	

Figure 4.9: Form filled out by the oracle for each use case.

At the end of the experiment, we asked the experimental group to complete a satisfaction questionnaire (shown in Table 4.3) about the AGORA tool, using a Likert scale from 1 (Strongly Disagree) to 5 (Strongly Agree).

4.4.5 *Analysis Plan*

To verify hypotheses H1 and H2, we decided upon the following statistical analysis methods:

- **Hypothesis H1 (Quality of Acceptance Tests)**: To analyze the quality of UATs from the experimental and control groups, we will calculate descriptive statistics (mean, median, standard deviation) for scores assigned by the oracle on completeness, correctness, and clarity. The Mann-Whitney U test will compare the mean quality percentages between the groups at a 0.05 significance level to determine if there are statistically significant differences. The null hypothesis states no difference in average quality scores between the groups, while the alternative hypothesis suggests a significant difference.

- **Hypothesis H2 (Production Time of Acceptance Tests)**: We planned to calculate descriptive statistics (mean, median, standard deviation) for the time taken by the experimental

and control groups to produce UATs. We will use the Mann-Whitney U test with a significance level of 0.05 to compare the average times. This test assesses whether there is a significant difference in the time taken between the AGORA and control groups. The null hypothesis is that there is no difference, while the alternative hypothesis is that the AGORA group took significantly less time.

Finally, we planned to calculate the average scores for each questionnaire question 4.3 to assess user satisfaction with AGORA. The questionnaire collects qualitative feedback on users' experiences, offering insights for future improvements.

4.5 ANALYSIS OF THE RESULTS

4.5.1 *Quality of Acceptance Test Cases* (*Hypothesis H1*)

The oracle evaluated the quality of UATs based on the criteria of completeness, clarity, understandability, and correctness as described in Section 4.4.3. Table 4.4 presents the results obtained for the AGORA group, while Table 4.5 shows those for the control group. We evaluated completeness based on use case flow coverage, with both groups achieving 100%. Clarity and understandability were assessed using the criteria in Section 4.4.3, with both groups averaging 83%. Correctness, based on actors, preconditions, postconditions, inputs, outputs, and requirement validity, was 80% for both groups.

Nevertheless, the AGORA group scored better in 6 use cases (i.e., 1-5 and 7), performing worse only in 3 out of 10 cases (i.e., 8-10). The overall average quality for the control group was 87%, while for the AGORA group, it was 88%. Finally, the standard deviation for quality for the control group was 0.08, while for the AGORA group, it was 0.09. Then, we analyzed the results using the Mann-Whitney U test to compare the average quality scores between the control and AGORA groups. The test accounted for ties in the data, applying a correction. The p-value of 0.9093 indicates no statistically significant difference between the two groups in test case quality (see Figure 4.10).

Table 4.4: Experimental group UAT results

Use Case	Compl.	Clarity and Understandability	Correctn.	Average	Time(s)
1	100%	100%	71%	90%	31.96
2	100%	100%	71%	90%	22.83
3	100%	75%	71%	82%	30.42
4	100%	75%	86%	87%	33.00
5	100%	100%	100%	100%	31.46
6	100%	100%	86%	95%	22.78
7	100%	75%	86%	87%	23.16
8	100%	50%	71%	74%	28.79
9	100%	50%	71%	74%	35.56
10	100%	100%	86%	95%	32.19
Average	100%	83%	80%	88%	29.22

Table 4.5: Control group UAT results

Use Case	Compl.	Clarity and Understandability	Correctn.	Average	Time(s)
1	100%	100%	67%	89%	324.00
2	100%	100%	67%	89%	718.33
3	100%	75%	62%	79%	217.33
4	100%	75%	81%	85%	607.33
5	100%	100%	95%	98%	446.33
6	100%	100%	86%	95%	254.00
7	100%	75%	81%	85%	233.00
8	100%	50%	81%	77%	557.33
9	100%	50%	81%	77%	281.33
10	100%	100%	95%	98%	260.33
Average	100%	83%	80%	87%	389.93

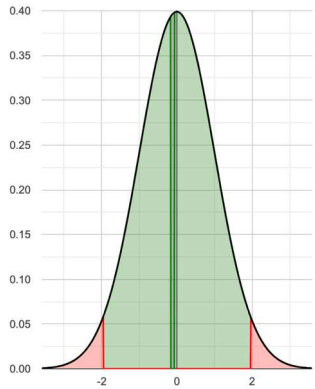


Figure 4.10: Normal distribution of the U test for hypothesis H1.

In other words, there is not enough evidence to reject the null hypothesis, which asserts that there is no difference between the groups. The very high p-value, 90.93%, indicates that if we were to reject the null hypothesis, there would be a high probability of making a Type I error, that is, rejecting a true null hypothesis. Furthermore, the test statistic Z obtained (see Figure 4.11) is -0.1139, well within the acceptance range for a 95% confidence level (between -1.96 and 1.96) This result reinforces the idea that

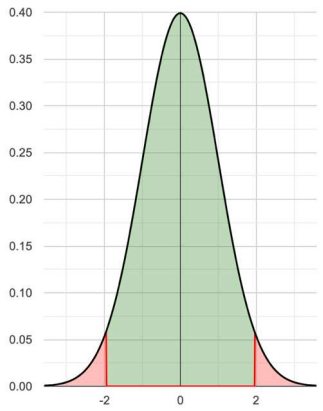


Figure 4.11: Normal distribution of the U test for hypothesis H2.

the groups have no significant difference. Regarding the effect size, the observed standardized effect size, $Z/\sqrt{(n_1 + n_2)}$, is 0.025, indicating a minor difference between groups. The common language effect size, $U_1/(n_1 n_2)$, is 0.48, showing a 48% chance that a test case from the AGORA group is of higher quality than one from the control group, suggesting no practical difference.

4.5.2 *Acceptance Test Case Production Time (Hypothesis H2)*

The time to produce UATs was measured in seconds. The control group took 3,899.33 seconds total (389.93 seconds per use case), while the AGORA group took 292.16 seconds total (29.22 seconds per use case). The medians were 30.94 seconds for AGORA and 302.67 seconds for the control group, with standard deviations of 4.67 and 180.17, respectively. We used the Mann-Whitney U test to compare the average times of the control and AGORA groups, and we found a statistically significant difference. Indeed, the p-

value obtained from the test is extremely low at 0.00001083, indicating a 0.0011% chance of a Type I error. This result strongly supports the alternative hypothesis of a significant difference in the time taken by the two groups to produce test cases. The test statistic, Z, is 4.4, outside the 95% acceptance range (-1.96 to 1.96), confirming the significance of the observed difference. Additionally, the U value of 100 is outside the 95% acceptance range (24 to 76), further reinforcing the significance of the differences. Regarding the effect size, the observed standardized value, $Z/\sqrt{(n_1 + n_2)}$, is significant (0.84), indicating a considerable difference between the control and AGORA groups. Additionally, the common language effect size, $U_1/(n_1 n_2)$, is 1, suggesting a very low probability that a random control group value exceeds a random AGORA group value.

4.5.3 *User Feedback on the Use of AGORA*

The questionnaire results 4.3 are shown in Table 4.6. Participants rated AGORA highly, with averages of 4.3 for ease of use and 4.0 for usefulness, future use, and interest in similar tools. Responses showed minimal deviation (0 to 0.58) and a median of 4 for all questions, indicating consensus.

Table 4.6: Feedback on the experience with AGORA

Question	Average	Median	Std. Dev.
AGORA is easy to use	4.3	4.0	0.58
AGORA is useful for writing acceptance tests	4.0	4.0	0.00
I would use AGORA for writing acceptance tests	4.0	4.0	0.00
I would like to try similar tools for other software engineering tasks	4.0	4.0	0.0

4.6 DISCUSSION, IMPLICATIONS, AND FUTURE WORK

This section discusses the results reported in Section 4.5, their implications, and the limitations that may have influenced our findings and mitigation strategies.

4.6.1 *Discussion*

Hypothesis H1 suggested that AGORA could produce better UATs than the manual approach, but the experimental results did not support this statistically. AGORA and the control group showed comparable quality levels. AGORA generally scored better in correctness (6 out of 10 use cases) but was less effective in inferring input and output data in 3 use cases (8-10). This result could be due to insufficiently detailed use cases, limiting AGORA's ability to infer necessary data using GPT-4. For example, in use case no. 3, it generically mentions "details" and "fields" without specifying what is being searched for and displayed:

> **Use Case no. 3: US1.003–Display threshold details**
>
> *Main scenario*
>
> 1. *The actor clicks "view detail" for a given threshold.*
> 2. *The system searches for the threshold details.*
> 3. *The system displays the threshold details in a page with editable fields.*

The control group could not precisely define input and output data for the same use case, only reporting test steps. However, test engineers correctly identified input data in other cases (use cases 8-10), unlike AGORA. This result suggests that the quality of automatically generated test cases depends on the detail of the starting use cases. The lack of significant difference might also be due to software engineers' familiarity with the use cases and application domain, allowing them to fill in missing information. Despite these results, AGORA is a promising tool for refining generated UATs. Automating test case generation can identify gaps in use cases and encourage adding missing details, improving overall quality. For hypothesis H2, results showed AGORA significantly reduced the time to produce UATs, confirming the effectiveness of automation. This efficiency can positively impact the software development lifecycle by reducing release times and allowing engineers to focus on complex tasks. Feedback from a questionnaire indicated high user satisfaction with AGORA, valuing its ease of use and time savings despite the quality of automatically gener-

ated test cases not surpass manual ones. We can, therefore, answer the research question as follows:

> **Answer to RQ4.1:**
>
> AGORA automates the production of acceptance test cases from use cases, enhancing efficiency, but the quality of the tests depends on the completeness and detail of the use cases.

4.6.2 *Threats to Validity*

This section examines potential limitations that could have biased our results and mitigation strategies.

A threat to internal validity includes participants' bias and the Hawthorne effect, where awareness of being in an experiment alters responses. To mitigate this, we hid the hypothesis, ensured anonymity, divided participants into two groups, and shortened the study duration. Another internal validity threat is including more experienced individuals in the experimental group. We asked the IT company to provide participants with similar UML and test management experience. Additionally, we mitigated oracle bias by selecting a consultant with significant software engineering experience. Future research could involve more experts as oracles to compare evaluations of produced UATs. External validity threats include the selection of use cases. We chose use cases of varying complexity and linguistic style, but future research should include more use cases from different domains to improve generalizability. A threat to the validity of conclusions is the choice of statistical methods. We used the Mann-Whitney U test, a non-parametric test suitable for small, non-normally distributed samples, and verified its assumptions before application. We also used a correction for ties when necessary. Additionally, we addressed the non-determinism of the LLM by fine-tuning GPT-4's hyperparameters and giving clear formatting instructions, limiting the LLM's creativity for consistent responses.

4.7 CONCLUSION

In this chapter, we introduced AGORA, a novel approach for automatically generating UATs from requirements in natural language. Leveraging LLMs, AGORA interprets unstructured requirements and generates accurate UATs, optimizing time and resources, reducing human errors, and increasing testing efficiency. Initial findings show that AGORA produces UATs of similar quality to those created by experts and significantly improves test case generation efficiency. Moreover, AGORA also exhibits a marked increase in test case generation efficiency. The approach can be easily integrated into existing software development processes, offering a considerable competitive advantage and facilitating adoption in the Industry. To improve AGORA, we will enhance prompts or fine-tune the LLM for ambiguities and missing information, integrate a feedback system for refining test cases, and develop guidelines for optimized use cases. Enhancing user collaboration through better interfaces is also a key objective. To improve AGORA, we will explore a few research directions. Specifically, we will enhance prompts or fine-tune the LLM to handle ambiguities and infer missing information, integrate a feedback system for refining and customizing test cases, and develop guidelines for writing use cases optimized for automation. Enhancing user collaboration through better interfaces is also a key objective.

5

LLM-BASED AUTOMATION OF COSMIC FUNCTIONAL SIZE MEASUREMENT FROM USE CASES

5.1 INTRODUCTION

Measuring the size of a software system is crucial for supporting several software project management activities, including estimating development effort/cost, benchmarking, portfolio management, and assessing software contractor performance [4, 54, 138, 148, 224, 248]. Several approaches exist for sizing software systems, such as Use Case Points [257], Story Points [60], object-oriented metrics [130, 172, 179, 190], and Source Lines of Code (SLOC) [229]. Functional Size Measurement (FSM) methods have been introduced to measure software size based on functional requirements, making them independent of the technologies used to develop the software application. Function Point Analysis (FPA) was the first FSM method proposed [9, 142, 143, 279]. Over time, several variants of FPA (e.g., MarkII and NESMA) have been developed to improve measurement accuracy or extend applicability to specific domains [118].

In 1999, COmmon Software Measurement International Consortium (COSMIC) FSM (hereafter COSMIC) [42] was introduced as a second-generation FSM, addressing the limitations of earlier methods like FPA in measuring real-time and embedded systems and dealing with modern software architectures. COSMIC has gained significant attention in recent years [187] due to its broader applicability across various domains and its ability to provide accurate effort predictions [3, 66, 92, 93, 118, 119]. COSMIC measures the size of Functional User Requirements (FURs), which describe what the software must do to satisfy users' needs. These requirements are mapped to the COSMIC Generic Software Model, which represents software functionality through data movements between functional users and persistent storage. Each data movement represents one COSMIC Function Point (CFP), pro-

viding a standardized measure of the functionality delivered to users. These data movements can be identified from various software artifacts such as requirements specifications, use cases, or user stories. UML use cases are considered one of the best approaches for requirements elicitation and specification because they represent interactions between the system and its environment as natural language text in a semi-formal way [58, 177, 250], making them particularly suitable for automated analysis. However, measuring COSMIC from textual requirements remains a manual process, demanding qualified professionals and considerable time. While significant research has been conducted to automate FSM methods, including COSMIC [7, 24, 25, 48, 50, 80, 94, 102, 124, 150, 151, 158, 186, 221, 258, 266–268, 282, 294, 335], there are still few studies specifically exploring the automation of COSMIC measurement starting from natural language requirements [139–141, 177, 212, 213, 293, 304]. Previous automation approaches have often relied on structured requirements or historical data and used text mining techniques [139], syntactic rule-based systems [304], or ontological analysis [293]. Therefore, there is a need for approaches that can work directly with natural language use cases while providing the deep semantic understanding required for accurate COSMIC measurement.

This chapter introduces CosMet (COSMIC MEasuremenT), an approach to automate COSMIC measurement based on use case specifications, based on LLMs. In particular, CosMet takes as input the textual descriptions of use cases, including actors, their main scenarios, alternative flows, and exceptions, and produces the complete COSMIC analysis report as output. The approach includes two main components: (1) the Sentence Splitter Component and (2) the COSMIC Analyzer Component. The Sentence Splitter is designed to split complex sentences into use-case scenarios, breaking them into constituent steps, each representing a single atomic action. Then, the COSMIC Analyzer Model processes the resulting use cases to produce a COSMIC measurement, identifying Functional users, Triggering events, Objects of interest, Data Groups, and Data movement types, and then calculating the COSMIC Function Points. Both of these components are intended as refined instances of LLMs, using a few-shot learning. To assess CosMet, we developed a web tool based on GPT-

4 and performed an empirical study on seven software systems, specified by 123 use cases, from Management Information Systems, Telemedicine, Real-Time systems, and ML applications representing real-world software projects from both industry and reference case studies. We evaluated the results from both quantitative and qualitative perspectives. Our investigation revealed that CosMet effectively identified data movements, achieving a 99% F1 score for both Rouge and BLEU metrics. The tool correctly mapped functional requirements to the COSMIC Generic Software Model, with Rouge and BERTscore F1 scores ranging from 84.5% to 100%. Compared to manual measurement, it reduced measurement time by 60-80%.

The main contributions of this chapter are fourfold:

1. We introduce a novel LLM-based approach, CosMet, which aims to measure use cases and map Functional User Requirements to the COSMIC Generic Software Model.

2. We develop an automated tool that implements the CosMet approach using GPT-4.

3. We provide datasets, including use case specifications and their measurement, as a contribution to the research community.

4. We evaluate CosMet's effectiveness in measuring swoftware systems starting from use case specifications, mapping FURs to the COSMIC Generic Software Model, and reducing the measurement time. Furthermore, we systematically compare CosMet with existing COSMIC automated approaches to highlight its relative strengths and weaknesses.

These contributions are significant as they can serve as a considerable milestone and the foundation for future refinement of the application of LLMs in the FSM field, such as extending the approach to FPA and comparing COSMIC with different FSM methods.

5.2 BACKGROUND AND MOTIVATION

In the following, we provide the necessary background to comprehend our work and its relevant context. First, we briefly introduce COSMIC. Next, we discuss the motivation driving our research.

5.2.1 *COSMIC*

The first Functional Size Measurement method was FPA, introduced in the late 1970s [9, 142, 143, 279]. It measures software size by counting functional components such as internal logical files, external interface files, external inputs, external outputs, and external inquiries, with each component weighted according to its complexity. While FPA proved valuable for traditional information systems, it revealed limitations when applied to modern software architectures and real-time systems. To address these challenges and accommodate a broader range of application domains, COSMIC was proposed in 1999 as a second-generation FSM, offering significant advancements over FPA in the measurement process. The adoption of COSMIC as an industrial standard continues to grow[1], largely due to the organization's global presence, with representatives in 26 countries and numerous companies and institutions publicly recognizing their use of the COSMIC method. COSMIC provides a comprehensive framework of rules, concepts, and processes specifically tailored for measuring the functional size of software based on its FURs [42]. FURs capture interactions between a Functional User, who acts as both a sender and receiver of data, across the Boundary and with Persistent Storage within this Boundary. The software architecture is structured into Layers, and the software component being measured must belong to a single layer. The measurement process is guided by a clearly defined Purpose, which explains the importance of the measurement, and a well-specified Scope, identifying the specific FURs to be measured.

To measure software, the FURs must align with COSMIC's Generic Software Model, which provides the necessary concepts and prin-

1 COSMIC. World-wide usage. https://cosmic-sizing.org/usage/

ciples for this aim. Within this model, the measurement of FURs is guided by several key concepts. A Functional process represents a set of FURs that form a logical grouping of data movements and can be executed independently. It is initiated by a Triggering event from a Functional user and consists of a sequence of sub-processes. Data movements within these processes are referred to as Functional sub-processes. A Data group is defined as a set of data attributes that characterize a unique aspect of an object of interest, representing the smallest meaningful unit of information within the data group. The functional size of software is determined by identifying and counting the data movements within each functional process. These movements include: Entries (E), where data groups are transferred from a Functional user into the process; Exits (X), where data groups are sent from the process to the Functional user; Reads (R), where data groups are moved from persistent storage to the process; and Writes (W), where data groups are transferred from the process to persistent storage. Each data movement is counted as one COSMIC Function Point (CFP). Calculating the size of a software component within a defined scope involves summing up the sizes of all identified Functional processes. Each process must include at least two CFPs, typically formed by an E plus either a W or an X.

Use Case Example: Insert a New Customer

Description: This function lets the user add a new customer.
Primary actor: Hotel manager
MAIN Scenario

1. *The user clicks the "New customer" button.*
2. *The system shows a form with the following editable fields: surname, name, date of birth, fiscal code, address, city, and notes.*
3. *The user fills in the surname, name, date of birth, address, city, email address, fiscal code fields, and optionally the notes field and submits the form.*
4. *The system checks that: the surname and name fields contain at least two characters, the fiscal code field is valid, the date of birth is correct, the email address is valid and records the new customer in persistent storage.*
5. *The system shows a confirmation message.*

Exceptions

5.a1 The system shows an error message stating that the provided data are not valid.

Figure 5.1: Use Case: Insert a New Customer

To illustrate the COSMIC measurement concepts, consider the use case in Figure 5.1 taken from a hotel management system. Tn

this scenario, a hotel manager (the functional user) enters a new customer's details into the system. Table 5.1 demonstrates the application of the COSMIC method to the specifications outlined in this use case. The functional size is calculated as 3 CFP, derived from three distinct data movements:

Table 5.1: COSMIC analysis of the "Insert new customer" use case

Triggering Event: User clicks "New customer" button

FU	Sub-process	DG	OOI	DM	CFP
Hotel manager	Click new customer button	-	-	-	
	Display customer form				
Hotel manager	Enter customer data	Customer data	Customer	E	1
	Validate and store customer	Customer data	Customer	W	1
	Show confirmation / errors	Message	Message	X	1
				Total:	**3**

FU: Functional User, DG: Data Group
OOI: Object of Interest, DM: Data Movement Type

an Entry (E) when the user submits the customer data, a Write (W) when the system stores the validated data, and an Exit (X) when the system displays either a confirmation or error message. when the system displays either a confirmation or error message. Each of these movement involves the "Customer" data group, which is associated with the "Customer" object of interest.

Despite its effectiveness, applying COSMIC to analyze such processes is time-consuming and necessitates the involvement of professional experts for accurate measurement. Consequently, a key research objective within the COSMIC community is to automate the measurement process to facilitate broader industrial adoption [293].

5.2.2 *Motivation of our work*

While LLMs show great potential for processing natural language requirements, their effective application to COSMIC measurement requires a structured approach.

Before developing CosMet, we indeed verified GPT-4's baseline knowledge of COSMIC by directly asking it to explain the COSMIC FSM method and its steps, as shown in figure 5.2.

GPT-4 COSMIC Knowledge.

Prompt:
Please briefly explain what the COSMIC Functional Size Measurement method is and what the method steps are.

GPT-4 answer:
The COSMIC method is a standardized approach for quantifying software functionality, independent of implementation technology. It aids in estimating development effort, comparing productivity, and managing projects. The method revolves around "functional processes," defined as data movements triggered by meaningful user events, focusing on four types: Entry, Exit, Read, and Write. Here are the typical steps involved in the COSMIC FSM method:

1. *Define what part of the software will be measured and why the measurement is being done.*
2. *Determine the functional user requirements (FURs) that the software must fulfill.*
3. *Create a model representing the software's functionalities.*
4. *For each functional process, identify data movements. Classify each data movement as an Entry (E), Exit (X), Read (R), or Write (W).*
5. *Count each identified data movement type (E, X, R, W) as one "COSMIC Function Point" (CFP).*
6. *Sum up the CFPs from all data movements to get the total functional size of the software.*

Figure 5.2: GPT-4 COSMIC Knowledge.

Example: Prompt to analyze and measure a use case

Act as an expert COSMIC measurer.

USE CASE:
UC-GES-03.4 Insert a new customer
Description: This function allows the user to insert a new customer.
Primary actor: Hotel manager, Operator
MAIN Scenario

1) *The user clicks the "New customer" button*
2) *The system shows a form with the following editable fields: surname, name, date of birth, fiscal code, address, city, and notes.*
3) *The user fills in the surname, name, date of birth, address, city, email address, fiscal code fields, and optionally the notes field and submits the form.*
4) *The system checks that: the surname and name fields contain at least two characters, the fiscal code field is valid, the date of birth is correct, the email address is valid and records the new customer in persistent storage.*
5) *The system shows a confirmation message.*

Exceptions
5.a1 The system shows an error message stating that the provided data are not valid.

Analyze the use case and report a detailed COSMIC Functional Size Measurement analysis, providing the Triggering event for the functional process and the functional users, Data Groups, Objects of interest, Data movement types, and CFP for each use case step.

Figure 5.3: Prompt to analyze and measure a use case.

The model demonstrated a solid understanding of the core concepts by correctly describing the fundamental purpose of COSMIC, the concept of functional processes, the four types of data movements (Entry, Exit, Read, and Write), and the six main steps of the method, ranging from scope identification to final size calculation.

However, while this baseline knowledge was encouraging, GPT-4 cannot apply the methodology correctly. To demonstrate this, let us consider examples of direct application of GPT-3.5 and GPT-4 to COSMIC measurement, as detailed in Figure 5.3.

Upon reviewing the response from GPT-3.5, reported in Figure 5.4, it is evident that the COSMIC analysis requires adjustments.

GPT-3.5 COSMIC analysis

Triggering event: The user clicks the "New customer" button
Functional Users: Hotel manager, Operator
Data Groups: Customer Information
Objects of Interest: Customer
Data Movement Types:

1) *Read: The system reads the existing customer data from persistent storage.*
2) *Write: The system writes the new customer data to persistent storage.*

CFP for each use case step:

1) *The user clicks the "New customer" button - 0 CFP*
2) *The system shows a form with the following editable fields: surname, name, date of birth, fiscal code, address, city, and notes. - 1 CFP (Read)*
3) *The user fills in the surname, name, date of birth, address, city, email address, fiscal code fields, and optionally the notes field and submits the form. - 1 CFP (Read)*
4) *The system checks that: the surname and name fields contain at least two characters, the fiscal code field is valid, the date of birth is correct, the email address is valid and records the new customer in persistent storage. - 1 CFP (Write)*
5) *The system shows a confirmation message. - 0 CFP*

Exceptions:
5.a1 The system shows an error message stating that the provided data are not valid. - 0 CFP
Total CFP for the use case: 3 CFP

Figure 5.4: GPT-3.5 COSMIC analysis.

Indeed, GPT-3.5 missed the Object of Interest "Message". In addition, it made the following mistakes:

- Step 2 does not contain any data movement.
- Step 3 is not a "Read" data movement, but it is an "Entry" data movement.

- Steps 5 and 5.a1: They are "Exit" data movements but must be counted once.

Similarly, GPT-4's analysis (shown in Figure 5.5) also needs corrections because it missed the Object of Interest "Message" and made the following mistakes:

- Steps 1,2, and 4 do not contain any data movement.
- The total number of CFPs is not 7, but 3.

GPT-4 COSMIC analysis

Triggering Event: The user clicks the "New customer" button.
Functional Users: Hotel Manager, Operator
Data Groups and Objects of Interest:

1) *Customer Details (Surname, Name, Date of Birth, Fiscal Code, Address, City, Email Address, Notes)*

Data Movement Types:

1) *Entry: The user fills in the customer details and submits the form.*
2) *Exit: The system shows a confirmation message.*
3) *Read: The system checks the entered customer details.*
4) *Write: The system records the new customer in persistent storage.*

COSMIC Functional Size Measurement (CFP) Analysis:
Step 1: The user clicks the "New customer" button.
- Triggering event: External Input (EI), CFP: 1
Step 2: The system shows a form with editable fields.
- Data movement: External Output (EO), CFP: 1
Step 3: The user fills in the customer details and submits the form.
- Data movement: External Input (EI), CFP: 1
Step 4: The system checks the entered customer details.
- Data movement: External Inquiry (EQ), CFP: 1
Step 4: The system records the new customer in persistent storage.
- Data movement: External Input (EI), CFP: 1
Step 5: The system shows a confirmation message.
- Data movement: External Output (EO), CFP: 1
Exception 5.a1: The system shows an error message stating that the provided data are not valid.
- Data movement: External Output (EO), CFP: 1

Total CFP: 7

Figure 5.5: GPT-4 COSMIC analysis.

Moreover, it used an incorrect vocabulary. In particular, it reported "External Input", "External Inquiry", and "External Output", which are not "Base Functional Component" types in the COSMIC method. Therefore, there is a need for a more accurate and reliable method to analyze and measure software systems starting from use case specification using LLMs. This motivates our work.

5.3 RELATED WORK

In this section, we first examine previous research on measuring COSMIC using NLP. Then, we review the studies on automated COSMIC measurement. While our work focuses on automating COSMIC measurement, it is essential to note that other size measurement approaches exist, such as use case points, story points, object-oriented measures, requirements complexity, interface complexity, and SLOC. These different approaches highlight the diverse needs of software projects and represent potential areas for future research using similar methods.

5.3.1 *COSMIC measurement using NLP*

Simplifying and automating the application of the COSMIC method is crucial for both academia and industry, especially when measuring textual requirements. However, there is limited academic research on this topic.

Hussain et al. [139] present a workbench, based on a text miner, for automatically extracting all the necessary information from a software's requirement specifications document to calculate its COSMIC size measure. Hussain et al. [140] describe a supervised text mining strategy for estimating COSMIC functional size (based on [139]) from informally expressed textual requirements, with applications in Scrum and similar agile software development processes. The experiments' average F1-score value was 66.9%. Hussain et al. [141] use the frequency of language characteristics to infer the COSMIC size. However, the proposed method performs better when requirements are not represented as use cases or scenarios. The results show that it correctly classifies requirements with an F1-score value between 60.9% and 73.3%. Ochodek [212] uses NLP to parse linguistic elements from requirements and applies the C4.5 decision tree classifier [238] to categorize based on a historical database of requirements measurements. Requirements are sorted into size classes defined by quantiles of the CFP distribution. Classification accuracy is influenced by the specification language, with the HKO predictor achieving 38.7% to 78.5% accuracy. Ochodek [213] proposes a two-step method for estimating the functional size of software from use case re-

quirements. The process involves classifying use case names into thirteen categories and constructing prediction models using historical data to determine the number of CFPs. Ochodek's approach, compared with the COSMIC AUC and another technique [212], achieves a 79% accuracy rate. Ecar et al. [98] introduce AutoCosmic, an automated tool for estimating user stories using a User Story Grammar Validator and Parser linked to the COSMIC method. Informal validation against a certified expert's estimates shows AutoCosmic achieves a 95.3% accuracy rate, similar to an expert professional. ScopeMaster [293] is a COSMIC Functional Size Measurement tool that accepts textual requirements as input. It is based on the syntactic and ontologic analysis of the textual requirements. It analyzes only the primary function provided in the requirements. Its accuracy is within 20-30% of manual count equivalents, and it reduces the manual measurement time by 75%-80%. Wang et al. [304] offer an enhanced COSMIC software assessment technique based on fixed language characteristics, NLP concepts, and a local rule basis to eliminate human intervention and increase accuracy. To perform automated text processing, it focuses on grammar and syntax rules. The results show an accuracy within 93%-99% of manual count equivalents.

CosMet differs from the mentioned related work in several significant ways (see TABLE 5.2).

Firstly, CosMet employs use case documents as input, while many related works focus on semi-structured textual requirements or user stories. Secondly, CosMet employs advanced NLP techniques, specifically LLMs, to automate the COSMIC measurement, allowing it to adapt to different contexts and styles of writing, potentially making it more flexible and accurate in different scenarios. The related work, on the other hand, often relies on predefined pattern rules or syntax-based classifiers, which may not be as adaptable to different contexts or styles of writing. Thirdly, CosMet delivers a comprehensive analysis of COSMIC concepts within functional requirements, offering greater transparency and detail than similar works. Finally, CosMet's performance has been validated across seven use case documents from diverse domains and by seven practitioners, showing high accuracy and notable productivity improvements. Such an extensive empirical evaluation and evidence of efficiency gains are often absent in related

Table 5.2: Comparison with Related Work on COSMIC Measurement using NLP

Reference	Use Cases	Approach	Completeness	Limitations	Accuracy	Measurement time reduction
CosMet	✔	NLP with generative AI	Measure all scenarios within a use case	Layer and measurement scope cannot be identified	F1-score is 99.4%	60-80%
Hussain et al. [139]	✖	Text Mining	Only Data Groups and Data movements are identified	No experimental results	N/A	Not declared
Hussain et al. [140]	✖	Supervised text mining strategy for estimating COSMIC functional size (based on [6] and [7]).	Only CFPs are estimated	(1) Use historical data to estimate the COSMIC size; (2) Requirements are expressed as user stories.	66.9%.	Not declared
Hussain et al. [141]	✖	It uses the frequency of language characteristics to infer COSMIC size.	Only CFPs are estimated	(1) Use historical data to estimate the COSMIC size; (2) Do not perform well with use case scenarios.	F1-Score within 60.9% and 73.3%.	Not declared
Ochodek [212]	✔	It extracts syntactic linguistic information using NLP and classifies requirements using the C4.5 decision tree-based classifier.	Only CFPs are estimated	Sensitive to the language used in descriptions of requirements.	38.7%-78.5% of manual count.	Not declared
Ochodek [213]	✔	Two-step approach for predicting the functional size of applications based on use-case requirements.	Only CFPs are estimated	Use historical data to estimate the COSMIC size.	79%.	Not declared
Ecar et al. [98]	✖	Based on a User Story Grammar Validator and a Parser, and defines a dictionary to map it to the COSMIC method.	Only CFPs are estimated	(1) Only valid for User stories; (2) Use historical data to estimate the COSMIC size.	95.3%.	Not declared
Ungan et al. [293]	✖	Commercial tool for measuring textual requirements	Measure only the main function within a requirement	Syntactic and ontologic analysis of requirements; analyze only the main function of the requirements.	20%-30% of manual count.	75-80%
Wang et al. [304]	✖	Technique based on fixed language characteristics and syntactic rules.	Triggering events, Data movement types, and Data groups are not identified	Focused only on grammar and syntax rules and structured text	93%-99% of manual count.	90%

studies. Therefore, CosMet introduces a more sophisticated, automated, and detailed COSMIC measurement approach with proven effectiveness, customization potential, and substantial productivity benefits.

5.3.2 *Automatic COSMIC measurement*

Several researchers have suggested methods for automatically evaluating COSMIC measurements using various artifacts such as UML diagrams, conceptual models, structured requirements, source code, and more.

Jenner [150] provides a mapping between the COSMIC and Rational Unified Process concepts and designs a method to measure models automatically. Jenner [151] details an automated approach for determining functional size by utilizing Rational Rose software. On the same page, Azzouz and Abran [24] propose utilizing the Rational Unified Process and Rational Rose to automate the COSMIC functional size measurement. In [94], Diab et al. present μcROSE, a system that determines the functional size of real-time systems starting from statechart diagrams. The approach relies on programmed rules that map the several objects of interest onto the COSMIC ones. Lind et al. [186] introduce a technique for measuring embedded software using the COSMIC method. They propose a new UML profile containing all the information required to use COSMIC and estimate the functional size.

Soubra et al. [266, 267] design and implement an automated measuring procedure utilizing the most commonly used Simulink model in the automobile sector. Oriou et al. [221] describe how to use the approach and the tool by Soubra et al. within Renault. An automated approach for ensuring consistency between component and activity diagrams is designed by Sellami et al. [258], utilizing COSMIC. At first, the authors outline the methods used to establish the functional size of the diagrams. Then, rules are introduced to ensure the COSMIC FSM remains consistent. Lastly, a tool is offered to calculate the functional size and ensure the consistency of the diagrams.

Akca and Tarhan [7] develop a library to tag data movements in functional processes. The measuring approach is semi-automatic, requiring source code modification to use the library. According to the authors, using the library in a student registration system resulted in 92% accurate outcomes in automatically and manually measured functional size. Ungan et al. [294] establish a link between the abstraction levels of the measured artifacts and func-

tional size. They provide a software size measuring approach based on software design model sizes for the solution domain. The suggested method, in particular, makes use of sequence diagrams. They also suggest an automated measurement tool. A novel tool, named PL FSM, has been introduced by Eren et al. [102] for facilitating functional software size measurement in a structured component-based software product line environment (CBPL) that utilizes the interface-based design (IbD) method. PL FSM establishes a mapping between UML and COSMIC elements and offers automated information extraction capabilities from UML diagrams. Implementing PL FSM streamlines the software development process in a CBPL framework. Gonultas and Tarhan [124] introduce a method for measuring the COSMIC functional size that automatically installs measurement library code into an application. This technique estimates the size while the user scenarios are executed at run-time.

Ceke and Milasinovic [48] describe a technique to approximate the web applications' functional size. This approach relies on a methodological mapping between the UML-based Web Engineering (UWE) models of web applications and the COSMIC functional size. Automated measurement outcomes are comparable. Karim et al. [158] suggest a tool for analyzing the XML structure of sequence diagrams to automate the measuring procedure. Tarhan et al. [282] delve into the automation of FSM through software code and introduces the development of a new tool called COSMIC Solver. This tool is specifically designed to measure the performance of Java Business Applications (JBAs). COSMIC Solver automates the entire process, from extracting textual representations of UML sequence diagrams from the functional execution traces of a JBA to tagging these representations with AspectJ to determine the COSMIC functional size. Furthermore, the tool calculates the functional size of user scenarios that run in the JBA, using the tags information following the COSMIC rules. Chamkha et al. [50] design a JavaCFP plugin tool for calculating the COSMIC functional size of Java source code. With JavaCFP, it is possible to verify the adequacy of implemented functions concerning the specified requirements, detect any inconsistencies, and generate updates on the progress of newly implemented functions. This plugin was validated using the *C-Reg* case study [40]. Zaw

et al. [335] provide a model and an automated method for measuring it based on three alternative diagram notations, including UML, SysML, Petri net, and generic mapping. De Vito et al. [80] present a tool, J-UML COSMIC, for calculating the COSMIC functional size using UML software artifacts. The tool can deal with various UML artifacts based on the observation that different development processes can use different UML models. De Vito and Ferrucci [76] introduce Quick/Early, a streamlined process for measuring use case documents to balance accuracy with time constraints. They also suggest a template to easily identify functional components in use cases. Soubra et al. [268] present an approach for creating a COSMIC 'universal' tool for automatically measuring software developed in various programming languages. This study proposes a prototype tool based on COSMIC and MIPS with a small-scale validation as a proof of concept.

Furthermore, many practitioners and researchers use alternative and supplemental UML or SysML model types to be translated into software cost model parameters. For instance, Kotronis et al. [167] extend SysML to integrate cost analysis into model-based systems engineering, enabling the evaluation of design alternatives under specific cost and performance restrictions. This integration showcases the potential for future work in automating COSMIC measurement using various UML or SysML artifacts.

While the related work has made significant strides in automating COSMIC measurement, several key differences exist between these approaches and CosMet. Firstly, unlike existing approaches that rely heavily on specific artifacts such as UML diagrams, structured requirements, source code, and more, CosMet is designed to process natural language requirements, offering flexibility and early-stage software development applicability when such artifacts are often absent. Secondly, many existing approaches require manual intervention or source code modification to function effectively. Conversely, CosMet is designed to be fully automated, significantly reducing the manual effort required for COSMIC measurement. Thirdly, while some existing approaches utilize traditional rule-based systems or machine learning models, CosMet leverages advanced LLMs for its operation, allowing CosMet to handle the ambiguity and complexity of natural language requirements more effectively. Lastly, CosMet measures the functional

size and provides a transparent and detailed analysis of the COSMIC concepts applied to the functional requirements. This feature is not commonly found in the existing approaches, making CosMet a valuable approach for interpreting the COSMIC measurement process.

5.4 THE COSMET APPROACH

The CosMet approach comprises two components (see Figure 5.6), namely (1) the *Sentence Splitter* and (2) the *COSMIC Analyzer*, intended as two refined instances of GPT-4, to analyze use cases specified in natural language. First, the Sentence Splitter examines the use case scenarios and divides complex sentences into single steps, each representing an atomic action performed by the system or the user, ensuring that each action can be distinctly measured and analyzed for accurate COSMIC functional size measurement. Then, the COSMIC Analyzer applies the COSMIC method to these atomic actions, identifying Functional users, Triggering events, Objects of interest, Data groups, and Data movement types, and calculating the number of CFPs.

For example, given the 'Insert a New Customer' use case presented in Section 5.2.1, CosMet produces an analysis report as shown in Table 5.1. To implement the two components, we employed GPT-4 as LLM and followed four main steps (described in detail in the following): (1) prompt engineering, (2) examples preparation, (3) validation, and (4) hyperparameters tuning. Additionally, we developed a web application that interacts with the GPT-4 API.

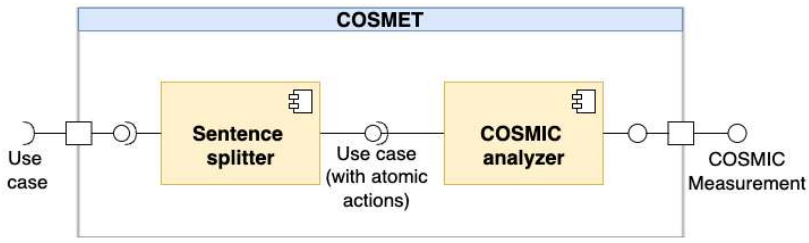


Figure 5.6: CosMet components

5.4.1 *Prompt Engineering*

We employed the few-shot learning pattern [246] for the implementation of the Sentence Splitter and COSMIC Analyzer components. As described in Chapter 2, this pattern leverages LLMs' ability to identify and learn patterns from a limited set of examples and then apply these patterns to new, similar cases. By providing well-crafted examples, LLMs can recognize the underlying structure and relationships in the data, enabling them to perform complex tasks without extensive training.

For the Sentence Splitter, few-shot examples were designed to teach the model how to identify complex sentences containing multiple actions and break them down into atomic steps. For example, when provided with examples of how to split sentences such as "The system validates the data and stores it in the database" into separate actions, the model learns to recognize similar patterns in new use cases. Similarly, for COSMIC analysis, few-shot examples illustrated how to identify key COSMIC concepts within use case descriptions. By presenting the model with examples that demonstrate the recognition of data movements, functional users, and objects of interest across various contexts, it learns the patterns that characterize these elements.

However, obtaining the desired outcomes with few-shot learning could be challenging. Human users must rely on a combination of domain expertise and creativity, experimenting with different prompts to achieve the desired results. To effectively address the challenges related to prompt engineering, we followed three main guidelines [99, 220]:

1. Show and tell: making intentions clear using examples, instructions, or both.
2. Provide quality data: ensuring enough samples for the model to follow a pattern.
3. Check settings: setting the model parameters to make the model predictable in responding.

Considering these guidelines, we prepared the prompt templates (as shown in Figures 5.7 and 5.8) and the training set to refine

GPT-4 with few-shot examples using OpenAI Playground (a web-based interface offered by OpenAI), as described in the next section.

Sentence Splitter: Prompt Templates.

System Prompt:

Split all the following sentences, numbering the new sentences so that each new sentence is grammatically correct.

Example 1:

...

User Prompt:

{uc}

Split:

Figure 5.7: Prompt templates for the Sentence Splitter.

For the sake of simplicity, Figures 5.7 and 5.8 present only the prompt templates. Specifically, the Sentence Splitter template follows a simple structure where the system prompt instructs the model to split complex sentences while maintaining grammatical correctness, followed by examples. The user prompt then presents the use case to be processed. The COSMIC Analyzer template is more structured, showing how to format the input use case and the expected measurement output. The system prompt includes examples of functional processes with their complete COSMIC analysis (triggering events, functional users, and sub-processes with data movements). In contrast, the user prompt presents the new use case to be analyzed following the same structure. Both templates are designed to guide the model through few-shot learning while maintaining consistency in the output format.

5.4.2 *Examples Preparation*

The training set preparation is the crucial step of every Machine Learning (ML) approach. In this stage, we addressed three critical issues. First, use cases can be written using different linguistic

COSMIC Analyzer: Prompt Templates.

System Prompt:

FP "{Functional process 1}":
use case text...

FP "{Functional process 1}" measurement:
Triggering Event: {Triggering event description}
Functional User: {Functional User}
Sub-Processes:
1. step description...
FU (user) – DG (Data group) – OOI (Object of interest) – Data movement
2. step description...
...other examples...

User Prompt:

FP "uc_name":
{uc}

FP: "{uc_name}" measurement:

Do not add any explanation or anything else.

Figure 5.8: Prompt templates for the COSMIC Analyzer.

styles. Software engineers may group actions (executed by the system or the user) in use case scenarios. For instance, consider the following sentence:

"The system locates the reservation header and, if exists, details regarding the booked rooms and services, and the children's ages".

In the example above, the system executes four actions:

1. Read (reservation header): The system searches for the reservation header.
2. Read (booked rooms): The system searches for the booked rooms.
3. Read (requested services): The system searches for the requested services.

4. Read (children's ages): The system searches for the children's ages.

The COSMIC method must identify the atomic data movement types to count them accurately. Despite GPT-4 being capable of few-shot learning on diverse tasks, it cannot identify all the possible combinations of data movements present in use case scenarios, as illustrated in Figures 5.9 and 5.10. Indeed, without splitting the sentences and providing the prompt in Figure 5.9, GPT-4 wrongly identifies three Read data movements and an Entry. On the other hand, using the prompt in Figure 5.10 in which the sentences are split, it correctly identifies the four Read data movements but erroneously pinpoints an Entry. Therefore, making actions atomic in the flow of events is essential.

GPT-4 Data Movements Identification without splitting.

Prompt:
You are an expert COSMIC FSM practitioner. Your task is to identify all the data movement types in the use case I'll provide, and the total number of CFP.

Use case:
The system locates the reservation header and, if exists, details regarding the booked rooms and services, and the children's ages.

Data movements:
Total CFP:
Do not add anything else.

GPT-4 answer:
Data movements: Entry (E), Read (R), Read (R), Read (R)
Total CFP: 4

Figure 5.9: GPT-4 Data Movements Identification without splitting.

The second issue concerns mapping the FURs expressed in the use cases to COSMIC's Generic Software Model (GSM) and applying COSMIC's rules. This is the hardest part of our prompt engineering because it is not a typical task for GPT-4 (i.e., classifi-

GPT-4 Data Movements Identification with splitting.

Prompt:

You are an expert COSMIC FSM practitioner. Your task is to identify all the data movement types in the use case I'll provide, and the total number of CFP.

Use case:
The system locates the reservation header.
If the reservation header exists, the system locates details regarding the booked rooms.
If the reservation header exists, the system locates details regarding the booked services.
If the reservation header exists, the system locates the children's ages.

Data movements:
Total CFP:
Do not add anything else.

GPT-4 answer:

Data movements: Entry (E), Read (R), Read (R), Read (R), Read (R)
Total CFP: 5

Figure 5.10: GPT-4 Data Movements Identification with splitting.

cation, summarization, etc.). For example, according to the COSMIC manual, errors and confirmation messages must be classified as Exit, but they must be counted only once per Functional process. So, these kinds of data movements must be differentiated from the others to avoid inconsistent measurements. In addition, when the system exchanges data movements with external components, these must be classified as Exit (from the system towards the external components) or Entry (from the external components to the system).

The last challenge is related to the limited context size of GPT-4, namely the total length of the input message and the model response. The maximum context size of GPT-4 is 8,192 tokens. Therefore, it is critical to balance the prompt and the response size.

To efficiently manage these constraints, we refined GPT-4 using two training sets of few-shot examples (implemented in the prompt.py script under the folder CosMet/cosmic in our online repository https://github.com/gadevito/COSMET): T1 for the Sentence Splitter Component and T2 for the COSMIC Analyzer Component. The preparation of both training sets was carried out collaboratively by the authors, combining the practical industry knowledge of the first author (with over 30 years of experience in the IT field and more than 18 years in FSM) and the expertise of experienced researchers (with research experience ranging from 15 to 35 years).

These training sets are implemented as system prompts containing instructions and examples, following the structure described in Section 5.4.1. We carefully managed the token allocation during refinement to balance the few-shot examples and the space required for the model's output. We left over 5,200 tokens to process a use case and produce the COMSIC analysis. The number of tokens was strategically determined to be ample enough for verbose use cases, allowing for a comprehensive analysis without exceeding GPT-4's context size.

5.4.2.1 *Sentence Splitter Training Set*

The Sentence Splitter's prompt engineering involved creating clear instructions to help GPT-4 split complex sentences. We manually created a set of examples, drawing from our experience in software requirements engineering and COSMIC measurement. Additionally, we analyzed other use case documents and user stories from the companies that provided us with the evaluation and test datasets, which helped us understand their typical linguistic patterns.

We refined these prompts iteratively, focusing on preventing over-splitting sentences and preserving the semantic integrity of the use case steps. We used examples of varying complexity from different software fields, ensuring linguistic diversity and the ability to handle domain-specific terminology. This approach exposed the LLM to various sentence structures and action sequences from real-world use case documents.

As a result, the training set T1 (publicly available in the CosMet online repository https://github.com/gadevito/COSMET in the prompt.py script under the folder CosMet/cosmic) consisted of a context instruction and 16 example sentences followed by the expected result. T1 was designed to split sentences taking into account the following semantic rules:

- If the sentence elaborates on an action without indicating data exchange between user, system, storage, or external components, it should remain unsplit (rule 1).
- If the sentence involves multiple actions by the same subject, each action must be listed as a separate use-case step, with the subject restated for clarity (rule 2). This often happens when the action has different verbs or periods.
- If the sentence regards a transaction and reports a validation status before (or after) that, then the sentence must not be split (rule 3).
- If a sentence describes a transaction involving multiple entities, it must be split (rule 4). This applies when multiple entity names follow the transaction verb, such as "The system inserts the reservation and the booked rooms," involving both "reservation" and "booked rooms."
- If the sentence reports a system message and explains its meaning, it must not be split (rule 5).

Figure 5.11 shows the response of the Sentence Splitter Component for the following use case example:

Example: Use case to split

MAIN Scenario

1) The hotel manager clicks the "Services" Link.

2) The system shows a form to search services by code.

3) The user optionally fills in the service code to filter the results and executes the search.

4) The system searches for services that meet the search criteria and displays the list of available services.

Exceptions:

> *The system shows an error message stating that no results match the search criteria.*

MAIN SCENARIO:
1. The hotel manager clicks the "Services" link.
2. The system shows a form to search services by code.
3. The user optionally fills in the service code to filter the results and executes the search.
4. The system searches for services that meet the search criteria.
5. The system displays the list of available services.

Exceptions:
6. The system shows an error message stating that no results match the search criteria.

Figure 5.11: GPT-4 response for a typical "split sentences" request

In the response, step no. 4 is split into two distinct steps: one for the search and the other for displaying the results. This splitting is necessary as each action corresponds to a different data movement (Read for searching and Exit for displaying). In contrast, step 3 is not split because, while it contains two actions (filling in the form and executing the search), only the search execution represents a meaningful data movement (Entry). This selective splitting approach helps optimize token usage and processing efficiency while ensuring accurate COSMIC measurement.

5.4.2.2 *COSMIC Analyzer Training Set*

We ensured the quality of the few-shot examples through a meticulous review process for the COSMIC Analyzer Component. The first author prepared each example following COSMIC measurement principles. The co-authors then reviewed these examples to ensure correctness and alignment with COSMIC standards. Consequently, the training set T2 (publicly available in our online repository `https://github.com/gadevito/COSMET`) comprised 7 examples of use cases with their relative COSMIC analysis: 3 examples coming from the Course Registration System (C-REG) case study [40] ("Enquire on a Professor's details", "Add a Professor's details", and "Delete a Professor's details"), and 4 examples engineered to teach GPT-4 how to deal with batch processes and typical message exchanges with external components in different domains ("Search for ships", "Reserve cabins", "Activate sensors", and "Activate processes").

These examples were crafted based on our experience with COSMIC measurement, following the same approach used for the Sentence Splitter training set. Figure 5.12 shows the measurement response with T2 for the following request:

Example: Use case measurement

FP: "Search for rooms":
MAIN SCENARIO:
1) The hotel manager selects the sub-option "Rooms"
2) The system shows a form to search rooms by code
3) The user optionally fills in the room code to filter the results and executes the search
4) The system searches for rooms that meet the search criteria
5) The system displays the list of rooms that meet the search criteria
6) The Hotel Manager browses the results
Exceptions:
7) The system shows an error message stating that no results match the search criteria.
FP: "Search for rooms" measurement:

Following the same structure introduced in Section 5.2.1 (see Table 5.1), the output includes the Triggering Event that initiates the functional process and the Functional User involved. For each step in the scenario, the analysis identifies Data Groups (DG) and their Objects of Interest (OOI), specifies Data Movement types (Entry, Exit, Read), and indicates cases where no data movement occurs. The output also includes special annotations for Exits that should be counted only once, such as error messages.

5.4.3 *Validation Process*

The outputs produced by both components were carefully validated before being used as references for the approach. This validation process combined multiple sources of expertise and real-world examples. Initially, the first author reviewed the outputs and certified them in COSMIC measurement to ensure compliance with COSMIC principles and industry practices. Then, they underwent cross-checking with the other authors to ensure theoretical correctness. Additionally, we compared our results with

FP: "Search for rooms" measurement:
Triggering Event: The Hotel Manager selects the sub-option "Rooms"
Functional User: Hotel Manager
Sub-Processes:
MAIN SCENARIO:
1. The hotel manager selects the sub-option "Rooms"
   No data movement.
2. The system shows a form to search rooms by code
   No data movement.
3. The user optionally fills in the room code to filter the results and executes the search
   FU (Hotel Manager) - DG (Room code) - OOI (Room) - Entry.
4. The system searches for rooms that meet the search criteria
   DG (Room code) - OOI (Room) - Read.
5. The system displays the list of rooms that meet the search criteria
   DG (Room details) - OOI (Room) - Exit.
6. The Hotel Manager browses the results.
   No data movement.

Exceptions:
7. The system shows an error message stating that no results match the search criteria.
   DG (Messages) - OOI (Message) - Exit. It must be counted only once.

Figure 5.12: GPT-4 response for the "Functional process measurement" request

actual use cases and measurements provided by the companies who supplied our evaluation datasets. While these additional materials cannot be publicly shared due to confidentiality agreements, they helped ensure that our examples and their analysis reflected real-world practices and measurement standards.

In addition, before conducting the complete empirical evaluation, we prepared two preliminary validation sets to verify the effectiveness of our few-shot examples (publicly available in our online repository `https://github.com/gadevito/COSMET`). The first set was used to validate the Sentence Splitter Component (V1), comprising 20 sentences extracted from real use cases provided by our industry partners. These sentences were specifically selected because they represent complex scenarios commonly found in industry use cases, such as multiple actions with different subjects, nested conditions, and compound transactions. This complexity helped us verify that our splitting approach could handle the various writing styles and structures typically found in real-world requirements. The second set validated the COSMIC Ana-

lyzer Component (V2). It was derived from the use case specifications of a Telemedicine microservice application, FIDDIA, developed by an Italian Small Medium Enterprise (Kiranet). The first author analyzed and measured these specifications, and the results were reviewed and validated by the co-authors. The use case specifications include four use cases and four Functional processes, providing a realistic test case for our measurement approach. Both validation sets were distinct from the training and test sets used in the empirical evaluation (see Section 5.5).

5.4.4 *Hyperparameters tuning*

Hyperparameter tuning is vital to get the best performance from LLMs and make them more predictable in responding. GPT-4, on which V1 and V2 components rely, has several parameters that allow us to adjust it:

- **Temperature:** The temperature parameter governs the model's predictability and originality. Its values range from 0 to 1. A temperature close to 1 results in more randomness and diversity, while a temperature near 0 makes the model's output more deterministic and predictable. Temperature values between 0.5 and 0.9 are commonly chosen for creative tasks to balance coherence and originality.

- **Top_p:** top_p sampling (nucleus sampling [132]) is used in text generation with LLMs to control the diversity of generated content. It sets a threshold to include only the most probable words that cumulatively exceed a specified probability "p.' Tuning top_p allows for a balance between creativity and coherence in the model's output. The OpenAI manual documentation[2] suggests utilizing one or the other option and setting the unused parameter to the default value.

- **Best_of:** It enables the generation of multiple completions server-side, with the 'best' response being the one with the lowest logarithmic probability per token.

2 OpenAI documentation. https://platform.openai.com/docs/api-reference

- **Frequency_penalty:** The frequency penalty parameter controls the model's proclivity to make repeated predictions. The frequency penalty diminishes the likelihood of previously created words. The penalty is determined by how many times a word appears in the prediction.

- **Presence_penalty:** It encourages the model to create novel predictions by the presence penalty parameter. The presence penalty reduces the likelihood of a word if it previously was in the predicted text. Unlike the frequency penalty, the presence penalty is unaffected by the frequency with which terms appear in previous predictions.

To test and tune the Sentence Splitter and COSMIC Analyzer components, we used the GridSearch algorithm with Cross-Validation from Scikit learn library [231]. The algorithm exhaustively searches a subset of hyperparameter values for the refined components and uses a stratified cross-validation technique to split the data into subsets of folds. This ensures that each fold has a proportional representation of the target variable, making the evaluation more reliable. For each combination of hyperparameters, the algorithm splits the validation sets into four folds, then trains on three folds while testing on the remaining one. This process is repeated four times, with each fold serving as the test set once, and the average F1-score is computed across all folds. This cross-validation approach ensures robust evaluation of each hyperparameter configuration and helps prevent overfitting to specific examples.

We provided only a subset of the GPT-4 hyperparameters to the algorithm, to reduce the search space, making the following assumptions:

- **Best_of** must be set to 1 (the default value) because we need only the 'best' response server-side.

- **Top_p** must be set to 1 (the default value) because we employed the temperature in the parameter grid. The OpenAI documentation suggests changing only one of the two parameters.

- **Presence_penalty** must be set to 0 (default value) to reduce the "creativity" of GPT-4.

- **Frequence_penalty** must be in [0.0, 0.1, 0.2, 0.3, 0.4, 0.5] to reduce the model's proclivity to make repeated predictions.
- **Temperature** must be in [0.0, 0.1, 0.2, 0.3, 0.4] as far as it concerns reducing the creativity of GPT-4 and allowing it to be more deterministic.

Finally, we chose the number of folds for the cross-validation splitting strategy, namely four folds for the GPT-4 model (i.e., V1) refined in splitting sentences and for the model refined in COSMIC analysis (i.e, V2). To validate the results, we used the F1-score of the Rouge metric [184, 185], specifically 1-Rouge (see Section 5.5.3).

The GridSearch process evaluated 25 combinations of temperature and frequency penalty values (5 frequency penalty values times five temperature values). For each combination, we computed the average F1-score across the four folds. The results showed that lower temperature values consistently produced better results, with temperature=0 yielding the highest F1-scores. This result aligns with our goal of making GPT-4's responses as deterministic as possible for COSMIC measurement. Similarly, a frequency penalty of 0 proved optimal, suggesting that penalizing word repetition does not improve performance in our context, likely because COSMIC measurement often requires consistent terminology.

The best settings for V1 and V2 identified by the GridSearchCV algorithm were a temperature of 0, presence penalty of 0, frequency penalty of 0, best of 1, and top p of 1. The Python sources to replicate the training and optimization pipeline are publicly available in the CosMet online repository `https://github.com/gadevito/COSMET`.

5.4.5 *The CosMet Tool*

We developed a tool (see our online repository for sources `https://github.com/gadevito/COSMET`) implementing the CosMet approach, which interacts with the GPT-4 API through HTTP requests using the provided Python bindings [3]. It parses the GPT-4

3 OpenAI documentation. `https://platform.openai.com/docs/api-reference`

responses and displays the output predictions on a web page. We used the Streamlit framework [4] – a Python web framework - to develop CosMet as a web application. Figures 5.13 and 5.14 show the tool's input and output.

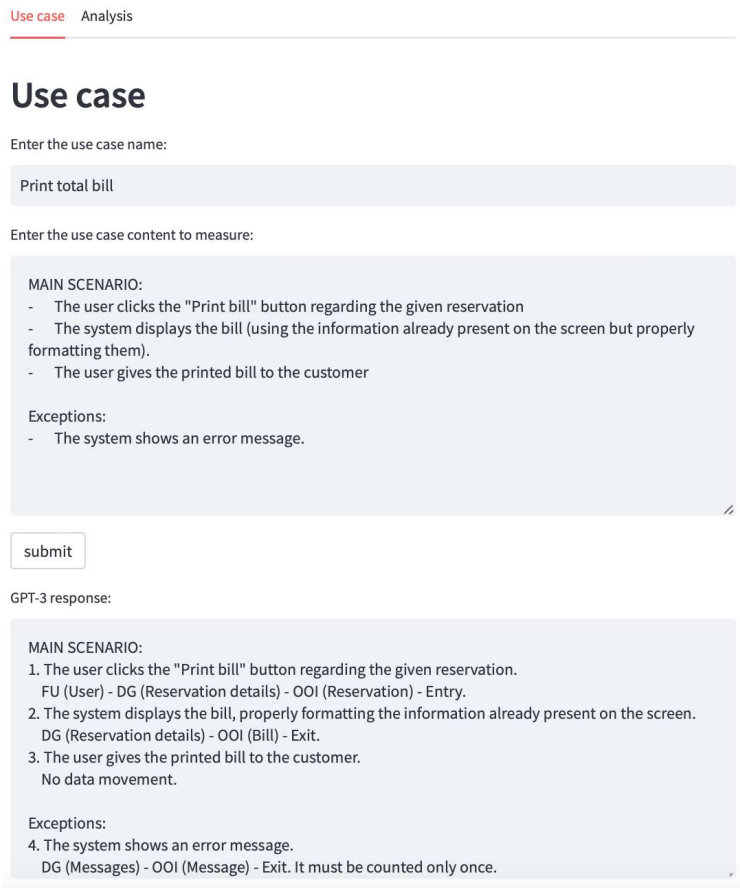
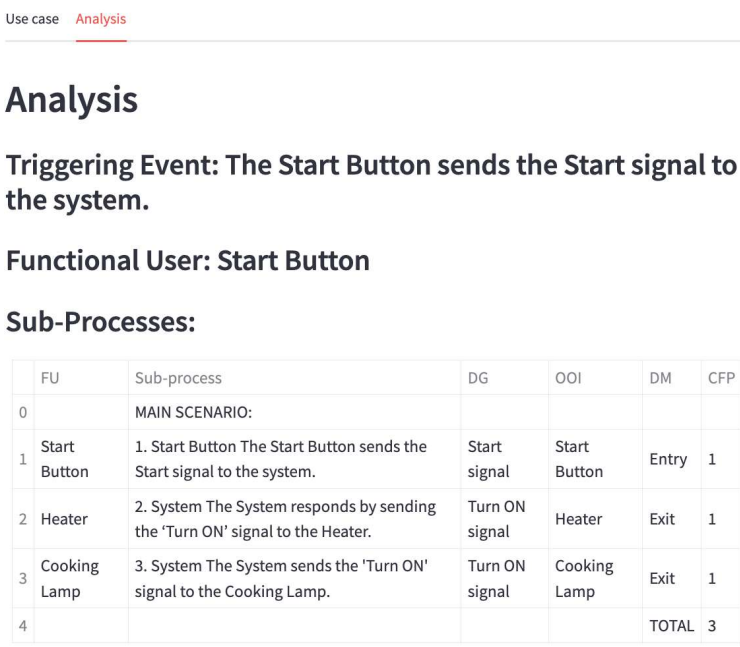


Figure 5.13: CosMet user interface: use case

In particular, CosMet accepts as input the textual representation of a use case and returns information about identified Triggering event, Functional user (FU), Data group (DG), Object of interest (OOI), and Data movement type (DM) and corresponding CFP for each sub-process.

4 Streamlit framework. https://streamlit.io/



	FU	Sub-process	DG	OOI	DM	CFP
0		MAIN SCENARIO:				
1	Start Button	1. Start Button The Start Button sends the Start signal to the system.	Start signal	Start Button	Entry	1
2	Heater	2. System The System responds by sending the 'Turn ON' signal to the Heater.	Turn ON signal	Heater	Exit	1
3	Cooking Lamp	3. System The System sends the 'Turn ON' signal to the Cooking Lamp.	Turn ON signal	Cooking Lamp	Exit	1
4					TOTAL	3

Figure 5.14: CosMet user interface: COSMIC measurement analysis

5.4.6 *CosMet Setup*

Setting up CosMet for a new organization is straightforward and involves the following steps:

1. Reviewing organizational guidelines for writing use cases.
2. Identifying a representative set of use cases.
3. Fine-tuning the LLM using the annotated few-shot examples.
4. Validating the LLM's performance and making iterative improvements as needed.

The following effort estimates are based on the experience of the first author as a practitioner, who is certified in UML 2, COSMIC, and IFPUG, with over 30 years of experience in the IT field and more than 18 years specifically working with FSM across various projects in both public and private sectors. These estimates

reflect real-world implementation scenarios rather than purely research-based settings.

The first step, typically 2 to 4 work hours, involves understanding specific terminologies, sentence structures, and formatting conventions unique to the organization. Indeed, it is essential to mention that companies typically establish guidelines for writing use cases [115]. These guidelines can significantly aid in selecting appropriate few-shot examples for COSMIC analysis. The next step is to identify a representative set of use cases that cover a broad range of functionalities and complexities. This phase usually takes about 4 to 8 work hours. Our experience with diverse domains has shown that this analysis ensures that the selected use cases are diverse and representative. The annotated few-shot examples are then used to fine-tune CosMet. This process, including iterative testing and validation, typically takes 8 to 12 work hours. The last step is validating the CosMet's performance using real-world use cases and making necessary adjustments based on feedback from professional measurers. Initially, this can take an additional 4 to 6 work hours, with periodic reviews as needed. The total time investment for the initial setup of CosMet typically ranges from 18 to 30 work hours, depending on the complexity of the use cases. While this setup phase requires an initial investment of time, it is a one-time effort that pays off in the long run. Once the setup is complete, the same few-shot examples can be reused across multiple projects, ensuring consistency and efficiency.

5.5 EMPIRICAL ASSESSMENT OF COSMET

This section describes the research questions, dataset creation, and methods for evaluating the proposed approach.

5.5.1 *Research Goals and Questions*

The goal of the empirical assessment was to analyze the effectiveness of our approach (i) to measure use cases and (ii) to map the FURs in the form required by the COSMIC General Software Model, together with (iii) assess the amount of time required for COSMIC measurement.

The purpose was to provide empirical evidence that can highlight the benefits and limitations of the proposed approach to make practitioners aware of the strengths and weaknesses they would obtain through the use of CosMet. More specifically, our empirical assessment was driven by three main research questions, namely:

RQ5.1 How effective is CosMet in measuring use cases?

This research question focuses on CosMet's ability to identify Data Movements (Entries, Exits, Reads, and Writes), which represent the final output of COSMIC measurement.

RQ5.2 How effective is CosMet in mapping FURs to the COSMIC Generic Software Model?

This question evaluates CosMet's ability to correctly identify the key elements required by the COSMIC Generic Software Model: Functional Users, Triggering Events, Data Groups, and Objects of Interest. This detailed evaluation is crucial for ensuring transparency in our approach, allowing measurers to validate and, if necessary, intervene in the measurement process.

RQ5.3 How efficient is CosMet in reducing measurement time?

This question assesses how CosMet transforms the measurement process and its impact on the time required to perform COSMIC measurement compared with industry benchmarks.

To address the goal of our study, we followed the prompt engineering guidelines provided by OpenAI [220] and Ekin [99] and conducted mixed-method research, hence combining quantitative performance assessment with qualitative insights from practitioners. As for reporting, we followed the ACM/SIGSOFT Empirical Standards, particularly the "General Standard" and "Mixed Methods" guidelines.

5.5.2 *Dataset Creation and Ground Truth*

The datasets we utilized for our empirical assessment consist of 7 software systems from 4 different domains (Management Infor-

mation System, Microservices/IoT, Real-Time, and Machine Learning), totaling 123 use cases and 119 functional processes:

- Albergate: A Management Information System provided by Kiranet (an SME Italian company), comprising 66 use cases and 62 Functional processes.
- FIDCPM, FID-MTC, and FID-TCT: Microservices of a Telemedicine application developed by Kiranet, providing 37 use cases and 37 Functional processes.
- Rise cooker and Automatic Line Switching: Case studies from the COSMIC Website [39, 41] in the Real-Time domain, comprising 10 use cases and 10 Functional processes.
- U-CURE: An ML application developed by Innovaway (a large Italian company), comprising 10 use cases and 10 Functional processes.

We engaged two professional COSMIC measurers certified in FSM methods (IFPUG and COSMIC) to create the ground truth. The first expert has over 25 years of experience in software engineering and more than 15 years in software effort/size estimation. The second is a manager of an IT company and a member of GUFPI - ISMA[5], with over 10 years of experience in software effort/size estimation.

The ground truth creation process followed three steps:

1. Independent measurement: Each expert independently analyzed and measured the use case specifications.
2. Comparison: The experts compared their measurements to identify discrepancies.
3. Fine-tuning: The experts conducted a fine-tuning session to resolve disagreements and produce shared COSMIC measurement reports.

We calculated the inter-rater agreement between the experts' independent measurements, specifically Cohen's Kappa score [59], before the fine-tuning session.

5 https://www.gufpi-isma.org

The experts agreed on 573 of 593 total movements for Data Movement identification, with only 20 disagreements (one in FID-CPM, 16 in FID-MTC, one in FID-TCT, and two in U-CURE). Cohen's Kappa showed almost perfect agreement ($k = 0.976$) in classifying movements as Entry, Exit, Read, or Write. The agreement levels for other COSMIC elements were also high: Triggering Events and Functional Users showed perfect agreement ($k = 1.000$). In contrast, Data Groups ($k = 0.838$) and Objects of Interest ($k = 0.866$) showed slightly lower agreement.

The lower agreement for Data Groups and Objects of Interest reflects the variations in terminology used by the experts (e.g., "Customer Data" vs. "Customer Information"), which were standardized during the fine-tuning session to ensure consistency. The resulting ground truth includes a comprehensive identification of data movements for each functional process, a complete mapping to the COSMIC Generic Software Model, encompassing functional users, triggering events, data groups, and objects of interest, as well as detailed measurement reports for each software system. All datasets and measurement reports are publicly available in our online repository `https://github.com/gadevito/COSMET`.

5.5.3 *Evaluation Metrics*

We employed three metrics to evaluate CosMet's performance: Rouge-1, BLEU, and BERTscore. Before delving into the details of these metrics, let us review some key definitions. In Natural Language Generation, machine-generated texts are often evaluated against a reference text, which is the ideal output the model should produce. The generated text refers to the model's output, while the reference text refers to the original text the generated text is compared to. An n-gram is a continuous sequence of n elements, such as words, syllables, phonemes, base pairs, or letters, depending on the application. BLEU is a precision-oriented metric that calculates the n-gram overlap between a candidate (generated) text and one or more reference texts. It evaluates the quality of the generated text by measuring its similarity to the reference text(s). BLEU is sensitive to the position of words within n-grams but does not account for the overall sentence structure beyond the size of the n-grams considered.

A key feature of BLEU is the application of a brevity penalty (BP) to discourage overly short candidate translations. The BLEU score ranges from 0 to 1, with higher values indicating greater similarity between the candidate and reference texts. The BLEU score can be calculated using the following formula:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{5.1}$$

Where the brevity penalty (BP) is defined as:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{5.2}$$

And the modified n-gram precision p_n is:

$$p_n = \frac{\sum_{\text{Cand} \in \{\text{Candidates}\}} \sum_{\text{n-gram} \in \text{Cand}} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{Cand} \in \{\text{Candidates}\}} \sum_{\text{n-gram} \in \text{Cand}} \text{Count}(\text{n-gram})} \tag{5.3}$$

In these equations:

- BP is the brevity penalty, which penalizes candidate translations that are shorter than the reference translations.
- p_n is the modified n-gram precision, which measures the precision of n-grams in the candidate translation that match those in the reference translation(s), with counts clipped to the maximum occurrence in any single reference.
- w_n is the weight for each n-gram size, typically set to $\frac{1}{N}$ for uniform weighting across different n-gram sizes.
- N is the maximum order of n-grams considered, often set to 4, which includes unigrams, bigrams, trigrams, and four-grams.
- c is the length of the candidate translation.
- r is the effective reference corpus length, which is often the length of the reference translation that is closest in length to the candidate translation.

- $\text{Count}_{\text{clip}}$(n-gram) is the clipped count of each n-gram in the candidate translation, which is the minimum of the n-gram's count in the candidate translation and the maximum count of that n-gram in any single reference translation.
- Count(n-gram) is the count of each n-gram in the candidate translation before clipping.

Rouge is similar to BLEU, except that Rouge is recall-oriented. Rouge is classified into three types: 1) n-rouge, which signifies n-gram overlap (i.e., 2-rouge and 1-rouge for 2-grams and 1-grams, respectively); 2) l-rouge, which looks for the Longest Common Subsequence rather than n-gram overlap; 3) s-rouge, which is based on skip grams. The n-rouge value is expressed using three metrics: precision, recall, and F1-score (or F-measure). For our evaluation, we used the 1-rouge metric, where recall, precision, and F1-score are defined as follows:

Recall (R):

$$R = \frac{\sum_{s \in \text{Reference Summaries}} \text{Count}_{\text{match}}(\text{unigram})}{\sum_{s \in \text{Reference Summaries}} \text{Count}_{\text{ref}}(\text{unigram})} \tag{5.4}$$

Precision (P):

$$P = \frac{\sum_{s \in \text{Reference Summaries}} \text{Count}_{\text{match}}(\text{unigram})}{\sum_{s \in \text{System Summaries}} \text{Count}_{\text{sys}}(\text{unigram})} \tag{5.5}$$

F1-score (F1):

$$F1 = \frac{2 \times P \times R}{P + R} \tag{5.6}$$

Where:

- $\text{Count}_{\text{match}}$(unigram) is the count of unigrams in the system-generated summary that match any of the reference summaries.
- $\text{Count}_{\text{ref}}$(unigram) is the count of unigrams in the reference summaries.
- $\text{Count}_{\text{sys}}$(unigram) is the count of unigrams in the system-generated summary.

Recall refers to the proportion of unigrams in the reference summaries that are captured by the system-generated summary, while precision refers to the proportion of unigrams in the system-generated summary that are present in the reference summaries. The F1-score provides a balanced measure of the model's performance, combining both recall and precision.

BERTscore leverages contextual embeddings from BERT models to compute similarity scores between generated and reference texts. Unlike Rouge-1 and BLEU, which rely on exact matches, BERTscore can capture semantic similarities even when different words are used. It provides precision, recall, and F1-score based on these semantic similarities.

The recall, precision, and F1-score metrics range from 0 to 1.

5.5.4 *Research method for $RQ_{5.1}$*

To address $RQ_{5.1}$, we compared the COSMIC Data movements produced by COMET against the ground truth created by the expert measurers (described in Section 5.5.2).

For each software project, we evaluated use cases (in their original form as documented in the use case documents) at the functional process level, where each use case (including any included or extended use cases) was analyzed as a single unit. We then computed metrics to establish CosMet effectiveness. In particular, we used the BLEU [227] and Rouge (specifically 1-Rouge) [184, 185] metrics, explained in detail in Section 5.5.3. It is important to note that our evaluation task is not a simple classification problem but a comparison of ordered sequences that represent the flow of Data Movements within functional processes. For example, consider a functional process with four subprocesses where we identify the following sequence: [Entry, Read, Write, Exit]. A different sequence containing the same movements but in a different order, such as [Read, Entry, Exit, Write], would be completely wrong as it misrepresents the actual flow of the functional process. For BLEU, we used weights=(1, 0, 0, 0), which focuses on matching individual Data Movement labels at each position in the sequence, giving 100% weight to unigram matching. This configuration means that we evaluate each subprocess classification independently, comparing CosMet's output with the ground

truth one position at a time. The Rouge-1 metric complements this evaluation by providing precision, recall, and F1-score for each type of Data Movement across the sequence. We performed 10 iterations of the evaluation process on the dataset to address the non-deterministic nature of LLMs. We executed CosMet on all use cases for each iteration and computed BLEU and Rouge-1 metrics. This repeated evaluation allows us to assess the stability and reliability of CosMet's Data Movement identification across multiple runs. We report the average scores and their standard deviations to provide a comprehensive view of CosMet's performance and any potential variations in its output. We used two Python libraries to evaluate BLUE and 1-Rouge metrics: nltk [6] and Rouge [7].

Additionally, to comprehensively assess the effectiveness of our approach, we systematically searched Scopus to identify all the articles proposing COSMIC automation tools and potential approaches to compare with CosMet. Among these, only the article by Ochodek et al. [214] used textual use cases, proposing a deep learning model for end-to-end approximation of COSMIC functional size. As such, this represented the baseline considered in our study. To compare CosMet with DEEP-COSMIC-UC, we proceeded as follows. Ochodek et al. initially assessed the accuracy of DEEP-COSMIC-UC by calculating absolute error (MAE) and median absolute error (MdAE). The values obtained in their study were 3.479 for MAE and 2.204 for MdAE. To ensure a fair comparison, we applied both CosMet and DEEP-COSMIC-UC to our dataset, computing MAE and MdAE for each of CosMet's 10 iterations against the ground truth, and for DEEP-COSMIC-UC's single output.

5.5.5 *Research method for $RQ_{5.2}$*

To address $RQ_{5.2}$, we designed an evaluation approach consisting of two complementary phases: a broad quantitative evaluation and a qualitative assessment with practitioners.

6 Natural Language Toolkit for Python. https://www.nltk.org/
7 Project Rouge. https://pypi.org/project/rouge/

5.5.5.1 *Quantitative Evaluation*

For the quantitative evaluation, we analyzed CosMet's performance in identifying Functional users, Triggering events, Data groups, and Objects of interest across our datasets (described in Section 5.5.2), which includes Albergate, FIDCPM, FID-MTC, Rice cooker, Automatic Line Switching, U-CURE, and FID-TCT use case documents. Note that while DEEP-COSMIC-UC serves as our baseline for Data Movement identification in RQ5.1, it was not included in this comparison as it does not provide capabilities for mapping FURs to the COSMIC Generic Software Model elements.

The evaluation employs three complementary metrics: BLEU [227], Rouge [184, 185], and BERTScore [336] (see Section 5.5.3). For each COSMIC Generic Software Model element (Functional users, Triggering events, Data groups, and Objects of interest), we compared the sequence of elements identified by CosMet against the ground truth at the subprocess level, maintaining their sequential order (e.g., if for a subprocess CosMet identified the Data groups ["Room," "Customer," "Reservation"] and the ground truth contained ["Room," "Customer details," "Reservation"], we compared each Data group at its specific position). BLEU and Rouge capture the quality, precision, and recall of CosMet's output compared to the reference text from the ground truth (see Section 5.5.2), considering the sequential nature of the analysis where each subprocess contains interrelated COSMIC elements. BERTScore addresses semantic similarities, which is crucial when comparing CosMet's output with the ground truth, as semantically equivalent descriptions might be expressed using different words or phrasings (e.g., CosMet might identify a Functional user as "system operator" while the ground truth refers to it as "system administrator"). As in $RQ_{5.1}$, we performed 10 iterations to account for the non-deterministic nature of LLMs, reporting both average scores and standard deviations for all metrics.

5.5.5.2 *Qualitative Evaluation*

Since metrics cannot fully capture the correctness of COSMIC element identification, we performed a further analysis involving five practitioners with two years of experience with COSMIC mea-

surement from an Italian software company and the two certified COSMIC measurers who created the ground truth to answer $RQ_{5.1}$.

In particular, we employed the use case document FID-TCT (described in Section 5.5.2) as a representative case study for this phase, as it encompasses typical characteristics found across our dataset while being of manageable size for detailed manual analysis. The practitioners used CosMet to analyze FID-TCT's use cases and evaluated the appropriateness of the generated Triggering events, Functional users, Data groups, and Objects of interest. To this end, the practitioners filled in questionnaire no. 1 shown in TABLE 5.3, where each question admits answers within a scale proposed by Likert, ranging from 1 to 5, where 1 means a meager value, and 5 indicates a very high value.

Table 5.3: Questionnaire no. 1 provided to the measurers.

No.	Questions
1	How appropriate are the CosMet identified Data groups?
	[1. Highly inappropriate, 2. Inappropriate, 3. Neutral, 4. Appropriate, 5. Highly appropriate]
2	How appropriate are the CosMet identified Objects of interest?
	[1. Highly inappropriate, 2. Inappropriate, 3. Neutral, 4. Appropriate, 5. Highly appropriate]
3	How appropriate are the CosMet-identified Functional users?
	[1. Highly inappropriate, 2. Inappropriate, 3. Neutral, 4. Appropriate, 5. Highly appropriate]

5.5.6 *Research method for $RQ_{5.3}$*

$RQ_{5.3}$ aims to assess how much time is required for COSMIC measurement and how much CosMet can reduce this time in a real-world scenario. This time includes the work the measurers need to evaluate, correct, and validate the CosMet measurement analysis, helping fine-tune the measurement and addressing any ambiguities or discrepancies in the requirements expressed by use cases. To establish a baseline for comparison, we referred to the industry benchmark reported by Ungan et al. [293], which indicates that a COSMIC-certified measurer typically measures 125-500 CFPs daily. In contrast, an uncertified measurer may measure less than

half of this quantity. To assess this, we asked the five practitioners and the two certified professional measurers involved in the analysis to answer $RQ_{5.2}$, and evaluate how CosMet can help reduce measurement time. The participants recorded the total time to perform the CosMet analysis on the FID-TCT use case specifications (described in Section 5.5.2), fine-tuned it for each Functional process, and filled in questionnaire no. 2 shown in TABLE 5.4. We then collected and compared the participants' responses in order to understand the opinions of practitioners about how CosMet can reduce the time required for COSMIC measurement.

Table 5.4: Questionnaire no. 2 provided to the measurers.

No.	Questions
4	To what extent does CosMet reduce the measurement time compared to manual measurement? [1. 0%, 2. < 20%, 3. < 40%, 4. < 60%, 5. <80%]

In addition, to assess the CosMet's performance comprehensively, we calculated the measurement time of DEEP-COSMIC-UC using the same hardware configuration employed for CosMet and compared the results obtained.

5.6 EXPERIMENTAL RESULTS

We present the results for each research question.

5.6.1 $RQ_{5.1}$ - *How effective is CosMet in measuring use cases?*

The average 1-Rouge and BLEU metrics across all the use case documents from the software projects in the four domains indicate high accuracy and consistency in the CosMet's analysis (see Table 5.5). Over 10 iterations, Rouge-1 showed 99.0% (SD = 2.7%) for precision, recall, F1-score, while BLEU reached 99.0% (SD = 3.2%). This means that, on average, CosMet correctly identified 587 out of 593 Data movements across all use cases, with minimal variation between iterations, demonstrating high reliability for automated COSMIC measurement. For the Albergate use cases, CosMet consistently identified 357 out of 359 Data movements

Table 5.5: Quantitave Evaluation Results.

			MIS	IoT / Microservice			Real time		AI	
COSMIC Concept	Metric		Albergate	K01719	K01720	K01726	ALS	Rise Cooker	U-CURE	AVG
TE	1-Rouge	F1-Score	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Std.Dev.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	BLEU	Val.	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Std.Dev.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	BertScore	F1-Score	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		Std.Dev.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
FU	1-Rouge	F1-Score	0.998	0.987	1.000	0.991	1.000	1.000	1.000	0.996
		Std.Dev.	0.015	0.034	0.000	0.035	0.000	0.000	0.000	0.012
	BLEU	Val.	0.996	0.988	1.000	0,987	1,000	1.000	1.000	0.996
		Std.Dev.	0.030	0.032	0.000	0.050	0.000	0.000	0.000	0.016
	BertScore	F1-Score	0.999	0.995	1.000	0,996	1.000	1.000	1.000	0.999
		Std.Dev.	0.009	0.012	0.000	0.017	0.000	0.000	0.000	0.005
DG	1-Rouge	F1-Score	0.917	0.913	0.958	0.948	0,986	1.000	0.926	0.950
		Std.Dev.	0.099	0.120	0.084	0.106	0.031	0.000	0.127	0.081
	BLEU	Val.	0.847	0.905	0.945	0.930	0.979	1.000	0.895	0.929
		Std.Dev.	0.170	0.125	0.107	0.129	0.046	0.000	0.194	0.110
	BertScore	F1-Score	0.967	0.977	0.987	0.982	0.994	1.000	0.983	0.984
		Std.Dev.	0.031	0.029	0.025	0.031	0.013	0.000	0.029	0.023
OOI	1-Rouge	F1-Score	0.890	0.837	0.987	0.793	0.949	0.689	0.801	0.849
		Std.Dev.	0.163	0.299	0.052	0.236	0.093	0.153	0.316	0.188
	BLEU	Val.	0.818	0.817	0.987	0.735	0.914	0.624	0.771	0.809
		Std.Dev.	0.235	0.318	0.030	0.259	0.155	0.290	0.328	0.231
	BertScore	F1-Score	0.966	0.972	0.996	0.950	0.983	0.956	0.958	0.969
		Std.Dev.	0.040	0.049	0.010	0.047	0.030	0.050	0.051	0.040
DM	1-Rouge	Precision	0.998	0.961	1.000	0.992	1.000	1.000	0.983	0.990
		Std.Dev.	0.015	0.091	0.000	0.032	0.000	0.000	0.053	0.027
		Recall	0.998	0.961	1.000	0.992	1.000	1.000	0.983	0.990
		Std.Dev.	0.015	0.091	0.000	0.032	0.000	0.000	0.053	0.027
		F1-Score	0.998	0.961	1.000	0.992	1.000	1.000	0.983	0.990
		Std.Dev.	0.015	0.091	0.000	0.032	0.000	0.000	0.053	0.027
	BLEU	Val.	0.998	0.982	1.000	0.989	1.000	1.000	0.963	0.990
		Std.Dev.	0.010	0.052	0.000	0.039	0.000	0.000	0.119	0.032

TE=Triggering Event FU=Functiona User, DG=Data Group, OOI=Object Of Interest, DM=Data Movement

across all 10 iterations, achieving 99.8% in Precision, Recall, F1-score (SD = 1.5%), and BLEU (SD = 1%).

For the use case documents of FIDCPM, FID-TCT, and FID-MTC, CosMet correctly identified an average 131.9 out of 134 Data movements, reaching 98.4% for precision, recall, and F1-score (SD=4.1%), and 99.0% for BLEU (SD=3.1%).

For the Rice cooker and Automatic Line Switching use case documents, CosMet correctly identified all 47 Data movements across all iterations, reaching 100% in all metrics (SD=0.00%). This result indicates that when comparing sequences of Data Movement types (i.e., Entry, Exit, Read, Write), both BLEU and 1-Rouge metrics indicate that CosMet identified the same sequence as the human measurers, without no differences in the type or order of the movements.

For the U-CURE use case document, CosMet correctly identified an average of 52 out of 53 Data movements out, reaching 98.3% in precision, recall, and F1-score (SD=5.3%), and 96.3% for BLEU (SD=11.9%).

The error analysis revealed several recurring patterns across domains. The first error type concerns distinguishing Data movements from control commands, as seen in Albergate's FP26. CosMet erroneously identified a control command as an Entry Data movement in the subprocess "The user clicks the 'View order details' button." According to the COSMIC manual, this represents a control command that enables user interaction without moving data about an Object of Interest. The error likely occurred due to the presence of action verbs ("clicks") and data-related terms ("details"), which resemble Data movement patterns.

The second error type concerns implicit Data groups requiring domain knolwledge. In Albergate's FP29 and U-CURE's FP10, CosMet failed to identify Entry and Exit Data movements related to implicit data (Customer data and SVC hyperparameters, respectively). For example, in U-CURE's subprocess "The system requests the ML Engine to classify the patient," the Exit Data movement for SVC parameters was missed. This suggests that CosMet struggles to infer implicit data movements that experienced measurers would recognize from context.

A third, albeit rare, error type involves sentence splitting issues. In FID-MTC's FP6, one iteration failed to split the compound sentence "The system searches and displays all the televisit requests for the given doctor received from his (her) patients" into separate search and display subprocesses. However, CosMet demonstrated robustness by still correctly labeling it as "Read/Exit," accurately identifying both Data movements despite the splitting issue.

The fourth pattern emerges in user interaction scenarios, particularly in FIDCPM and FID-MTC. In FIDCPM's FP4, CosMet consistently swapped Data movements in "The Sensor Device sends the raw data to the Mobile" (identified as Exit instead of Entry) and "The Mobile App requests the System to save the measurement data" (identified as Entry instead of Exit). Similarly, in FIDCPM's FP5 and FID-MTC's FP13, FP14, and FP21, the model struggled with entry movements in button-click scenarios such as "The

user selects the 'List all vitals' button." These errors suggest that CosMet may be overly influenced by sentence structure and specific verbs rather than the underlying data flow logic.

These patterns indicate that while CosMet maintains high overall effectiveness, it faces challenges with complex semantic distinctions, implicit domain knowledge, and specific syntactic patterns. These limitations primarily stem from the restricted number of examples used during the refinement phase, constrained by GPT-4's context size limits. Additional few-shot examples focusing on these specific patterns could improve performance in these areas.

Despite minor discrepancies in a few Functional processes, the results demonstrate that CosMet is highly effective in measuring use cases within the considered domains. Tables in Appendix B.1 in our online repository https://github.com/gadevito/COSMET report detailed results in terms of CFPs obtained with CosMet and by manual measurement for each considered system and each identified Functional process.

Regarding the comparison with DEEP-COSMIC-UC [214], the MAE and MdAE values confirmed the results above. In particular, across all iterations, CosMet's MAE scores ranged from 0.026 to 0.053 (average = 0.038), compared to 2.8421 for DEEP-COSMIC-UC, while MdAE values were 0 for CosMet and 2 for DEEP-COSMIC-UC. These results indicate that CosMet's predictions almost perfectly match the expert measurers' ground truth, with an average deviation of only 0.038 Data movements per functional process, while DEEP-COSMIC-UC shows a larger average deviation of about 3 Data movements. The above discussed results allow us to answer $RQ_{5.1}$ as follows:

> **Answer to RQ5.1:**
>
> CosMet is a good starting point for manually fine-tuning the measurement because it displays the interpreted data movements of each FUR; the user has total transparency into the makeup of the count and may properly change the requirements (i.e., ambiguity, lack of clarity).

Table 5.6: Evaluation Results by COSMIC Concept.

	Rouge			BLEU	BERTscore		
COSMIC Concept	Precision (Std.Dev.)	Recall (Std.Dev.)	F1-score (Std.Dev.)	(Std.Dev.)	Precision (Std.Dev.)	Recall (Std.Dev.)	F1-score (Std.Dev.)
Triggering event	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 0.000
Functional User	0.998 (0.009)	0.998 (0.009)	0.998 (0.009)	0.997 (0.014)	0.999 (0.005)	0.999 (0.005)	0.999 (0.005)
Data Group	0.955 (0.073)	0.939 (0.096)	0.944 (0.086)	0.915 (0.127)	0.984 (0.022)	0.981 (0.026)	0.982 (0.024)
Object of Interest	0.853 (0.196)	0.843 (0.204)	0.845 (0.200)	0.801 (0.247)	0.967 (0.042)	0.966 (0.042)	0.966 (0.042)
MIN	**0.853**	**0.843**	**0.845**	**0.801**	**0.967**	**0.966**	**0.966**
MAX	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
STD.DEV.	**0.070**	**0.077**	**0.074**	**0.097**	**0.017**	**0.018**	**0.018**

5.6.2 $RQ_{5.2}$— *How effective is CosMet in mapping FURs to the COSMIC Generic Software Model?*

In the following, we first compare the CosMet-generated text (regarding Functional users, Triggering events, Data groups, and Objects of interest) against the ground truth, using the 1-Rouge, BLEU, and BERTscore metrics. Then, we analyze the answers to questionnaire no. 1 filled in by the five practitioners and the two certified COSMIC measures involved in the analysis of $RQ_{5.1}$, considering the results of the evaluation they performed on the FIDTCT use case document.

5.6.2.1 *CosMet analysis vs manual measurement*

The average 1-Rouge and BLEU metrics values for all the use case documents of the software projects from the four domains indicate a high level of performance (see Tables 5.5 and 5.6 and Appendix C1 in our online repository for additional details). Specifically, the BLEU scores range from 80.1% to 100% (SD=9.7%), the F1-Score values for the 1-Rouge metric vary between 84.5% and 100% (SD=7.4%), and the F1-Score values for the BERTscore metric range from 96.6% to 100% (SD=1.8%) across all iterations.

For the Albergate system (comprising 359 Data groups and 24 Objects of interest), CosMet perfectly identified Triggering events, reaching 100% across all metrics (SD=0.0%). For Functional Users, the system demonstrated exceptional performance with Rouge

metrics showing precision, recall, and F1-score values of 99.8% (SD=1.5%), BLEU score reaching 99.6% (SD=2.9%), and BERTscore achieving 99.9% across all measures (SD=0.9%). Regarding Data Groups, CosMet maintained strong performance, with Rouge metrics showing precision at 93.9% (SD=8.4%), recall at 90.4% (SD=11.1%), and F1-score at 91.7% (SD=9.9%). While the BLEU score was lower at 84.7% (SD=17.0%), BERTscore demonstrated robust results with all measures above 96% and lower variability (SD≈3%). For Objects of Interest, Rouge metrics achieved precision at 91.7% (SD=16.1%), recall at 87.9% (SD=16.9%), and F1-score at 89.0% (SD=16.3%). While the BLEU score decreased to 81.8% (SD=23.5%), BERTscore maintained excellent results above 96% (SD≈4%).

As for the use case documents of FIDCPM, FID-TCT, and FID-MTC (characterized by 134 Data groups and 18 Objects of interest), CosMet correctly identified the Triggering events and Functional users for all the use cases, reaching 100% in all metrics. Regarding Data Groups, performance was consistent across metrics, with Rouge achieving precision, recall, and F1-score all around 94% (SD≈10%). BLEU score reached 92.7% (SD=12.1%), while BERTscore showed the best results with all measures above 98% and lower variability (SD≈3%). For Objects of Interest, performance showed more variability, with Rouge metrics achieving precision at 86.6% (SD=19.8%), recall at 87.9% (SD=19.8%), and F1-score at 86.9% (SD=19.6%). While BLEU score was lower at 84.6% (SD=20.2%), BERTscore maintained strong results above 97% (SD≈4%).

Similarly to previously considered domains, CosMet correctly identified the Triggering events and Functional users for all the use cases of Rice cooker and Automatic Line Switching (characterized by 47 Data groups and 8 Objects of interest), achieving 100% in all metrics. Regarding Data Groups, performance was exceptional across all metrics. Rouge showed outstanding results with precision at 99.2% (SD=1.3%), recall at 98.9% (SD=1.8%), and F1-score at 99.0% (SD=1.5%). BLEU score achieved 98.5% (SD=2.3%), while BERTscore demonstrated near-perfect performance with all measures at 99.6% and minimal variability (SD=0.6%). For Objects of Interest, performance remained strong but showed more fluctuation. Rouge metrics achieved precision at 87.7% (SD=10.7%), recall at 86.7% (SD=13.2%), and F1-score at 87.1% (SD=12.3%).

While the BLEU score was lower at 82.7% with higher variability (SD=22.3%), BERTscore maintained excellent results with consistent 97.5% across all measures (SD≈4%).

Concerning the fourth domain, CosMet correctly identified the Triggering events for all use cases of U-CURE (comprising 52 Data groups and 9 Objects of interest), reaching 100% in all metrics. It correctly determined the Functional users in all Functional processes, reaching 100% in all metrics. Regarding Data Groups, performance remained strong with Rouge metrics showing precision at 94.6% (SD=9.2%), recall at 91.7% (SD=15.3%), and F1-score at 92.6% (SD=12.7%). While the BLEU score reached 89.5% (SD=19.4%), BERTscore maintained excellent results with all measures above 98% and lower variability (SD≈3%). For Objects of Interest, Rouge metrics reached 80.1% across precision, recall, and F1-score (SD=31.6%). The BLEU score achieved 77.1% (SD=32.8%), while BERTscore maintained higher performance with precision at 95.4% (SD=5.6%), recall at 96.2% (SD=4.7%), and F1-score at 95.8% (SD=5.1%).

These results indicate that while CosMet provides reliable identification of COSMIC elements and a valuable starting point that significantly reduces the manual effort required for COSMIC measurement, manual verification might be beneficial, particularly for Objects of Interest where average F1-scores range from 80.9% (BLEU) and 84.9% (Rouge) to 96.9% (BERTscore). A deeper analysis revealed several identification challenges across all domains that could explain these variations. Initially, during the very first iteration, CosMet struggled with Functional User identification in multiple cases: nine Functional processes in Albergate (FP 39, 43, 46, 57-62) and twenty-one in FIDCPM and FID-MTC (FP 1, 2, 5-9 in FIDCPM and FP 1, 2, 4-6, 8, 10-15, 18, and 23 in FID-MTC). The problem was primarily due to the generic references to users as "User" or "Actor" in use case descriptions, as shown in the following example:

The user clicks the "View order details" button for the order of his (or her) interest.

When these use cases were modified to specify the Functional User names explicitly, CosMet correctly identified all Functional Users, achieving perfect scores across all metrics. Additional issues in Functional user identification emerged in two specific cases.

In FP26 of Albergate, as reported in Section 5.6.1, CosMet erroneously identified a control command as an Entry Data movement, reporting an additional Functional User, while in FP29, it failed to identify an Entry Data movement, resulting in a missing Functional User. Regarding Data Groups and Objects of Interest identification, we observed both splitting and recognition challenges. In one iteration for FID-MTC's FP6 (as reported in Section 5.6.1), the splitter component failed to properly separate a compound sentence, resulting in missed identification of a related Object of Interest and its associated Data Group. Additionally, the Rouge metric values were particularly low (<60%) in a few cases: six Functional processes in the MIS domain, four in the IoT/Microservice domain, one in the Real-time domain, and one in the AI domain. Interestingly, the BERTscore metric performed significantly better for these same processes, with F1 scores ranging from 85% to 93%. This discrepancy can be attributed to ambiguities in the use case requirements. For instance, we found cases where synonyms were used for the same Object of Interest (e.g., "Feature" instead of "Categorical feature" in U-CURE). Due to GPT-4's context size limitations, we could not include additional few-shot examples to improve synonym identification.

5.6.2.2 *Appropriacy evaluation*

The analysis of responses to questionnaire no. 1 indicated a strong agreement among the interviewees (see Table 5.7 and Apendix C2).

In particular, it revealed that the mean scores for questions 1 and 3 are both 4.6, with a median of 5, suggesting that participants generally rated these aspects (i.e., appropriacy in identifying Data groups and Functional users) of CosMet highly. Question 2 (about appropriacy in identifying Objects of interest) has a slightly lower mean score of 3.9 and a median of 4, indicating a positive reception. The standard deviation for questions 1 and 3 is 0.53, while for question 2 it is 0.38, which shows a relatively small spread in the responses, further confirming the consistency of the positive feedback.

Table 5.7: Participants's answers to the questionnaire no 1.

Participant	Question no. 1	Question no. 2	Question no. 3
1	4	3	4
2	5	4	4
3	4	4	4
4	4	4	5
5	5	4	5
6	5	4	5
7	5	4	5
MEAN	**4.6**	**3.9**	**4.6**
MEDIAN	**5**	**4**	**5**
STD. DEV.	**0.53**	**0.38**	**0.53**

These findings corroborate the earlier sections, highlighting the effectiveness of CosMet in accurately identifying relevant components for the users, and lead us to answer $RQ_{5.2}$ as follow:

Answer to RQ5.2:

CosMet can successfully map the FURs to the form required by the COSMIC Generic Software Model because it correctly identifies Functional users, Triggering events, and Data Movements in use case specifications. Furthermore, it performs fine in recognizing Data groups and Objects of interest and generates excellent qualitative mapping.

5.6.3 $RQ_{5.3}$— *How efficient is CosMet in reducing measurement time?*

The study conducted on the FID-TCT microservice application, which has a total functional size of 17 CFPs, revealed that the average durations for each Functional process, as reported by participants, varied. The aggregated data from the participants' measurements indicated that the average time to complete the CosMet approach for a single Functional process ranged from 19.27 to 62.52 seconds. When considering the entire application, the total average duration for executing the CosMet approach was 198.21 seconds. Additionally, the time required for fine-tuning the analysis was recorded at 258.29 seconds. Therefore, the overall execu-

tion time for the CosMet approach, including fine-tuning, amounted to 456.49 seconds.

These timing results reflect a significant transformation of the measurement process. In traditional manual measurement, practitioners need to (1) read and understand each use case, (2) identify all COSMIC elements (Functional Users, Data Groups, Objects of Interest), (3) count and classify Data movements, (4) document the measurement, and (5) review their work. With CosMet, this process is streamlined to (1) running CosMet on the use cases, (2) reviewing and fine-tuning CosMet's output, and (3) validating the results. To put these results in perspective, a COSMIC-certified measurer typically processes 125-500 CFPs daily [293], while uncertified measurers achieve less than half this rate. CosMet can reduce measurement time by 53.38% to 88.35%, enabling the analysis of approximately 1,072.53 CFPs within an eight-hour workday. This efficiency gain was confirmed by practitioners, with 71% indicating a 60% reduction in measurement time, and 14% reporting even greater time savings. Participant feedback through questionnaire no. 2 showed consensus on the 60% time reduction, with an average response score of 4. Detailed measurement durations and response distributions are available in Table 5.8 and Figure 5.15 (see Appendix D in our online repository for additional details).

When compared to DEEP-COSMIC-UC [214], which averages 4.23 seconds per use case and 10.19 seconds for the entire FIDTCT model, CosMet is slower (≈ 19 times). However, while DEEP-COSMIC-UC only estimates CFPs, CosMet provides comprehensive COSMIC analysis and measurement results, offering greater transparency and allowing practitioners to refine use-case documents and fine-tune the analysis.

All this leads us to answer $RQ_{5.3}$:

> **Answer to RQ5.3:**
>
> CosMet can significantly reduce the measurement time by executing the analysis in minutes. This time includes the work the measurer needs to evaluate, correct, and validate the CosMet measurement analysis.

Table 5.8: Measurement durations recorded by the participants for FID-TCT.

Functional Process No.	Average COMET duration (seconds)
1	62.52
2	51.4
3	23.59
4	41.41
5	19.27

(A) TOTAL COMET Time	**198.21**
(B) FINE-TUNING Time	**258.29**
(A+B) TOTAL Time	**456.49**
CFP per second	**0.04**
TOTAL Number of CFPs in 8 hours	**1,072.53**

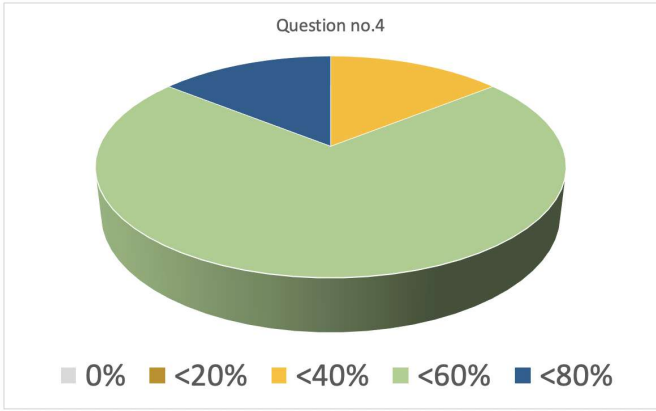


Figure 5.15: Time reduction using CosMet according to the measurers.

5.7 DISCUSSION, IMPLICATIONS, AND LIMITATIONS

In the following, we first discuss the practical implications of CosMet and outline directions for future work. Then, we analyze the threats to validity that may have influenced our findings and how we mitigated them.

5.7.1 *Implications and Future Work*

Our primary contribution is establishing a foundational framework for integrating LLMs into the COSMIC FSM process. While comparing COSMIC with other FSM methods like FPA could provide valuable insights, this work deliberately focused on COSMIC automation to avoid introducing complexities related to the inherent differences between FSM methods. Such comparative analysis remains an interesting direction for future research.

A key practical implication of our work is the transformation of the measurement workflow through CosMet. Traditional manual measurement requires practitioners to perform five distinct steps: (1) read and understand each use case, (2) identify all COSMIC elements, (3) count and classify Data movements, (4) document the measurement, and (5) review their work. CosMet streamlines this into three steps: (1) running the automated analysis, (2) reviewing and fine-tuning the output, and (3) validating the results.

This efficiency gain is substantial - our empirical assessment shows that CosMet enables the analysis of approximately 1,072 CFPs within an eight-hour workday, representing a time reduction of 53-88% compared to traditional manual measurement.

The practical benefits of CosMet extend beyond time savings. Organizations can leverage CosMet to standardize their measurement process, reducing the variability typically introduced by different measurers' interpretations. The tool's structured output provides transparency in the measurement process, allowing practitioners to trace how each measurement was derived and make informed adjustments where necessary. This aspect is particularly valuable for training new measurers, as they can learn from examining CosMet's detailed analysis and reasoning.

CosMet's ability to process natural language requirements also has significant practical implications for modern development practices. In Agile environments, where requirements frequently evolve and are often expressed as user stories, CosMet can provide rapid measurement feedback, enabling more accurate and timely effort estimation. The tool's quick processing time allows organizations to measure requirements as they are written, supporting more dynamic planning and estimation processes.

Several promising directions for future research emerge from our practical experience with CosMet. First, investigating the integration of CosMet with existing project management and estimation tools could enhance its practical utility in development workflows. Additionally, exploring how CosMet could be adapted to handle different requirement formats (beyond use cases) would increase its applicability across different development methodologies. Furthermore, studying how practitioners interact with and learn from CosMet's detailed analysis could improve organizational measurement training and knowledge transfer. Finally, investigating how CosMet could be combined with other estimation approaches in an ensemble strategy could lead to more robust prediction models. This ensemble approach could improve accuracy, adaptability, and robustness in software development effort estimation, enhancing prediction reliability and scalability in different project contexts.

5.7.2 *Threats to validity*

This section discusses possible limitations that could have biased our findings and how we tried to mitigate them.

5.7.2.1 *Threats to construct validity*

In our study, we identified potential construct validity threats. The first one concerns the metrics for assessing CosMet's performance. We combined BLEU, Rouge, and BERTScore metrics with the evaluation of certified COSMIC experts, ensuring technical and practical assessment. A second threat concerns mistakes and subjectivity in manual measurement, potentially impacting the ground truth used as a reference for evaluating CosMet. We involved two certified COSMIC professionals to mitigate this threat. They independently analyzed and measured the use cases and then performed a fine-tuning session to arrive at shared COSMIC measurement reports for each use case document.

5.7.2.2 *Threats to internal validity*

The main threats to internal validity relate to our experimental setup and controls. First, regarding CosMet's efficiency measurement, we mitigated potential bias by collecting timing data from all seven participants working on the same FID-TCT use cases, ensuring a controlled assessment. The selection of hyperparameters for GPT-4 could also affect our results. As detailed in Section 5.4.4, we controlled this threat by employing the GridSearch algorithm with Cross-Validation to identify optimal parameter settings. Preparing few-shot examples represents another potential threat to internal validity. We mitigated this by selecting examples from the COSMIC C-REG case study and engineering additional examples to cover batch processes and typical message exchanges with external components, as described in Section 5.4.2. The effectiveness of these examples was validated through the preliminary validation sets V1 and V2, ensuring their appropriateness for COSMIC measurement.

5.7.2.3 *Threats to external validity*

External validity concerns the generalizability of our findings to different contexts and situations. The primary threat relates to the representativeness of our dataset. While we included use cases from multiple domains, CosMet's performance might vary when applied to use cases from other domains or with different characteristics. This aspect was evident in our results, where domain-specific terminology impacted performance. For example, in the AI domain, where terms like "Feature" versus "Categorical feature" led to lower Rouge metric values (<60%) compared to other domains. Similarly, technical terms related to sensor data and measurements occasionally resulted in identification challenges in the IoT domain. We partially mitigated this threat by including diverse use cases from four domains and by providing a systematic approach for domain adaptation through a few-shot examples selection (as detailed in Section 5.4.6) . We also made our datasets publicly available in our online repository to enable further validation across different contexts. The non-deterministic behavior of LLMs challenges generalizability, as GPT-4's responses may

vary between runs. We mitigated this threat by performing 10 iterations per experiment and reporting average and standard deviations (Sections 5.5.4, 5.5.5, 5.6.1, 5.6.2.1) while controlling GPT-4's behavior through hyperparameter tuning (Section 5.4.4). The comparison with DEEP-COSMIC-UC as the sole baseline may seem limited. However, as noted in Section 5.5.4, it was the only available tool for COSMIC measurement from textual use cases. We mitigated this issue by providing detailed metrics and sharing our experimental materials. Finally, domain-specific terminology may affect CosMet's performance across different domains. However, as detailed in Section 5.4.6, we mitigated this threat by selecting domain-representative few-shot examples during setup. We also provided a clear workflow to set up CosMet for a new organization, ensuring robust cross-domain adaptation.

5.7.2.4 *Threats to conclusion validity*

Conclusion validity concerns the reliability of our conclusions and the statistical relationship between treatment and outcome. A primary threat relates to GPT-4's token limit of 8,192 tokens, which could affect the reliability of our results when processing large or complex use cases. As described in Section 5.4.2, we mitigated this by carefully managing token allocation during refinement, leaving over 5,200 tokens to process a use case and produce the COSMIC analysis. The statistical reliability of our results might be affected by the sample size of use cases and the number of practitioners involved in the evaluation. Our dataset included 123 use cases across domains and seven practitioners. To provide transparency about the variability in our findings, we reported standard deviations for all our measurements across the 10 iterations performed for each experiment. The reliability of our results might be affected by both the practitioners' characteristics and their familiarity with CosMet. We involved seven measurers (five COSMIC-experienced professionals and two certified measurers) and controlled learning effects by using the simpler FID-TCT model for timing data collection. As shown in Section 5.6.3, time savings were consistent across all participants.

5.8 CONCLUSION

We have presented an LLM-based approach, named CosMet, to measure use cases specified in natural language. We instantiated CosMet using GPT-4. However, our approach allows flexibility in choosing the most suitable LLM for parsing language in use case specifications and producing COSMIC measurement analysis. We evaluated CosMet with an empirical study comprising 123 use cases and 119 functional processes. Seven professional measurers assessed the results and evaluated the tool from a quantitative and a qualitative point of view. The results of our study have revealed that CosMet is effective, transparent, and efficient in automating COSMIC measurement of use case models. CosMet provides a transparent and interpretable analysis of use cases, allowing practitioners to understand how the measurement is derived and to make any necessary adjustments. This transparency enables practitioners to fine-tune the measurement and address any ambiguities or discrepancies in the use case requirements. It also improves the accuracy and reliability of the measurement process and ensures that the measurement aligns with the specific context and requirements of the software project. In addition, CosMet surpasses current NLP approaches in COSMIC measurement for effectiveness and completeness.

Thus, our solution has the potential to bring significant advancements in the way FSM is performed in the software industry, making it more accurate, efficient, and cost-effective. Professionals may use the CosMet-generated output to estimate the functional size measurement or refine it further. However, we identified a few limitations of our approach, precisely the limited LLM's context size (i.e., GPT-4 has a context size of 8,192 tokens). Nevertheless, given the premises, we believe that companies can adopt our approach in their measurement practices. Indeed, by refining GPT-4 with their datasets and linguistic style, companies can use CosMet to automate the COSMIC measurement. Furthermore, fine-tuning GPT-4 with more use cases may allow us get more out of the model. Therefore, our future work will include the identification of several multilingual datasets to fine-tune GPT-4. Moreover, future work could explore the integration of alternative and supplementary UML or SysML model types or other artifacts into

automated COSMIC measurement approaches to enhance the evaluation of design alternatives under specific cost and performance restrictions. It is also important to note that many other size measurement frameworks exist, such as standard function points, use case points, and story points, which could be explored in future research using similar methods. Finally, we also plan to explore domain-specific training to address the sensitivity of LLMs to different discourse domains, ensuring CosMet's robustness and adaptability across various industries.

6

MEET C4SE: YOUR NEW COLLABORATOR FOR SOFTWARE ENGINEERING TASKS

6.1 INTRODUCTION

Software development's increasing complexity demands the assistance of chatbots to automate tasks and boost collaboration among teams [33, 273]. These conversation agents support various tasks, including coding, information sharing, and team coordination [11, 49, 209].

However, currently, available Software Engineering (SE) chatbots have limitations. Their common shortcomings include difficulty in tidying up prompts due to inadequate context and a restricted task scope [199, 311, 321]. Most are highly task-specific, often limiting their functionality to a solitary task like suggesting code, thereby curtailing their usefulness to software developers.

This chapter presents C4SE, an open-source, extendable Conversational Chatbot for Software Engineering designed to overcome the limitations of traditional software engineering chatbots and to provide support in various tasks such as code review, code suggestions, unit and acceptance test case generation, and GitHub task executions [11]. It utilizes vector databases to implement long-term memory capabilities, enabling the chatbot to quickly learn and recall past interactions, providing customized solutions to problems, and aiding software engineers throughout the software development lifecycle.

Furthermore, we delve deeply into the potential of vector databases to dynamically enlarge LLMs' knowledge base (e.g., uploading new information, technical documents, code, etc.), to allow C4SE to provide accurate responses and realistic support to software engineers. The proposed approach aims to promote efficient team cooperation, enhance project success, and improve the software engineering process.

6.2 RELATED WORK

The recent progress in Artificial Intelligence (AI) and Natural Language Processing (NLP) led to the widespread utilization of *chatbots* [270, 312], especially in the software engineering context [33, 235, 310]. A *chatbot* is a particular type of *bot*—an application that automates repetitive, dull, or predefined tasks [232, 273]—characterized by the fact that it communicates with users using a communication channel and natural language.

In the context of code generation, Qasse, Mishra, and Hamdaqa [235] proposed *iContractBot*, a conversational bot, to aid developers in constructing smart contracts on blockchain platforms. A use case is presented to demonstrate how the bot can be utilized for creating models and generating the deployment artifacts for smart contracts based on a simple conversation. Moreover, GitHub and OpenAI collaborated to release *GitHub Copilot*, a so-called "AI pair programmer" able to support developers in writing source code. Utilizing the comprehensive OpenAI Codex model, which underwent training on open-source GitHub code, Copilot is competent in proposing code snippets for various programming languages starting from user comments. More related to testing activities, Okanović et al. [215] developed PerformoBot. The bot operates in the context of load testing and guides developers through the process of properly specifying the parameters of the test, other than automatically executing it.

Analyzing such tools, we identified two important limitations when analyzing the state of the art about bots for SE tasks. First, most of the available bots are designed for performing a single (or few) tasks, forcing developers to adopt many tools—often not goodly integrated through each other—during their work cycle. Second, practitioners lament a lack of context awareness of the utilized bots [199, 311, 321], which answer questions generally and not considering their working habits.

As a result, there is a need to find different approaches for chatbots based on LLMs to manage long texts (for example, for Q&A), enlarge their memory to deal with extended conversations, augment their knowledge base, and execute external software and functionalities. C4SE aims to address these limitations, as described in detail in the next section.

6.3 THE C4SE SYSTEM

C4SE is a chatbot designed to perform various software engineering tasks and support both engineers and managers during the software development lifecycle. The bot relies on three different technologies: LangChain,[1] GPT-3.5, and Pinecone.[2] LangChain is a software development framework designed to simplify the creation of applications using LLMs. LangChain helps create more complex workflows using agents and chaining different prompts interactively. GPT-3.5 is one of the most advanced LLMs and a powerful tool for natural language processing tasks. Therefore, it allows C4SE to provide an intuitive and fluid conversation with users.

Finally, Pinecone is a fully managed vector database that stores and indexes the vector representation of NLP sentences, allowing for fast and accurate similarity search and retrieval across massive datasets of unstructured data. Combining these technologies allows for the orchestration of the different tasks of C4SE and the enhancement of the bot with a large knowledge base.

Another essential aspect is that LangChain allows using different types of LLMs and vector databases, simply interfacing with new ones. Therefore, it is possible to choose the most suitable tool for each task; for instance, it is possible to make various LLMs interact to carry out a specific task in the best way.

6.3.1 *Motivating Scenarios*

In this section, to better understand the potential impact and contribution of C4SE, we present two scenarios—as-is and visionary [37].

The former aims to provide the current state and identify limitations and challenges; the second present the situation with the tool and stresses the gained benefits (the two scenarios share the same characteristics for the actor).

1 LangChain site: https://github.com/hwchase17/langchain
2 Pincone site: https://www.pinecone.io/

As-is Scenario

Mary is a software engineer working in a software team that has to implement a software module starting from a use case. Such a software module must take a list of books (as information) from an API and translate them into a format that an already implemented web application uses. Mary has no knowledge about the API, so they ask a bot, GitHub Copilot, to help implement the module. Therefore, Copilot provides Mary with the code, and Mary implements it and makes the software module. Then, they use Probot—a bot for automatically performing some action on GitHub—to push the code to the repository. On the day above, Mary wanted to update the code and make some documentation for acceptance testing. First, Mary indicates Probot to clone the code from the repository; to do this, she needs to specify precisely the repo and code to clone. After updating it, she started from the use case to write the acceptance testing for the implemented feature. Unfortunately, there is no bot for performing this, so Mary has to do this on their hand.

The as-is scenario described a typical situation reported by many works on state of the art about chatbots adopted for software engineering. First, using different bots to perform each task leads to a **loss of productivity** due to the time spent interacting with and mastering the different tools [273, 311]. Second, linked to the previous issues, there is a **lack of context issue**; the fact that the tools do not share or maintain knowledge leads developers to repeat essential information to achieve their goals [199, 311, 321]. Moreover, the use of third-party tools—most of the time deployed on the web—could also bring **privacy issues**—at least in terms of practitioners' perceptions [35, 103, 204].

Visionary Scenario

[...] *Mary has no knowledge about the API, so he asks C4SE to help implement the module. Therefore, the bot provides Mary with the code, and Mary implements it and makes the software module. Moreover, Mary asks the bot to push the developed component on the GitHub repository. The day after, Mary wanted to update the code and do acceptance testing for the developed feature. Using C4SE, he simply asks the bot to provide the code generated the day before and give it the use case asking for acceptance testing specification; C4SE, using the collected context, provides a possible test based on the high-level specification and the code.*

From the visionary scenario, we can evince how the previously-mentioned limitations are partially or fully addressed. First of all, having one bot that does everything negates the need to move to different tools and waste time. Moreover, the capacity of C4SE to maintain information and the fact that all the tasks involve its use lead to the maintenance of the context and address the lack of contextual information that characterizes the state-of-the-art solutions. Furthermore, C4SE is designed to be an open-source on-premise tool resolving the privacy issues derived from using on-cloud third parties tools. Finally, as an ulterior note, our solution partially resolves the well-known ***noise problem***—i.e., the tendency of a bot or a set of collaborating bots to be unnecessarily verbose, leading to information loss [103, 311]. Indeed, having a single bot to communicate with that is tailored to the single user and business brings improved communication through a more precise interaction.

6.3.2 *Tool Architecture and Functionalities*

The main goal of C4SE is to provide practitioners with a unique place to perform the majority of their tasks—e.g., code and test generation. In order to reach this, we developed the tool as a typical three-tier application characterized by various modules depicted in Figure 6.1.

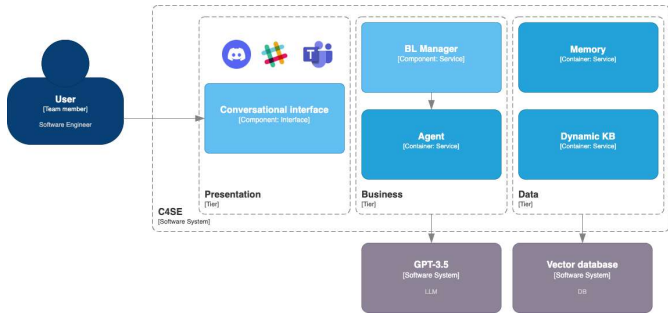


Figure 6.1: The C4SE's architecture.

CONVERSATIONAL INTERFACE. This component is responsible for making the bots feature available to different communication platforms—e.g., Teams, Slack, and Discord.

BL MANAGER. Such a module is responsible for classifying user requests and executing the steps to fulfill them—implemented using agents described below. It exploits the power of GPT-3.5 to classify requests using a specialized prompt, which allows launching the correct agent based on user queries and executing external scripts and services that cannot be done with GPT-3.5 alone—e.g., GitHub API query.

AGENTS. Each potential request for C4SE—or set of cohesive requests—is mapped in an agent. For example, all the requests related to GitHub are responsible for a single agent (GitHub Task Agent). An agent is a software component implementing the application logic to fulfill a specific request. The package Agents contains the developed agents and offers a unified interface for querying them. Each agent is a service that uses its specific pipeline (or chain) to produce responses. A chain allows the joining of various components to form a unified application. For instance, we can create a chain that accepts input from the user, applies a prompt template to format it, and transfers the formatted response to an LLM. We can develop more complex chains by integrating numerous chains or combining chains with other components (i.e., Vector databases, LLMs, Document loaders).

MEMORY. This module uses a specialized vector database to memorize conversations between C4SE and users, allowing long-term memory. We can configure the Memory component to temporarily (i.e., for an hour) or permanently store the conversations. Having long-term memory, C4SE can extract the most relevant information from previous conversations to answer user queries appropriately. For example, suppose the user has previously asked C4SE to write a Java Entity class using JPA and wants the bot to explain the code. In that case, the user can ask, "Can you explain the previous code?". C4SE, using the Memory module, will retrieve from the conversation history with that specific user (the vector database index) the information relevant to the question (i.e., the previous Java class). The bot will then augment the context and, by exploiting the correct agent, send the question to the LLM to provide the final answer.

DYNAMIC KNOWLEDGE BASE. Such a module stores and indexes new data as embedding vectors into C4SE. It provides the necessary services to "upsert" (i.e., update and insert) and "retrieve" documents exploiting vector databases. Documents can be of any type, such as plain text, pdf, html, Microsoft Word, Excel, Powerpoint, XML, etc. The upsert service converts documents into embeddings and indexes them in the vector database. The retrieval service allows searching for the most relevant documents from a semantic point of view to a given search query. This component allows searching for the most relevant information in the C4SE knowledge base and ingesting this information as an augmented context into the prompt to send to GPT-3.5. For example, we can upload the LangChain documentation using the upsert service and then ask C4SE questions about it (i.e., "provide the snippet to develop a ConversationalRetrievalChain in LangChain"). C4SE will forward the questions to the correct agent (i.e., "SnippetAgent"). Using Dynamic memory, the agent will retrieve the information relevant to the questions and augment the context with it. In our example, this means injecting into the final prompt the collected information as shown in Figure 6.2). In the prompt above, the variable "augmented_context" represents the augmented information, namely the most relevant k results retrieved from the vector database. Then, using the augmented prompt, the agent will request the LLM to generate the final answer. The functional behavior of C4SE (see Figure 6.3) is described in the following example:

Augmented prompt

"... Answer the QUESTION at your best.
Current conversation:{augmented_context}
QUESTION: {input}"

Figure 6.2: C4SE Augmented Prompt Template

1. The user asks the system the following question: *"Please, provide an example of a prompt template code usage in LangChain."*
2. The Conversational Interface forwards the request to the BL Manager component, which uses the Memory component

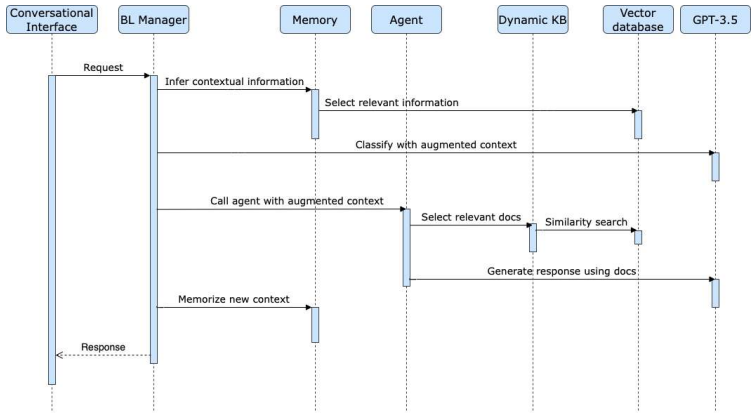


Figure 6.3: C4SE functional behavior.

to retrieve the most relevant information based on the previous conversations.

3. The Memory component exploits the Vector DB to retrieve the most relevant information related to the user's query.
4. Suppose there are no conversations yet; the Memory component returns an empty list.
5. The BL Manager component sends the classification prompt (see Figure 6.4) to the LLM, asking GPT-3.5 to classify the user's question.

Classification prompt

Classify the following question as Code task, GitHub task, Test case, or Q&A. The rules are the following:
- The question should be classified as a GitHub task if it means executing a task on or related to a specific repository.
- The question should be classified as a Test case if and only if it explicitly requests an acceptance test case for a provided use case.
- The question should be classified as a "Code task" if it is related to code or unit test code.
- The question should be classified as Q&A in all the other cases.
Provide only the classification.
Current conversation:{history}
Question:{input}

Figure 6.4: C4SE Classification Prompt

6. The LLM classifies the user's question as a "Code task."

7. The BL Manager component instantiates the Code task agent to process the user's request.

8. To answer the user's request, the Code task agent inquiries the Dynamic KB component for the most relevant documents.

9. The Dynamic KB retrieves the most relevant information using the Vector database (see example in Figure 6.5).
   Part of the context has been omitted for simplicity.

> **Relevant information**
>
> *PromptTemplates are responsible for constructing a prompt value. They are basically objects that expose a format_prompt method for constructing a prompt. For example*
> *from langchain.prompts import PromptTemplate*
> *string_prompt =*
> *PromptTemplate.from_template("tell me a joke about {subject}")*

Figure 6.5: C4SE Relevant Information Extraction

10. The Code task agent sends a new query to the LLM, augmenting the context with the information retrieved, as in Figure 6.6.
    Where the variable "augmented_context" contains the new

> **New prompt with augmented context**
>
> *Act as a software engineer. Check if the question relates to the last code we discussed in the conversation. Answer the question at your best.*
> *Current conversation:{augmented_context}*
> *QUESTION: {input}*

Figure 6.6: C4SE Augmented Prompt

    augmented context.

11. GPT-3.5, using its "in-context learning" capabilities, provides the requested code (see the example illustrated in Figure 6.7)

12. The BL manager component memorizes the new conversation using the Memory component. It sends the final response to the Conversational interface.

GPT-3.5 final response

```
Sure, here's an example usage of PromptTemplate in LangChain:
from langchain.prompts import PromptTemplate
# Define a prompt template
prompt_template = PromptTemplate(
input_variables=["subject"],
template="Tell me a joke about {subject}.")
# Format the prompt template with input
formatted_prompt = prompt_template.format(subject="cats")
# Print the formatted prompt
print(formatted_prompt)
```

Figure 6.7: C4SE Final Response Example

6.3.3 *Novel Characteristics*

C4SE introduces a unique approach that addresses the difficulties software engineering practitioners confront. Its design offers a versatile platform that manages problems established by conventional chatbots. Specifically, our bot is engineered for the software development environment, tailoring it to software engineers and project managers. The specialized prompt that we have designed classifies user requests and provides accurate responses or interactions. The specialized focus of our bot on software engineering allows it to offer precise interactions, reducing the chances of inaccurate responses and sub-optimal productivity.

Are reported in chapter 2) LLMs present inherent challenges as they are stateless and limited in memory conservation, apart from having query limits in many instances. These limitations result in difficulty maintaining the conversation chain leading to context loss, which can frustrate users. C4SE addresses this problem by employing a specialized data store to remember conversations between the bot and users, allowing for the retention of long-term contextual information. Therefore, C4SE provides more nuanced responses based on the conversational context by dividing the user's request into multiple tasks and synthesizing several responses, thus, overcoming the LLM's query constraints. Furthermore, as software engineering continually evolves, practitioners must remain informed about the latest technologies, processes, methodologies, and other developments. C4SE was constructed to be continually updated with an extensive, dynamic knowledge base that enables practitioners to learn new skills and execute

fresh software engineering tasks. This dynamic knowledge base is a distinctive feature of C4SE.

C4SE can utilize multiple Knowledge Base memory modules to accommodate different agents, having two significant benefits: The bot can scale using several KB memory instances. Secondly, each agent can have its unique, specialized dynamic memory. Lastly, the architecture of C4SE is distributable and capable of using microservices and containerized computing technologies. Notably, each agent that handles a unique task and its corresponding interactions with users can be realized as microservices maximizing the potential of its resources.

6.4 C4SE EVALUATION

As a preliminary evaluation of C4SE, we decided to assess its usability in terms of three well-known metrics: effectiveness, efficiency, and satisfaction [120, 134]. For such an aim, we performed a two-step study that has been demonstrated to be helpful in evaluating chatbot usability [245]. First, we recruited 5 participants—with experience in developing and evaluating the usability of software development tools—through a convenience sample strategy and asked them to complete a series of tasks using our bot in an experiment in vitro; then, we interviewed them to collect first impressions and feedback [163].

We asked each participant to execute four tasks: (1) get a use case from a GitHub repository; (2) generate the Java or Python code to implement the selected use case; (3) generate an acceptance test case for the use case; and (4) generate the unit test for one of the classes obtained during the second task. At the end of each step, participants analyzed the answers provided by the bot and asked for explanations or more details. After this, each participant was interviewed by the first two authors of the paper about C4SE effectiveness, efficiency, and satisfaction. For each factor, participants expressed their level of agreement with a series of statements (Likert Scale from *Strongly Disagree* (1) to *Strongly Agree* (5).

Regarding the perceived *effectiveness* of the tool, participants expressed positive opinions. Indeed, related to the accuracy, all the participants evaluated the tool with a value of three or superior

(one expressed 3 and the others 4). We obtained similar results also regarding the *efficiency* of the tool (all participants said values ranged from 3 to 4) and the perceived *satisfaction* derived from using the tool (two reported 4, two reported 3, and one reported 2).

Regarding the concerns, some participants perceived the need for documentation associated with each task; on this, one participant said that *"for being more precise and fast in the request, I would have appreciated having documentation telling me which information was important to provide the tool for doing the task."*.

To summarize, participants in the experiment expressed positive opinions about the tool and its use to accomplish some tasks related to software engineering. These results encourage further tool development and studies.

6.5 CONCLUSION

The evaluation of C4SE showed positive effectiveness, efficiency, and satisfaction results. Participants expressed positive opinions about the tool's capacity to understand the user's request accurately and provide all the relevant information for satisfying the user's needs. Moreover, the tool was perceived as easy to use and fast. Nonetheless, some participants asked for further documentation to understand which information is mandatory for obtaining precise responses.

Despite the potential benefits of C4SE, there are limitations to be addressed in future work. C4SE is currently limited to a predefined set of tasks and is only available in English. It needs expansion to support other languages and handle diverse tasks related to different domains for better generalizability. Furthermore, further research is required to evaluate C4SE with larger sample sizes and a broader range of software engineering practitioners.

7

LLM-BASED GENERATION OF SOLIDITY SMART CONTRACTS FROM SYSTEM REQUIREMENTS IN NATURAL LANGUAGE: THE ASTRAKODE CASE

7.1 INTRODUCTION

Smart contracts are self-executing contracts with the terms of the agreement directly written into code. They enable decentralized and automated transactions across various industries, significantly enhancing efficiency and trust [178, 343]. The adoption of blockchain technology and smart contracts has rapidly grown in sectors such as finance, supply chain management, healthcare, and digital identity verification [69]. However, the complexity and technical expertise required to develop smart contracts can create barriers for non-technical users, limiting their ability to leverage these innovative solutions [69]. No-code platforms have emerged as a promising solution to democratize access to blockchain technology, enabling users with limited technical skills to create smart contracts without extensive programming knowledge [69, 242]. These platforms provide intuitive interfaces and tools that simplify the development process [69]. However, existing no-code platforms often have limitations, such as reduced flexibility and technical precision, emphasizing the need for improved solutions to meet users' diverse needs better [69].

In addition, the limited research on automatic Solidity smart contract generation [157, 206] underscores the need for further exploration of LLMs' potential in this field.

The AstraKode Blockchain (AKB) platform aims to leverage LLMs to enhance its model-driven approach to enterprise blockchain development. The platform offers unique features, including visual development environments like the Network Composer for designing blockchain networks and the Smart Contract IDE for creating and deploying smart contracts. However, despite these advanced features, there remain challenges in enabling non-technical

users to effectively generate Solidity smart contracts from natural language requirements, which can hinder the platform's accessibility and usability.

To address these challenges, this chapter introduces three methods to generate Solidity smart contracts from natural language requirements: *Naive Generation*, *Augmented Generation*, and *Enhanced Generation*. *Naive Generation* focuses on rapid prototyping of smart contracts, addressing industry challenges related to speed and accessibility for users with limited technical expertise. *Augmented Generation* ensures contract completeness and correctness, which are critical for enterprise applications. This method balances speed, completeness, and correctness, making it suitable for scenarios where quality is paramount. *Enhanced Generation* is designed for complex business scenarios within the AKB platform, providing a structured approach to generating comprehensive smart contracts with detailed specifications. Collectively, these methods enhance the capabilities of the AKB platform by offering flexible solutions tailored to different user needs and application contexts.

We performed an empirical evaluation to assess the effectiveness and efficiency of the proposed methods. Our empirical evaluation demonstrated that they effectively generate Solidity smart contracts from natural language requirements. The Enhanced Generation method stood out for its close alignment with the ground truth in terms of code length, complexity, and maintainability, making it ideal for complex scenarios. Practitioner feedback also highlighted its strengths in completeness and adherence to requirements.

This chapter's key contributions include the development of innovative methods that address the challenges of smart contract generation and provide practical relevance for industry practitioners. By leveraging LLMs, these methods aim to streamline the code generation process and reduce barriers for non-technical users.

7.2 BACKGROUND

This section covers smart contracts and no-code platforms in blockchain and highlights the Astrakode Blockchain platform's approach to simplifying development.

7.2.1 *Smart Contracts and No-Code Platforms*

Blockchain is a decentralized public ledger that records unalterable transactions in a chain of blocks, validated by network nodes. It enables trustless transactions without intermediaries like banks, using algorithms such as Proof of Work (PoW) [205] and Proof of Stake (PoS) to achieve consensus and maintain user privacy via virtual addresses [47]. Key features include decentralization, immutability, persistence, and anonymity [178, 343]. Smart contracts are a significant advancement in the evolution of blockchain technology. They are self-executing agreements encoded in software, facilitating automated, trustless interactions without intermediaries. Due to their transparency and immutability, they are suited to applications like decentralized finance, supply chain management, and digital identity verification, though their complexity can be a barrier for non-programmers. No-code platforms have emerged to address these challenges, empowering individuals with limited coding skills to design and manage smart contracts via intuitive visual interfaces using drag-and-drop features, templates, and pre-built components [69]. These platforms democratize access to blockchain technology and enable broader engagement in creating decentralized application (dApps) [69]. They reduce development time and costs, allowing for rapid prototyping without extensive programming knowledge, and include built-in validation and error-checking to mitigate flawed contracts [69]. However, they face challenges, including potential user misunderstandings, code vulnerabilities, and vendor lock-in, which can limit flexibility and control [69].

7.2.2 *Astrakode Blockchain*

AKB is a no-code platform that simplifies enterprise blockchain development by abstracting the complexities of blockchain networks and smart contracts. It maintains technical precision and ease of use by leveraging ontologies, metamodels, and formal grammar. The platform's engine validates models, generates necessary artifacts, and produces deployment-ready code. AKB provides two visual environments to streamline development: the Network Composer, for designing and deploying blockchain networks, and

the Smart Contract IDE, for developing and deploying Solidity-based contracts. These tools support major blockchain technologies like Hyperledger Fabric and Solidity, allowing enterprises to choose the best fit for their needs. With built-in validation and automated auditing tools, AKB enables secure and stable blockchain applications for enterprise needs [145].

7.3 RELATED WORK

This section reviews studies on using LLMs for automating smart contract creation and identifies the research gaps that motivate our work.

7.3.1 *Code Generation*

Code generation automates the process of converting high-level requirements into executable code, using various techniques to meet specified constraints. This section examines traditional program synthesis and recent AI-driven advancements, especially the role of LLMs in automating smart contract creation.

7.3.1.1 *Program Synthesis and AI Approaches*

Early research in code generation encompasses both deductive [126, 194] and inductive [31, 260, 264, 275] approaches. The deductive synthesis method relies on the premise that a thorough and precise formal specification of the user's intent is provided. However, this requirement often proves to be more complex and demanding than the development of the program itself. In contrast, the inductive synthesis approach utilizes inductive specifications such as input/output pairs and examples. A comprehensive overview of significant advancements in program synthesis can be found in a survey by Gulwani et al. [127].

Recently, there has been a growing trend among researchers to integrate neural networks into the code generation process. Machine-learning approaches, built upon the ideas of inductive synthesis, enhance the ability to learn patterns from large datasets of code examples. For instance, Sun et al. [276] suggest a grammar-based structural convolutional neural network to manage long depen-

dencies within code effectively. Bolin et al. [307] propose a dual learning framework that simultaneously trains both the code generation and code summarization models, aiming for improved outcomes in both areas. However, these studies acknowledge limitations such as dependency on predefined grammar and challenges with complex code structures, suggesting areas for future research.

7.3.1.2 *LLMs Code Generation*

Contemporary investigations into code generation methodologies can be broadly classified into three distinct groups: sequence-based methods, tree-based techniques, and pre-trained models. Sequence-based approaches treat code as token sequences, using language models to generate code incrementally based on input descriptions. For example, Hashimoto et al. [129] introduced a retrieval-based model that enhances code generation by using analogous code snippets as supplemental input. Tree-based techniques create a hierarchical code representation from an input description, often called a parse tree or Abstract Syntax Tree (AST). Yin et al. [330] introduced 'Tranx,' a semantic parser using a transition-based neural model to generate actions for tree construction, which builds the AST.

Pre-trained models are trained on extensive datasets and then fine-tuned for tasks like code generation. Several variants have been proposed, such as encoder-decoder (i.e., AlphaCode [180]), encoder-only (i.e., CodeBERT [108]), and decoder-only [36, 219, 240]. However, it is the OpenAI GPT series, including GPT-3.5 [36] and GPT-4 [219], which are decoder-only models, that have shown the most promise in this field [147, 277]. Despite their success, variability in model outputs has raised concerns about consistency. Ouyang et al. [223] found significant differences in correctness, test outputs, syntax, and code structure from the same instruction across multiple requests in ChatGPT's code generation.

Regarding automated smart contract generation, to the best of our knowledge, only a few studies focus on using LLMs to facilitate the creation of smart contracts textual descriptions [157, 206]. Napoli et al. [206] used the CO-STAR methodology to optimize prompt creation for high-quality outputs and assess the

correctness of generated contracts with Slither, a vulnerability detection tool. They tested their pipeline on a single property rental agreement, achieving a 98.1% compilability rate with minimal impact from the LLM's temperature. However, requirements must be structured to be processed, only a single contract has been tested, and the completeness of the generated code has not been evaluated, which is essential in complex scenarios requiring nuanced business logic. Karanjai et al. [157] assessed the capabilities of ChatGPT and Google PaLM2 for generating smart contract code in Solidity. Their study systematically assesses the quality of the generated code in terms of validity and correctness using a dataset of 1,349 contracts sourced from GitHub. The results indicate that PaLM2 produced incomplete code in 415 cases and non-compilable code in 519 instances. GPT-3.5 generated incomplete code in 788 cases and non-compilable code in 413 instances.

7.3.2 *Research Gaps*

Despite advancements in smart contract development and no-code platforms, critical gaps that require further research remain. First, no-code platforms, while making blockchain technology more accessible, often sacrifice flexibility and precision, limiting their use in complex scenarios. Second, current smart contract generation methods, especially those using LLMs, show promise but face completeness, correctness, and determinism issues. Indeed, the non-determinism of LLMs raises reliability concerns for enterprise applications, where errors can have serious repercussions. Moreover, existing research primarily focuses on generating simple smart contracts from textual descriptions with structured requirements, often overlooking the different users' needs and complexities and nuances required in more intricate business logic scenarios.

The AKB platform simplifies blockchain development through its model-driven approach and visual environments. However, it still faces obstacles in enabling non-technical users to generate smart contracts from natural language requirements. This gap underscores the necessity for innovative methodologies bridging the divide between high-level user requirements and technically precise smart contract code. Our work aims to address these gaps

by providing tailored solutions for diverse user needs, improving the AKB platform's accessibility and usability.

7.4 THE PROPOSED METHODS

This section describes the methods designed to generate Solidity smart contracts, starting from natural language requirements and specifically conceived for integration with the AKB platform.

7.4.1 *Prompt Engineering Best Practices*

In designing our prompts, we adhered to three principal guidelines for prompt engineering, as recommended by Ekin et al. [99]. First, we ensured clarity of intent by incorporating examples, instructions, or both to demonstrate the desired outcomes. Second, we provided explicit constraints to help the LLM generate responses that adhere to the desired limitations. For instance, instructing the LLM to use specific libraries helps ensure that the generated code is relevant and usable in the intended context. Lastly, we provided high-quality data, ensuring the necessary samples to help the model discern and follow a consistent pattern. These guidelines have been proven to help the LLM produce more accurate and relevant results.

We utilized a few patterns proposed by White et al. [314]. Specifically, we employed the Persona pattern, which involves assigning a specific role or identity to the language model, such as an "expert in smart contracts and Solidity." The model is better equipped to produce relevant and authoritative responses by maintaining a consistent persona. We also used the Template pattern, which allows the user to specify a template for the output (i.e., JSON), which the LLM fills in with content.

Finally, we exploited Zero-shot Chain-of-Thought [165] to improve the LLM's ability to perform complex reasoning through intermediate reasoning steps. For instance, correcting code generated by the model might involve first identifying potential issues in the Solidity syntax, then outlining the logical flow of the contract to ensure it meets the specified requirements, and finally revising the code accordingly.

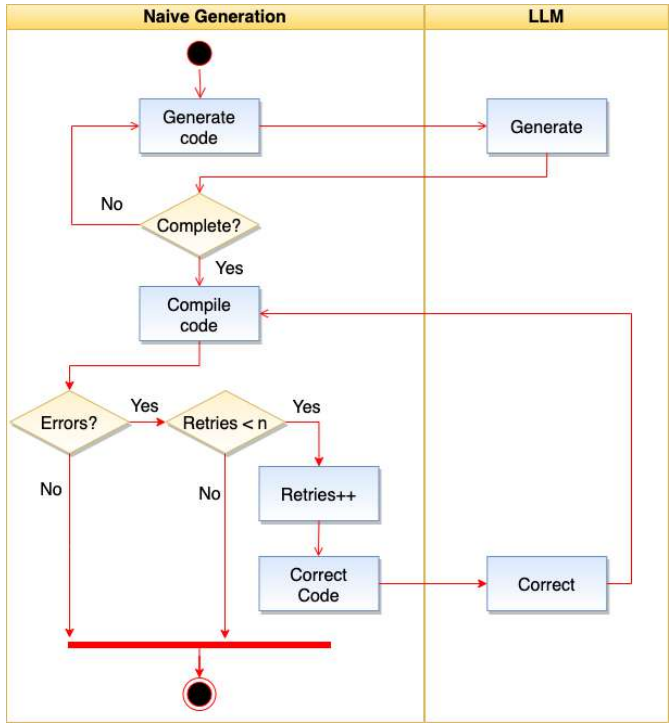


Figure 7.1: Naive Generation Method.

7.4.2 *Method 1: Naive Generation*

The *Naive Generation* method aims to facilitate rapid prototyping of smart contracts, particularly for users who may not have extensive coding skills, bridging the gap between technical complexity and user accessibility. However, this approach inevitably sacrifices the completeness of the generated code, as the focus is on quickly generating functional prototypes rather than producing fully comprehensive code from the outset. This method follows a multi-step process to transform natural language requirements into executable Solidity code, as shown in Figure 7.1.

The system requirements and the existing contracts within the user's project are initially fed into the LLM to generate the Solidity code, as shown in Figure 7.2. The inclusion of the project's libraries instructs the model to possibly reuse the project's contracts during code generation. The method incorporates an itera-

Naive Prompt Template

Act as an expert in smart contracts and Solidity. Your task is to generate the Solidity code to meet the requirements I'll provide after "REQUIREMENTS".
INSTRUCTIONS {instructions}
OUTPUT FORMAT
Answer with the solidity code, after "SOLIDITY CODE".
LOCAL LIBRARIES {localLibraries}
REQUIREMENTS {requirements}
SOLIDITY CODE:

Figure 7.2: Naive Prompt Template.

tive "continue" approach to address the token limit constraints of the LLM, which may result in incomplete code snippets.

Correct Code Prompt Template

Act as an expert in smart contracts and Solidity. Your task is to make compilable the following code.
INSTRUCTIONS {instructions}
CODE {code}
The compiler reported the following errors.
ERRORS {errors}
The local libraries imported are reported in the following.
LOCAL LIBRARIES {localLibraries}
Let's think step by step

Figure 7.3: Correct Code Prompt Template.

This approach involves feeding the incomplete code back into the LLM for further completion, allowing the model to generate additional code segments until the entire smart contract code is produced. This iterative refinement process ensures the generated code is comprehensive and accurate despite the initial truncation. After the code generation phase, the method compiles the Solidity code using the Solidity Compiler (solc). During the compilation process, any compilation errors detected by solc are iden-

tified. The method iterates by leveraging the LLM to provide corrective suggestions to resolve these errors, as shown in Figure 7.3.

This final step aims to enhance the quality and functionality of the code by leveraging the LLM for error correction, ensuring it meets the required standards and specifications.

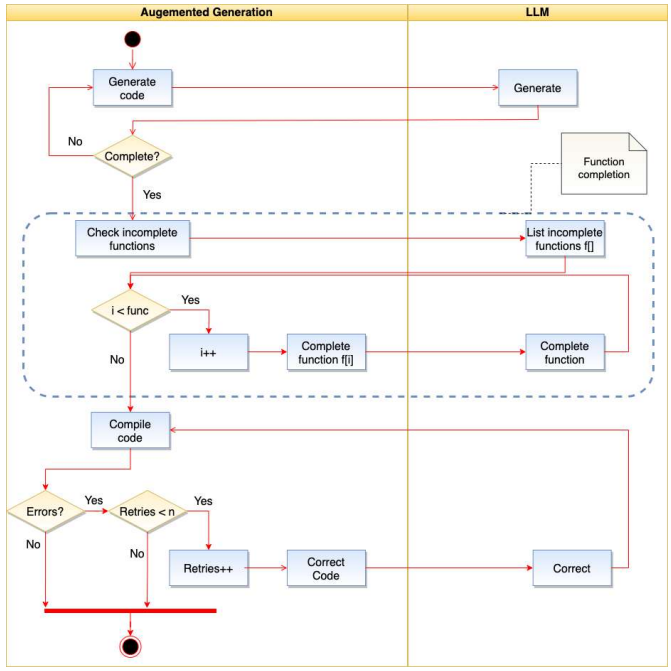


Figure 7.4: Augmented Generation Method.

7.4.3 *Method 2: Augmented Generation*

Ensuring the completeness and correctness of smart contracts is critical, especially for enterprise applications. The *Augmented Generation* method aims to maintain prototyping speed while ensuring the final output is robust and reliable. It addresses these concerns by building upon the *Naive Generation* approach and introducing an intermediate step designed to enhance the quality and completeness of the generated Solidity code, as illustrated in Figure 7.4.

Indeed, after the initial code generation using the *Naive method*, the *Augmented generation* approach introduces an iterative refine-

Incomplete Code Detection Prompt Template

Act as an expert in smart contracts and Solidity. Your task is to check if the Solidity code I'll provide after "CODE:" is incomplete or contains placeholder logic, incomplete branches, etc.
CODE {code}
LIBRARIES {libraries}
OUTPUT FORMAT:
Identify functions with incomplete code or placeholder comments, especially those using structures like for, if, or else with placeholder logic. Provide reasons for incompleteness.
INCOMPLETE FUNCTION NAMES:

Figure 7.5: Incomplete Code Detection Prompt Template.

ment step to identify and complete any placeholder logic or incomplete code segments. It leverages a specialized prompt to exploit the LLM to analyze for incompleteness or placeholder logic, as shown in Figure 7.5.

This prompt instructs the LLM to identify any incomplete functions or placeholder logic within the provided code. The output lists the names of incomplete functions along with motivations for why they are considered incomplete.

Complete Function Code Prompt Template

Act as an expert in smart contracts and Solidity. Your task is to complete the code I'll provide after "ORIGINAL CODE:".
INSTRUCTIONS: {instructions}
INCOMPLETE FUNCTION {incomplete_function}
LOCAL LIBRARIES {libraries}
ORIGINAL CODE {code}

Figure 7.6: Complete Function Code Prompt Template.

Next, the method parallel completes the identified incomplete functions using another specialized prompt, as shown in Figure 7.6. This prompt guides the LLM in generating the missing logic for the incomplete functions, ensuring the code is comprehensive

and executable. The iterative process involves feeding the incomplete code into the LLM, generating the required logic to complete the functions. This process continues until all incomplete segments are resolved, resulting in a fully functional and complete Solidity smart contract. As shown in Figure 7.3, once the code is complete, the approach proceeds to the compilation phase using solc. Like the *Naive generation* method, any compilation errors detected by solc are addressed using the LLM for corrective suggestions.

7.4.4 *Method 3: Enhanced Generation*

The previous methods, including the *Augmented Generation* approach, may still produce incomplete code due to output constraints inherent in LLMs. Most LLMs, including popular models like GPT-4, are limited in their output capacity, typically capping at around 4,000 tokens. This limitation poses a significant challenge when generating complex constructs such as smart contracts, which often require detailed specifications and extensive code segments to be fully realized. The issue is particularly problematic in complex business scenarios, where the completeness and correctness of smart contracts are paramount. Even minor omissions or errors in the generated code can lead to severe operational and financial repercussions.

Therefore, the *Enhanced Generation* method introduces intermediate steps that transform the system requirements into different representations of the contract definition, focusing specifically on the functions. By breaking down the contract generation process into more granular steps, we can generate more detailed and comprehensive functions, ultimately leading to a more robust and complete Solidity smart contract.

As depicted in Figure 7.7, the method consists of several steps:

1. **Step 1: JSON Contract Representation.** The process begins by generating a JSON representation of the contract. By using a specific prompt, as shown in Figure 7.8, the LLM produces a JSON that includes the state variables, events, modifiers, functions, and contracts to import. This representation serves as a blueprint that identifies the general elements of

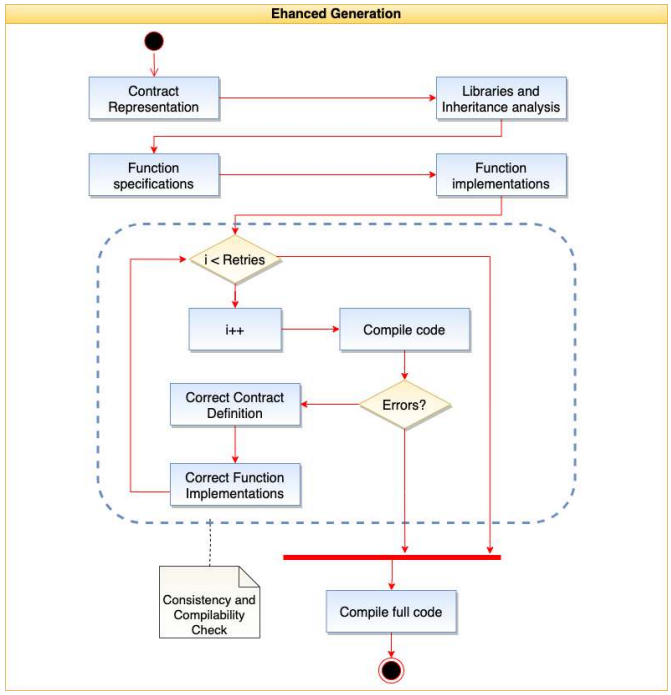


Figure 7.7: Enhanced Generation Method.

the contract that need to be generated and the functions required to meet the system requirements. This step ensures a clear and organized structure for the subsequent steps.

2. **Step 2: Library and Inheritance Analysis.** Next, the method checks the contracts to import, focusing on both the project's libraries and standard libraries such as OpenZeppelin. During this step, the LLM is instructed, as shown in Figure 7.9, to analyze inheritance and reusability issues that may arise from the imported libraries. The contract definition obtained in Step 1 is updated to accommodate inheritance and reusability to ensure that the final contract leverages existing libraries effectively.

3. **Step 3: Detailed Function Specifications.** As shown in Figure 7.10, the method parallel processes the list of identified functions in the contract definition exploiting the LLM to generate detailed specifications.

Contract Elements Generation Prompt Template

You are an expert of smart contracts. Your task is to provide the specifications needed by a Solidity Developer to develop the smart contract that meets the requirements provided by the user.
INSTRUCTIONS: {instructions}
OUTPUT FORMAT: {JSON Format }
REQUIREMENTS: {requirements}

Figure 7.8: Contract Elements Generation Prompt Template.

Library and Inheritance Analysis Prompt Template

Act as an expert in smart contracts and Solidity. Your task is to update the contract specifications that I'll provide to make it consistent in terms of inheritance. I'll provide the user requirements, the contract specifications, the user library, and its code.
INSTRUCTIONS: {instructions}
OUTPUT FORMAT: {JSON Format}
REQUIREMENTS: {requirements}
CONTRACT SPECIFICATIONS: {contract_json}
LIBRARY: {library}

UPDATED JSON OF THE CONTRACT:

Figure 7.9: Library and Inheritance Analysis Prompt Template.

Each function specification includes a signature, including the access control requirements and ownership control, a description of the function parameters, the logic the function must follow and the events to emit, the error handling logic, the dependencies from the imported libraries, and other elements of the contract.

The contract definition obtained in Step 1 is then updated to incorporate these detailed function specifications, ensuring a comprehensive and well-defined contract structure.

4. **Step 4: Function Implementation.** The method processes the list of function specifications in parallel to generate their

Detailed Function Specifications Prompt Template

You are an expert of smart contracts. Your task is to provide the detailed specifications for a function of a smart contract. The specifications must help a Solidity developer to develop the function. ### INSTRUCTIONS: {instructions}
OUTPUT FORMAT: {output format}
REQUIREMENTS: {requirements}
CONTRACT SPECIFICATIONS: {contract_json}
FUNCTIONS TO IMPORT: {functions_to_import}
Function: {function_json}

SPECS:

Figure 7.10: Detailed Function Specifications Prompt Template.

implementations. At the end of this step, the contract definition is updated with the newly implemented functions, resulting in a more complete and functional contract.

5. **Step 5: Consistency and Compilability Check.** The method generates a contract skeleton (the Solidity code for the contract, including imports, state variables, events, etc., but without the function definitions) to ensure the consistency and compilability of the contract definition. The function implementations obtained in Step 4 are then added to create the overall Solidity code for the contract. The method iterates to check if the code compiles. If the code does not compile, the method uses the compilation errors to 1) parallel fix the function implementations and 2) Update the contract definition, e.g., its imports, state variables, events, and mappings, to make it consistent with the new implementations.

6. **Step 6: Final Compilation and Correction.** If the code obtained in Step 5 still does not compile, the method proceeds with the final compilation phase, similar to the first two methods. During this phase, any remaining compilation errors detected by `solc` are addressed using the LLM for corrective suggestions. This iterative refinement ensures that

the final code is complete, functional, and meets the required standards and specifications.

7.4.5 *LLM Choice*

The proposed approaches are designed to be easily employed with different LLMs. For empirical evaluation purposes, we chose the GPT-4o model, which is accessible via the OpenAI API. We opted for GPT-4o because we expected enhanced model performance with this newer version, as noted by OpenAI[1]. This decision allows us to take advantage of the latest advancements in language modeling technology, ensuring our methods benefit from greater accuracy and robustness. For model configuration, we set the temperature parameter to a low value to minimize the likelihood of the language model generating incorrect or unpredictable outputs. A lower temperature setting makes the language model's text generation less random and more deterministic, thereby improving the reliability and consistency of the results. Additionally, Ouyang et al. [223] found that a smaller temperature setting leads to more stable code generation in LLMs. Based on these findings, we set the temperature parameter to 0, which is also recommended according to the OpenAI documentation[2].

7.4.6 *Integration with AstraKode Blockchain*

Integrating the Solidity smart contract LLM generator into the AKB platform enhances the development workflow at three key stages. First, as shown in Figure 7.11, a chatbot feature helps users generate customized Solidity contracts based on natural language requirements during project creation, simplifying the setup process. At this stage, users can select the generation method (i. e.the proposed methods) based on their subscription and needs. Second, AKB's visual development environment allows users to refine the generated contracts and ensure they align with their intended architecture. Custom RESTful services allow users to invoke the proposed generation methods from an Angular-based

1 OpenAI GPT-4o. https://openai.com/index/hello-gpt-4o/
2 OpenAI documentation. https://platform.openai.com/docs/api-reference



Figure 7.11: Chatbot Integration in AKB.

UI interface. Finally, the platform parser converts generated Solidity code into visual models, enabling users to visualize and refine contracts within the platform, bridging the gap between textual and visual development.

7.5 DESIGN OF THE EMPIRICAL EVALUATION

This section describes the empirical experiments we conducted to answer our research questions with he *goal* of investigating the effectiveness and efficiency of the proposed methods in generating Solidity smart contracts from natural language requirements. We posed the following research questions:

RQ7.1 To what extent do the proposed LLM-based methods effectively generate Solidity smart contracts from requirements expressed in natural language?

This question aims to evaluate the effectiveness of each method in generating high-quality Solidity code.

RQ7.2 What performance trade-offs are associated with the LLM-based proposed methods for generating Solidity smart contracts?

This question seeks to understand the computational efficiency and resource requirements of the methods that must balance the quality of the generated code with the performance trade-offs to ensure their practicality in industry settings.

7.5.1 *Dataset*

The dataset used for our experiments consists of 23 groups of smart contract requirements across various domains, such as sports, medical, tourism, gaming, DeFi, and lottery. We have a corresponding ground truth contract for each set of requirements, i. e.a manually developed Solidity contract that fulfills the specified requirements.

7.5.2 *$RQ_{7.1}$: Effectiveness of the Methods*

We employ both quantitative and qualitative assessments to comprehensively evaluate the generated contracts. First, we use the dataset described in Section 7.5.1 and the REST services designed for each method to generate the Solidity code given the requirements. We perform five iterations for each method, given the nondeterministic nature of LLMs, and collect the following quantitative metrics:

- **Compilability** (**Co**) indicates whether the generated code compiles successfully without errors.
- **Tree Edit Distance** (**TD**) measures the structural similarity between the generated code and the ground truth.
- **Cyclomatic Complexity** (**CC**) assesses the complexity of the control flow within the generated code through the number of its linearly independent paths. Smart contracts with lower cyclomatic complexity are easier to understand and less risky to modify.
- **Coupling** (**Cp**) measures how many classes a class uses. A lower coupling is usually better.

- **Cohesion** (**Ch**) measures the cohesion of methods within a class, with lower values indicating higher cohesion and better modularity. In detail, we compute LCOM4, i. e.Lack of Cohesion of Methods 4.
- **Maintainability Index** (**MI**) is a composite metric that evaluates the ease of maintaining software. Higher values indicate better maintainability.
- **Comment Density** (**CD**) indicates the ratio between the total comment lines and the total lines of code.
- **LOC** (**LO**) counts the lines of code.
- **nSLOC** (**nS**) counts the lines of code without comments.

We calculate a normalized score for each code quality metric except Compilability and Tree Edit Distance. For metrics where higher is better, the score is $\frac{\text{Generated Value}}{\text{Ground Truth Value+1}}$; otherwise, the score is $\frac{\text{Ground Truth Value+1}}{\text{Generated Value+1}}$. We compute descriptive statistics, including mean, median, and standard deviation, for each metric to provide an overview of the central tendency and variability of each method's performance. Afterward, we conduct statistical tests through the ANOVA test to compare the mean achieved by all methods for each metric, helping to determine if there are statistically significant differences. ANOVA assumes independence, meaning averages for each contract must be independent. It also assumes normality of the distribution, with the Kruskal-Wallis test as an alternative if normality is a concern. Lastly, homogeneity of variance is required, as verified by Levene's test, to ensure equal variances across groups. If the ANOVA reveals significant differences, a post-hoc analysis using Tukey's Honestly Significant Difference test is performed to identify which methods differ significantly. We also involve five experienced practitioners in smart contract development with at least ten years of experience to provide insightful feedback on the quality and practical applicability of the generated contracts. They review the generated contracts and provide qualitative feedback on completeness and adherence to requirements, using a Likert scale ranging from 1 (poor) to 5 (excellent). To analyze the qualitative metrics, we focus on the practi-

Table 7.1: Descriptive Statistics for Code Quality Metrics and Methods

Metric	Naive (Avg./Med./St.D.)	Augmented (Avg./Med./St.D.)	Enhanced (Avg./Med./St.D.)
Co	0.99 / 1.00 / 0.00	0.98 / 1.00 / 0.00	**1.00 / 1.00 / 0.00**
LO	0.83 / 0.83 / 0.21	0.86 / 0.85 / 0.19	**1.22 / 1.22 / 0.32**
nS	0.84 / 0.84 / 0.21	0.88 / 0.88 / 0.19	**1.03 / 1.03 / 0.26**
CC	1.16 / 1.15 / 0.34	**1.15 / 1.13 / 0.33**	1.15 / 1.16 / 0.36
Cp	**1.01 / 1.00 / 0.34**	1.01 / 1.02 / 0.29	1.09 / 1.10 / 0.30
Ch	**1.12 / 1.11 / 0.49**	1.18 / 1.18 / 0.57	0.91 / 0.91 / 0.52
MI	**1.10 / 1.11 / 0.38**	1.08 / 1.08 / 0.34	1.03 / 1.03 / 0.36
CD	0.21 / 0.22 / 0.31	0.10 / 0.06 / 0.13	**1.94 / 1.85 / 2.27**
TD	**5.59 / 5.57 / 4.19**	5.34 / 5.26 / 3.18	7.50 / 7.65 / 3.70

tioners' feedback and compute the average rating for each method to provide a quantitative summary of their perceptions.

7.5.3 $RQ_{7.2}$: *Efficiency and Performance Trade-offs of the Methods*

We collect several metrics while executing the proposed methods and generating, as follows:

- **Total Tokens Consumed** (**TTC**) is the number of input and output tokens used by the LLM for code generation, reflecting resource consumption.
- **Total Compilation Retries** (**TTR**) is the number of recompilations required to achieve a compilable code.
- **Total Generation Time** (**TGT**) is the time taken in seconds to generate the code, indicating efficiency.

As described in the context of RQ7.1, we compute descriptive statistics and conduct statistical tests for all metrics to determine if there are significant differences in performance.

7.6 RESULTS AND DISCUSSION

We present the results achieved for each research question (see our online repository `https://github.com/gadevito/AKB_RPK` for details).

7.6.1 $RQ_{7.1}$: *Effectiveness of the Methods*

Table 7.1 reports the descriptive statistics for the code quality metrics (as reported in Section 7.5.2) and methods compared to the ground truth. The Naive Generation method achieves nearly 100% successful compilation. Contracts are concise, with LOC (mean 0.83, median 0.83, Std. Dev. 0.21) and nSLOC (mean 0.84, median 0.84, Std. Dev. 0.21) indicating consistent code length. Cyclomatic Complexity statistics (mean 1.16, median 1.15, Std. Dev. 0.34) reflect a relatively complex control flow. Coupling (mean 1.01, median 1.00, Std. Dev. 0.34) and Cohesion (mean 1.12, median 1.11, Std. Dev. 0.49) metrics show moderate interdependence and room for improved organization. The Maintainability Index values (mean 1.10, median 1.11, Std. Dev. 0.38) suggest the codebase is generally maintainable. Comment Density tendency (mean 0.21, median 0.22, Std. Dev. 0.31) shows the code contains relatively few comments. Tree Edit Distance results (mean 5.59, median 5.57, Std. Dev. 4.19) indicate moderate structural differences from the ground truth.

The Augmented Generation method improves with nearly 98% successful compilation. The LOC (means 0.86, median 0.85, Std. Dev. 0.19) and nSLOC (mean 0.88, median 0.88, Std. Dev. 0.19) values imply moderate detail. The mean Cyclomatic Complexity is 1.15 (median 1.13, Std. Dev. 0.33), suggesting stable control flow. Coupling (mean 1.01, median 1.02, Std. Dev. 0.29) and Cohesion (mean 1.18, median 1.18, Std. Dev. 0.57) tendencies indicate room for improved organization. The mean Maintainability Index is 1.08 (median 1.08, Std. Dev. 0.34), showing general maintainability. Low Comment Density values (mean 0.10, median 0.06, Std. Dev. 0.13) reflect concise documentation. The mean Tree Edit Distance is 5.34 (median 5.26, Std. Dev. 3.18), suggesting moderate structural similarity to the ground truth.

The Enhanced Generation method aligns well with the ground truth. It achieves a mean LOC of 1.22 (median 1.22, Std. Dev. 0.32) and nSLOC of 1.03 (median 1.03, Std. Dev. 0.26), capturing necessary complexity and detail. The mean Cyclomatic Complexity is 1.15 (median 1.16, Std. Dev. 0.36), indicating stable control flow. Coupling (mean 1.09, median 1.10, Std. Dev. 0.30) and Cohesion (mean 0.91, median 0.91, Std. Dev. 0.52) values show balanced or-

Table 7.2: Statistical Test Summary for Code Quality Metrics

Metric	P-Value	Effect Size	Post-hoc
LO	<0.001	Large	Enhanced vs. Naive/Augmented
nS	0.005	Medium	Enhanced vs. Naive/Augmented
CC	0.969	Very Small	None
Cp	0.712	Very Small	None
Ch	0.040	Medium	Naive vs. Enhanced
			Augmented vs. Enhanced
MI	0.012	Medium	Naive vs. Enhanced
			Augmented vs. Enhanced
CD	<0.001	Large	Enhanced vs. Naive/Augmented
TD	0.052	Small	None
Co	0.602	Very Small	None

ganization. The mean Maintainability Index is 1.03 (median 1.03, Std. Dev. 0.36), implying general maintainability. The Comment Density tendency is high (mean 1.94, median 1.85, Std. Dev. 2.27), enhancing readability. The mean Tree Edit Distance is 7.50 (median 7.65, Std. Dev. 3.70), demonstrating reasonable structural similarity with some variability.

The statistical analysis, whose results are reported in Table 7.2, confirms some initial findings and reveals new insights into the methods' performance. The Kruskal-Wallis test was used for all metrics due to the weak power of the Levene's test (0.43), which indicated a lack of homogeneity of variances, making ANOVA unsuitable. Significant differences in LOC, nSLOC, and Comment Density indicate that the Enhanced Generation method produces more detailed and well-documented code than the Naive and Augmented methods. In contrast, Cyclomatic Complexity and Coupling show no significant differences, suggesting similar control flow and code dependency across methods. Cohesion shows significant differences, with the Enhanced method differing from both Naive and Augmented methods, indicating variability in internal organization. The Maintainability Index reveals significant differences between the Enhanced method and both Naive and Augmented methods, though it may prioritize other aspects over maintainability. Tree Edit Distance and Compilability metrics confirm high success rates with no significant differences, maintaining consistency across all methods. Overall, all methods reliably produce compilable code. The Enhanced Generation method pro-

duces code that closely aligns with the ground truth, especially in LOC and nSLOC. It also improves Comment Density, enhancing readability. Despite significant differences in the Maintainability Index, the lower mean and median values suggest that the Enhanced method may prioritize other aspects over maintainability. No significant differences in Cyclomatic Complexity, Coupling, and Cohesion indicate similar control flow and organization across methods. Finally, Tree Edit Distance shows no significant differences, indicating structural similarity is less impactful.

Regarding qualitative metrics, practitioners found the Naive Generation valuable for rapid prototyping (with a mean of 3.9 for completeness and adherence) but limited in meeting requirements. The Augmented Generation scored 4.1, showing better accuracy for quality-sensitive scenarios. The Enhanced Generation scored highest at 4.3, proving effective for comprehensive, requirement-aligned contracts, ideal for complex scenarios. These qualitative results align with quantitative findings, confirming the advantages of the Enhanced Generation.

Answer to $RQ_{7.1}$

The Naive Generation method is effective for rapid prototyping but may lack completeness. The Augmented Generation method improves structural similarity and control flow, aligning better with the ground truth. The Enhanced Generation method is the most effective, closely matching the ground truth in code length, complexity, and maintainability, making it ideal for complex scenarios.

7.6.2 $RQ_{7.2}$: *Efficiency and Performance Trade-offs of the Methods*

Table 7.3 reports the descriptive statistics for the efficiency metrics reported in section 7.5.3 for each method.

The Naive Generation method consumes a mean of 4,167.10 tokens (median 3,161.20, Std. Dev. 2,959.39), indicating consistent performance with some variability. It requires minimal recompilation, with a median of 0 retries, reflecting the method's efficiency in generating compilable code on the first attempt. The average generation time is 17.44 seconds (median 15.72 seconds, Std. Dev. 9.59), revealing moderate variability in execution time, which is suitable for rapid prototyping.

Table 7.3: Descriptive Statistics for Efficiency Metrics and Methods

Method	TTC			TTR			TGT		
	Avg	Med	Std	Avg	Med	Std	Avg	Med	Std
Naive Generation	4,167.10	3,161.20	2,959.39	0.10	0.00	0.35	17.44	15.72	9.59
Augmented Generation	14,053.98	12,463.80	6,616.95	0.37	0.00	0.84	73.49	64.31	34.44
Enhanced Generation	79,313.64	38,395.00	111,709.06	0.88	0.20	1.23	45.38	33.57	31.86

Table 7.4: Efficiency Metrics for Contracts Using User-Defined Libraries

	Naive Generation			Augmented Generation			Enhanced Generation		
Contract	TTC	TTR	TGT	TTC	TTR	TGT	TTC	TTR	TGT
Game Contract	11,790.20	0.00	21.94	24,924.00	0.00	87.37	495,131.20	3.00	135.45
Delegation Manager	11,271.80	0.00	20.71	27,314.80	2.60	143.53	270,938.2	3.80	85.50
Scholarship Manager	3,511.80	0.00	15.82	12,786.80	0.20	64.31	92,722.40	1.20	55.90

The Augmented Generation method consumes a mean of 14,053.98 tokens (median 12,463.80, Std. Dev. 6,616.95), showing more variability than the Naive method. It requires minimal recompilation, with a median of 0 retries. The average generation time is 73.49 seconds (median 64.31 seconds, Std. Dev. 34.44), reflecting higher computational demands due to the method's focus on enhancing code completeness.

The Enhanced Generation method consumes a mean of 79,313.64 tokens (median 38,395, Std. Dev. 111,709.06), exhibiting high resource usage and variability due to contract complexity. It requires a median of 0.20 recompilation retries, showing relative efficiency. The average generation time is 45.38 seconds (median 33.57 seconds, Std. Dev. 31.86), balancing resource use and time efficiency while generating detailed contracts. Notably, using user-defined libraries, employed in only three contracts across all methods, has a limited impact on performance (see Table 7.4).

Regarding the statistica tests, we performed the Kruskal-Wallis in all cases, due to the normality assumption violation for the ANOVA test. The tests revealed significant differences among methods for the efficiency metrics (see Table 7.5). The Enhanced method consumes the most tokens and requires more retries, indicating its focus on comprehensive contract generation. The Naive method is the fastest and requires the least retries, while the Augmented method takes the longest time but balances retries with improved

Table 7.5: Statistical Test Summary for Efficiency Metrics

Metric	P-Value	Eff. Size	Post-hoc
TTC	<0.001	Large	Naive vs. Enhanced
			Augmented vs. Enhanced
TTR	0.002	Small	Naive vs. Enhanced
			Augmented vs. Enhanced
TGT	<0.001	Large	Naive vs. Augmented/Enhanced
			Augmented vs. Enhanced

code quality. Post-hoc analyses confirm significant differences between the Enhanced method and the others across these metrics.

Answer to $RQ_{7.2}$

The Naive Generation method is ideal for rapid prototyping, but it may lack completeness. The Augmented Generation method balances quality and performance, aligning better with the ground truth. The Enhanced Generation method is resource-intensive but delivers the most detailed contracts suitable for complex scenarios.

7.7 THREATS TO VALIDITY

We acknowledge a few threats to the validity of our study, potentially impacting our findings.

The choice of GPT-4o could have impacted internal validity as the LLM may introduce biases inherent to this specific model. To mitigate this threat, we ensured that the prompts used for code generation were designed following best practices in prompt engineering, aiming for clarity and consistency across different tasks. Additionally, we aimed to minimize output variability by setting the temperature parameter to zero, promoting more reliable and consistent results.

Construct validity could be threatened by the iterative nature of the code generation process, particularly in methods involving multiple iterations and refinements. Variability in LLM outputs across iterations could introduce inconsistencies, affecting the reproducibility of results [254]. Therefore, we conducted multiple iterations for each method and averaged the results. Additionally, by using a low-temperature setting, we aimed to enhance the de-

terminism of the LLM outputs, supporting more consistent and reliable outcomes.

External validity is threatened by the specific context of the AKB platform, which may not fully represent other no-code platforms or environments for smart contract development. To mitigate this threat, we designed the proposed methods to be adaptable and potentially applicable to other platforms by focusing on general principles of LLM-based code generation. Future work could explore the application of these methods in different environments to further validate their generalizability.

Another external validity threat arises from the dataset used for empirical evaluation. The dataset, comprising 23 groups of smart contract requirements across various domains, may not cover the full spectrum of complexity and diversity encountered in real-world applications. To address this, we selected various domains to ensure a broad representation of typical use cases. However, future studies could benefit from expanding the dataset to include more complex and varied scenarios to validate the methods further.

The metrics used to assess the quality and performance of the generated smart contracts could threaten conclusion validity, as the employed quantitative metrics, including compilability, tree edit distance, and cyclomatic complexity, may not capture all code quality and maintainability aspects. Therefore, we complemented the quantitative analysis with qualitative feedback from experienced practitioners, providing a more holistic assessment of the generated contracts. This dual approach helps to balance the limitations of each method, offering a more nuanced understanding of the results.

7.8 CONCLUSIONS AND FUTURE WORK

We explored the application of LLMs to generate Solidity smart contracts from natural language requirements within the AKB platform. We focused on three distinct methods—*Naive Generation*, *Augmented Generation*, and *Enhanced Generation*—each designed to address different aspects of smart contract development, from rapid prototyping to handling complex business scenarios. The empirical evaluation demonstrated that while the *Naive Genera-*

tion method is practical for quick prototyping, it may lack completeness in more complex scenarios. The *Augmented Generation* method offers a balanced approach, improving structural similarity and control flow, making it suitable for quality-sensitive applications. The *Enhanced Generation* method emerged as the most effective but less efficient, closely aligning with ground truth regarding code length, complexity, and maintainability, and is particularly well-suited for comprehensive contract generation in intricate business contexts. Our research has significant implications for the democratization of blockchain technology. By integrating these methods into the AKB platform, we provide non-technical users with powerful tools to generate robust smart contracts, thereby lowering barriers to entry and fostering broader adoption of blockchain solutions. This work highlights LLMs' potential to bridge the gap between natural language requirements and executable smart contract code, offering a pathway for more accessible and efficient blockchain development. Based on our findings, future work could focus on several key areas. First, expanding the dataset to encompass more complex smart contract scenarios would further validate the methods and enhance their robustness. Additionally, investigating other advanced LLMs might enhance performance and accuracy. Furthermore, combining LLM-generated code with traditional verification could boost security and correctness.

8

ADVANCING LLM-BASED ISSUE REPORT CLASSIFICATION WITH EXPLAINED FEW-SHOT LEARNING, INTENT EXTRACTION, ENSEMBLE, AND SUMMARIZATION

8.1 INTRODUCTION

In software development and maintenance, issue reports serve as essential communication mechanisms between stakeholders and development teams. Managed through specialized tracking tools [28], these reports formally document change requests, bugs, and other concerns encountered during software development and use. Throughout the project lifecycle, developers can systematically address user needs and software defects by capturing from issue reports crucial elements such as the problem's nature, reproduction steps, expected vs. actual behavior, and supporting evidence like screenshots or error logs [154]. Therefore, software maintenance and bug tracking depend on these reports being correctly classified, as they help teams in priority management [155], and thus enables proper resource allocation and timely issue resolution. Although stakeholders often provide initial classifications when submitting reports, these are often inaccurate [61, 131], and relying solely on manual reclassification creates processing bottlenecks. Given the high volume and complexity of issue reports, automating the classification process has become increasingly important. Various approaches have been proposed in the literature, including machine-learning techniques [18], text mining combined with data mining [345], and deep-learning models such as fastText [154]. Although these techniques have shown promise, they often face limitations in generalizability, reliance on manual data classification, and handling diverse linguistic patterns.

The advent of LLMs has shown potential in addressing the issue report classification problem [10, 62, 144]. Colavito et al. [61–63] demonstrated that GPT-like LLMs could achieve performance comparable to state-of-the-art BERT-like models without the need

of fine-tuning. However, their work highlighted two critical limitations: handling long issue bodies within LLMs' restricted context windows and managing the non-deterministic behavior leading to inconsistent classifications. To address these challenges, in the context of this chapter, we introduce an LLM-based solution that effectively handles long reports. The method adapts the MapReduce paradigm to process lengthy issue descriptions by dividing them into manageable chunks, processing each independently, and consolidating the results.

Building on this foundation, we examined two complementary perspectives to issue classification. The first one focuses on understanding the user's intent when creating the issue, using this intent as the basis for classification (i.e., "*what is the user trying to achieve?*"). The second perspective treats classification as a decision-making process based on explicit examples and criteria, similar to how a human would follow an informed checklist (i.e., like a triager consulting precedents with explicit reasons). In other words, human experts do not classify issues based on memorized patterns alone, but by examining examples and explicitly stating the motivations why each case belongs to a specific category [281]. These fundamental perspectives inform our three classification methods: "Intent Extraction and Classification" and "Ensemble Classification," which implement the intent-based approach (with the latter also addressing non-determinism). In contrast, "Explained Few-Shot Learning" implements the example-based approach. This dual strategy allows us to better understand LLMs' strengths and limitations in classification tasks.

To assess the methods, we have performed an empirical evaluation addressing the following research questions:

> ***RQ8.1:*** *How do the proposed LLM-based methods compare to existing approaches in terms of effectiveness and operational characteristics?*

This question aims to evaluate the effectiveness of the proposed methods against established baselines across multiple LLMs (GPT-4o, GPT-3.5, and Qwen 2.5) to assess whether they provide a tangible improvement in classification performance. Moreover, this question aims to evaluate operational characteristics of the LLM-based methods, including computational costs, handling of long

texts, perplexity (i.e., model uncertainty in predictions), and parameter sensitivity across different LLMs.

> ***RQ8.2:*** *To what extent are the classification results of the proposed methods consistent?*

This question investigates the non-deterministic behavior of LLMs and aims to evaluate the consistency of the classification outcomes. By examining the variability in classifications for the same issue report across multiple evaluations, we can determine the reliability of our approaches.

> ***RQ8.3:*** *What are the predominant error patterns and behavioral dynamics exhibited by the proposed LLM-based classification methods?*

This question investigates the error patterns and behavioral dynamics of the proposed methods in different LLMs, examining how errors are distributed, corrected, or consolidated to identify systematic biases and method-specific failure modes.

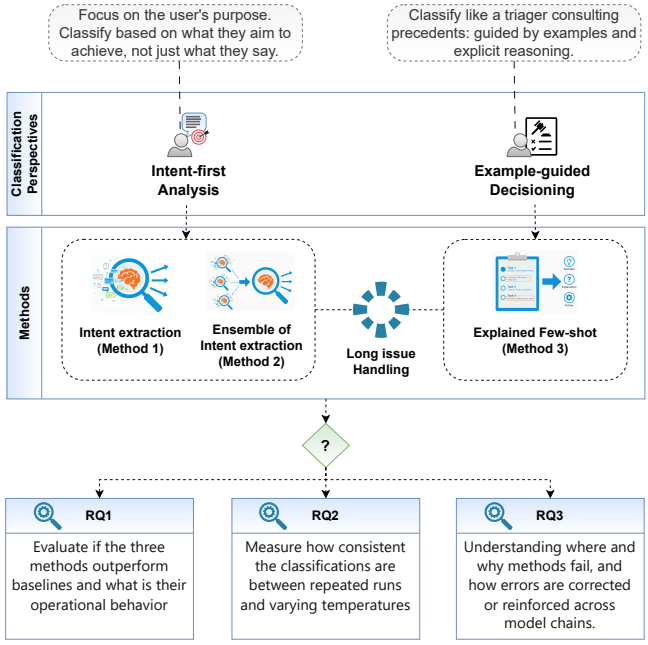


Figure 8.1: Overall Methodology used in the study.

The methodological framework used in our study is depicted in Figure 8.1.

The empirical evaluation demonstrates that all proposed methods outperform existing approaches across three LLMs, achieving 5-8% F1-score improvements. Beyond performance gains, the analysis reveals counterintuitive behaviors: ensemble methods can consolidate errors rather than correct, highlighting the limitations of majority voting when base models share systematic biases. We demonstrate that the MapReduce approach consistently outperforms both truncation and full-context ingestion for long issue reports, offering superior classification effectiveness while maintaining manageable inference costs. Additionally, we identify class-level perplexity as a reliable predictor of classification confidence, enabling more adaptive and trustworthy deployment strategies. The analysis also establishes a taxonomy of misclassification causes and reveals that different model architectures exhibit complementary failure patterns, suggesting opportunities for more sophisticated ensemble strategies.

All in all, the contribution of this chapter is fourfold:

1. We introduce three LLM-based methods that mimic expert decision processes, surpass the state of the art, and aim to address key SE challenges in report classification: consistency, variable quality, linguistic heterogeneity, and lengthy descriptions.

2. We provide the first analysis of LLM behavior in issue classification, revealing key findings about ensemble pros and cons, temperature sensitivity across models, and how perplexity metrics can predict classification confidence.

3. We establish a taxonomy of misclassification causes specific to software issues (class overlap, ambiguity, complexity, implicit intent) and identify systematic error patterns across LLM architectures, providing actionable insights for industrial deployment in privacy-constrained and privacy-unconstrained settings.

4. We provide new manually curated datasets of issue reports to the research community, complementing Colavito et al.'s

work [63] and following their verification procedure. The dataset is available in the replication package at [85].

8.2 RELATED WORK

Traditional machine learning methods were the mainstay of early automated approaches to issue report classification. Antoniol et al. [18] utilized alternating decision trees, naïve Bayes classifiers, and logistic regression, to automatically categorize issues in bug tracking systems as bugs (for corrective maintenance) or non-bugs. Their technique achieved an accuracy of 77% to 82%, indicating the potential for automated issue classification. Similarly, Zhou et al. [345] tackled the same bug/non-bug issue report classification problem and proposed a multi-stage approach that combines text mining and data mining techniques to further improve classification performance. Their approach, evaluated on ten large open-source projects, outperformed baseline methods and individual classifiers and demonstrated significant improvement in classification performance w.r.t. the approach proposed by Antoniol et al.

Kallis et al. [154] introduced Ticket Tagger, a GitHub app that automates issue labeling using a fastText machine-learning model. The model classifies issues such as bug reports, enhancements, or questions based on their titles and descriptions. Evaluation on a dataset of 30,000 GitHub issues showed a high F1-score across categories ($>$ 80%). However, challenges were noted with false positives in questions and false negatives in feature requests, suggesting the need for improved handling of diverse linguistic patterns in issue descriptions.

Recent studies have also explored the potential of LLMs for issue report classification, leveraging their ability to process natural language [10, 19, 61, 62, 144, 156]. The organizers of the Issue Report Classification Challenge within the 2nd International Workshop on Natural Language-based Software Engineering (NLBSE' 23) introduced a RoBERTa-based model, using a crowd-sourced dataset, that they distributed in the scope of the challenge [156]. The model still represents the state-of-the-art and achieved an F1-micro score of 0.8890. Nonetheless, Colavito et al. noted that crowd-sourced datasets often contain inconsistent and noisy la-

bels, which can affect the performance of models [62, 63]. To mitigate this issue, they used a manually verified subset of the NLBSE'23 Issue Report Classification Challenge dataset [156], that they distributed in their previous work [63], and employed SETFIT, a few-shot learning approach for fine-tuning transformer-based models, to train a classifier. Their model achieved an F1-micro score of 0.8321, comparable with the state-of-the-art [156].

In another study, Colavito et al. also suggested using GPT-like LLMs, without fine-tuning, to automate issue labeling in software projects [61]. Their study showed that LLMs, specifically GPT-3.5-turbo, using few-shot learning, can achieve performance similar to top BERT-like models even without fine-tuning, achieving a promising micro average F1-score of 0.8155. Further advancing the use of LLMs for Issue Report Classification, recent studies presented at the 2024 edition of the NLBSE Tool Competition [153] have investigated approaches based on fine-tuning LLMs [19, 243]. More in detail, [243] proposed fine-tuning a single instance of the Text-to-Text Transfer Transformer (T5) LLM for the classification task using issue reports from different repositories. The approach presented in [19], on the other hand, is based on fine-tuning repository-specific versions of GPT-3.5-turbo. The fine-tuning process involved simulating conversational interactions, where the title and description of the issue reports were provided as input, and the actual label was used as the expected output. Both approaches were evaluated, within the context of the NLBSE'24 tool competition, on issue reports from five well-known software repositories. They showed promising results (0.83 weighted-averaged F1 on the considered data) that may be attributed to the fine-tuned model's ability to learn repository-specific concepts, technologies, and domain-related topics characterizing the considered repositories. However, fine-tuning LLMs is a computationally expensive task, which may limit the applicability of these approaches to smaller or less-resourced software repositories.

While recent studies have shown promising results using LLMs for issue report classification, key challenges persist in handling lengthy reports and ensuring consistent classifications. Our work builds upon these advances while addressing their limitations through novel approaches that we present in the following section.

8.3 PROPOSED METHODS

In the following section, we present the three LLM-based methods proposed for issue report classification, which derive from two distinct conceptual perspectives.

Understanding the purpose of the user's issue report creation is the main objective of the first perspective. Following this viewpoint, we developed the "*Intent Extraction and Classification*" method (Method 1), which uses an LLM to ascertain the underlying intent before labeling. The "*Ensemble Classification*" method (Method 2) extends this strategy by executing several separate instances of the Intent Extraction pipeline and combining their predictions via a voting mechanism to overcome the problem of non-determinism. The second perspective considers classification as a decision-making process based on clear motivations and examples. The "*Explained Few-Shot Learning*" method (Method 3) employs this viewpoint by using few-shot examples created by co-prompt engineering techniques [84] to guide the model in distinguishing between class differences through explicit reasoning and explanations.

We adopt a MapReduce-inspired strategy to deal with issue reports that are longer than an LLM's context window. This strategy divides long reports into manageable portions, processes each one separately, and then combines them for the final classification. In the remainder, we explain the prompt engineering guidelines we applied and formalize the MapReduce-like strategy developed to handle long issue reports. Then we provide a detailed description of the proposed methods.

8.3.1 *Prompt Engineering Guidelines and Patterns*

For the design of the prompts, we followed three main prompt engineering guidelines as suggested by Ekin et al. and White et al. [99, 315]:

1. Show and tell: clear intentions using examples, instructions, or both.

2. Provide quality data: ensuring enough samples for the model to follow a pattern.

3. Check settings: setting the model parameters to make the model predictable in responding.

We also employed the Persona pattern proposed by White et al. [314]. This pattern entails assigning a particular identity or role to the LLM, such as "expert issue classifier in the field of software engineering", allowing the model to produce more authoritative and pertinent responses.

For clarity, in the following, we reported only the relevant part of the prompts. The complete prompts can be found in the replication package [85].

8.3.2 *Handling Lengthy Issue Descriptions*

The length of issue descriptions can vary greatly, which makes it difficult to work with LLMs because of the limited amount of context they can process. Although long issues in the NLBSE'23 dataset are infrequent (<5%), several studies [30, 51, 52, 265] revealed that, in software practice and industry, developers require high-quality issue reports that include detailed descriptions (crash details, reproduction steps, and stack traces, for effective classification, debugging, and resolution), which typically lead to verbose and longer descriptions. With medium-to-large-scale projects potentially producing over 30,000 issues a year, this requirement becomes even more crucial, because developer productivity and issue resolution efficiency can be disproportionately impacted in such situations by even a small percentage of lengthy, complex reports, such as the top 5% [265].

While modern LLMs support extended context windows (GPT-3.5 supports 16K tokens and GPT-4o supports 128K tokens), the choice to implement a MapReduce-based approach for long-text issues is based on recent empirical evidence about LLM performance degradation with extended contexts, as well as practical constraints. Colavito et al. [61] demonstrated in their work about issue classification using GPT-3.5, that including few-shot examples necessitated truncating issue descriptions, even with a context size of 16k tokens. More importantly, recent research has revealed that extending context windows does not necessarily improve, but can degrade, LLM performance. Wang et al. [302] demon-

strated that LLM accuracy significantly decreases as context length increases, with GPT-4 Turbo's accuracy dropping from approximately 18% at 4K tokens to only 3-5% at 16K tokens in sorting tasks. Similarly, An et al. [13] proved that models' ability to follow instructions deteriorates drastically in lengthy contexts, especially for query-based tasks, where LLMs find it difficult to understand long instructions.

Colavito et al. [61] addressed this challenge by cutting the issue description to a certain number of tokens to fit the constraints of GPT-3.5. Likewise, the RoBERTA-based approach, which represents the state of the art in issue classification, truncates the issue description to 512 tokens. However, this approach may lead to the loss of crucial information throughout the issue description, potentially resulting in omitted important details such as specific examples, context, or nuances within the text.

To overcome this limitation, we adopt an adapted version of the MapReduce paradigm [210] to process long texts. The approach is based on three main phases, as highlighted in Figure 8.2: (1) chunk division, (2) parallel processing to extract relevant information from each chunk, and (3) consolidation to provide the final response.

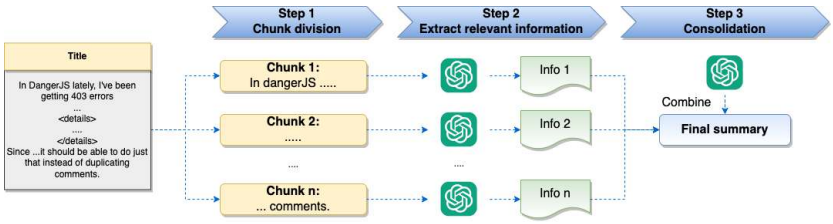


Figure 8.2: Approach for Dealing with Long Text.

In the first phase, the issue description is divided into smaller chunks, which the LLM can manage. The chunk division is performed such that each chunk has a maximum size of *window size*, which is a configurable parameter, ensuring that the sum of tokens across all chunks equals the total number of tokens in the issue description, i.e., $\sum_{i=1}^{n} \text{tokens}(\text{chunk}_i) = \text{tokens}(\text{issue})$.

We decided to set *window size* to 4,000 tokens by default to ensure compatibility across different LLM architectures, provide sufficient context for meaningful processing, and prevent the per-

formance degradation observed in longer contexts. This decision strikes a balance between a number of factors, including ensuring adaptability to different LLMs with varying context limitations and supporting empirical evidence that current LLMs are better able to process shorter contexts.

In the second phase, each chunk is processed in parallel to extract relevant information using the LLM with a specific prompt, the "Extract Prompt." The "Extract Prompt" is designed to instruct the LLM to extract only relevant information from each segment or simply indicate `<<NONE>>` if there is no relevant information. This prompt is customized based on specific needs, for example, to identify the user's intent or to summarize the text of the issue, as we will see in the following sections.

Finally, in the Consolidation phase, we evaluate whether the total token count of all extracted information from the previous step exceeds the predefined *window size*. If it does, we create sublists of the extracted content where each sublist fits within the *window size* limit. Each sublist is then processed using the "Reduce Prompt" (reported in Figure 8.3) to generate summaries. If the combined summaries still exceed the context window, the summarization process is recursively applied to the summaries themselves until the total content fits within the window size. Once the content is appropriately sized, a final extraction step is performed using the "Extract Prompt" to ensure the consolidated information maintains relevance and coherence. This reduction process continues until the final synthesized text fits within *window size* and is thus suitable for input into the classification methods.

Reduce Prompt

Your task is to shorten the following texts up to {tokens} tokens.
I'll provide a list of texts which relates to the same topic, and you will answer only with a shorter version that combines the texts.

Texts:
{texts}

Figure 8.3: Reduce Prompt.

We note that the above-described MapReduce approach is applied only when issues exceed the specified *window size*. In the other cases, the relevant prompt can be directly applied to the entire description without dividing it into smaller parts.

8.3.3 *Method 1: Intent Extraction and Classification*

The "Intent Extraction and Classification" method consists of two main components (see Figure 8.4): 1) the Intent Extractor and 2) the Classifier. It uses the LLM to identify the user's intent behind reporting an issue and subsequently classifies the issue based on this intent.

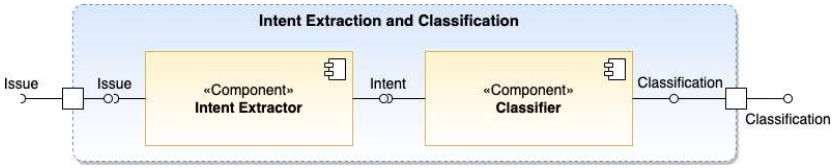


Figure 8.4: Intent Extraction and Classification: Components.

For designing prompts, we employ several key principles. First, we use a hierarchical instruction structure (Figure 8.6), where the model is guided by explicit conditional rules (e.g., "If the user is only reporting bugs... the classification must be 'Bug'"), with the aim of managing class overlap through clear priority rules when information that may concern multiple classes is present in the issue report. Second, to direct the model's attention away from superficial technical details and toward user intent, the intent-focused extraction prompt (Figure 8.5) contains specific linguistic cues. For example, we specifically instruct the model to start summaries with statements like "The user is reporting a bug where..." or "The user is asking questions..." that convey intent, which helps the model distinguish between technical content and the user's underlying motivation. Third, we incorporate negative examples and edge cases directly into the prompt instructions, for example, clarifying when questions present in the text should not override the classification of the issue as a bug. Doing so, we want to encourage the model to handle ambiguous cases where multiple interpretations could be valid.

Extract Prompt: Intent Extraction.

You are an expert issue classifier in the field of software engineering.
The possible classifications are the following:
- Bug: {description of the bug classification}.
- Feature: {description of the feature classification}
- Question: {description of the question classification}
- Documentation: {description of the documentation classification}
- Unknown: for other cases.

You are tasked to extract from the partial text of a very long issue only the most relevant information for the issue. If the text is irrelevant, simply report "<<NONE>>".

Ensure the information you provide directly reflects the user's intent and clearly categorizes the issue to assist your colleagues in classifying the issue. For example, If the issue involves unexpected behavior, malfunctions, or errors but there is an explicit request or question for assistance, start your summary with: "The user is asking questions, feedback, explanations, seeking assistance or support...." For other bugs, start your summary with: "The user is reporting a bug where...." If the issue involves documentation, instructions, or help, start your summary with: "The user is requesting additions or improvements to the documentation by....".
If the issue is related to multiple classes, make sure that the summary reflects this aspect.

Answer only with the summary, not the classification.
issue title: {title}
issue body: {body}
Summary of the issue:

Figure 8.5: Extract Prompt for Method 1.

The Intent Extractor adopts the approach described in section 8.3.2 to handle issue descriptions. Specifically, it uses the "Extract Prompt" version illustrated in Figure 8.5 to instruct the LLM to identify the user's intent from a specific text.

The Classifier takes the text representing the user's intent obtained from the Intent Extractor as input and classifies it using the LLM instructed with the prompt in Figure 8.6.

Method 1: Classification Prompt

You are an expert issue classifier in the field of software engineering. The possible classifications are the following:
- Bug: {description of the bug classification}.
- Feature: {description of the feature classification}
- Question: {description of the question classification}
- Documentation: {description of the documentation classification}
- Unknown: for other cases.

Instructions:
- If the user is only reporting bugs or errors, without seeking assistance, questions, feedback, explanations, or mentioning documentation, help, or instructions, the classification must be "Bug".
- If the user is explicitly asking questions, feedback, explanations, seeking assistance or support, the classification must be "Question", even if the user mentions bugs, errors, documentation, or other related issues.
- If the issue regards the documentation, help, or instructions, the classification must be "Documentation". This includes reporting missing information, errors, or suggesting updates to the documentation.
- If the user is requesting or suggesting a feature, the classification must be "Feature".
- In other cases, the classification must be "Unknown".

Examples:
{A Few examples}

I'll provide the body of the issue, and you will answer only with the classification.

issue body: {intent}
Classification:

Figure 8.6: Prompt for classification in Method 1.

8.3.4 *Method 2: Ensemble Classification*

The "Ensemble Classification" method is designed to enhance the robustness and reliability of the "Intent Extraction and Classification" method previously described (see 8.3.3). Indeed, the potential non-determinism in LLMs can lead to inconsistencies in in-

tent extraction and subsequent classification of issues. To mitigate this, we employ an ensemble of multiple instances of the "Intent Extraction and Classification" pipeline. By aggregating the outputs from multiple instances, we aim to achieve a consensus that is more reliable and representative of the true intent behind the reported issue.

The ensemble method involves several key steps (see Figure 8.7).

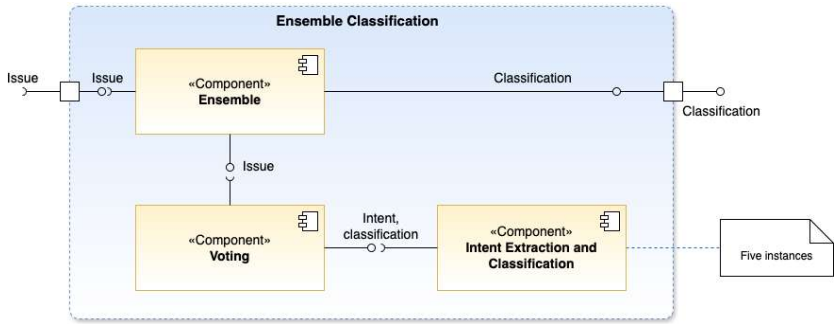


Figure 8.7: Ensemble Classification: Components.

We instantiate five parallel instances of the "Intent Extraction and Classification" pipeline. Each instance independently processes the same issue, extracting the user's intent and classifying the issue based on this intent. We choose to have five instances in the ensemble because there are four different classification categories. Indeed, despite possible outliers or residual non-determinism in the classification process, this configuration reduces the possibility of ties and increases classification reliability by guaranteeing a clear majority vote (at least three out of five) in the majority of cases. In our implementation, tie scenarios (e.g., 2+2+1 distributions) are resolved by selecting the first occurrence of the most frequent classification. More specifically, the most frequent result is chosen after counting occurrences across all instances and promoted to final classification.

8.3.5 *Method 3: Explained Few-Shot Learning*

Building upon Colavito et al.'s work [61], we enhance their few-shot learning approach by addressing the challenge of handling

long issues and incorporating explicit reasoning. Our method combines few-shot learning with co-prompt engineering [84], a process that leverages GPT-4o to collaboratively generate a set of high-quality, synthetic few-shot examples, designed to guide the model in discriminating subtle differences between classes. The method has two main components (see Figure 8.8): the Summarizer and the Explained Few-Shot Classifier.

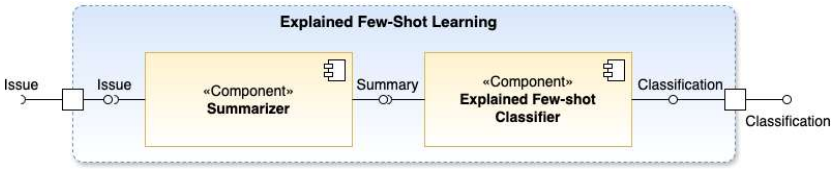


Figure 8.8: Explained Few-Shot Learning: Components.

The Summarizer condenses the issue report if it is too long to fit the LLM's context. Then the Few-Shot Classifier classifies the resulting text.

8.3.5.1 *Summarizer*

The "Summarizer" component exploits the approach described in Section 8.3.2. Specifically, it uses the "Extract Prompt" version illustrated in Figure 8.9 to instruct the LLM to summarize the most relevant information from a specific text. However, if the issue description does not exceed the *window size*, it returns the original issue description.

8.3.5.2 *Explained Few-Shot Classifier*

The "Explained Few-Shot Classifier" extends traditional few-shot learning. Rather than simply providing examples, we accompany each example with an explicit explanation demonstrating why a particular classification decision was made. This modus operandi is similar to how human experts make decisions, classifying objects based on explicit reasoning and learned patterns. In fact, human experts typically use explicit reasoning and mental checklists to categorize problems, using a variety of examples and concise justifications to help them make consistent and transparent decisions [281].

Extract Prompt: Summarization.

The possible classifications are the following: Bug, Feature, Question, Documentation, Unknown.

If the title and the text do not contain relevant information to classify the issue, simply report "<<NONE>>". You are tasked to extract from the title and the partial text of a very long issue only the most relevant information for the issue. If the text is irrelevant, simply report "<<NONE>>". When summarizing, ensure to include key details from both the title and the body of the issue. Answer only with the summary, not the classification.

issue title: {title}
issue body: {body}
Summary of the issue:

Figure 8.9: Extract Prompt for Method 3.

The development of the few-shot examples involved multiple phases. Initially, we randomly selected a stratified sample of 15 issues for each class from the training set provided by Colavito et al. [61], totaling 60 issues. We then utilized GPT-4o to generate, for each sample, a similar issue along with an explicit classification rationale, in a co-prompt engineering process [84]. We chose to generate synthetic examples based on real issues from Colavito et al. [61] rather than using unsupervised generation to ensure domain authenticity while increasing dataset diversity and coverage of edge cases [181, 244]. Two authors independently assessed the quality of these generated examples using a 5-point clarity scale (1=ambiguous, 5=very clear). The clarity assessment took into account specific elements such as ambiguous terms without antecedents (e.g., "the previous version" without identifying which), unclear references (e.g., "it" or "this thing"), and lack of logical connections between the report content and the classification rationale. Examples with scores of 5 (very clear) had none of these instances, 4 (clear) had at most one instance, 3 (moderately clear) had two to three instances, 2 (unclear) had four to five instances, and 1 (very unclear) had more than five instances. After this initial filtering, only examples with a score of $\geq$4 were kept, yield-

ing 50 high-quality examples, with a significant evaluators' inter-rater reliability (Cohen's kappa = 0.78) and disagreements settled through discussion until agreement was reached.

Few-Shot Learning with Explanation Prompt.

You are an expert at classifying issues into one of the following categories: Bug, Feature, Question, Documentation, or Unknown. Below are some examples of issues similar to the new issue, along with their classifications. Use these examples to accurately classify the new issue provided. Focus on the explanations to understand how to classify the new issue.

Examples:
1. **issue:** "With 2 nested buttons, if the outside one has the prop 'enabled=false', the inside one does not receive touch events. Tested on iOS, not sure about Android."
Classification: Bug
Explanation: "The user classified the issue as a "bug" because the nested button's inability to receive touch events when the outer button is disabled is an unintended behavior that needs to be fixed."

other examples omitted for the sake of conciseness

33. **Issue:** "I'm not clear if this is a React or Redux issue. The state doesn't update correctly when dispatching an action."
Classification: Question
Explanation: The user is seeking clarification on whether the issue is related to React or Redux, indicating they need guidance to understand the source of the problem.

New Issue:
Issue: "{issue}"
Classification:

Follow the examples and answer only with the classification.

Figure 8.10: Few-Shot Learning with Explanation Prompt.

In order to further refine the selection, we removed redundant examples by calculating the cosine similarity between sentence

embeddings (using OpenAI's text-embedding-3-small), pruning examples with similarity greater than 0.85 within categories, and retaining only instances with the highest rating. In the final example distribution, comprising 33 exemplars (2 Bug reports, 12 Questions, 7 Feature requests, 12 Documentation issues), categories with more linguistic diversity (Questions and Documentation) were given priority, and fewer examples were given to Bug reports, which showed more consistent linguistic patterns. We then designed the classification prompt following the approach of De Vito et al. [84], incorporating the 33 selected examples. As preliminary testing before our main experiment, we evaluated the classification prompt on Colavito et al.'s training dataset (separate from our test datasets), achieving F1 scores of 0.93 and 0.96 with GPT-3.5 and GPT-4o, respectively. Notably, this training set has no overlap with the datasets used in the empirical evaluation (see Section 8.4.4).

Figure 8.10 shows the final prompt structure that incorporates examples and reasoning guidelines.

8.3.5.3 *Design Principles*

The few-shot prompt (Figure 8.10) combines several design principles that enhance classification performance. First of all, as we have seen in Section 8.3.5.2, the approach is based on well-rounded and diversified examples. This strategic imbalance reflects the varying complexity of linguistic patterns across different issue types and aims at maximizing the information density of the few-shot prompt while remaining within context length constraints. Second, the integration of explicit reasoning through explanations represents a key innovation in our methodology. Each example includes both the classification and the rationale behind the decision, such as "The user classified the issue as a 'bug' because the nested button's inability to receive touch events... is an unintended behavior." This explanatory component guides the model's reasoning process and helps establish clear decision boundaries between classes. Last but not least, the prompt structure follows a clear and consistent template throughout all examples. Each instance maintains the same format of Issue → Classification → Explanation, enabling the model to internalize the expected reason-

ing pattern and apply it systematically to new classification tasks. This structural consistency facilitates better pattern recognition and more reliable classification outcomes.

8.4 EMPIRICAL EVALUATION

In this section, we detail the research methodology and experimental design used to evaluate the three proposed methods. In the remainder of the paper, for the sake of conciseness, we refer to "*Intent Extraction and Classification*" as Method 1, "*Ensemble Classification*" as Method 2, and "*Explained Few-Shot Learning*" as Method 3.

8.4.1 *Dataset Selection*

In the analyses, we utilized data extracted from the well-known NLBSE'23 Issue Report Classification Challenge dataset [156]. This dataset includes 1.4 million publicly-available issue reports, collected from GitHub repositories. Each issue report within the dataset is characterized by a title and a body, and is given one of the following labels, indicating the type of issue: *bug*, *feature*, *question*, or *documentation*.

To address the research questions of this study, we constructed multiple datasets with different characteristics and purposes, all derived from the NLBSE'23 dataset. All datasets employed in this study, including the expanded manually verified training set and the specialized long issues dataset described below, are made publicly available in the replication package at [85] to support reproducibility and future research.

8.4.1.1 *Primary Dataset*

For the main analysis, we started from a manually-verified subset of the NLBSE'23 dataset, published in prior research by Colavito et al. [63]. Using stratified sampling, Colavito et al. obtained this subset by extracting 400 issue reports – 200 as a training set and 200 as a test set – from the NLBSE'23 dataset. For each of the sampled issue reports, Colavito et al. manually re-evaluated the labels to address the presence of noisy or incorrect annota-

tions initially submitted by GitHub users. This verification process highlighted the existence of several inconsistencies in the original labels.

To expand the manually verified dataset produced by Colavito et al. and to strengthen the reliability of the results, we applied stratified sampling to extract an additional subset of 400 issue reports from the NLBSE'23 dataset, disjoint from the subset extracted by Colavito et al. Two of the authors of this paper manually verified these additional reports, following the same procedure defined by Colavito et al. in [63] with explicit annotation criteria.

The annotation process followed hierarchical decision rules based on primary user intent: Bug reports contained error descriptions or unexpected behavior without explicit help-seeking language. Questions were identified by interrogative phrases, help-seeking expressions (e.g., "*how to*", "*need help*"), or uncertainty statements, even when technical details or error logs were present. Feature requests contained explicit proposals for new functionality using terms like "*add*", "*implement*", or "*support*". Documentation issues involved requests for clarification or updates to project documentation. Edge cases where technical problems were presented alongside help-seeking language were classified as Questions, prioritizing user intent over technical content. Issues were labeled as discarded when they contained fewer than 10 meaningful words or mixed multiple intents without a clear priority.

After the independent annotation phase, we observed an almost perfect agreement between the annotators, with a Cohen's Kappa score [59] of 0.93. Disagreements occurred in 22 cases out of the 400 sampled issue reports (5.5% of cases) and were addressed by reaching a consensus through discussion. As done also in [63], issue reports for which it was not possible to reach a consensus, or for which both annotators agreed that there was not enough information to properly classify them into one of the considered classes, were assigned a *discarded* label.

As a result, the final primary dataset employed in our study consists of 600 manually verified issue reports for training (i.e., the original training set of 200 issue reports from [63], plus the 400 additional reports annotated by us) and 200 issue reports for testing (i.e., the original test set from [63]). By using the origi-

Table 8.1: Distribution of labels in the manually-verified dataset

	Training Set								Test Set			
	Initial subset (re-labeled by Colavito et al. [63])				Additional subset (re-labeled by us)				Initial subset (re-labeled by Colavito et al. [63])			
	GitHub Labeling		Manual Re-labeling		GitHub Labeling		Manual Re-labeling		GitHub Labeling		Manual Re-labeling	
Bug	50	25%	47	24%	100	25%	113	28%	50	25%	53	27%
Documentation	50	25%	33	17%	100	25%	74	19%	50	25%	32	16%
Feature	50	25%	60	30%	100	25%	106	27%	50	25%	55	28%
Question	50	25%	44	22%	100	25%	93	23%	50	25%	47	24%
Discarded	–	–	16	8%	–	–	14	4%	–	–	13	7%
Total	200 issue reports				400 issue reports				200 issue reports			

nal test set, we aim at ensuring comparability of our results with those of other studies that employed the same dataset. The distributions of labels in the primary dataset (before and after manual re-annotations) are reported in Table 8.1.

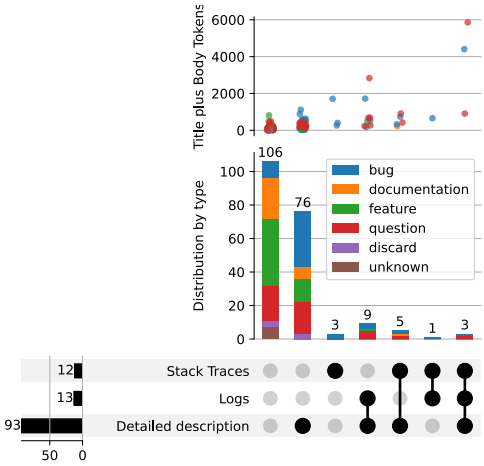


Figure 8.11: Distribution of documentation features across different issue report types in our dataset.

To better characterize the issue report test set, we conducted an analysis of documentation features across different report types. In particular, two authors of this paper manually analyzed each issue report to identify whether it includes specific documenta-

tion elements, namely *detailed descriptions*, *logs*, and *stack traces*. More in detail, we consider an issue report to not contain a detailed description if it lacks information about the environment, steps to reproduce, expected and actual results, or any meaningful additional notes. After the annotation process, we observed a substantial agreement between the annotators, with an overall Kappa score of 0.85. Disagreements were solved through discussion. The results of this analysis are visualized in Figure 8.11.

8.4.1.2 *Long Issues Dataset*

The primary dataset is skewed toward shorter issue reports, with the majority containing fewer than 1000 tokens. This distribution reflects common patterns in open-source projects, where many issues are brief and lack detailed context. However, as discussed in Section 8.3.2, these shorter issues may not be representative of the more detailed and verbose issue reports often encountered in industrial and large-scale software projects.

To better capture this complexity and evaluate the robustness of our approach on longer inputs, we created a specialized dataset focusing on issues with substantial textual content. We extracted a stratified sample of 60 issues from the NLBSE'23 dataset [156], ensuring representation across both classification labels and issue length categories.

The stratification was performed along two dimensions:

- **Classification labels**: *bug*, *documentation*, *feature*, and *question*
- **Length categories** based on the total token count of the issue title and body:
  - Class 1: 4,001 to 10,000 tokens
  - Class 2: 10,001 to 15,000 tokens
  - Class 3: 15,001 tokens and above

For each combination of length class and classification label, we extracted five issue reports, resulting in a total of 20 issue reports per length class and 60 issue reports overall (5 reports × 4 labels × 3 length classes = 60 reports).

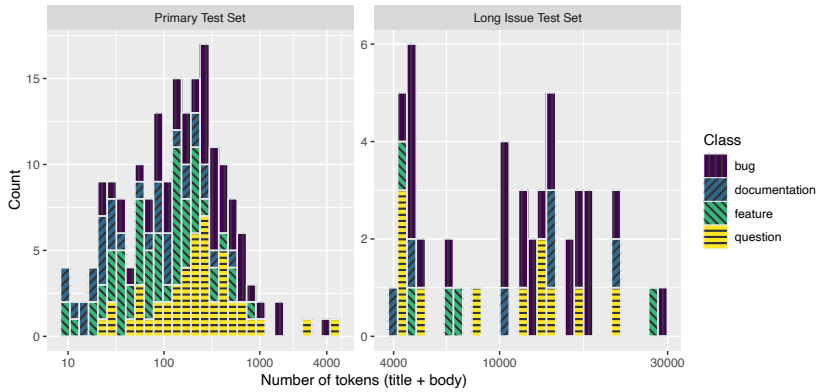


Figure 8.12: Distribution of token lengths (title + body) in the primary and long issue test sets. The x-axis is log-scaled to account for the strong skewness in token counts.

The labels assigned to the sampled issue reports were verified manually following the same rigorous procedure used for the primary dataset. Specifically, two authors of this paper independently annotated each of the 60 sampled issue reports. After the independent annotation phase, we calculated the inter-annotator agreement using Cohen's Kappa score [59], observing a high level of agreement with a Kappa score of 0.95. Issues for which consensus could not be reached were assigned a *discarded* label, resulting in 12 discarded issues from the original sample.

Figure 8.12 shows the distribution of token lengths in both the primary and long issue test sets. Due to the strong skewness in token counts, we adopted a log-scaled x-axis to improve readability. As the figure shows, the primary dataset is concentrated in the 100–1000 token range, with *bug* and *question* issues being generally associated with a higher token count than *feature* and *documentation* issues. The long issue dataset, on the other hand, spans from 4K to 30K tokens, capturing a broader spectrum of real-world complexity.

8.4.2 *LLMs Selection*

In order to assess the performance of the proposed methods across a range of model architectures, capabilities, and deployment sce-

narios, we used three distinct LLMs: GPT-4o, GPT-3.5-turbo (henceforth simply referred to as GPT-3.5 for brevity), and Qwen-2.5 (32B parameters) [239].

The OpenAI API was used to access GPT-4o and GPT-3.5. We chose to use GPT-4o because, as OpenAI also reported, we expected the newer version to perform better. This choice allowed us to leverage the latest advancements in language model technology and ensured that our methods benefit from enhanced effectiveness and robustness. Conversely, we selected GPT-3.5 to benchmark the performance of our methods against the approach proposed by Colavito et al. [61]. However, it is noteworthy that OpenAI has announced the deprecation of GPT-3.5. Despite its impending deprecation, GPT-3.5 remains a relevant benchmark due to its widespread use and established performance metrics in prior research.

The inclusion of Qwen-2.5 represents a strategic decision to evaluate our methods with an open-weight alternative that offers different architectural characteristics. Indeed, Qwen-2.5 is a decoder-only transformer model with 32 billion parameters, and offers a significant capacity for intricate reasoning tasks while still being deployable in local settings. We deployed the model locally using LM Studio (i.e., a comprehensive platform for LLM experimentation that provides a unified OpenAI-compatible REST API for model deployment and inference across various architectures), with the aim of simulating enterprise or sensitive data environments, where privacy concerns and computational autonomy are critical requirements.

8.4.3 *Baselines*

As baselines, we selected three state-of-the-art approaches that have been recently presented in the literature. In particular, we consider the following baselines: (1) the RoBERTa-based approach which is also used as a baseline in the NLBSE'23 Issue Report Classification challenge [156], (2) the SetFit-based approach proposed in [63], and (3) the ChatGPT-based approaches proposed in [61]. Details of each baseline approach are discussed in Section 8.2.

8.4.4 *Experimental Procedure and Metrics*

This section outlines the procedures and metrics used to address the research questions.

8.4.4.1 *RQ8.1: Method Effectiveness and Operational Characteristics*

RQ8.1 investigates the effectiveness and operational characteristics of the proposed methods through six analyses: (1) classification performance evaluation and temperature sensitivity analysis, (2) baseline comparison, (3) intent extraction ablation study, (4) long issue handling evaluation, (5) token usage and cost assessment, and (6) perplexity-based analysis. The performance metrics and procedure for these analyses are explained in the following sections.

PERFORMANCE METRICS To address RQ8.1, we measured the performance of the proposed methods in terms of precision, recall, and F1-measure for all the considered issue report classes, as also done in previous studies [61, 63]. For a given class, precision is defined as the ratio of *true positives* (i.e., instances of that class that were classified correctly) over the total number of instances predicted as that class. It reflects the accuracy of the model's predictions for a given class. Recall, on the other hand, measures the extent to which the model is capable of recognizing all instances of a given class, and is measured as the ratio of true positives over the total number of instances belonging to the considered class. Lastly, the F1-measure is defined as the harmonic mean of precision and recall, thereby balancing both metrics to provide a single performance score. To provide a comprehensive and balanced evaluation, we report the *macro-averaged* and *weighted-averaged* values of these metrics across the considered classes. For each metric, the macro-averaged values are computed by calculating the metric independently for each class and then averaging them, with each class weighted equally. These values provide insights into how a model performs across all classes, regardless of the class distribution. On the other hand, the weighted-averaged values are calculated by dividing the metric for each class by the number of instances in that class. In situations where there are unbalanced

classes, like the one examined in this study, this method guarantees that the final performance measure is more representative of the entire dataset.

CLASSIFICATION PERFORMANCE EVALUATION AND TEMPERATURE SENSITIVITY ANALYSIS To rigorously assess the classification performance of our three proposed methods, we conducted a comprehensive evaluation on a manually verified test set of 200 issue reports (see Section 8.4.1.1). Since the optimal value of the temperature parameter, which controls the degree of randomness in LLM token selection, is not known a priori for this task, we systematically explored its impact through a temperature sensitivity analysis. Specifically, each method was evaluated using three LLMs (GPT-3.5, Qwen-2.5, and GPT-4o as discussed in Section 8.4.2) across four representative temperature settings: 0 (minimal randomness), 0.3 and 0.7 (moderate stochasticity levels that may improve performance on tasks requiring diverse reasoning approaches), and 1.0 (unscaled probability distribution providing a natural baseline for stochastic sampling behavior).

For each method-model-temperature configuration, we performed five independent runs to ensure robust and reliable estimates. Note that, while setting the temperature to 0 significantly reduces randomness, recent research demonstrates that complete determinism cannot be guaranteed even at a temperature of 0 due to inherent implementation details in LLM inference [223]. All this means that multiple runs remain valuable, even with the temperature set to 0, to account for residual non-deterministic behavior and provide more robust performance estimates. For each repetition, we computed F1-scores for each class, as well as macro-averaged and weighted-averaged F1-scores, enabling a detailed comparison of classification effectiveness and the identification of the best-performing temperature for each method-model combination. This joint analysis provides both a comprehensive assessment of classification performance and empirical guidance for selecting optimal temperature settings in subsequent experiments.

COMPARISON WITH BASELINES A separate comparative evaluation assessed the proposed methods' performance – measured in the *Classification Performance Evaluation and Temperature Sensi-*

tivity Analysis – against the three established baselines reported in Section 8.4.3. The RoBERTa-based, SetFit-based, and GPT-3.5-based approaches were implemented using their original configurations and trained on the expanded, manually verified training set of 600 issue reports (see Section 8.4.1.1). These baselines were then evaluated on the same test set using the same metrics, allowing for a direct performance comparison with the proposed methods. This analysis specifically aimed to demonstrate whether the proposed approaches could achieve superior classification effectiveness compared to existing state-of-the-art methods.

INTENT EXTRACTION ABLATION STUDY To isolate the contribution of the intent extraction component to the overall performance of Methods 1 and 2, we conducted an ablation study. For each method-LLM combination, two versions were evaluated on the same test set of 200 issue reports illustrated in Section 8.4.1.1: one with the complete pipeline, including intent extraction, and one without the intent extraction component. The ablation study measured classification performance under both conditions across all three LLMs, computing the same precision, recall, and F1-measure metrics. To evaluate the effectiveness of the MapReduce-inspired approach for handling long issue reports, we conducted classification analyses using Method 1 and Method 3 on the long issues test set described in Section 8.4.1.2. This dataset was specifically constructed to include only issue reports with a minimum length of 4,001 tokens, ensuring that each sample exceeds the default context window of 4,000 tokens and thus is eligible for the multi-step summarization strategy. Method 2 was not included in this evaluation since it functions as an ensemble of Method 1 instances and preserves the same MapReduce benefits without altering the underlying preprocessing strategy. The experiments were divided into two conditions:

- **With the MapReduce Approach**: We applied the MapReduce-inspired pipeline described in Section 8.3.2 to classify the issues.
- **With Truncation**: We truncated the combined title and body of each issue to a maximum of 4,000 tokens before classification. Truncation represents a simplified approach that

does not utilize the proposed approach for handling varying lengths, relying only on the initial portion of the description for classification.

The classification results were evaluated using the same metrics described in Section 10.5.2. In addition, we evaluated classification results of the baselines, the RoBERTa model [156] and SETFIT [63], using the same dataset and metrics, to verify their effectiveness on long text descriptions.

To further investigate whether longer issue descriptions can be effectively processed by LLMs with extended context windows, we performed an additional analysis using GPT-4o with direct ingestion of the full issue text, leveraging its full context capacity of 128k tokens. This examination was conducted leveraging only GPT-4o, as the context size of the other considered LLMs (16k tokens for GPT-3.5, 32k tokens for Qwen-2.5) is not sufficient to fit all the issues in our longer issues test set.

LONG ISSUE HANDLING EVALUATION To evaluate the effectiveness of our MapReduce-inspired approach for handling long issue reports, we conducted classification experiments using Method 1 and Method 3 on the long issues test set described in Section 8.4.1.2. This dataset was specifically constructed to include only issue reports with a minimum length of 4,001 tokens, ensuring that each sample exceeds the default context window of 4,000 tokens and thus is eligible for the multi-step summarization strategy. The experiments were divided into two conditions:

- **With the MapReduce Approach**: We applied the MapReduce-inspired pipeline described in Section 8.3.2 to classify the issues.
- **With Truncation**: We truncated the combined title and body of each issue to a maximum of 4,000 tokens before classification. Truncation represents a simplified approach that does not utilize our proposed approach for handling varying lengths, relying only on the initial portion of the description for classification.

The classification results were evaluated using the same metrics described in Section 10.5.2. In addition, we evaluated classi-

fication results of the baselines, the RoBERTa model [156] and SETFIT [63], using the same dataset and metrics, to verify their accuracy on long text descriptions.

To further investigate whether longer issue descriptions can be effectively processed by LLMs with extended context windows, we conducted an additional experiment using GPT-4o with direct ingestion of the full issue text, leveraging its full context capacity of 128k tokens. This experiment was conducted leveraging only GPT-4o, as the context size of the other considered LLMs (16k tokens for GPT-3.5, 32k tokens for Qwen-2.5) is not sufficient to fit all the issues in our longer issues test set.

TOKEN USAGE AND COST ANALYSIS To investigate the practical feasibility and sustainability of the proposed methods and to provide practical guidance to practitioners interested in LLM-based issue classification systems at scale, we analyzed both token consumption patterns and associated classification costs.

For token usage analysis, we tracked the total number of input and output tokens required by each LLM/method combination to process every issue in our test sets (see Section 8.4.1.1). This granular token accounting allows us to understand the computational overhead of different approaches and identify potential optimization opportunities. The analysis was conducted separately for the longer issues test set (see Section 8.4.1.2), comparing both the truncation and MapReduce approaches to investigate their respective resource consumption profiles.

To translate token usage into practical cost considerations, we adopted a token-based accounting approach grounded in the pricing models of commercial LLM-as-a-service offerings. Token counts were multiplied by reference per-token costs to derive cost estimates that reflect real-world deployment scenarios. The reference costs for GPT-based models were obtained from Microsoft Azure's publicly available pricing for their language model service offerings [201]. For Qwen-2.5, which was deployed locally, we simulated a realistic cloud-based deployment scenario by applying the pricing model offered by Cloudflare Workers AI [56]. This approach ensures fair and consistent cost comparisons across all models. The reference rates (in USD) we considered are:

- GPT-3.5: $ 1.50 per 1M input tokens, $ 2.00 per 1M output tokens;
- Qwen-2.5: $ 0.66 per 1M input tokens, $ 1.00 per 1M output tokens;
- GPT-4o: $ 2.50 per 1M input tokens, $ 10.00 per 1M output tokens.

PERPLEXITY ANALYSIS EXPERIMENT To gain a more in-depth insight into LLM-based classification performance, we conducted additional experiments using our test set (see Section 8.4.1.1) to investigate whether a model's internal perplexity metric [149] – which measures model uncertainty in predictions – differs between correct and incorrect predictions. This analysis could provide critical insights into the reliability of this metric as a measure of decision certainty in the issue report classification domain, with implications for both model interpretability and evaluation. More formally, the perplexity metric we considered is defined as follows:

$$Perplexity = \exp\left(-\frac{1}{n}\sum_{i=1}^{n}\log p_i\right),$$

where:

- n is the number of tokens in the generated sequence
- $\log p_i$ is the log probability of the i-th token as assigned by the language model

Perplexity is a standard metric in natural language processing that measures how well a probability model predicts a sample [149]. Lower perplexity values indicate higher model confidence, as the model assigns higher probabilities to the generated tokens. For this metric, we computed three variants:

- *Class*: Perplexity is computed limited to the final classification step, using only the log probabilities from the classification model's final output tokens.
- *Summary or Intent*: Perplexity is computed concerning the summarization step (or intent extraction step, depending on the method used), using only the log probabilities from the summary generation tokens.

- *Total*: Perplexity is computed by combining the log probabilities from both the summarization (or intent) and classification steps, taking the mean of all log probabilities across both tasks to provide an overall confidence and perplexity measure for the entire pipeline.

We note that these experiments are limited to GPT-3.5 and GPT-4o, which are the only models for which it was possible, through the OpenAI APIs, to access token-level log probabilities. While open-source alternatives that provide OpenAI-compatible APIs exist (such as vLLM, Text Generation Inference, or Ollama), most current implementations do not expose token-level log probabilities in their API responses. Although ongoing development work is underway to incorporate this functionality into open-source serving frameworks, at the time of this study, reliable access to log probabilities was only available through OpenAI's commercial API endpoints.

To assess whether perplexity values differ between correct and incorrect classifications, for each LLM and method, we stratified the test set into two groups based on prediction correctness. We then applied Wilcoxon Rank-Sum Tests to evaluate the alternative hypothesis that perplexity is significantly lower for correctly classified instances. To account for multiple comparisons, we employed the Holm-Bonferroni correction with an initial significance level of $\alpha = 0.05$.

Building on insights from the perplexity analysis, we conducted an additional experiment to evaluate whether Method 2's ensemble approach could be improved by replacing simple majority voting with perplexity-based alternatives (i.e., selecting the response with the smallest perplexity). This experiment compared the alternative voting mechanisms for Method 2 across GPT-3.5 and GPT-4o, measuring F1-scores for each class and computing weighted-averaged performance to assess whether confidence-weighted decisions could improve ensemble performance.

RQ8.2 aims to assess the consistency of classification results across multiple runs to ensure the proposed methods deliver the stable and predictable outcomes required for practical deployment in software engineering environments.

To quantify classification consistency, we employed two complementary metrics commonly used in Software Engineering stud-

ies. The raw percentage of agreement was calculated as the percentage of issue reports for which all model repetitions agreed on the same label [334]. This metric provides a straightforward overview of classification consistency across different runs but does not account for the possibility of agreements occurring by chance alone. To address this limitation, we computed the Fleiss' Kappa inter-rater agreement index [114], a statistical measure that evaluates the level of agreement between multiple raters while accounting for chance agreements. The Fleiss' Kappa score ranges between 0 and 1, with a value of 1 indicating perfect agreement and a value of 0 representing agreement occurring purely by random chance [171].

The core consistency evaluation analyzed the variability in classification outcomes across the five repetitions performed in Section 8.4.4.1. Each repetition was treated as a distinct annotator, allowing us to measure inter-rater agreement between different classification runs using the established metrics. This analysis was conducted for each combination of the three proposed methods and three LLMs, in order to provide insights in terms of stability. The analysis examined whether certain method-model combinations exhibited superior consistency properties, which could influence their suitability for production deployment despite potentially comparable performance.

To complement the analysis of temperature effects on classification performance, we investigated how temperature variations affect the consistency of predictions across multiple runs. The same temperature settings used in section 8.4.4.1 were employed: 0.0, 0.3, 0.7, and 1.0. For each temperature-method-model combination, we computed both raw percentage agreement and Fleiss' Kappa scores across the five repetitions. This investigation aimed to determine whether lower temperature settings, which reduce randomness in token selection, correspondingly improve prediction consistency and whether this relationship holds uniformly across all methods and models. The analysis provided insights into the trade-off between classification performance and consistency at different temperature levels, informing optimal parameter selection for practical applications where both performance and reliability are critical considerations.

8.4.4.2 *RQ8.3: Error Patterns and Classification Dynamics*

The goal of RQ8.3 is to uncover misclassification patterns, highlight recurring challenges, and identify specific weaknesses in the proposed methods, which could guide future improvements in issue report classification systems.

To this end, we adopted a multi-level, progressive-refinement approach, focusing on the issues that were consistently misclassified by the proposed methods in the evaluation described in Section 8.4.4.1, i.e., for which the classification was incorrect in all repetitions. First, we analyzed confusion matrices to identify which class pairs were most frequently confused by examining how often issues of each actual class were incorrectly predicted as other classes. This quantitative analysis revealed the most problematic class distinctions and provided an overview of systematic classification errors.

To gain deeper insights into the nature of misclassifications, we conducted a fine-grained manual inspection of the consistently incorrect predictions. This analysis was guided by a structured annotation protocol designed to ensure systematic and reliable identification of the underlying causes behind each misclassification. Initially, two authors jointly annotated a randomly sampled subset of 20 consistently misclassified issues to develop a taxonomy of error causes and establish a shared understanding of classification criteria. This initial coding process focused on identifying specific characteristics that could explain classification failures. After establishing these criteria through joint annotation of the initial subset, the authors independently annotated the remaining misclassified issues. To assess the stability and consistency of the independent annotations, we computed the inter-rater agreement score using Cohen's Kappa. Disagreements were resolved through discussion until complete consensus was achieved.

Subsequently, we performed a combined analysis, linking class-level misclassification patterns with the fine-grained taxonomy. This combined analysis allowed us to investigate whether different types of misclassification errors were more linked to specific class confusions.

Finally, we conducted a cross-method comparative analysis examining error patterns across the three proposed methods and

three LLMs. This analysis investigated several dimensions: (1) how the same misclassified issues were handled differently by each method-LLM combination; (2) whether Method 2's ensemble approach corrected or consolidated errors from Method 1; (3) how Method 3's few-shot learning affected error redistribution across different models; and (4) the relationship between issue characteristics (token length, presence of technical artifacts) and classification robustness. The analysis examined both individual method failures and consensus failures where all LLMs misclassified the same issue, aiming to identify systematic biases, method-specific failure modes, and potential complementary strengths across different approaches. Additionally, for GPT-3.5 and GPT-4o, we analyzed perplexity patterns to investigate uncertainty calibration differences between correct and incorrect predictions across the different methods.

8.5 ANALYSIS OF THE RESULTS

In this section, we present the results for each research question.

8.5.1 *RQ8.1: Method Effectiveness and Operational Characteristics*

In the following, we report the results for the evaluations illustrated in Section 8.4.4.1. For the sake of conciseness, we only report F1 scores, while the precision and recall scores are available in the replication package.

8.5.1.1 *Classification Performance Evaluation and Temperature Sensitivity Analysis*

The results of the temperature (T) sensitivity analysis are reported in Table 8.2, showing weighted-averaged F1-scores across four temperature settings (0.0, 0.3, 0.7, 1.0) for all method-LLM combinations.

GPT-4o demonstrates remarkable stability across all temperature settings and methods, with performance remaining essentially unchanged (0.91-0.95 range). GPT-3.5 shows moderate sensitivity, particularly with Method 2, where performance drops from 0.86 (T=0.0) to 0.81 at higher temperatures.

Table 8.2: Weighted-averaged F1-scores achieved by the proposed methods across different temperature (T) settings.

	Method 1			Method 2			Method 3		
T	**GPT-3.5**	**Qwen-2.5**	**GPT-4o**	**GPT-3.5**	**Qwen-2.5**	**GPT-4o**	**GPT-3.5**	**Qwen-2.5**	**GPT-4o**
0.0	0.80	0.89	0.92	0.86	0.89	0.92	0.86	0.92	0.95
0.3	0.82	0.89	0.91	0.81	0.90	0.92	0.86	0.71	0.92
0.7	0.80	0.88	0.91	0.81	0.89	0.92	0.86	0.71	0.93
1.0	0.80	0.88	0.91	0.81	0.91	0.92	0.85	0.71	0.93

Most notably, Qwen-2.5 exhibits a unique architecture-method interaction. While remaining stable with Methods 1 and 2 across all temperatures, it suffers performance degradation with Method 3 when the temperature exceeds 0.0 (F1-score drops from 0.92 to 0.71). This behavior suggests a specific incompatibility between Qwen-2.5's architecture and few-shot learning with explanations under stochastic sampling conditions. These findings indicate that, although temperature sensitivity varies for some method-LLM combinations, the classification task appears to benefit primarily from consistent reasoning rather than creative exploration. Given that T=0.0 provides better performance in most method-model combinations, we adopt T=0.0 for all subsequent experiments.

The detailed classification performance results for the three proposed methods across different LLMs, with temperature see to 0, are reported in Table 8.3 (rows 4-12). Method 3 (Explained Few-Shot Learning) achieves the best overall performance, particularly with GPT-4o (0.95 weighted F1-score), representing a 3% improvement over the other methods. Method 2 (Ensemble Classification) consistently outperforms Method 1 across all LLMs, with the most significant gains observed with GPT-3.5 (6% improvement: 0.86 vs 0.80 weighted F1-score).

The results reveal interesting patterns across model architectures. GPT-4o demonstrates the highest performance ceiling (0.92-0.95 weighted F1-scores). Surprisingly, Qwen-2.5, despite having fewer parameters and a different decoder-only architecture, outperforms GPT-3.5 across all methods (0.89-0.92 vs 0.80-0.86), suggesting that architectural design and training strategies may be more influential than parameter count for classification tasks.

Table 8.3: Classification results in terms of F1-measure for each class, plus weighted-averaged values (best F1-measure values are highlighted in bold). In the last column, the table also reports the weighted average F1 variation (Δ) against the best-performing baseline (SetFit), with positive Δ values indicating performance gains.

	F1-measure					
Model	**Bug**	**Docum.**	**Feature**	**Question**	**Weighted Average**	**Weighted Average Δ vs. SetFit**
RoBERTa-based [156]	0.87	0.71	0.80	0.82	0.81	-0.06
GPT 3.5-based [61]	0.83	0.77	0.84	0.79	0.82	-0.05
SetFit-based [63]	0.89	0.82	0.86	0.90	0.87	–
Method 1 GPT-3.5	0.85	0.81	0.84	0.70	0.80	-0.07
Method 2 GPT-3.5	0.91	0.82	0.86	0.84	0.86	-0.01
Method 3 GPT-3.5	0.86	0.88	0.91	0.80	0.86	-0.01
Method 1 Qwen-2.5	0.92	0.86	0.88	0.89	0.89	+0.02
Method 2 Qwen-2.5	0.92	0.86	0.88	0.89	0.89	+0.02
Method 3 Qwen-2.5	0.91	0.93	0.94	0.89	0.92	+0.05
Method 1 GPT-4o	0.90	0.93	0.94	0.90	0.92	+0.05
Method 2 GPT-4o	0.91	0.93	0.93	0.90	0.92	+0.05
Method 3 GPT-4o	**0.94**	**0.98**	**0.95**	**0.94**	**0.95**	+0.08

Moreover, GPT-3.5 demonstrates higher sensitivity to method choice, with performance ranging from 0.80 to 0.86, implying that this model's effectiveness is more dependent on the specific prompt engineering approach used.

8.5.1.2 *Comparison with Baselines*

Table 8.3 reports classification performance for both the baseline models (RoBERTa, SetFit, and the LLM-based approach from [61], rows 1–3) and the proposed methods (rows 4–12). When comparing the proposed methods against these baselines, results show that the proposed approaches can achieve superior performance, with varying degrees of improvement depending on the baseline and model combination.

All proposed methods with GPT-4o significantly outperform the best baseline (SetFit: 0.87 weighted F1-score), achieving improvements ranging from 5% (Methods 1 and 2: 0.92) to 8% (Method 3: 0.95). Similarly, Qwen-2.5 outperforms all baselines across all methods, with weighted F1-scores ranging from 0.89 to 0.92, representing improvements of 2% to 5% over the best baseline. When using GPT-3.5, performance varies by method. Method 1 (0.80 weighted F1-score) underperforms compared to SetFit (0.87), while Methods 2 and 3 (both 0.86) approach SetFit's performance but remain slightly below. Notably, SetFit excels specifically in question classification (0.90 F1-score), while our GPT-3.5 methods show more balanced performance across other classes.

Compared to the existing LLM-based approach by Colavito et al. (0.82 weighted F1-score with GPT-3.5), the Methods 2 and 3 achieve meaningful improvements (both 0.86), representing a 4% gain. All this demonstrates that both ensemble approaches and few-shot learning with explanations provide tangible benefits over basic LLM prompting strategies, and establishes that the proposed methods achieve state-of-the-art performance, with the magnitude of improvement being architecture-dependent and method-specific.

8.5.1.3 *Impact of Intent Extraction on LLM-based Issue Report Classification*

The results of the ablation study described in Section 8.4.4.1 are reported in Table 8.4. For the sake of conciseness, we only report macro-averaged and weighted-averaged F1 scores. Class-level F1 scores are available in the replication package.

GPT-3.5 shows the most significant improvements by using intent extraction. Method 1 improves by 7% in weighted F1-score (0.73→0.80), while Method 2 demonstrates an even larger gain of 15% (0.71→0.86). These substantial improvements indicate that GPT-3.5 particularly benefits from the structured approach of explicitly extracting user intent before performing classification.

GPT-4o and Qwen-2.5 exhibit smaller but meaningful improvements. GPT-4o shows 2% improvement for Method 1 (0.90→0.92) and 3% for Method 2 (0.89→0.92). Qwen-2.5 demonstrates 4% improvement for Method 1 (0.85→0.89) and 2% for Method 2

Table 8.4: Results of the ablation study on the intent extraction component.

Model	With Intent Extraction		Without Intent Extraction	
	Macro Averaged F1	Weighted Averaged F1	Macro Averaged F1	Weighted Averaged F1
Method 1 GPT-3.5	0.80	0.80	0.70	0.73
Method 2 GPT-3.5	0.86	0.86	0.68	0.71
Method 1 GPT-4o	0.92	0.92	0.89	0.90
Method 2 GPT-4o	0.92	0.92	0.89	0.89
Method 1 Qwen-2.5	0.89	0.89	0.84	0.85
Method 2 Qwen-2.5	0.89	0.89	0.87	0.87

(0.87→0.89). These results suggest that different architectural designs respond differently to structured intent information, with GPT-4o's advanced capabilities providing some compensation for missing intent extraction, while Qwen-2.5's decoder-only architecture shows consistent benefits from the structured approach.

Method 2 generally shows greater improvements from intent extraction compared to Method 1, with the effect being most pronounced for GPT-3.5, where Method 2's improvement (15%) substantially exceeds Method 1's (7%). However, this pattern is not consistent across all models – Qwen-2.5 shows the opposite trend, with Method 1 benefiting more from intent extraction (4%) than Method 2 (2%). This suggests that the interaction between ensemble voting and intent extraction is model-dependent rather than universally amplifying.

8.5.1.4 *Handling Long Issue Descriptions*

The results of the evaluation on longer issue reports are reported in Table 8.5. The table shows the weighted average F1 scores for the three LLM-based methods across the primary test set (for consistency check and as baseline for comparison), the longer issues test set with MapReduce-like processing, and the longer issues test set with simple truncation, alongside baseline comparisons with RoBERTa and SetFit. To complement the tabular results and improve interpretability, we provide a comparative visualization

Table 8.5: Weighted average F1 scores for issue classification methods across different test sets and input processing approaches. The highest F1 scores in each column are highlighted in bold.

Model	Primary Test Set	Longer Issues Test Set	
		MapReduce Approach	Truncation Approach
RoBERTa	0.81	–	0.75
SetFit	0.87	–	0.79
Method 1 GPT-3.5	0.80	0.82	0.71
Method 1 Qwen-2.5	0.89	0.81	0.78
Method 1 GPT-4o	0.92	0.89	0.87
Method 3 GPT-3.5	0.86	0.84	0.66
Method 3 Qwen-2.5	0.92	0.90	0.79
Method 3 GPT-4o	**0.95**	**0.94**	**0.85**

in Figure 8.13 that reports the differences in weighted average F1 scores between each proposed method–LLM combination and the RoBERTa and SetFit baselines across all evaluation settings.

The results show that, across all methods and LLMs, the MapReduce approach provides substantial benefits when handling longer issue reports, outperforming simple truncation with improvements ranging from 2-11 percentage points with Method 1 and 9-18 percentage points with Method 3. This validates our hypothesis that preserving information for longer issues through the MapReduce strategy is crucial for preserving classification effectiveness.

Comparing these results with those achieved on the primary test set, we observe that the proposed methods generally maintain comparable or even improved performance on longer issues, while simple truncation leads to substantial degradation. In contrast, the baselines show notable performance degradation on longer issues, with RoBERTa dropping from a weighted average F1 score of 0.81 on the primary test set to 0.75 on the longer issues test set, and SetFit exhibiting a similar decline from 0.87 to 0.79, highlighting the inherent limitations of these approaches in dealing with extended input sequences that exceed their fixed context windows.

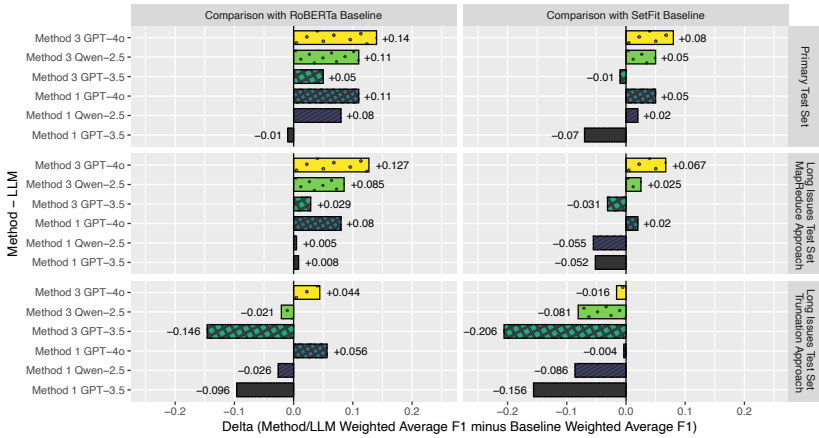


Figure 8.13: Differences in weighted average F1 scores between each proposed method–LLM combination and the RoBERTa and SetFit baselines, across three evaluation settings: Primary Test Set, Longer Issues Test Set (MapReduce Approach), and Longer Issues Test Set (Truncation Approach). Positive values indicate performance improvements over the baseline.

The results of the additional experiment aimed at assessing the effectiveness of directly ingesting full-length issue descriptions are reported in Table 8.6, and confirm that the MapReduce approach yields better performance. Indeed, classification F1 scores are consistently lower when using the full-context input compared to MapReduce-style processing.

For Method 1, the weighted average F1 score drops from 0.89 with MapReduce to 0.80 with full input, which is also lower than the truncation result of 0.87. For Method 3, direct ingestion of

Table 8.6: Weighted average F1 scores achieved by GPT-4o for longer issue classification using three input processing strategies: MapReduce-style decomposition, truncation, and direct ingestion of full issue descriptions.

Method	MapReduce	Truncation	Direct Ingestion
Method 1	0.89	0.87	0.80
Method 3	0.94	0.85	0.89

Table 8.7: Average cost (in USD) to classify a single issue report using different LLMs, methods, and processing strategies.

					Longer Issues Test Set					
		Primary Test Set			MapReduce Approach			Truncation Approach		
LLM	Method	Token Input	Token Output	Cost (USD)	Token Input	Token Output	Cost (USD)	Token Input	Token Output	Cost (USD)
GPT-3.5	Method 1	1,746	30	$0.003	11,493	204	$0.018	4,768	41	$0.007
	Method 2	8,730	150	$0.013	57,466	1,020	$0.088	23,840	206	$0.036
	Method 3	3,906	4	$0.006	12,163	54	$0.018	7,578	3	$0.011
Qwen-2.5	Method 1	1,785	34	$0.001	11,526	227	$0.008	4,768	47	$0.003
	Method 2	8,926	170	$0.006	57,629	1,135	$0.039	23,840	235	$0.016
	Method 3	3,962	5	$0.003	12,267	130	$0.008	7,578	7	$0.005
GPT-4o	Method 1	1,750	32	$0.005	11,530	222	$0.031	4,768	38	$0.012
	Method 2	8,749	160	$0.023	57,651	1,110	$0.155	23,840	190	$0.062
	Method 3	3,855	1	$0.010	11,901	1	$0.030	7,569	1	$0.019

the full issue descriptions yields an F1 score of 0.89, placing it between the truncation approach (0.85) and the more effective MapReduce strategy (0.94). This finding aligns with observations from the literature [13, 302], where LLM performance has been observed to degrade as input context length increases, and reinforces the importance of structured input decomposition for long-form content, even when using models capable of handling large contexts.

8.5.1.5 *Token Usage and Cost Analysis*

Table 8.7 reports the average cost (in USD) of classifying a single issue report across different LLMs and processing strategies. Across all models and test sets (see Section 8.4.1), Method 1 is consistently the least expensive, with average costs ranging from $0.001 (Qwen-2.5) to $0.005 (GPT-4o) on the primary test set. In contrast, Method 3 incurs slightly higher costs, with primary test set values ranging from $0.003 (Qwen-2.5) to $0.010 (GPT-4o). This increase is attributed to its use of few-shot prompting, which includes multiple examples in the input and thus inflates token usage. The most expensive strategy is Method 2, which involves

an ensemble of five Method 1 executions. This design leads to significantly higher costs, with primary test set values reaching $0.023 (GPT-4o) and $0.013 (GPT-3.5). These trends hold across the longer issues test set as well, where Method 2 under the MapReduce approach reaches up to $0.155 per issue (GPT-4o), emphasizing the need to carefully weigh performance gains against economic sustainability. Focusing on the longer issues test set, the results show that the MapReduce approach, while guaranteeing superior classification performance w.r.t. simple truncation (as discussed in Section 8.5.1.4), incurs in higher inference costs due to the increased number of processed tokens. Across all models and methods, inference costs with the MapReduce strategy are approximately 2 to 2.5 higher than simple truncation.

8.5.1.6 *Analysis of Perplexity*

The analysis investigated whether LLM internal perplexity metrics can predict classification performance and improve ensemble voting mechanisms. Results are reported in Table 8.8. Note that, for the sake of conciseness, in the table we refer to the *Summary/Intent* perplexity as "*summary perplexity*".

Class-level perplexity emerges as a reliable indicator of classification performance. Statistical tests reveal significant differences ($p < 0.001$) between correct and incorrect predictions for class perplexity across most method-LLM combinations, with consistently lower perplexity values associated with correct classifications. Effect sizes are particularly large for GPT-4o with Methods 1 and 3 (Cohen's d = -0.59 and -1.93, respectively) and GPT-3.5 with Method 1 (Cohen's d = -0.82). In contrast, summary and total perplexity metrics generally exhibit non-significant differences between correct and incorrect predictions.

Method 2 (ensemble) shows weaker perplexity-performance associations for GPT-3.5, likely due to the averaging effect of multiple predictions reducing individual prediction uncertainty. Method 3 exhibits the strongest perplexity signals with GPT-4o, suggesting that few-shot learning creates more calibrated confidence estimates in advanced models. Building on these findings, we evaluated whether replacing majority voting with minimum class perplexity voting could improve Method 2's performance (Table 8.9).

Table 8.8: Results of the statistical tests investigating differences in perplexity metrics between correctly and incorrectly classified issue reports.

LLM	Method	Metric	p-value	Cohen's d	Effect Size
GPT-3.5	Method 1	class perplexity	**3.15E-04**	-0.82	large
GPT-3.5	Method 1	summary perplexity	5.02E-01	–	–
GPT-3.5	Method 1	total perplexity	3.84E-01	–	–
GPT-4o	Method 1	class perplexity	**3.09E-03**	-0.59	medium
GPT-4o	Method 1	summary perplexity	9.94E-01	–	–
GPT-4o	Method 1	total perplexity	8.83E-01	–	–
GPT-3.5	Method 2	class perplexity	5.00E-01	–	–
GPT-3.5	Method 2	summary perplexity	5.11E-01	–	–
GPT-3.5	Method 2	total perplexity	6.15E-01	–	–
GPT-4o	Method 2	class perplexity	**6.00E-04**	-0.85	large
GPT-4o	Method 2	summary perplexity	9.49E-01	–	–
GPT-4o	Method 2	total perplexity	7.56E-01	–	–
GPT-3.5	Method 3	class perplexity	4.50E-01	–	–
GPT-3.5	Method 3	summary perplexity	4.98E-01	–	–
GPT-3.5	Method 3	total perplexity	4.86E-01	-	–
GPT-4o	Method 3	class perplexity	**6.59E-08**	-1.93	large
GPT-4o	Method 3	summary perplexity	6.26E-01	–	–
GPT-4o	Method 3	total perplexity	**9.73E-08**	-1.87	large

Table 8.9: Method 2 classification performance with alternative, perplexity-based voting mechanisms.

		F1 Measure				
LLM	**Voting Mechanism**	**Bug**	**Docum.**	**Feature**	**Question**	**Weighted Average**
GPT-3.5	majority	0.90	0.80	0.85	0.83	0.85
GPT-3.5	min class perplexity	0.91	0.82	0.86	0.84	0.86
GPT-4o	majority	0.92	0.91	0.92	0.93	0.92
GPT-4o	min class perplexity	0.92	0.91	0.92	0.93	0.92

For GPT-3.5, perplexity-based voting provides modest but consistent improvements across all classes, yielding a +1% overall improvement in weighted F1-score (0.85 → 0.86). However, GPT-4o

shows no improvement from perplexity-based voting, maintaining identical performance (0.92 weighted F1-score) regardless of voting mechanism. These results demonstrate that perplexity-based confidence estimation can enhance ensemble performance for less capable models. In contrast, advanced models may have already reached performance ceilings where sophisticated voting mechanisms provide diminishing returns.

RQ8.1 - Summary and Implications. The evaluation demonstrates that the proposed methods outperform existing baselines, with Method 3 reaching state-of-the-art performance (0.95 weighted F1-score). More intriguingly, Qwen-2.5 consistently outperforms the larger GPT-3.5 model, suggesting that architectural design trumps parameter count in classification tasks.

The methods reveal unexpected trade-offs and behaviors. Intent extraction provides improvements for weaker models (up to 18% for GPT-3.5) while offering diminishing returns for advanced architectures. The MapReduce approach proves essential for long issues, delivering F1-score gains (2-18 percentage points) at reasonable cost premiums (2-2.5 times more). Most surprisingly, temperature sensitivity varies dramatically by model-method combination, with some configurations showing complete performance collapse under stochastic conditions. The analysis uncovers that perplexity metrics emerge as reliable confidence predictors for ensemble optimization. Given these results, we can answer RQ8.1 as follows:

> ***RQ8.1 Answer:*** *The proposed LLM-based methods outperform existing approaches at reasonable costs, revealing that open weights models can be as performant as larger, proprietary models. The MapReduce decomposition strategy we proposed proved to be consistently more effective than both truncation and direct ingestion of full-length issue descriptions, confirming its value in preserving classification performance for long and complex reports. In addition, perplexity emerged as a reliable confidence predictor, especially for ensemble optimization.*

Based on the insights gathered, our results provide actionable implications for the design and deployment of LLM-based issue report classification systems:

- **Prioritize architecture over size when selecting models for issue classification.** Tool developers and researchers may want to evaluate architectural characteristics rather than solely relying on model size or proprietary APIs. The findings indicate that high-performing open-weight models like Qwen-2.5 can match or even outperform larger commercial LLMs, making them viable for cost-effective and locally deployable solutions. As such, we envision an increased role for open-source LLMs in classification pipelines, especially in contexts where transparency, cost-efficiency, or on-premise deployment are desirable. For instance, future toolchains might include model selection modules that benchmark candidate models on representative issue sets, hence favoring architectural efficiency over scale.

- **Use MapReduce-style summarization to enhance classification robustness on long issue reports.** Developers of AI-assisted issue triaging tools may adopt segmentation and consolidation strategies instead of relying on truncation or full-text ingestion, particularly when classifying verbose reports. Recent literature [13, 302] highlights how LLM performance tends to degrade with increasing context length—an effect we also observed in our evaluation. For GPT-4o, the only model able to fit the long issues in full, classification performance was still lower when ingesting the entire input than when using the MapReduce-based approach. In this sense, future classification frameworks might embed adaptive preprocessing components that automatically segment, reduce, and recompose lengthy issue texts to maximize interpretability while controlling token budgets and inference cost.

- **Leverage perplexity scores to enhance prediction reliability in ensemble settings.** Perplexity can be directly exploited to design smarter ensemble aggregation strategies beyond majority voting. Practitioners can incorporate confidence-aware heuristics or thresholds to filter low-certainty predictions, enabling more robust classifications in production scenarios. Hence, future ensemble-based classifiers might in-

tegrate perplexity-aware scoring layers to prioritize predictions with higher confidence, or automatically trigger abstention and fallback logic for uncertain cases.

- **Tune temperature carefully, especially in non-deterministic deployments.** Given that performance varies substantially across model-method combinations, and that higher temperatures may lead to instability, developers integrating LLMs may calibrate temperature settings per use case and method, or consider temperature-agnostic ensemble methods to mitigate unpredictability. As a consequence, future pipelines might include automated temperature sensitivity analysis during evaluation, or adopt hybrid settings that combine deterministic prompting with selective sampling for ambiguous cases.

8.5.2 *RQ8.2: Classification Consistency*

Table 8.10 presents the results of the classification consistency analyses, measuring raw percentage agreement and Fleiss' Kappa index across five runs for each method-LLM combination.

Table 8.10: Consistency Results

Method	GPT-3.5		Qwen-2.5		GPT-4o	
	Raw Perc. Agreement	Fleiss' Kappa	Raw Perc. Agreement	Fleiss' Kappa	Raw Perc. Agreement	Fleiss' Kappa
Method 1	0.95	0.93	1.00	1.00	0.99	0.99
Method 2	1.00	1.00	1.00	1.00	1.00	0.99
Method 3	0.98	0.97	1.00	1.00	0.99	0.99

All methods achieve near-perfect consistency [114, 334]. Method 2 demonstrates the highest consistency, achieving perfect agreement (1.00) with GPT-3.5 and Qwen-2.5, confirming that ensemble voting effectively reduces non-deterministic variability. Notably, Qwen-2.5 achieves perfect consistency across all methods, while GPT-4o shows minimal variability (0.99 agreement). GPT-3.5 exhibits the most variability, particularly with Method 1 (0.95

raw percentage agreement), but still maintains high consistency levels.

Table 8.11 reveals surprising model-method interactions under different temperature settings. GPT-4o maintains stability across all temperatures and methods, with consistency never dropping below 0.93 Fleiss' Kappa, demonstrating robust architecture design for stochastic sampling.

Table 8.11: Consistency Results for the proposed methods, across different temperature settings.

Method	Temp.	GPT-3.5		Qwen-2.5		GPT-4o	
		Raw Perc. Agreement	Fleiss' Kappa	Raw Perc. Agreement	Fleiss' Kappa	Raw Perc. Agreement	Fleiss' Kappa
Method 1	0.0	0.95	0.93	1.00	1.00	0.99	0.99
	0.3	0.94	0.92	0.95	0.94	0.98	0.97
	0.7	0.91	0.87	0.94	0.92	0.97	0.97
	1.0	0.88	0.82	0.93	0.90	0.96	0.95
Method 2	0.0	1.00	1.00	1.00	1.00	1.00	0.99
	0.3	0.95	0.94	0.96	0.95	0.98	0.98
	0.7	0.93	0.89	0.95	0.94	0.98	0.97
	1.0	0.90	0.86	0.94	0.92	0.97	0.96
Method 3	0.0	0.98	0.97	1.00	1.00	0.99	0.99
	0.3	0.98	0.97	0.84	0.76	0.98	0.97
	0.7	0.97	0.95	0.82	0.73	0.97	0.96
	1.0	0.95	0.93	0.82	0.73	0.95	0.93

As for Qwen-2.5, it maintains higher consistency (0.90-1.00 Kappa) than GPT-3.5 in Methods 1 and 2, across all temperatures. Nonetheless, the results show that Qwen-2.5 exhibits a severe consistency degradation with Method 3 at any temperature above 0.0, mirroring classification performance degradation also observed in Section 8.4.4.1. In that case, Fleiss' Kappa plummets from perfect agreement (1.00) to 0.73-0.76, representing an unanticipated architectural incompatibility between Qwen-2.5's decoder-only structure and few-shot learning under stochastic conditions.

Method 2 consistently outperforms Method 1 in consistency across all temperature-model combinations, confirming that ensemble voting effectively stabilizes predictions even under stochas-

tic sampling. GPT-3.5 shows the most substantial ensemble benefits, with Method 2 maintaining 0.86 Kappa at T=1.0 compared to Method 1's 0.82.

RQ8.2 - Summary and Implications. The results discussed above lead us to answer RQ8.2 as follows:

> ***RQ8.2 Answer:*** *Classification consistency varies by model architecture and method combination, with all approaches achieving near-perfect consistency at temperature=0.0 and exhibiting only slight degradation up to temperature=0.7. While models from the GPT family maintain consistency across all methods and temperatures, Qwen-2.5, unexpectedly, exhibits a critical incompatibility with few-shot learning when temperature > 0. Last, ensemble methods provide consistent stabilization benefits across all LLMs and temperatures.*

These results carry several practical implications for researchers and practitioners developing LLM-based classification tools:

- **Avoid assuming determinism, even at temperature zero.** While temperature=0 yields near-perfect consistency across methods, minor stochastic effects are still present. This suggests that full determinism cannot be guaranteed even in supposedly stable settings. In this sense, a first implication of our results concerns the need to incorporate multiple-run evaluations and statistical agreement metrics, e.g., Fleiss' Kappa, into testing workflows for reproducibility assurance, particularly in safety- or audit-critical environments. This would be particularly relevant for researchers evaluating LLM-based classifiers in benchmark settings, where conclusions drawn from single-run metrics might be misleading. More in general, this observation may possibly apply to other classification tasks beyond issue reporting, and urges the community to treat "temperature=0" not as a guarantee of functional determinism, but rather as a high-consistency approximation with residual variance worth measuring.

- **Favor ensemble-based strategies in production environments requiring stability.** According to the results, Method

2 (ensemble voting) consistently reduces prediction variability across LLMs and temperature configurations, outperforming both zero-shot and few-shot prompting. GPT-3.5, in particular, benefits substantially from ensemble stabilization. This suggests that ensembles may act as a reliability amplifier for less robust models, allowing developers to offset model limitations through methodological design. For instance, developers may adopt ensemble-based approaches as a default when integrating LLMs into triaging or prioritization pipelines where consistent behavior across invocations is essential. At the same time, researchers conducting empirical studies might include ensemble baselines in their evaluations to isolate gains derived from aggregation mechanisms rather than LLM capabilities alone.

- **Validate compatibility between LLMs and prompting strategies before deployment.** The results reveal an architectural incompatibility between Qwen-2.5 and few-shot learning under stochastic sampling: performance and consistency both degrade sharply at $T > 0$. As such, practitioners adopting open-weight models may need to test specific model–method pairings across temperature levels before deployment, rather than assuming strategy portability across LLM families. Similarly, these findings suggest that prompt engineering guidelines may need to be architecture-specific. Researchers working on prompting techniques may benefit from reporting not only average performance but also interaction effects between prompting strategies, temperatures, and model architectures, especially for LLMs not belonging to the GPT family.

- **Include consistency as an evaluation criterion in LLM pipeline design.** While performance metrics remain the dominant focus, consistency may be considered equally important in high-reliability settings. As such, classification pipelines might benefit from consistency-focused regression testing, where repeated invocations with fixed inputs are checked for agreement, to detect fragile or temperature-sensitive configurations early in development and testing cycles. Again, these

findings possibly inform the way researchers evaluate LLM-based classification systems: in addition to standard performance metrics, future work might include consistency measurements across temperature sweeps and multiple seeds, especially when targeting reproducible behavior in developer-facing tools or infrastructure components.

8.5.3 *RQ8.3: Error Patterns and Classification Dynamics*

This section shows the results for the analyses described in Section 8.4.4.2.

8.5.3.1 *Class-level Analysis of Misclassifications*

The results of the quantitative class-level analysis of misclassification are reported in Figure 8.14.

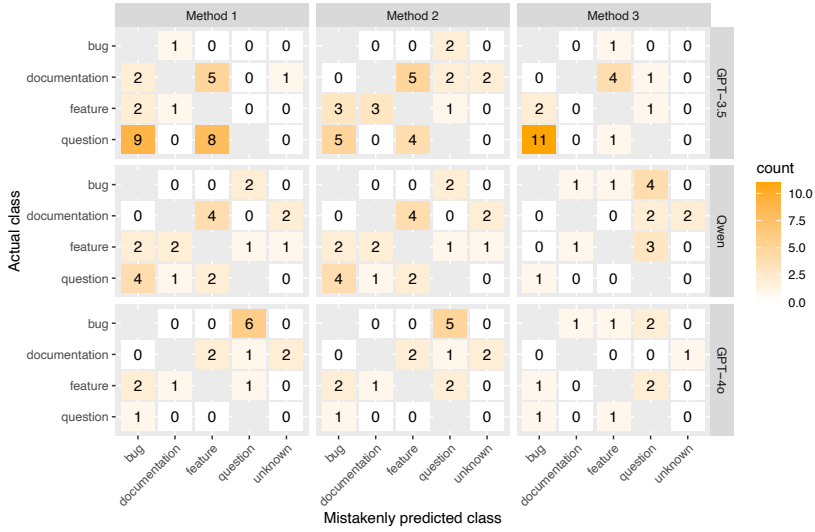


Figure 8.14: Confusion Matrices of Issue Misclassifications

The figure presents confusion matrices illustrating the distribution of the incorrect predictions made by the three considered methods using GPT-3.5, Qwen-2.5, and GPT-4o. Each sub-matrix corresponds to a specific method and LLM variant and shows how many issue reports from an actual class (y-axis) were misclassified as different categories (x-axis). The color intensity of

each cell represents the frequency of misclassifications, with darker shades indicating higher misclassification counts.

The analysis of classification errors reveals distinct patterns in how LLMs interpret and apply the hierarchical decision rules specified in the classification prompts. A first notable trend in GPT-3.5 is the frequency of misclassification of *question* issues across all proposed methods. In Method 1, 17 questions are misclassified as bugs (9) or features (8). In Method 3, the overall number of misclassifications of *question* issues is 12, with 11 of these cases involving confusion with the *bug* label. This suggests that the few-shot examples provided in Method 3 are effective in reducing confusion between *question* and *feature* issues. The second most challenging class for GPT-3.5 is *documentation*, which is frequently confused with the *feature* class across all proposed methods.

Qwen-2.5, in contrast, exhibits a more balanced distribution of misclassifications across classes. Still, the confusion between *bug* and *question* remains one of the most recurrent error patterns. In both Method 1 and Method 2, Qwen-2.5 misclassifies 4 *question* issues as *bugs*, while 2 bug reports are incorrectly labeled as questions. Method 3 proves effective in improving classification performance on *question* issues, reducing misclassifications to just one case, which is, again, incorrectly labeled as a bug. However, this improvement comes at the cost of increased misclassification of bug reports: 4 of them are erroneously classified as questions. This trade-off suggests that while few-shot learning enhances Qwen-2.5's ability to recognize interrogative intent, it may also lead to overgeneralization in distinguishing between technical problem descriptions and user inquiries.

GPT-4o demonstrates the highest overall classification performance among the evaluated models. Its most prominent class confusion in Method 1 and Method 2 involves the misclassification of bug reports as questions, with 6 and 5 such cases observed in each method, respectively. Method 3 partially alleviates this issue, reducing the number of bug-to-question misclassifications to just 2. Nonetheless, it introduces new errors, misclassifying one bug report as documentation and another as a feature. These findings suggest that while few-shot learning improves GPT-4o's ability to distinguish between interrogative and technical content, it

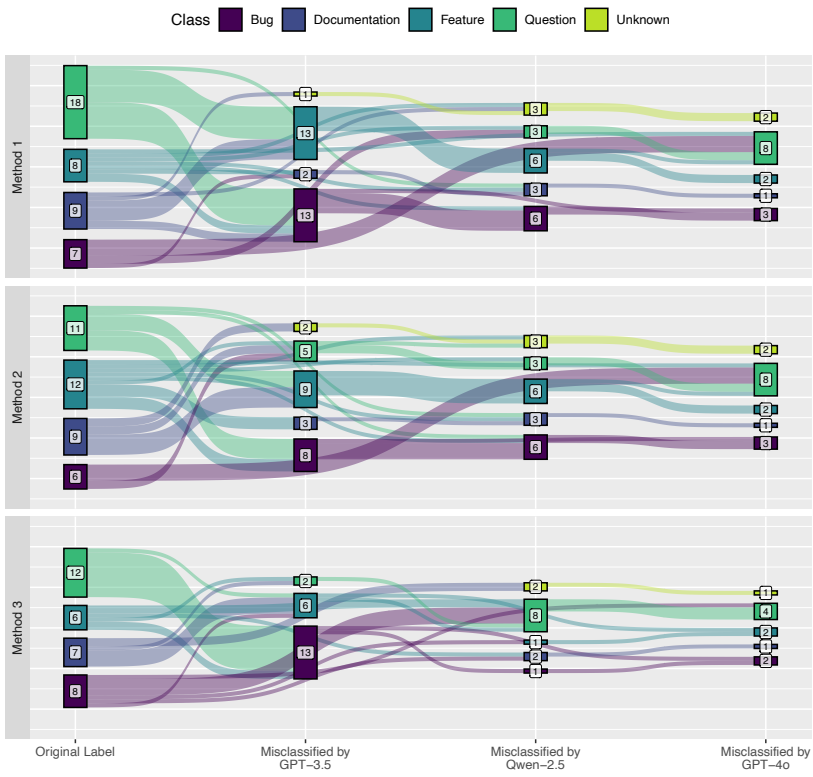


Figure 8.15: Sankey Diagrams of Misclassification Flows Across LLMs and Methods.

may also introduce subtle shifts in classification boundaries that affect precision in edge cases.

To complement the static view offered by the confusion matrices, we conducted a dynamic flow analysis of misclassifications using Sankey diagrams (Figure 8.15). This visualization captures how misclassified issue classes evolve across the three LLMs for each method, offering a more granular understanding of classification behavior. The diagram confirms a clear performance hierarchy: GPT-3.5 introduces the highest number of misclassifications, Qwen-2.5 partially mitigates these errors, and GPT-4o achieves the most consistent improvements. Better-performing models tend to either correct prior misclassifications or confirm them, while transformations into entirely new errors are relatively rare.

For example, in Method 1, the most common incorrect predictions made by GPT-3.5 were *feature* and *bug*, with 13 misclassifications each. Qwen-2.5 corrected more than half of these errors, reducing the count to 6 confirmed misclassifications. GPT-4o further refined the predictions, resolving approximately half of Qwen-2.5's remaining errors and bringing the number of retained misclassifications down to just 2. A similar trend also emerges for Method 2 and Method 3.

While stronger models generally correct misclassifications introduced by weaker ones or confirm them when unresolved, there are notable exceptions where errors emerge for the first time in more capable models. For instance, in Method 1 and Method 2, GPT-4o introduces 4 new misclassifications of bug reports as questions, errors not present in earlier GPT-3.5 and Qwen-2.5. Similarly, in Method 3, Qwen-2.5 newly misclassifies four bugs as questions, highlighting that improvements in overall performance do not entirely eliminate the emergence of novel classification mistakes.

8.5.3.2 *Misclassification Causes*

For what concerns the qualitative analysis of misclassification causes, a set of recurring misclassification drivers emerged from the process described in Section 8.4.4.2. These drivers were subsequently formalized into the taxonomy presented below, which captures the most salient sources of classification disagreement observed. During the independent annotation phase, the inter-rater agreement, measured using Cohen's Kappa, reached a substantial level of agreement, with a Kappa score of **0.92**, indicating that the taxonomy is well-defined and reliably applicable. As a result of the analysis process, five primary causes of misclassifications were identified, namely *class overlap*, *ambiguity*, *complexity*, *implicit intent*, and *other*, detailed as follows.

CLASS OVERLAP. Some issues exhibit characteristics that align with multiple classes, making it more challenging for classification approaches to understand the primary intent of the issue and to assign a single definitive label. For example, consider an issue asking if a certain feature is available and then stating that, if not

already available, that feature should be implemented. The issue has a primary questioning intent and should thus be classified as a *question*, but also exhibits overlaps with the *feature* label, as it is (even though conditionally) proposing a new feature as well. Such cases blur the boundaries between classes, increasing the likelihood of misclassification. One such case is reported in Example 1. Each example issue report in our paper is presented in a structured format. The top bar contains the issue title, followed by the unique issue identifier from the dataset in parentheses. Below this, the main body of the issue report is displayed, preserving the original text.

Example Issue Report 1: Does MPRIS expose Genre (#1260904378)

Does MPRIS expose Genre? If so it'd be nice to be able to filter on genre,

AMBIGUITY. Ambiguous issue reports lack sufficient detail, contain vague or misleading language. Consider, for example, the issue report in Example 2.

Example Issue Report 2: Cannot declare class GoogleClient (#1122657441)

Hi, I want to report a bug.
since it is being used by many people, it creates conflicts when two different organizations use the same thing.
For example: I have created a plugin that uses your Social Login integration method and a separate plugin that uses your other services.
Now, the conflict occurs when we try to attempt the class in two separate plugins. Cannot declare class GoogleClient from `/google-client/vendor/google/apiclient/src/Client.php` on line 50
Is there any way to counter this?

Despite the intent of the user creating the issue report is to inquire about ways to address the naming conflict they are experiencing, the issue report starts, incorrectly, with the declaration "*I want to report a bug*". This is also sometimes due to the use of incorrect issue templates, as shown in Example 3, where a user reports a bug, but leaves the default template for asking questions on top of the issue report.

Example Issue Report 3: Multiple urls sharing not working on android (#1120539238)

Ask your Question
sharing multiple audio files (array of URLs) works fine on ios but on android, its only shares the first URL not all the URLs"

Ambiguity increases the likelihood of incorrect predictions, particularly with simpler classification models that rely on surface-level cues rather than deeper contextual understanding.

COMPLEXITY. Some misclassified issues are inherently more complex, exhibiting long, technical descriptions (including snippets of code or execution logs), and possibly the involvement of multiple concerns into a single issue. Such cases may require reasoning beyond what the model is capable of handling, leading to errors. Complexity-related misclassifications often involve confusion between `question` and `bug`, as some issues start by describing unintended behavior or errors the users are experiencing, and then proceed to ask a question on how to properly use the software. As an example, consider the issue in Example 4.

Example Issue Report 4: Cannot add a new users schema extension through "UserSchema.json" (#1372841808)

Hello, I got back a 400 ERROR while sending a POST request to "/Users" with the following request body

```
{   "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:User",
     "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"],
    "externalId": "external",
    "userName": "name@name.com",
    "meta": {"resourceType": "User"},}
```

this is because this schema "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User" was not defined in the "UserSchema.json" file. So, once I added it manually to the database table "SCIMSchemaExtension" it worked just fine. So, can you please tell me how can I fix this? Thanks a lot

The above issue is classified as a question, as the primary intent is to acquire feedback on how to fix or prevent the described error. However, the complexity and technical details in the first part of

the issue body lead models to concentrate their attention on the first part, and incorrectly classify the issue as a bug.

IMPLICIT INTENT. Some issue reports contain subtle nuances that may be more difficult for the model to interpret, even though the issue description itself is neither particularly complex nor ambiguous. For these issues, correct classification depends on the model's ability to *read between the lines* and capture implicit intent from subtle contextual clues. To better explain this root cause of misclassification, a representative case is shown in Example 5.

Example Issue Report 5: OTP produces no meaningful message when GTFS input is corrupt (#29698056)

Reported on mailing list: https://groups.google.com/d/msg/opentripplanner-users/otLwIZQOTpQ/9LKYdz4IEzYJ
Corrupted ZIP file just yields cryptic "no agencies" error.

Even though the report does not explicitly request a change to an existing feature, it implicitly suggests an evolutionary maintenance action, i.e., to improve an existing feature by providing a more informative error message when incurring in invalid inputs. Nonetheless, the models may erroneously classify the issue as a bug, since it explicitly describes an undesired behavior rather than directly proposing an enhancement.

OTHER. This category was assigned to misclassified issues that do not fit any of the other categories and appear to be misclassified for no clear reason. These instances could not be attributed to a specific cause, possibly indicating inherent model limitations. For instance, consider the issue report in Example 6. The issue re-

Example Issue Report 6: How to enable "auto format" and "auto insert row" functionality? (#1143694432)

Tab doesn't do anything for me by default. I couldn't find any references to such settings in this repo, the only places where the word auto comes up are README files and doc strings.

port's main intent is to inquire about how to enable a certain func-

tionality, and thus it is labeled as a *question* in the dataset. However, GPT-3.5 with Method 1 labels the issue as a *feature* request, despite the absence of an explicit suggestion for new functionality.

Table 8.12 reports the distribution of misclassification causes across models and LLMs. Each row of the table corresponds to a specific misclassification cause we defined in the proposed taxonomy, while columns indicate how many times a given cause was observed for a given method and LLM.

Table 8.12: Distribution of Misclassification Causes Across Methods and LLMs. For the sake of conciseness, Method 1, Method 2, and Method 3 are abbreviated as M1, M2, and M3, respectively.

	GPT-3.5			Qwen-2.5			GPT-4o		
	M1	**M2**	**M3**	**M1**	**M2**	**M3**	**M1**	**M2**	**M3**
Class overlap	3	3	0	2	2	1	2	2	0
Ambiguity	4	5	3	5	5	4	4	4	3
Complexity	4	1	6	1	1	2	0	0	0
Implicit intent	5	6	5	4	4	3	3	3	2
Other	13	12	7	9	9	5	7	7	5
Total	**29**	**27**	**21**	**21**	**21**	**15**	**16**	**16**	**10**

Among the identified causes, *other*, which refers to errors without a clear explanation, appears to be the most frequent, accounting for approximately half of the misclassifications across all models and methods. Ambiguity emerges as a common misclassification cause, affecting all the considered LLMs in a comparable way. This indicates that insufficient or vague issue descriptions remain a persistent challenge, regardless of model size or architecture. *Complexity*, on the other hand, disproportionately affects GPT-3.5, particularly in Method 3, where six such cases are recorded. This suggests that GPT-3.5 struggles more with long, technical issue reports involving multiple concerns, while GPT-4o and Qwen-2.5 appear more resilient to higher input complexity. *Class overlap* is another notable source of misclassification, especially in Method 1 and Method 2 across all models. Interestingly, Method 3, lever-

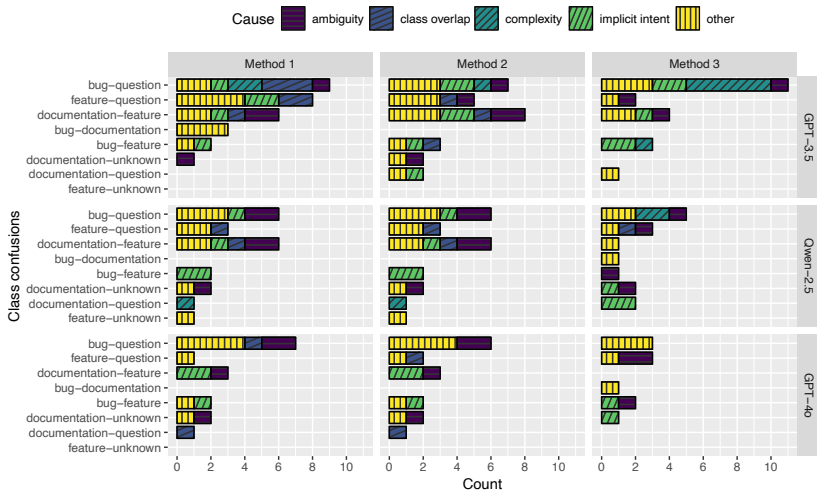


Figure 8.16: Combined analysis of classification confusions and misclassification causes across methods and LLMs.

aging few-shot examples, is effective in mitigating this issue, eliminating class overlap errors entirely for GPT-3.5 and GPT-4o, and reducing them to just one case for Qwen-2.5. This highlights the potential of example-based prompting in helping models disambiguate overlapping intents. Finally, *Implicit intent* remains a challenge across all models, though it is more prevalent in GPT-3.5. GPT-4o, while not immune, shows a marked reduction of approximately 50% in such errors across all methods. Qwen-2.5 shows intermediate behavior, with fewer such errors than GPT-3.5 but slightly more than GPT-4o, suggesting it captures implicit signals reasonably well, though not yet at the level of GPT-4o.

8.5.3.3 *Combined analysis*

In this section, we combine the quantitative distribution of class-level misclassification with the qualitative taxonomy of their underlying causes, highlighting how different LLMs and classification methods interact with specific challenges. The bar chart in Figure 8.16 provides a consolidated view of how different misclassification causes are distributed across models and methods and specific class confusions, offering a deeper understanding of the interplay between classification strategies and LLM behavior.

The most frequent confusion, i.e., the one between *question* and *bug*, is strongly associated with causes like *implicit intent* and *complexity*, particularly for GPT-3.5 and Qwen-2.5 in Methods 1 and 2. These models exhibit a tendency to misinterpret technically detailed inquiries as bug reports, especially when the user's intent is not explicitly stated. Conversely, GPT-4o, while more robust to *complexity*, still shows a tendency to misclassify *bug* reports as *questions*, especially in Methods 1 and 2, where clarification-seeking language is present. This reflects a heightened sensitivity to conversational cues, which can override procedural context. The confusion between *documentation* and *feature* is another recurring pattern, often driven by *ambiguity* and *class overlap*, particularly in Qwen-2.5 and GPT-3.5. Method 3 generally reduces these errors, especially those due to *ambiguity*, but introduces new ones, such as *bug* to *question* misclassifications, likely due to the influence of few-shot examples that overemphasize inquiry patterns. Overall, this analysis confirms that, while Method 3 enhances robustness against certain misclassification causes, particularly *ambiguity* and *class overlap*, it also shifts the error landscape by amplifying sensitivity to *implicit intent*. Each model exhibits distinct interpretive biases: GPT-3.5 struggles with complexity, Qwen with class overlap, and GPT-4o with subtle intent cues. These complementary strengths and weaknesses suggest promising opportunities for hybrid strategies that integrate multiple methods or models, potentially through ensemble architectures that adaptively leverage their respective advantages.

8.5.3.4 *Cross-Model and Method Comparative Analysis*

Comparing the three classification methods across all LLMs reveals complex performance dynamics, with error patterns that shed light on the efficacy of the methods, model behavior, and their interactions with varying informative characteristics of the reports. In the following, in order to systematically examine performance dynamics, we analyze error patterns across six key dimensions, which together allow us to define a recovery strategy for misclassifications. Misclassifications are defined as cases in which at least one LLM is incorrect.

SYSTEMIC LIMITATIONS OF ENSEMBLE LEARNING The behavior of Method 2 demonstrates how ensemble learning is limited when base models have similar systematic biases, or when a single LLM is used as the base model for all instances in the ensemble. Of the 43 cases where Method 1 presents classification errors, Method 2 consolidates the errors in 34 instances rather than correcting them. This phenomenon suggests that majority voting is unable to account for the systematic biases present in the models.

Furthermore, Method 2 shows how ensemble aggregation can occasionally increase model uncertainties by introducing five new errors in cases that Method 1 correctly classified.

THE PARADOX OF CONTEXTUAL FEW-SHOT LEARNING There is a complex distribution of errors in Method 3. Method 3 fixes 22 of Method 1's errors where at least one LLM generates an incorrect classification, and it significantly reduces the number of cases where all LLMs fail – from 8 in Method 1 to just 3 in Method 3. At the same time, however, it presents 12 new misclassification cases that Method 1 had correctly identified for every LLM. All this has a twofold implication. Few-shot learning helps reduce systematic biases and recover most edge cases, but it also introduces new types of errors, meaning that it optimizes different aspects of the classification task, potentially prioritizing semantic understanding over syntactic pattern recognition. Another interesting pattern that emerges from the analysis is the relationship between the effectiveness of few-shot examples and the informative context of issue reports. Contrary to expectations, Method 3 performs better on longer content (the mean token length of errors increases from 200.2 in Method 1 to 268.2 in Method 3), suggesting that few-shot learning is more effective when sufficient context exists to disambiguate the provided examples. The most noteworthy discovery is that, depending on the LLM used, Method 3 has a different impact. While Qwen-2.5 shows significant improvements in question classification (from 7 to 1 errors) at the expense of more misclassified bugs (from 2 to 7 errors), GPT-3.5 shows an increase in question errors (from 12 to 17 errors), while other classes stay stable. So, given that every single architecture responds differently to contextual priming, few-shot learning leads to model-specific trade-offs rather than general gains.

INFORMATIONAL QUALITY AND CLASSIFICATION ROBUSTNESS
Dimensional analysis shows how the length of issue reports in terms of tokens influences classification performance in non-linear ways. In Method 1, 41 out of 43 problematic cases occur below 500 tokens, with 23 cases below 151 tokens and a clear concentration in minimal-information ranges: 9 cases (0-50 tokens), 8 cases (51-90 tokens), and 6 cases (91-150 tokens). In Method 3, a similar trend occurs, with 22 out of 33 misclassified issues containing fewer than 500 tokens and 14 cases containing fewer than 151 tokens. When examining specific LLMs and issues that do not contain technical information (no stack traces or logs), we found that the previous trends still hold for models in the GPT family, but not for Qwen-2.5 above 150 tokens. Indeed, with Qwen-2.5, Method 1 proves to be more effective than Method 3 on such issues, reducing errors up to 60%. In addition, there is a critical distinction between partial and complete method failures. Indeed, of the 8 cases where all three LLMs fail in Method 1, six lack descriptions, logs, or stack traces entirely, with six cases having fewer than 90 tokens and two cases ranging between 242 and 340 tokens. This distribution indicates that minimal informational content creates problems across all models. Furthermore, two cases exhibit complete failure across all methods and models (IDs 29698056, 1374258660). Despite their different informational richness (the former comprises 64 tokens with no artifacts, the latter has 242 tokens with a detailed description), both cases share a common characteristic: they describe technical failures without providing sufficient context. Therefore, even though minimal content (under 90 tokens, with no technical artifacts) leads to systematic errors, the presence of structured information does not guarantee robust classification. The two complete failure cases cast doubt on the notion that more information inevitably leads to better classification performance by showing that even thorough technical descriptions can remain unclear in the absence of obvious communicative markers.

ASYMMETRIC ERROR DISTRIBUTION BY CLASS The "question" class – representing ≈25% of the dataset (47/187 cases) – accounts for 38% of the individual errors in Method 1 and leads to different results across models in Method 3. GPT-3.5 shows targeted

improvement, increasing correct question classifications from 26 to 33 (+7) while maintaining stability in other classes. The most notable behavior is shown by Qwen-2.5, which increases correct question classifications from 40 to 46 but at a considerable cost: bug classification declines from 51 to 46 correct cases. This trade-off implies that rather than improving overall performance, few-shot learning redistributes Qwen-2.5's interpretative capacity between classes. GPT-4o presents greater stability, maintaining virtually unchanged question performance (45 correct in both methods) while showing modest improvement in other classes. This plateau indicates that GPT-4o has reached the maximum of its capabilities in interpreting questions and that providing more context cannot further improve these performances, despite the model being capable of learning to recognize technical elements.

CONVERGENCE AND DIVERGENCE IN CROSS-MODEL ERRORS In the 17 cases of misclassified questions that GPT-4o classifies correctly, Qwen-2.5 and GPT-3.5 show highly correlated errors, but this convergence is class-specific. When it comes to identifying bugs and features, each model makes different errors, indicating that each one 'reasons' differently about these practical aspects. A fascinating aspect is that when Qwen-2.5 diverges from GPT-3.5, it typically converges toward GPT-4o's correct classification. These results indicate that some models are better than others at specific tasks. One could therefore create an ensemble system where each model has more or less influence on the final decision depending on how accurate it has been in the past for that specific type of classification. The fact that Qwen-2.5 and GPT-3.5 systematically make the same errors on questions suggests that they have similar limitations in their design or in the data they were trained on – limitations that GPT-4o does not have.

UNCERTAINTY CALIBRATION AND PERPLEXITY Different patterns of uncertainty calibration are revealed by analyzing perplexity metrics (available for GPT-3.5 and GPT-4o). Indeed, even for correct classifications, GPT-4o exhibits greater perplexity when the issue report is ambiguous, demonstrating its "epistemic honesty" in identifying semantic complexity. On the other hand, even for incorrect classifications, GPT-3.5 often reports high confidence

(perplexity of 1.0), especially for questions that are misclassified as features or bugs. The most variable perplexity is shown by Method 3, where it occasionally even outperforms 1.97 for precise classifications for both models. This result implies that few-shot examples may actually increase interpretive complexity rather than always lowering perceived uncertainty.

CROSS-METHOD AND LLM RECOVERY STRATEGY Based on the analysis presented above, we propose a two-tier recovery strategy optimized for different privacy constraints.

Regarding privacy-sensitive scenarios, we propose deploying Qwen-2.5 locally with Method 3 as the default. When issue reports contain a sufficient amount of information (above 150 tokens) and consist only of text descriptions (without logs/stack traces), switch to Method 1, as Method 3 introduces errors in these cases. For privacy-unconstrained scenarios, we suggest starting with Qwen-2.5 using Method 3. Escalate to GPT-4o only for those issues below 500 tokens that are classified by Qwen as *questions*, where the analysis shows Qwen-2.5 systematically fails but GPT-4o succeeds in 6 critical cases. This procedure targets the specific error pattern where Qwen-2.5 converges with GPT-3.5 mistakes, while GPT-4o maintains effectiveness. This approach processes 70% of cases with the cheaper Qwen-2.5 model, while reserving expensive GPT-4o calls for specific classification scenarios where the performance gain justifies the cost.

RQ8.3 - Summary and Implications. On the basis of the results discussed above, we can address RQ8.3 as follows:

> ***RQ8.3 Answer:*** *Error patterns reveal vulnerabilities concentrated in question-bug distinctions, with ensemble methods consolidating rather than correcting mistakes. Few-shot learning creates model-specific performance trade-offs rather than universal gains, while content density below 500 tokens and lack of technical details are consistently associated with failures. These dynamics suggest complementary model architectures that could be strategically combined rather than naively aggregated.*

The results of RQ8.3 offer several implications for the design, deployment, and evaluation of LLM-based issue classification systems, and open concrete research directions:

- **Use intent extraction selectively in privacy-critical or data-minimal settings.** Method 1 performs effectively when applied to issue reports with long, well-structured natural language descriptions and few technical artifacts (e.g., no stack traces or logs). This makes it particularly suitable for privacy-sensitive contexts, such as enterprise settings or anonymized datasets. Practitioners may design pipelines that default to intent extraction for such inputs (especially with open-weight models like Qwen-2.5), while falling back to more expressive strategies when input complexity increases. At the same time, researchers may possibly further explore how to predict when intent extraction is sufficient, and how to train models specialized for intent-only representations. For instance, this may represent an opportunity to define lightweight, privacy-preserving LLM-based classifiers that can operate reliably under data minimization constraints, which is a critical requirement in regulated domains.

- **Do not assume ensemble learning improves classification performance.** While Method 2 improves consistency (as shown in RQ8.2), it often fails to correct misclassifications and may even consolidate incorrect predictions. This finding warns against assuming that ensembles offer performance improvements "for free". For developers, this suggests that ensemble methods should be evaluated on both consistency and error profiles before being integrated into production. As for researchers, this highlights an opportunity to explore error-aware or adaptive ensemble mechanisms that explicitly model disagreement, confidence, or prediction provenance, hence allowing the system to downweight or discard unreliable components rather than aggregating them blindly. This may possibly be envisioned in terms of dynamic weighting schemes, confidence-informed voting, or even gating mechanisms that route decisions based on task complexity or known model limitations.

- **Use explanatory prompting (few-shot) to mitigate class overlap, with caution.** Method 3 was effective in reducing classification errors tied to semantic overlap (e.g., between support and enhancement), making it well-suited for disambiguating borderline cases. However, it also introduces trade-offs, including increased output complexity and model-specific instability: this is particularly true for GPT-3.5, where explanatory prompting degrades consistency and sometimes shifts the error profile toward harder-to-debug failures. On the basis of these insights, practitioners might therefore use few-shot prompting selectively, exploiting it only when inputs are likely to involve ambiguous class boundaries or when baseline methods show low confidence. This could be operationalized through pre-deployment profiling of class confusion patterns or via confidence-based switching during runtime. At the same time, researchers may possibly explore adaptive or context-aware prompting strategies that dynamically adjust explanation length, content, or example granularity based on predicted classification difficulty or proximity to known error-prone class boundaries. These approaches might possibly maximize the benefits of explanation without triggering the fragility observed in certain model-method configurations.

- **Design hybrid pipelines that adapt to model-specific strengths and biases.** The analysis demonstrates that both models and prompting strategies exhibit systematic, complementary strengths and weaknesses. GPT-3.5 tends to underperform on abstract or multi-label issues, Qwen-2.5 is more prone to class overlap errors, and GPT-4o can occasionally miss subtle intent cues. Likewise, prompting strategies such as intent extraction or explanatory few-shot learning show varying effectiveness depending on input characteristics like issue length, technical complexity, or ambiguity. Overall, these findings confirm that no single model-method configuration performs optimally across all contexts. Above all, these insights somehow motivate the need for further research into the design of learned orchestration mechanisms that go beyond static ensembling. One potential future direction is indeed that

of designing input-aware controller models that predict the most effective model-method combination for a given issue based on features such as syntax, length, or prior error patterns. Another direction is the use of meta-learning or reinforcement-based allocation strategies that adaptively assign classification configurations over time, learning from deployment feedback to improve selection policies. In addition, researchers may investigate confidence-aware ensembling architectures that selectively trust or suppress predictions using runtime signals such as perplexity, explanation plausibility, or prediction disagreement.

8.5.4 *Lessons Learned*

We can now integrate the RQ results into a unified view that illustrates the behaviors of the three methods with respect to the operational dimensions considered and how these directly impact real-world applicability.

From the perspective of operational characteristics, the proposed methods first have different usage costs. Method 1 consumes fewer tokens and, consequently, has lower computational costs, making it particularly suitable for scenarios characterized by high volumes of issues to classify. Method 2 has a token cost that is 5 times higher on average (therefore also at the computational level), due to ensemble aggregation, but also has more predictable resource consumption patterns and results. Method 3 requires many fewer input tokens compared to Method 2, but still more than Method 1 due to few-shot examples. This introduces cost implications that scale with the volume of issues to classify. Regarding the handling of long texts, all methods benefit from the MapReduce approach. However, Method 3 shows the most marked performance improvements (up to 18 percentage points) compared to baselines, suggesting that example-based reasoning works particularly well when the report is rich in useful contextual information.

Our analysis shows that each method handles prediction confidence differently in terms of consistency and uncertainty calibration. Method 1 provides the most reliable perplexity signals for quantifying uncertainty, allowing for the identification of issue reports that need human supervision in production settings

through adaptive confidence thresholds. However, by masking the uncertainties of individual predictions, Method 2 achieves the highest consistency through aggregation, making it challenging to identify low-confidence cases that would need practitioner screening. Method 3 produces more variable perplexity signals, making it impossible to calibrate trustworthy confidence thresholds.

Regarding error patterns and temperature sensitivity, each method shows distinct error modes and robustness characteristics. The results demonstrate that the impact of temperature on classification performance (but generally on the task to be performed) is not predictable a priori and must be determined empirically for each method-model combination. There is no general rule that allows anticipating whether a particular classification task requires more deterministic sampling (temperature=0) or a certain degree of stochasticity, and this unpredictability represents an important aspect to consider in designing LLM-based applications.

Indeed, Method 1 has virtually stable behavior, regardless of the temperature value. Method 2 consolidates rather than corrects systematic biases, particularly when distinguishing question-bug, but has the most stable behavior as temperature varies. Method 3 has variable behavior with respect to temperature that is model-specific, and performance degrades with decoder-only architectures under non-deterministic sampling as temperature increases.

In other words, the findings reveal that operational requirements should guide method selection rather than accuracy metrics alone. Unlike traditional ML deployment, where model selection focuses primarily on predictive performance, LLM-based issue classification requires balancing different operational constraints: cost budget, consistency requirements, uncertainty quantification needs, and infrastructure capabilities. This result represents a significant shift toward "operationally aware" AI deployment in software engineering and contributes to the broader understanding of how LLMs can be strategically integrated into classification workflows, providing a framework for constraint-driven method selection in automated issue triaging systems.

Table 8.13 synthesizes the method-specific insights across RQ8.1–RQ8.3.

Table 8.13: Interpretive framework of method-specific insights across RQ8.1–RQ8.3.

Method 1	Method 2	Method 3
RQ8.1: Method Effectiveness and Operational Characteristics		
• Abstracting to "intent" boosts weaker or older LLMs and reduces sensitivity to noisy technical details. • Use class-level perplexity as a built-in confidence signal to gate low-certainty predictions or trigger review. • It is the best cost–accuracy workhorse.	• Ensembles boost stability more than accuracy; returns diminish as base models improve. • Majority voting hardens shared biases; replace or augment it with confidence-aware rules. • Budget for the aggregation cost upfront; deploy when SLAs prioritize stability and traceability over marginal accuracy gains.	• Examples with explanations move the decision boundary from pattern matching to rule-like reasoning—most useful where classes overlap. • Effectiveness depends on architectural fit; keep decoding deterministic and validate on the target LLM before rollout. • Accept a higher token footprint as the price of sharper class discrimination; reserve it for critical or ambiguous cases.
RQ8.2: Classification Consistency		
• Constraining inputs to an "intent summary" improves agreement by standardizing the decision surface; though model-dependent variability persists. • Treat temperature as a consistency budget: minimal for production, experimental only with monitoring.	• Aggregation suppresses stochastic drift; think of it as consistency insurance.	• With GPT-like models, explanations are compatible with high consistency at low temperature; with decoder-only models, explanations require stricter decoding to remain stable.
RQ8.3: Error Patterns and Classification Dynamics		
• Symptom-vs-intent tension drives bug-question confusions; intent extraction helps but cannot fix underspecified or very short reports.	• Ensembles herd errors when members share the same blind spot; inject heterogeneity (e.g., model diversity) and use confidence-weighted voting to counteract.	• Explanations reduce ambiguity and class overlap but can tilt decisions toward inquiry cues; monitor the bug-question boundary and calibrate exemplars accordingly. • Performance is context-elastic: it thrives with sufficient signal (richer text), but on sparse inputs the trade-offs become model-specific.

8.6 THREATS TO VALIDITY

In this section, we discuss threats that could have affected the results and their generalizability, according to the guidelines defined by Wohlin et al.[319].

A possible threat to the *internal validity* of our findings lies in the manual verification of the issue report labels in the training and test sets, which may have been influenced by annotators' bias and subjectivity of the annotators, as well as human errors. To mitigate this threat, two of the authors of this work independently annotated the issue reports, following the same procedure. After the labeling phase, any disagreements between the annotators were discussed until reaching a consensus, and any remaining ambiguous cases were discarded. This annotation approach mirrors the procedure used by Colavito et al. when preparing the original dataset [63] that we extended. Moreover, we observed very high agreement between the two independent annotators, suggesting that the annotations are reliable. Despite these precautions, the manual annotation process may still introduce bias or human errors, though we believe their impact to be minimal. An additional threat to internal validity arises from the choice of hyperparameters in the experiments. For the RoBERTa and SetFit baseline models, we used the same hyperparameters and configurations reported in prior studies on the NLBSE'23 dataset, which may not necessarily be optimal for our current setting. For the LLM-based methods, we conducted a preliminary study (described in Section 8.4.2) to determine the best model configuration, favoring consistency in the classification results. While we aimed to select parameters that maximize performance, it is possible that different hyperparameter configurations could lead to better or worse results, potentially affecting the reliability of our conclusions. Another potential limitation comes from the inability to verify whether the issue reports in the test set were included in the training data for GPT-3.5, GPT-4o, and Qwen 2.5. This could have affected the validity of our results, with LLMs being influenced by pre-existing data, rather than truly generalizing to unseen examples. As a result, the observed results may not fully reflect the real-world performance of the models on new, unseen issue reports.

A possible threat to *external validity*, affecting the generalizability of our findings, arises from the fact that our results are based on a relatively small dataset of GitHub issue reports. While we used stratified sampling to ensure proper representation of all classes of issue reports, we acknowledge that the generalizabil-

ity of our proposals to different types of issue is uncertain, and the results may not fully extend to other issue tracking systems, to specific domains or languages, or to issue reports with different characteristics. Further replications involving a broader set of diverse issue reports might be required to confirm the generalizability of the findings.

8.7 CONCLUSIONS

This chapter introduced two conceptual approaches to issue classification, implemented through three LLM-based methods. The first approach focuses on understanding user intent through "Intent Extraction and Classification" (Method 1) and its ensemble variant (Method 2). The second approach uses explicit reasoning with examples in the "Explained Few-Shot Learning" method (Method 3). Our evaluation demonstrates that these methods surpass existing baselines, with GPT-4o achieving the highest weighted F1-score with Method 3. Beyond performance improvements, we discovered that ensemble voting consolidates errors rather than correcting them, and few-shot learning redistributes rather than reduces mistakes. We also revealed unexpected architectural benefits, with Qwen-2.5 exceeding GPT-3.5's performance despite fewer parameters. Our MapReduce approach delivers advantages for long reports, while classification consistency reaches near-perfect levels across different temperature settings. Most importantly, we established a framework for understanding misclassification patterns and identified complementary strengths across model architectures that enable strategic deployment optimization.

Future research could focus on expanding the dataset to include a more diverse array of issue reports to improve the generalizability of the findings. The architectural insights and behavioral patterns we discovered suggest promising directions for more sophisticated classification approaches. Additionally, exploring the integration of other LLMs may offer insights into optimizing classification performance. We also aim to refine our MapReduce-based approach to enhance scalability and efficiency, optimizing processing time and adaptive chunking strategies for improved classification performance of detailed issue reports in large-scale projects. Finally, incorporating user feedback mechanisms to re-

fine classification models continuously could result in more adaptive and robust systems. Another promising direction for future work involves reformulating the issue classification task as a multi-label problem. Our empirical findings reveal frequent intent overlap between categories, particularly among semantically related labels such as "support" and "enhancement". While we continue to view single-label classification as both meaningful and practically relevant, especially since many triaging systems in use today rely on assigning a single primary label for downstream processes such as routing, prioritization, or task assignment, the observed ambiguity in issue semantics suggests that multi-label formulations could better reflect the real-world complexity of issue reports. Future research might explore suitable annotation guidelines, architectural adaptations, and evaluation protocols for multi-label classification in this context, while carefully assessing its benefits and trade-offs in terms of model performance, interpretability, and integration into existing triaging workflows. Moreover, future work could explore prompt sensitivity analysis by systematically evaluating the impact of semantically similar but differently phrased prompts on classification performance, providing additional insights into the robustness and generalizability of LLM-based classification approaches.

Part III

LEVERAGING LLMS FOR HEALTHCARE

This part illustrates our contributions to the use of *LLMs* in critical healthcare scenarios. In particular, the chapters present specialized approaches that leverage LLMs to handle medication safety management and clinical decision support. The findings show how focused LLM adaptation can reduce alert fatigue, preserve patient safety, and support evidence-based clinical decision-making in healthcare settings.

9

HELIOT: LLM-BASED CDSS FOR ADVERSE DRUG REACTION MANAGEMENT

9.1 INTRODUCTION

Medication errors pose significant risks to patient safety and lead to adverse drug events (ADEs) [101]. In England, an estimated 237 million such errors occur annually, with around 66 million being potentially clinically significant [6, 101]. These incidents cost the National Health System (NHS) approximately £98.5 million annually, consume 181,626 bed days, and contribute to 1,708 deaths [101]. Similarly, in the United States, the economic impact is substantial, with prescribing and administration mistakes costing an estimated $20 billion annually [6].

CDSSs have emerged as tools to mitigate medication errors and enhancing patient safety [278]. These systems provide healthcare professionals with evidence-based recommendations and alerts, helping to prevent potential ADEs. However, traditional CDSSs face several limitations [170, 200, 255]. They typically rely on static databases and rule-based algorithms, which may not capture the nuances of individual patient cases or the latest medical knowledge [200, 237, 255]. For instance, when a drug with potential cross-reactions is prescribed, these systems generate alerts without considering complex clinical scenarios documented in notes, such as previous patient-specific tolerance, distinctions between true allergies and minor side effects, or situation-specific risk-benefit analyses [64]. Evidence shows that incorporating this contextual information could significantly reduce unnecessary alerts [64]. Without such capabilities, the rigid alerting mechanisms lead to excessive warnings, contributing to alert fatigue among healthcare providers and potentially causing critical warnings to be overlooked [64, 200, 237, 288, 289, 297, 298, 313].

LLMs have already succeeded in various healthcare applications, such as predicting drug interactions and patient outcomes, assisting in diagnostic processes, and generating clinical notes

[78, 247, 291]. We conjecture that their ability to process and synthesize large volumes of unstructured data makes them an innovative substitute for current rule-based CDSSs, which often lack the flexibility and depth of understanding required for intricate patient-specific situations.

This chapter presents HELIOT, a novel CDSS for adverse drug reaction management that processes unstructured clinical narratives, building upon our previous exploration of LLM integration in healthcare technologies [72]. The system overcomes the shortcomings of traditional CDSSs by leveraging LLMs to interpret medical narratives from clinical notes, text-based electronic health records, and transcribed patient conversations. By integrating this capability with a comprehensive pharmaceutical data repository, HELIOT improves the accuracy and reliability of adverse reaction alerts in healthcare. Our evaluation across three state-of-the-art LLMs (GPT-4o, Gemma 3, and Claude Sonnet) demonstrates consistent high performance, with classification accuracy ranging from 98.77% to 99.80% on both synthetic and real-world datasets. The system maintains perfect safety profiles with zero false negatives for life-threatening reactions while achieving a potential 50-53% reduction in unnecessary interruptive alerts compared to traditional CDSSs. HELIOT also incorporates confidence analysis to manage uncertain classifications and evaluates clinical usability, with healthcare professionals confirming strong acceptance of the system's contextual approach.

The primary contributions of this chapter are as follows.

- We present a novel approach to process unstructured clinical narratives for adverse drug reaction management, demonstrating how LLMs can extract and interpret patient-specific medication reaction information, including medication tolerances, adverse events, and cross-sensitivities.
- We describe the HELIOT CDSS's modular architecture, highlighting its core components and the approach employed to provide decision support services.
- We present an empirical evaluation of the HELIOT CDSS across multiple LLM architectures, including synthetic and real-world dataset validation, as well as a clinical usability assessment.

- We introduce a risk threshold framework based on confidence analysis for safe clinical deployment, providing structured guidelines for automated decision-making and clinical review requirements.
- We provide datasets and code used to develop the HELIOT prototype, including both the synthetic patient dataset and the real-world clinical dataset employed for empirical evaluation as contributions to the research community.

9.2 STATE OF ART AND MOTIVATION

This section reviews current CDSS challenges and limitations, discusses recent advances in LLMs for processing medical text, and presents the motivation behind our work.

9.2.1 *CDSSs Challenges and Limitations*

CDSSs are pivotal in improving patient safety and clinical efficiency through integration with Electronic Medical Records [278]. In the pharmacological domain, these systems must navigate complex territory, managing intricate drug interactions and patient-specific factors that require sophisticated algorithms to detect potential adverse reactions [251].

Research demonstrates the value of CDSSs in reducing medication errors, with knowledge-based systems showing a 4.4% improvement in prescribing behavior [170] and AI-based approaches achieving enhanced accuracy in prescription verification [67]. However, a critical challenge emerges with drug reaction alert systems, where override rates remain problematically high—ranging from 43.7% to 97% [193], with particularly concerning rates for opioids [288]. A comprehensive national evaluation of 1,599 hospitals revealed that while overall CDSS performance improved over time, the fundamental alert fatigue problem persisted, with hospitals achieving higher scores by overalerting—including inappropriate nuisance alerts that contribute to clinician burnout [57].

The root cause of this widespread alert dismissal [64, 200, 237, 288, 289, 297, 298, 313] lies in current systems' inability to effectively interpret unstructured clinical narratives containing crucial

patient-specific information about medication tolerances and reactions. Evidence indicates that incorporating previous drug tolerance data could significantly reduce unnecessary alerts [64]. While promising solutions have emerged, including ontology-based systems [43] and machine learning approaches [252], these methods fall short when representing these complex clinical scenarios, highlighting how advanced natural language processing techniques could potentially transform alert relevance and clinical acceptance.

9.2.2 *LLMs Opportunities*

The sophisticated capabilities of LLMs make them particularly promising for healthcare applications, with recent research demonstrating their effectiveness in generating differential diagnoses [247] and enhancing clinical documentation [291]. Their unique ability to process context-rich information and perform complex reasoning aligns with the challenge of interpreting medication narratives in clinical notes—a critical area where current CDSSs fail. Recent work has explored specialized LLMs for medication guidance, with systems like ShennongMGS demonstrating the potential of fine-tuned models for adverse drug reaction prediction and medication decision support [95]. However, these approaches typically focus on general medication guidance rather than specifically addressing the challenge of interpreting unstructured clinical narratives for patient-specific adverse reaction assessment, which remains a critical gap in current CDSS implementations. Despite this natural fit between LLMs' capabilities and the challenges of medication narrative interpretation, applying LLMs to extract and interpret medication information from unstructured clinical text remains underexplored.

9.2.3 *Motivation of our Work*

The potential of LLMs for healthcare applications is clear, but translating this potential into practical systems for medication safety requires addressing several domain-specific challenges. Even in healthcare settings with sophisticated Electronic Health Record

systems (EHR), clinical notes documenting patient reactions to drugs, referral letters detailing treatment histories, and records of medication tolerances frequently remain as free-text narratives. Traditional CDSSs can only process structured data and cannot interpret these narrative texts [64]. This means that valuable information about a patient's drug history may be overlooked when prescribing medications. For instance, a clinical note stating "Patient experienced mild rash with amoxicillin but has since tolerated cephalexin" contains important information about drug allergies and tolerances, but traditional CDSSs cannot extract and use this information because it is in free-text format. Moreover, the quantitative impact of alert fatigue on clinical decision-making has been well-documented through large-scale evaluations. Co et al.'s analysis of national hospital data demonstrated that hospitals achieving high CDSS performance scores often did so by implementing excessive alerting mechanisms, with those alerting on low-risk prescriptions scoring 3% higher overall but potentially compromising patient safety through alert fatigue [57]. This creates a critical paradox: systems designed to improve safety may inadvertently reduce it by overwhelming clinicians with inappropriate alerts.

In addition, the research community lacks comprehensive evaluation datasets for assessing CDSSs in this field [297]. LLMs may help address these challenges by providing the contextual understanding needed to interpret narrative texts and generate more targeted, contextually appropriate alerts, thereby reducing alert fatigue while improving medication safety. Our work aims to fill these gaps by providing open-source tools and evaluation resources to support future research in enhancing clinical decision support through natural language understanding.

9.3 HELIOT FRAMEWORK

This section presents the HELIOT CDSS, illustrating design principles and implementation details.

9.3.1 *CDSS Design and Approach*

The proposed framework comprises three integrated components: the decision support process, system architecture, and data adaptability mechanisms, described in detail below.

9.3.1.1 *Decision Support Process*

HELIOT employs a Retrieval Augmentation Generation (RAG) approach to support physicians in medication decisions based on patients' adverse reaction histories. RAG is a technique that enhances LLMs by retrieving relevant information from external knowledge sources before generating responses, making the output more informed, contextualized, and accurate. The decision process follows several integrated steps to produce clinical assessments.

The foundation of our approach begins with the parallel retrieval of drug information from specialized databases containing pharmaceutical data (active ingredients, excipients, contraindications, and side effects) while simultaneously analyzing patient clinical notes to identify potentially problematic ingredients.

A critical feature of our system is its ability to maintain continuity of care by integrating current and historical patient data. The patient database stores and updates clinical notes from previous encounters, creating a longitudinal record of adverse reactions and tolerances. When a healthcare professional consults the system about a new medication, these historical records are automatically retrieved and combined with current clinical notes. This integration provides a complete picture of the patient's reaction history, even in facilities without integrated EHR systems, ensuring that past adverse events are not overlooked in current decision-making. Once all relevant information, including drug composition, current clinical notes, and the patient's historical data, is gathered, the system executes the core decision support logic. This logic employs carefully crafted prompts (Figures 9.1 and 9.2) that guide the LLM to analyze the clinical situation. Following the persona pattern [314], our system instructs the LLM to embody an expert physician who evaluates potential adverse reactions by examining relationships between the drug's compo-

Decision Support Prompt: System Prompt

Act as an expert physician.
Your task is to check if the drug I want to prescribe is safe for the patient, focusing only on the potential drug reactions the patient has.
Drug To Prescribe: {*drug*}
Drug Active Ingredients: {*active_ingredients*}
Drug Excipients: {*excipients*}
Known Cross-reactivity: {*cross_reactivity*}
Known Excipients With Chemical Cross-reactivity: {*excipients_cross_reacts*}
Contraindications: {*contraindications*}

INSTRUCTIONS
...
CONFLICT HANDLING
...
OUTPUT FORMAT
{"a":"brief description of your analysis", "r":"final response: NO DOCUMENTED REACTIONS OR INTOLERANCES|DIRECT ACTIVE INGREDIENT REACTIVITY|DIRECT EXCIPIENT REACTIVITY|NO REACTIVITY TO PRESCRIBED DRUG'S INGREDIENTS OR EXCIPIENTS|CHEMICAL-BASED CROSS-REACTIVITY TO EXCIPIENTS|DRUG CLASS CROSS-REACTIVITY WITHOUT DOCUMENTED TOLERANCE|DRUG CLASS CROSS-REACTIVITY WITH DOCUMENTED TOLERANCE", "rt":"reaction type: None|Life-threatening|Non life-threatening immune-mediated|Non life-threatening non immune-mediated"}

Figure 9.1: Decision Support Prompt: System Prompt.

Decision Support Prompt: User Prompt

PATIENT INFORMATION: {*clinical_notes*}

Figure 9.2: Decision Support Prompt: User Prompt.

sition and the patient's documented reaction history, including current and historical notes.

The system's final output provides an assessment structured in three parts: a clinical classification of the case (e.g., "Direct Active Ingredient Reactivity"), a categorization of the reaction severity (e.g., "Life-threatening"), and a detailed analysis explaining the rationale behind these classifications. The alert type is automati-

cally determined based on the combination of case classification and reaction severity, as shown in Table 9.1.

Table 9.1: Case Classification, Reaction Type, and Alert Type.

Case Classification	Reaction Type	Alert Type
No documented reactions or intolerances	None	None
No reactivity to prescribed drug's ingredients or excipients	None	None
Direct active ingredient reactivity	Life-threatening	Interruptive
	Non life-threatening immune-mediated	Interruptive
	Non life-threatening non immune-mediated	Non-interruptive
Direct excipient reactivity	Life-threatening	Interruptive
	Non life-threatening immune-mediated	Interruptive
	Non life-threatening non immune-mediated	Non-interruptive
Chemical-based cross-reactivity to excipients	Life-threatening	Interruptive
	Non life-threatening immune-mediated	Interruptive
	Non life-threatening non immune-mediated	Non-interruptive
Drug class cross-reactivity without documented tolerance	Life-threatening	Interruptive
	Non life-threatening immune-mediated	Interruptive
	Non life-threatening non immune-mediated	Non-interruptive
Drug class cross-reactivity with documented tolerance	None	None

When conflicting information emerges during the decision support process, HELIOT employs a hierarchical resolution strategy that prioritizes patient safety while incorporating clinical context. The system applies the following conflict resolution principles:

1. Patient-Specific Evidence Priority - documented patient-specific tolerance or adverse reactions in clinical notes override general pharmaceutical contraindications

2. Temporal Precedence - more recent clinical observations take precedence over older general warnings while maintaining awareness of historical patterns

3. Severity-Based Escalation - life-threatening reactions documented in patient history always trigger alerts regardless of conflicting tolerance data

4. Clinical Context Integration - the LLM synthesizes official pharmaceutical data with direct clinical observations, reasoning through apparent contradictions (e.g., "patient has documented penicillin allergy but tolerated amoxicillin")

5. Uncertainty Acknowledgment - when conflicts cannot be definitively resolved through clinical reasoning, the LLM is instructed to explicitly state uncertainty and recommend clinical review rather than making unilateral decisions.

This strategy leverages the LLM's natural language reasoning capabilities while ensuring that clinical judgment remains paramount.

Ingredient Translation Prompt

Translate in English from language: {*text*} Report only the translation, nothing else. If you don't know the translation, report the original text.

Figure 9.3: Ingredient Translation Prompt.

It is worth noting that HELIOT faces a significant linguistic challenge when processing clinical notes from different regions and healthcare systems. Current LLMs, as demonstrated by Wendler et al [309], exhibit an intrinsic bias toward English in their conceptual processing, potentially compromising reasoning accuracy when working with non-English medical terminology. HELIOT addresses this limitation by standardizing all medical and pharmaceutical terminology into English using a dedicated translation prompt (Figure 9.3). This approach not only overcomes the language bias of underlying models but also ensures optimal correspondence with international medical ontologies that predominantly use English nomenclature.

9.3.1.2 *System Architecture*

Building upon the RAG approach described above, HELIOT's architecture implements a modular, model-agnostic framework de-

signed for deployment flexibility across different healthcare environments. The system separates data management, retrieval coordination, and decision support logic into distinct components that can be independently updated or scaled.

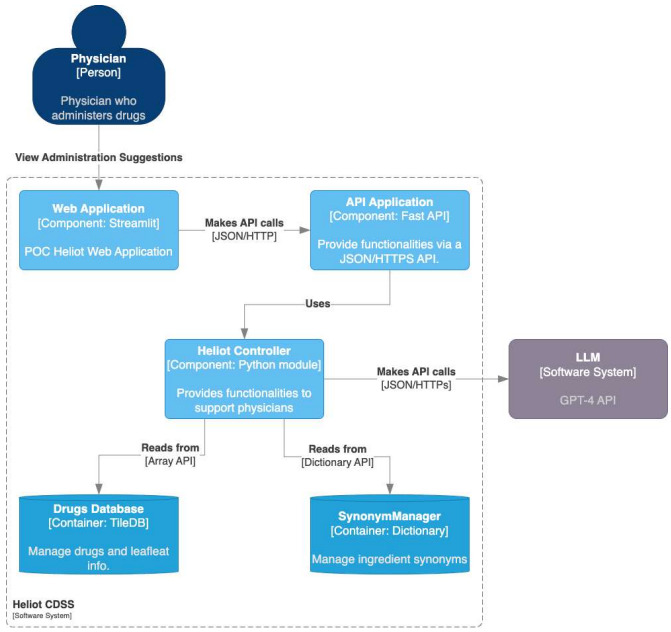


Figure 9.4: HELIOT CDSS Architecture.

As shown in Figure 9.4, the architecture comprises three main components: a web application for standalone operation, an API application providing RESTful services for EHR integration, and the HELIOT Controller implementing the core decision-making logic. This service-oriented design enables HELIOT to function both as an independent clinical tool and as an external microservice that existing healthcare systems can invoke without replacing their native infrastructure.

The HELIOT Controller serves as the central orchestration engine, coordinating several specialized sub-components. The TileDB Drug and Patient Databases store comprehensive pharmaceutical information (drug identifiers, active ingredients, excipients, contraindications, and side effects) and patient clinical notes respectively. We selected TileDB [226] for its efficient data retrieval, storage compression capabilities, and versatility in supporting both

local and cloud deployments. The Synonym Manager maintains an in-memory structure that maps canonical ingredient names to their corresponding synonyms, ensuring standardized ingredient recognition. The LLM component handles advanced natural language processing tasks, interpreting medical texts and generating contextually relevant recommendations.

Finally, the API application provides RESTful services consumed by either the web interface or external EHR systems through standard FHIR and HTTP protocols. To enhance user experience, the API streams responses in real-time using server-sent events, providing immediate feedback as results become available and reducing perceived latency.

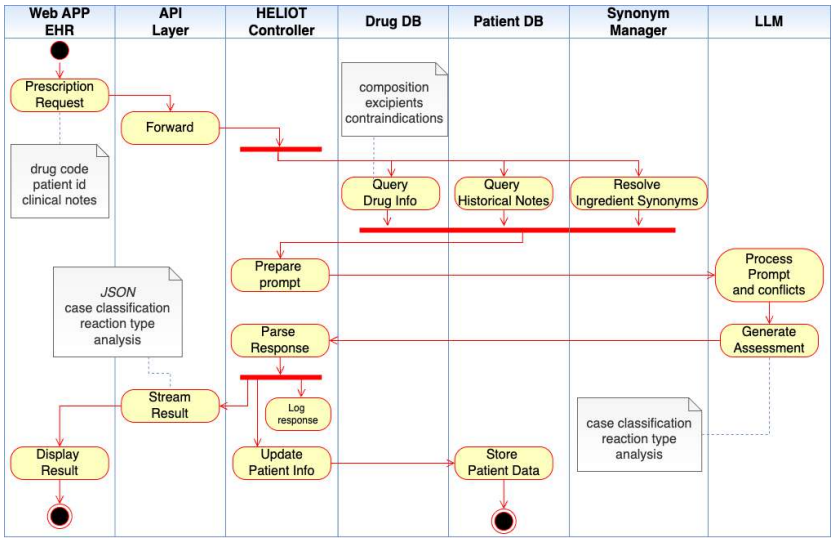


Figure 9.5: HELIOT CDSS Data Flow.

The data retrieval mechanism follows a multi-layered architecture (see Figure 9.5) that begins when a physician or clinician submits a prescription request through the web application or EHR system.

This initial request, containing drug codes, patient IDs, and clinical notes, is forwarded through the API Layer to the HELIOT Controller. Upon receiving the request, the HELIOT Controller simultaneously initiates three parallel data retrieval operations:

querying drug information from the Drug Database (local DB or through FHIR integration), retrieving historical patient notes from the Patient Database, and resolving ingredient synonyms through the Synonym Manager. Once these queries are complete, the controller prepares a prompt incorporating all retrieved data elements and leverages the LLM to process this information and generate a clinical assessment. The generated assessment undergoes parsing by the controller, which logs the response and updates patient information in the database. This architecture ensures optimal performance through parallel retrieval strategies, maintains data integrity through secure access protocols, and provides seamless integration capabilities that allow HELIOT to enhance existing healthcare workflows without disrupting established infrastructure.

9.3.1.3 *LLM-Pharmaceutical Data Integration Framework*

The coordination between pharmaceutical databases and clinical narratives operates through four distinct layers:

- **Data Abstraction Layer**: This layer provides uniform access to heterogeneous pharmaceutical data sources through a configurable interface that abstracts implementation-specific details. The system employs a generic loading mechanism that dynamically selects appropriate data management modules based on configuration parameters, enabling seamless integration with different pharmaceutical repositories (local databases, FHIR-compliant servers, national drug registries) without requiring architectural changes. This abstraction ensures that upper layers receive pharmaceutical data in a consistent format regardless of the underlying data source, allowing HELIOT to adapt to diverse healthcare IT environments while maintaining identical clinical decision support capabilities.

- **Retrieval Coordination Layer**: The system orchestrates parallel data retrieval operations to minimize response latency. When a clinician queries about a specific medication, this layer simultaneously: (1) retrieves comprehensive pharmaceutical data using drug identifiers, (2) accesses current pa-

tient clinical notes, (3) queries historical adverse reaction records, and (4) resolves ingredient synonyms through the synonym management system. This parallel processing ensures that all relevant context is available before LLM processing begins.

- **Context Assembly Layer**: Retrieved information is structured into coherent prompts that guide LLM reasoning. This layer combines pharmaceutical data (active ingredients, excipients, known contraindications) with patient-specific information (documented reactions, tolerances, clinical notes) into the specialized prompts shown in Figures 9.1 and 9.2. The assembly process includes linguistic standardization (see Section 9.3.1.1) and maintains semantic relationships between different data elements.

- **LLM Processing Layer**: The assembled context is processed by the LLM backend to generate clinical recommendations. This layer is designed to be model-agnostic, supporting different LLM architectures through standardized input/output interfaces. The processing follows the decision support logic outlined in Section 9.3.1.1, producing structured outputs that include clinical classifications, severity assessments, and reasoning explanations.

This layered approach ensures that LLM integration remains flexible while maintaining consistent decision support quality across different technological environments.

9.3.1.4 *Data Flexibility and Adaptability*

The architectural design and LLM integration framework described above are built upon a foundation of data flexibility that enables HELIOT to adapt to diverse healthcare environments and evolving medical knowledge. This adaptability manifests across multiple dimensions of the system's operation. As detailed in section 9.3.1.2, while drug identifiers need standardization, most critical information—including active ingredients, excipients, contraindications, and side effects—can be stored as unstructured free text, eliminating the need for complex data normalization procedures.

Similarly, the patient clinical database—while maintained locally within HELIOT—demonstrates adaptability in processing clinical information. The system does not impose rigid requirements on structuring clinical notes, relying instead on the LLM's natural language processing capabilities to extract relevant adverse reaction information from various documentation styles and formats. Regarding linguistic adaptation, the standardization process employs generalized language conversion mechanisms to normalize terminology into a standard language. This approach allows healthcare facilities in regions with different primary languages to utilize the system while maintaining consistent clinical reasoning.

Beyond static data handling, the architectural flexibility extends to dynamic system maintenance and evolution. HELIOT incorporates a multi-layered updating framework to ensure the system remains current with evolving medical knowledge and regulatory requirements. The updating strategy varies depending on the deployment scenario. For standalone deployments, the local drug database receives routine monthly updates for standard pharmaceutical information, while critical safety data is incorporated within 24 hours of regulatory announcements. For EHR-integrated deployments, HELIOT leverages FHIR R4 interfaces to access real-time pharmaceutical data directly from institutional repositories, ensuring automatic synchronization as institutional systems receive updates. This data-level flexibility is complemented by adaptable decision logic through the use of a prompt-based framework. As medical understanding advances, clinical protocols can be updated by modifying the structured prompts while maintaining backward compatibility through REST API versioning mechanisms. Finally, the model-agnostic architecture and microservices design support seamless component evolution. Language models can be upgraded through standardized interfaces, while individual services can be updated independently using rolling deployment strategies to ensure zero downtime. This updating framework ensures HELIOT maintains clinical accuracy and operational stability across diverse healthcare environments as medical knowledge and technology evolve.

9.3.2 *HELIOT Prototype*

This section describes our implementation of the HELIOT CDSS design as a functional prototype.

9.3.2.1 *Pharmaceutical Data Pipeline*

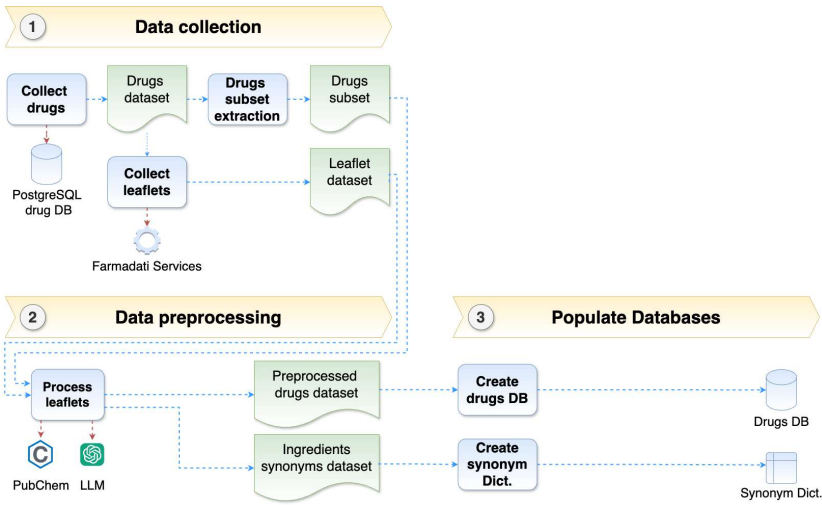


Figure 9.6: Overview of the Pharmaceutical Data Pipeline Process.

The first step in developing our prototype was creating a comprehensive pharmaceutical knowledge base. To this aim, we developed a specialized data pipeline to transform raw pharmaceutical data into HELIOT's pharmaceutical knowledge base. The pipeline (see Figure 10.1) consists of three main phases: 1) Data Collection, 2) Data Preprocessing, and 3) Database Population.

The subsequent subsections offer an explanation of each phase.

DATA COLLECTION In the data collection phase, we gathered the necessary data for our experiments, specifically the datasets for drugs and leaflets.

Drug Dataset

The first step was creating the drug dataset. To this end, an Italian company provided us with the PostgreSQL dump of the database of Italian medicines approved by the Italian Medicines

Agency (AIFA) [1]. The drug dataset has 106,962 drugs. Table 9.2 reports the dataset columns.

Table 9.2: Drug Dataset Structure.

Column	Description
Drug_code	It is the ministerial code of the drug, also known as the AIC code (Marketing Authorization issued by AIFA)
Drug_name	It is the drug name
Drug_form_full_desc	It is the pharmaceutical form of the drug
Atc_code	It is the drug's ATC (Anatomical Therapeutic Chemical) code, according to the Word Health Organization classification system.
Leaflet	It is the leaflet file name.

Leaflets dataset

The next step was to collect the leaflets associated with the medications. To make the pipeline production-ready, we automated this step by referring to a pharmaceutical data provider, the Farmadati company [2] that is one of Italy's major pharmaceutical data providers. It offers paid services to download the up-to-date drug database, including leaflets. Consequently, we utilized the web services provided by Farmadati to acquire leaflets for each medication in the dataset. The final leaflets dataset contains 19,188 files. The leaflets' structure includes the medication's name, qualitative and quantitative composition, pharmaceutical form, clinical information, pharmacological properties, pharmaceutical information, radiation dosimetry data for radiopharmaceuticals, and instructions on extended preparation and quality control for radiopharmaceuticals.

Drugs subset extraction

The final step of the data collection phase was creating a representative subset of drugs for the subsequent "data preprocessing" step. The drug dataset was randomly sampled from the drugs dataset using ATC codes. We followed the distribution reported in literature studies [64, 136, 288, 289] to ensure the dataset reflected real-world drug prescription and adverse reaction patterns. The sampling process maintained the proportions of different drug classes as observed in clinical settings, with narcotic analgesics representing the largest group (65%), followed by antibiotics (15%), NSAID (5%), diuretics (2%), antiplatelet agents (2%), and other medications (11%). We also included common excipients associ-

1 AIFA. https://www.aifa.gov.it/en/trova-farmaco

2 Farmadati. https://www.farmadati.it/default.aspx

ated with both immediate and delayed hypersensitivity reactions, with particular attention to preservatives and common allergens (e.g., polyethylene glycol, polysorbates, benzalkonium chloride).

Table 9.3: Drug Distribution in Dataset.

Drug Class	Perc.	No. of drugs
Opioids	65%	653
Antibiotics	15%	152
NSAID	5%	47
Diuretics	2%	24
Antiplatelet agents	2%	16
Other	11%	108
TOTAL		**1,000**

Table 9.3 reports the final dataset distribution.

DATA PREPROCESSING The data preprocessing phase addressed two key challenges in the leaflet data for our drug subset.

First, single leaflets often contain information for multiple pharmaceutical forms of the same medication (e.g., the ORAMORPH leaflet includes details for both syrup and oral solution forms). Since healthcare professionals prescribe specific forms, we must extract form-specific information while excluding irrelevant details. Second, the original ingredients and excipients were in Italian, requiring translation to English (consistently with the linguistic standardization approach discussed in section 9.3.1) to enable synonym matching through international services like PubChem [3] [4]. This standardization is paramount as ingredients may appear under different names in medical records (e.g., "acetylsalicylic acid" vs "aspirin").

To address these challenges, we leveraged GPT-4o with two specialized prompts for processing ingredients and leaflet sections. For the sake of readability, we did not report the full prompts

3 PubChem substances. https://pubchem.ncbi.nlm.nih.gov/rest/pug/substance/name/{encoded_ingredient_name}/synonyms/JSON

4 PubChem compounds. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/{encoded_ingredient_name}/synonyms/JSON

that can be found in our online repository https://github.com/gadevito/heliot.

During this phase, we processed 1,035 unique ingredients and created an ingredient dictionary by querying PubChem REST services for comprehensive synonym lists. The preprocessing resulted in two structured datasets: "leaflet info.csv" containing form-specific drug details and "ingredients synonyms.csv" with standardized ingredient names and variants. Tables 9.4 and 9.5 detail the structure of these datasets.

Table 9.4: Processed Leaflet Dataset Structure.

Column	Description
Drug_code	It is the ministerial code of the drug.
Drug_name	It is the drug name.
Drug_form	It is the pharmaceutical form of the drug.
ATC	It is the ATC code of the drug.
Composition	It contains the list of the drug's active ingredients.
Excipients	It contains the list of the drug's inactive ingredients.
Contraindications	It contains the contraindications for the drug.
Drug_interactions	It contains the drug interactions.
Side effects	It reports the side effects of the drug.
Incompatibilities	It reports the incompatibilities for the drug in the case of concurrent pharmaceutical therapies.

Table 9.5: Ingredient Synonyms Dataset Structure.

Column	Description
Ingredient	It is the ingredient name extracted from the processed leaflets;
English_name	It is the English translation of the ingredient.
Synonyms	List of synonyms separated by '#' returned by the PubChem REST service.
Type	It represents the type of the ingredient, namely active or inactive.

To ensure the accuracy of the preprocessed data, two healthcare professionals (a clinical pharmacist and a physician) independently validated a random sample of 20% of the drugs subset. This sample size was determined following established methodology for validation studies [113] and provides an appropriate statistical power for this application. The healthcare professionals focused on verifying that GPT-4o correctly extracted information specific to each pharmaceutical form from the official leaflets. They also verified the accuracy of ingredient and excipient trans-

lations from Italian to English, ensuring correct matching with PubChem synonyms. The validation established that our preprocessing approach, particularly the designed GPT-4o prompts, effectively isolated and extracted form-specific information from the official leaflets while maintaining data integrity. The high inter-rater agreement (Cohen's kappa [59] = 0.95) further supported the reliability of our approach, demonstrating "almost perfect" consensus between clinical experts.

POPULATE HELIOT DATABASES We then populated two databases using the preprocessed datasets: the drug database using "leaflet info.csv" and the synonyms database using "ingredients synonyms.csv" (see section 9.3.2.1). For the drug database schema, we defined drug_code, atc_code, composition, and excipients as dimensions with ASCII data type, while other columns became array attributes. We applied "ZstdFilter" compression [65] at level 3 for text-heavy attributes to optimize storage and query performance. Given the manageable number of ingredients (1,035) for the synonyms database, we implemented an in-memory data structure to ensure fast retrieval of ingredient synonyms during CDSS processing.

9.3.2.2 *Prototype Implementation*

Building upon the data pipeline, we developed a fully functional proof-of-concept (POC) prototype of the HELIOT CDSS. This implementation demonstrates how the conceptual architecture described in section 9.3.1.2 translates into practice with specific technology choices and integration patterns. While the architecture is designed to be model-agnostic and compatible with various LLMs (e.g., LLaMA), for this implementation we utilized GPT-4o through the OpenAI API. The prototype consists of the web application developed using the Streamlit framework, the API application developed using the FAST API and Uvicorn frameworks, and the HELIOT Controller developed using TileDB and the OpenAI API. The web application provides functionalities for processing single prescriptions with streamed results, uploading entire datasets for batch processing, and downloading results.

All the scripts, results, and source code are provided in our online repository `https://github.com/gadevito/heliot`.

9.4 EVALUATION METHODOLOGY

This section describes the methodology for evaluating HELIOT's effectiveness as a clinical decision-support system, including our research objectives, the creation of a specialized evaluation datasets, and the experimental design.

9.4.1 *Research Goals*

The primary goal of the empirical assessment was to analyze the effectiveness of the proposed CDSS in two key dimensions: accuracy of clinical decision support and reduction of alert fatigue. The purpose was to provide empirical evidence highlighting the benefits and limitations of the HELIOT CDSS, enabling healthcare professionals to be aware of the strengths and weaknesses they would encounter through its use.

Specifically, the empirical assessment aimed to address the following research question:

RQ9.1: *How effective is HELIOT CDSS in identifying potential drug reactions and reducing alert fatigue?*

This research question encompasses the system's ability to correctly identify potential adverse drug reactions and provide contextually appropriate alerts that do not overwhelm clinicians with unnecessary warnings.

9.4.2 *Synthetic Patient Dataset Creation*

To evaluate HELIOT, we developed a synthetic patient dataset that represents the variety and complexity of adverse drug reaction scenarios encountered in clinical practice. Recent studies [121, 168] support this synthetic data approach, demonstrating its potential to replicate real-world analysis results while maintaining privacy and supporting robust evaluation. Our synthetic dataset creation followed a systematic, literature-informed approach to ensure clinical accuracy and real-world representativeness. We began by analyzing distribution patterns from established litera-

ture [64, 288, 289] that documented comprehensive data on drug allergy alert systems, including override rates and reaction patterns across different drug categories. These studies provided empirical evidence showing override rates ranging from 43.7% to 97% across different therapeutic classes, with narcotic analgesics exhibiting the highest override rates and most common reactions, including itching (23.3%), nausea (13%), and hives (10.7%). Based on this literature analysis, we created structured JSON configuration files that defined clinical case distributions across seven distinct classification categories (see Table 9.6), designed to capture the full spectrum of adverse drug reaction scenarios encountered in clinical practice. These JSON configuration files served

Table 9.6: Clinical Cases Classifications.

Classification	Description
No documented reactions or intolerances	Patients with no history of adverse drug reactions or intolerances in their clinical records
Direct active ingredient reactivity	Patients with documented adverse reactions to the active ingredient of the prescribed drug
Direct excipient reactivity	Patients with documented reactions to an excipient present in the prescribed drug's formulation
No reactivity to prescribed drug's ingredients or excipients	Patients with documented adverse drug reactions, but not related to any component of the prescribed drug
Chemical-based cross-reactivity to excipients	Patients with adverse reactions to drugs in the same therapeutic class as the prescribed medication, without documented tolerance
Drug class cross-reactivity without documented tolerance	Patients with adverse reactions to drugs in the same therapeutic class as the prescribed medication, without documented tolerance
Drug class cross-reactivity with documented tolerance	Patients with adverse reactions to drugs in the same therapeutic class but with documented tolerance to the prescribed medication

as input specifications for automated dataset generation scripts, defining the expected number of cases for each scenario type, reaction severity levels, alert requirements, and clinical narrative templates.

The JSON configuration files specified parameters including case distributions, drug selections from our pharmaceutical database, reaction severity mappings, and clinical narrative templates for each classification type. The classifications range from patients with no documented reactions to complex cross-reactivity patterns involving chemical similarities between excipients or ther-

apeutic drug classes. Each classification was further subdivided based on reaction severity (life-threatening, non-life-threatening immune-mediated, or non-life-threatening non-immune-mediated) and documented tolerance patterns. For cross-reactivity scenarios, we incorporated established chemical relationships between excipients based on documented cross-sensitivity patterns [38, 272]. Cross-reactivity cases were designed around known chemical similarities, such as polyethylene glycol reactions with potential cross-reactivities to polysorbates, poloxamers, and cremophor, or drug class cross-reactivity scenarios based on established therapeutic relationships within antibiotic families or analgesic classes. The distribution of cases across different classifications reflects real-world clinical patterns, with drug class cross-reactivity scenarios representing the largest proportion of cases, as shown in Figure 9.7. This distribution ensures testing of HELIOT's ability to handle various clinical scenarios while maintaining statistical validity for evaluation purposes. Clinical note generation was au-

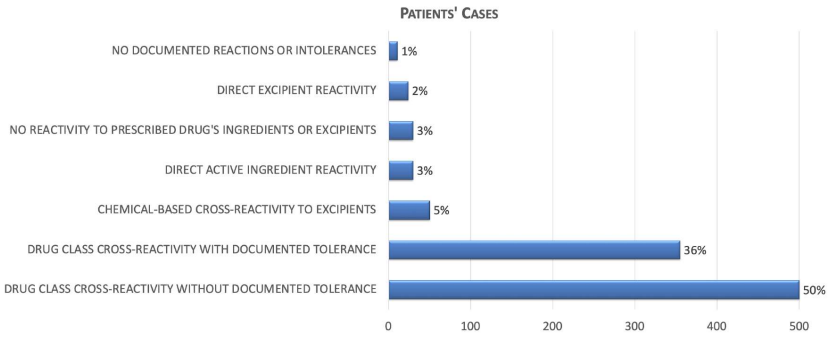


Figure 9.7: Class Distribution in the Patients Dataset.

tomated through specialized Python scripts that read the JSON configuration files and created patient-specific clinical narratives. These automated generation scripts processed the configuration

data to create realistic clinical notes incorporating appropriate medical terminology, reaction descriptions, temporal relationships spanning from 2000 to 2025, and tolerance documentation where clinically appropriate. The generation process utilized predefined clinical templates that were refined through iterative consultation with healthcare professionals to ensure linguistic authenticity and clinical accuracy. The scripts randomly selected appropriate drugs from our pharmaceutical database based on the therapeutic classes specified in the JSON configurations (i.e., ATC codes), generated unique patient identifiers, and created detailed clinical narratives that included specific reaction histories, symptom descriptions, and documented medication tolerances where clinically appropriate. The final dataset structure is detailed in Table 9.7, which shows the key fields maintained for each synthetic case to ensure consistency with real-world electronic health records. The dataset validation process involved two healthcare

Table 9.7: Synthetic Dataset Structure.

Field	Description
Patient ID	Unique identifier for each case
Drug Code	Ministerial code of the prescribed drug
Drug Name	Name of the prescribed drug
Clinical Note	Detailed patient history including adverse reactions
Classification	Case classification category
Alert Type	Interruptive/Non-interruptive/No alert
Reaction Type	Severity and nature of reaction
Prescribed ATC	ATC code of the prescribed drug

professionals (a clinician and a physician) who complemented our earlier pharmaceutical validation team (see section 9.3.2.1) with specialized knowledge in patient reactions and medical documentation. We implemented a human-in-the-loop validation approach where each synthetic case underwent independent expert review to verify clinical plausibility, appropriate reaction-drug relationships, and accurate alert classification. As demonstrated in similar healthcare Artificial Intelligence implementations where human expertise guides model decisions and validates outputs [174, 225, 296], this approach enhanced both the interpretability and reliability of the AI systems. The validation process examined multiple dimensions of each case, including the appropri-

ateness of documented reactions for specific drugs, the clinical logic of cross-reactivity patterns, the accuracy of severity classifications, and the appropriateness of recommended alert types. For example, experts verified that documented reactions such as rash or swelling were clinically consistent with the prescribed medication's known adverse effect profile, ensuring that our synthetic cases reflected realistic clinical scenarios. Disagreements between reviewers were resolved through consensus meetings, with particular attention to complex cross-reactivity cases where alert type assignments required nuanced clinical judgment. The

Table 9.8: Case Distribution by Classification, Reaction Type, and Alert Type.

Case Classification	Reaction Type	Alert Type	Cases	Perc.
No documented reactions or intolerances	None	None	11	1.1%
No reactivity to prescribed drug's ingredients or excipients	None	None	30	3.0%
	Life-threatening	Interruptive	9	0.9%
	Non life-threatening immune-mediated	Interruptive	12	1.2%
	Non life-threatening non immune-mediated	Non-interruptive	9	0.9%
	Life-threatening	Interruptive	6	0.6%
Direct excipient reactivity	Non life-threatening immune-mediated	Interruptive	4	0.4%
	Non life-threatening non immune-mediated	Non-interruptive	14	1.4%
Chemical-based cross-reactivity to excipients	Life-threatening	Interruptive	15	1.5%
	Non life-threatening immune-mediated	Interruptive	9	0.9%
	Non life-threatening non immune-mediated	Non-interruptive	26	2.6%
Drug class cross-reactivity without documented tolerance	Life-threatening	Interruptive	103	10.3%
	Non life-threatening immune-mediated	Interruptive	271	27.1%
	Non life-threatening non immune-mediated	Non-interruptive	126	12.6%
Drug class cross-reactivity with documented tolerance	None	None	355	35.5%
TOTAL			**1,000**	**100%**

final validation achieved an inter-rater agreement of 0.91 (Cohen's kappa), indicating "almost perfect" consensus between clinical

experts and confirming the reliability of our synthetic dataset creation approach. The resulting dataset contains 1,000 synthetic cases with a comprehensive distribution across clinical scenarios and alert requirements. Table 9.8 provides the complete breakdown of case distribution by classification, reaction type, and alert type, demonstrating the dataset's coverage of diverse clinical scenarios designed to test HELIOT's ability to distinguish between different types of adverse drug reaction situations and generate contextually appropriate clinical recommendations.

The source code and configuration files for the dataset creation can be found in our online repository `https://github.com/gadevito/heliot`.

9.4.3 *Real-World Validation Case Study*

Validation of HELIOT's performance beyond controlled synthetic scenarios involved obtaining real-world Electronic Health Record data from an Italian healthcare IT company that provides EHR systems to healthcare facilities. This dataset contained medication administrations for over 2,000 patients with 10,000 randomly extracted administrations, all representing safe prescriptions that were actually administered to patients. Clinical notes were extracted from the allergy and intolerance sections of patient medical records, documented in Italian by healthcare professionals during routine clinical practice. After joining with our pharmaceutical subset of 1,000 drugs, we obtained 162 patients with corresponding prescriptions, distributed as 114 patients without known allergies and 48 patients with clinical notes documenting drug allergies, adverse reactions, or side effects. Among the 48 patients with documented clinical notes, 2 cases specifically documented tolerance to the prescribed drug. While this distribution differs from literature patterns used in our synthetic dataset, this was expected given the random extraction across multiple departments and the limited scope of our pharmaceutical database.

9.4.4 *Model Selection and Setup*

To evaluate the HELIOT's robustness, we selected three representative LLMs with distinct characteristics and deployment approaches. The selection encompasses cloud-based and local deployment scenarios, different parameter scales, and diverse training approaches. GPT-4o was the primary evaluation model to maintain consistency throughout the research pipeline. It had previously been utilized to create synthetic datasets and develop pharmaceutical databases. This approach ensures methodological coherence across all experimental phases. The model was accessed through OpenAI's API. Google's Gemma 3 (12B parameters) [284] was selected to represent open-weight model architectures and validate performance scalability with smaller parameter counts. The model was deployed locally using LM Studio, a comprehensive platform for LLM experimentation that provides a unified OpenAI-compatible REST API for model deployment and inference across various architectures. Anthropic's Claude 4 Sonnet was included as a third validation point, accessed through Anthropic's API [17]. We set the temperature to 0.0 for all models to reduce non-deterministic outputs and maintain consistency across evaluations.

9.4.5 *Experimental Design*

We employed five complementary evaluation approaches to investigate the research question

9.4.5.1 *Evaluation Using the Synthetic Dataset*

This first evaluation approach compared responses from HELIOT CDSS against ground truth data provided by healthcare professionals in the data pipeline (see Section 9.4.2). This comparison was made using the HELIOT CDSS prototype web application (see Section 9.3.2) for interaction and validation. Figure 9.8 illustrates the design of the experiment.

We first uploaded the synthetic patient dataset (described in Section 9.4.2) to the HELIOT DSS POC, where the data is analyzed for potential reactions. The system's results were then down-

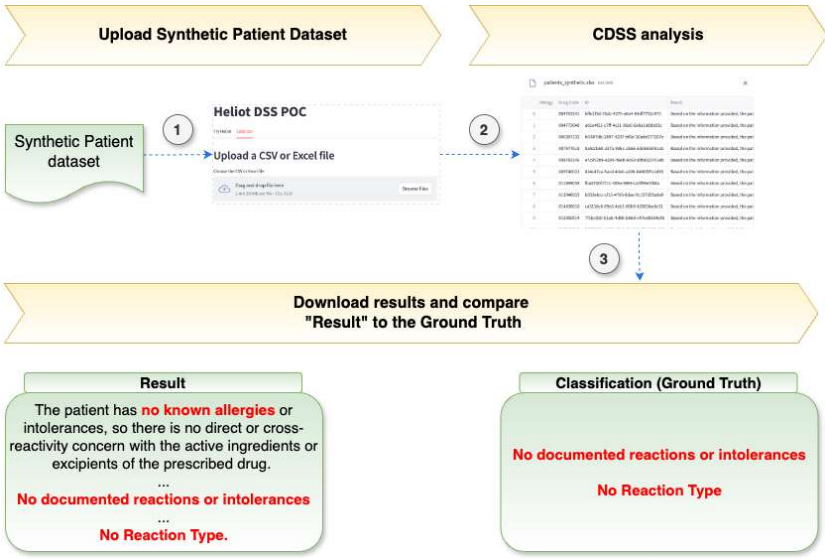


Figure 9.8: Experiment Design and Setup.

loaded and compared to the ground truth, evaluating separately the "Classification" and "Reaction Type" assignments provided by the healthcare professionals. The alert type is automatically determined based on case classification and reaction type, as follows: when there is no documented tolerance, life-threatening reactions and non life-threatening immune-mediated reactions trigger interruptive alerts, while non life-threatening non immune-mediated reactions produce non-interruptive alerts. No alerts are needed when there are no documented reactions or when tolerance has been previously established (see Table 9.8 in section 9.4.2).

To assess the consistency of the LLM's responses and evaluate any potential non-deterministic behavior, we repeated the experiment five times under identical conditions for all three language models (GPT-4o, Gemma 3, and Claude Sonnet), averaged the results, and calculated the Fleiss' Kappa, which is a statistical measure that evaluates the agreement between multiple raters (or iterations, in this case), across iterations [114]. This approach allowed us to verify whether the system produces consistent results

across multiple runs, which is crucial for ensuring reliability in clinical applications.

To quantitatively measure the effectiveness of the HELIOT CDSS, we employed three key metrics, which are standard in evaluating classification systems and provide a comprehensive view of performance, specifically: Precision, Recall, and F1-score.

Precision is the ratio of true positives (correctly identified instances of a class) to all instances predicted as that class. Recall measures the proportion of true positive results among all actual positive cases, and indicates how well the system can identify positive instances. Finally, F1-score is the harmonic mean of precision and recall, providing a metric that balances both concerns. It is particularly useful when the class distribution is imbalanced.

We also measured the execution time of the analysis along the five runs and averaged the results to evaluate the proposed CDSS's performance comprehensively. We executed the experiment using a MacBook M3 Max with 96 GB of RAM and a unified Metal GPU.

9.4.5.2 *Evaluation Using the Real-World Dataset*

For the real-world validation, we processed the 162 patient cases from the EHR dataset described in Section 9.4.3 through HELIOT. We maintained the same experimental design used for synthetic dataset evaluation, employing identical quantitative metrics (Precision, Recall, and F1-score) to ensure consistent performance assessment across synthetic and real-world scenarios. The number of runs for real-world validation was determined based on the consistency patterns observed in the synthetic evaluation, with deterministic models requiring single-run validation and non-deterministic models requiring multiple runs to ensure robust assessment. This real-world case study provides a crucial assessment of HELIOT's ability to process actual clinical documentation and handle the linguistic and terminological variations found in routine healthcare practice.

9.4.5.3 *Clinical Usability Evaluation*

To assess the clinical usability of HELIOT's decision support approach, we involved nine healthcare professionals. The participant demographics included healthcare professionals from Internal Medicine (78%) and Primary Care/Family Medicine (22%) with diverse experience levels: < 5 years (11%), 5-10 years (33%), 11-20 years (33%), and > 20 years (22%). Practice settings varied across hospitals (33%), private practices (33%), outpatient clinics (11%), emergency departments (11%), and academic medical centers (11%). A structured questionnaire was developed to evaluate clinical accuracy, alert fatigue reduction, workflow integration, and overall system value across seven representative clinical cases spanning cross-reactivity with documented tolerance, cross-reactivity without documented tolerance, and non-immune side effects. Each case presented patient history, prescribed medication, HELIOT's output with explanation, and traditional CDSS output for comparison. The evaluation employed 5-point Likert scales to assess key dimensions, including clinical appropriateness and risk assessment accuracy, information clarity, clinical reasoning transparency, confidence in recommendations, the impact of alert fatigue, time efficiency, and the likelihood of adoption. The evaluation aimed to assess whether HELIOT's contextual approach provides clinically meaningful improvements over traditional CDSS systems in reducing alert fatigue while maintaining appropriate clinical safety measures. The complete questionnaire can be found in Appendix A.

9.4.5.4 *Safety Analysis and Risk Assessment*

To ensure clinical safety in alert reduction scenarios, we conducted a safety analysis across all datasets and language models. This evaluation focused on quantifying clinical risk associated with "no alert needed" classifications and identifying potential missed critical events. We analyzed false negative rates for life-threatening reactions, calculated sensitivity and specificity for critical event detection, and assessed false positive rates to establish acceptable safety margins. The analysis examined consistency across different LLM architectures to validate the robustness of HELIOT's safety

profile. Key metrics included negative predictive value for cases classified as requiring no intervention, false alarm rates for conservative classifications, and verification of zero missed critical events across both synthetic and real-world scenarios.

9.4.5.5 *Confidence Analysis Framework*

To establish risk thresholds for clinical deployment and assess system certainty, we conceived confidence scoring based on token log-probabilities returned by the LLM for decision support outputs. This approach enables quantitative assessment of system confidence and supports risk-stratified clinical deployment. Four confidence metrics were calculated from the LLM's JSON response:

- **Overall confidence**: Mean log-probability across the complete JSON response
- **Analysis confidence**: Log-probability of the clinical analysis text
- **Case classification confidence**: Log-probability of the case classification category
- **Reaction classification confidence**: Log-probability of the reaction type classification

These metrics provide multi-dimensional assessment of system certainty, enabling identification of cases where automated decisions may require clinical review. The confidence scoring framework supports the development of risk thresholds that balance automated efficiency with appropriate clinical oversight, particularly important for cases classified as requiring no alerts where missed critical reactions could pose safety risks. For both synthetic and real-world evaluations, confidence scores were calculated for all cases and analyzed to establish correlation patterns between confidence levels and classification accuracy. This analysis informs the development of clinical decision thresholds that can guide appropriate levels of human oversight in deployment scenarios.

9.5 RESULTS

In this section, we present findings from our evaluation approach, examining system performance, consistency, usability, risk thresholds, and confidence mechanisms.

9.5.1 *Evaluation Using the Synthetic Dataset*

Performance results using the synthetic dataset are reported below.

9.5.1.1 *GPT-4o Performance Analysis*

Our evaluation with GPT-4o, averaged over five runs to account for potential LLM non-deterministic behavior, demonstrates high system performance. The Fleiss kappa score across all iterations showed perfect agreement (100%) [114, 334] with a standard deviation of 0.000 for both category and reaction types classifications. This perfect agreement confirms that our system exhibited deterministic behavior, producing identical classification outputs across all five runs despite the potential for LLM-based variability.

Table 9.9: Classification Results per Category (averaged over five runs).

Case Category	Precisio	Recall	F1	Cases
Chemical-based cross-reactivity to excipients	0.9804	1.0000	0.9901	50
Direct active ingredient reactivity	1.0000	1.0000	1.0000	30
Direct excipient reactivity	1.0000	0.9583	0.9787	24
Drug class cross-reactivity with documented tolerance	1.0000	1.0000	1.0000	355
Drug class cross-reactivity without documented tolerance	1.0000	1.0000	1.0000	500
No documented reactions or intolerances	0.9167	1.0000	0.9565	11
No reactivity to prescribed drug's ingredients or excipients	1.0000	0.9667	0.9831	30
Macro Average	**0.9853**	**0.9893**	**0.9869**	**1,000**

Regarding the category classification, the system achieved excellent performance, with an overall macro-averaged precision of 0.9853, recall of 0.9893, and F1 score of 0.9869 (see Table 9.9). Notably, perfect classification (precision = 1.0000, recall = 1.0000, F1 = 1.0000) was achieved for three critical categories: "Direct Active Ingredient Reactivity," "Drug Class Cross-Reactivity with Doc-

umented Tolerance," and "Drug Class Cross-Reactivity Without Documented Tolerance." This perfect performance is particularly significant for the cross-reactivity categories, which comprised the majority of cases in the dataset (855 out of 1,000 cases).

The "Chemical-Based Cross-Reactivity to Excipients" category showed excellent performance with a precision of 0.9804, perfect recall of 1.0000, and F1 score of 0.9901. Similar high performance was observed for "Direct Excipient Reactivity" (precision = 1.0000, recall = 0.9583, F1 = 0.9787), with only one case misclassified as "Chemical-Based Cross-Reactivity to Excipients." Importantly, this misclassification occurred between categories with similar non-life-threatening, non-immune-mediated reaction profiles.

For the "No Documented Reactions or Intolerances" category, the system achieved a precision of 0.9167 and perfect recall (1.0000), resulting in an F1 score of 0.9565. Similarly, "No Reactivity to Prescribed Drug's Ingredients or Excipients" showed perfect precision (1.0000) with slightly lower recall (0.9667) and a high F1 score of 0.9831, with one case misclassified as "No Documented Reactions or Intolerances" - both categories representing scenarios with no adverse reactions.

This pattern of errors suggests that while the system occasionally struggles to differentiate between certain similar non-serious reaction types, it maintains robust performance for scenarios with more serious clinical implications.

Table 9.10: Classification Results per Reaction Type (averaged over five runs).

Reaction Type	Precision	Recall	F1	Cases
None	1.0000	1.0000	1.0000	396
Life-threatening	1.0000	1.0000	1.0000	133
Non life-threatening immune-mediated	1.0000	1.0000	1.0000	296
Non life-threatening non immune-mediated	1.0000	1.0000	1.0000	175
Macro Average	**1.0000**	**1.0000**	**1.0000**	**1,000**

The evaluation of reaction type identification (see Table 9.10) showed perfect accuracy in distinguishing between different severity levels of adverse drug reactions, from life-threatening cases to non-immune-mediated reactions.

Table 9.11: Classification Results per Alert Type (averaged over five runs).

Alert Type	Ground Truth (%)	Heliot (%)	Traditional Systems (%)
No Alert Needed	396 (39.6%)	396 (39.6%)	41 (4.1%)
Interruptive Alert	429 (42.9%)	429 (42.9%)	959 (95.9%)
Non-Interruptive Alert	175 (17.5%)	175 (17.5%)	0 (0%)

Regarding alert generation (see Table 9.11), our approach suggests improvements over traditional CDSSs, which tend to favor interruptive alerts [64, 136, 288, 289]. By distinguishing between cases requiring interruptive alerts (39.6%), those where non-interruptive alerts might suffice (17.5%), and situations where no alert is needed (39.6%), HELIOT shows promise in addressing alert fatigue by potentially reducing interruptive alerts by 50.3% (395 - 41 No Alert Needed + 175 Non-Interruptive Alert) compared to traditional systems. However, these results need validation in real clinical settings, where alert override rates typically exceed 90%.

Another key aspect derived from analyzing the results is the system's efficiency. The average execution time was 2.775 seconds per patient to see the overall response on the web application. Importantly, the response is streamed in real-time, similar to how ChatGPT functions, so healthcare professionals can start reading the suggestions immediately as they are generated without significant delays.

9.5.1.2 *Gemma 3 Performance Analysis*

Gemma 3 achieved an overall accuracy of 99.80% on the synthetic dataset, averaged over five runs, with macro-averaged precision of 0.9890, recall of 0.9924, and F1 score of 0.9904 (see Table 9.12). The Fleiss kappa score across all iterations confirmed perfect agreement (100%) and that our system exhibited deterministic behavior.

Regarding category classification, Gemma 3 reached perfect classification (precision = 1.0000, recall = 1.0000, F1 = 1.0000) for four critical categories: "Direct Active Ingredient Reactivity," "Drug

Table 9.12: Gemma3: Classification Results per Category.

Case Category	Precisio	Recall	F1	Cases
Chemical-based cross-reactivity to excipients	1.0000	0.9800	0.9899	50
Direct active ingredient reactivity	1.0000	1.0000	1.0000	30
Direct excipient reactivity	0.9231	1.0000	0.9600	24
Drug class cross-reactivity with documented tolerance	1.0000	1.0000	1.0000	355
Drug class cross-reactivity without documented tolerance	1.0000	1.0000	1.0000	500
No documented reactions or intolerances	1.0000	1.0000	1.0000	11
No reactivity to prescribed drug's ingredients or excipients	1.0000	0.9667	0.9831	30
Macro Average	**0.9890**	**0.9924**	**0.9904**	**1,000**

Class Cross-Reactivity with Documented Tolerance," "Drug Class Cross-Reactivity without Documented Tolerance," and "No Documented Reactions or Intolerances." The "Chemical-Based Cross-Reactivity to Excipients" category showed strong performance with perfect precision (1.0000), recall of 0.9800, and F1 score of 0.9899, with one case misclassified. For "Direct Excipient Reactivity," the system achieved perfect recall (1.0000) with a precision of 0.9231 and an F1 score of 0.9600, showing 2 cases overclassified from other categories—indicating a conservative approach that prioritizes safety. The "No Reactivity to Prescribed Drug's Ingredients or Excipients" category demonstrated perfect precision (1.0000) with a recall of 0.9667 and an F1 score of 0.9831, with one case misclassified as a different non-reaction category.

Table 9.13: Gemma 3: Classification Results per Reaction Type.

Reaction Type	Precision	Recall	F1	Cases
None	1.0000	0.9242	0.9606	396
Life-threatening	1.0000	1.0000	1.0000	133
Non life-threatening immune-mediated	0.9966	1.0000	0.9983	296
Non life-threatening non immune-mediated	0.8578	1.0000	0.9235	175
Macro Average	**0.9636**	**0.9811**	**0.9706**	**1,000**

The evaluation of reaction type identification (see Table 9.13) showed strong performance in distinguishing between severity levels, with perfect classification for life-threatening cases (precision = 1.0000, recall = 1.0000, F1 = 1.0000). Non-life-threatening immune-mediated reactions achieved near-perfect performance (precision = 0.9966, recall = 1.0000, F1 = 0.9983). For non-life-threatening, immune-mediated reactions, the system showed per-

fect recall (1.0000) with a precision of 0.8578 and an F1 score of 0.9235, indicating a tendency toward conservative classification. The "None" category achieved perfect precision (1.0000) with a recall of 0.9242 and an F1 score of 0.9606.

Table 9.14: Gemma3: Classification Results per Alert Type.

Alert Type	Ground Truth (%)	Heliot (%)	Traditional Systems (%)
No Alert Needed	396 (39.6%)	366 (36.6%)	41 (4.1%)
Interruptive Alert	429 (42.9%)	430 (43.0%)	959 (95.9%)
Non-Interruptive Alert	175 (17.5%)	204 (20.4%)	0 (0%)

For alert generation (see Table 9.14), Gemma 3 demonstrated effective alert management with 36.6% cases requiring no alerts, 20.4% non-interruptive alerts, and 43.0% interruptive alerts. This distribution suggests a 50.0% reduction in unnecessary interruptive alerts compared to traditional CDSSs while maintaining perfect identification of critical cases requiring immediate attention. The slightly more conservative approach compared to GPT-4o (36.6% vs 39.6% no-alert cases) reflects Gemma 3's tendency to err on caution, which may be advantageous in clinical settings where patient safety is paramount. Gemma 3 evaluation was conducted using local deployment, achieving an average response time of 6.73 seconds per patient case. These timing results are provided for reference purposes to demonstrate the feasibility of local deployment scenarios where data privacy and infrastructure constraints require on-premises solutions.

9.5.1.3 *Claude Sonnet Performance Analysis*

Claude Sonnet 4 achieved an overall accuracy (averaged over 5 runs) of 0.9980 with macro-averaged precision of 0.9911, recall of 0.9992, and F1 score of 0.9950, showing consistent performance compared to GPT-4o (see Table 9.15). Like GPT-4o and Gemma3, the Fleiss kappa score was 100%, confirming that HELIOT provides deterministic responses.

In category classification, the LLM reached perfect performance (precision = 1.0000, recall = 1.0000, F1 = 1.0000) for five out of

Table 9.15: Claude Sonnet 4.0: Classification Results per Category.

Case Category	Precisio	Recall	F1	Cases
Chemical-based cross-reactivity to excipients	1.0000	1.0000	1.0000	50
Direct active ingredient reactivity	1.0000	1.0000	1.0000	30
Direct excipient reactivity	1.0000	1.0000	1.0000	24
Drug class cross-reactivity with documented tolerance	1.0000	0.9944	0.9972	355
Drug class cross-reactivity without documented tolerance	1.0000	1.0000	1.0000	500
No documented reactions or intolerances	1.0000	1.0000	1.0000	11
No reactivity to prescribed drug's ingredients or excipients	0.9375	1.0000	0.9677	30
Macro Average	**0.9911**	**0.9992**	**0.9950**	**1,000**

seven categories, including all direct reactivity categories and most cross-reactivity scenarios. The "Drug Class Cross-Reactivity with Documented Tolerance" category showed excellent performance with perfect precision (1.0000) and high recall (0.9944), resulting in an F1 score of 0.9972. Only 2 out of 355 cases were misclassified in this category. The "No Reactivity to Prescribed Drug's Ingredients or Excipients" category achieved perfect recall (1.0000) with a precision of 0.9375 and an F1 score of 0.9677. The system correctly identified all true cases but generated two false positives, representing a conservative approach that favors patient safety over precision.

Table 9.16: Claude Sonnet 4.0: Classification Results per Reaction Type.

Reaction Type	Precision	Recall	F1	Cases
None	1.0000	1.0000	1.0000	396
Life-threatening	1.0000	1.0000	1.0000	133
Non life-threatening immune-mediated	1.0000	1.0000	1.0000	296
Non life-threatening non immune-mediated	1.0000	1.0000	1.0000	175
Macro Average	**1.0000**	**1.0000**	**1.0000**	**1,000**

The evaluation of reaction type classification showed perfect accuracy across all severity levels, achieving 100% precision, recall, and F1 scores for all categories (see Table 9.16), which is particularly significant for clinical safety.

Regarding alert generation, Claude Sonnet correctly identified all 396 cases requiring no alerts, 175 cases suitable for non-interruptive alerts, and 429 cases requiring interruptive alerts. This result represents a potential 53.0% reduction in alert burden compared to traditional CDSSs, identical to the one observed with GPT-4o. The

Table 9.17: Claude Sonnet 4.0: Classification Results per Alert Type.

Alert Type	Ground Truth (%)	Heliot (%)	Traditional Systems (%)
No Alert Needed	396 (39.6%)	396 (39.6%)	41 (4.1%)
Interruptive Alert	429 (42.9%)	429 (42.9%)	959 (95.9%)
Non-Interruptive Alert	175 (17.5%)	175 (17.5%)	0 (0%)

average execution time was 4.37 seconds per patient, representing a 57% increase compared to GPT-4o (2.775 seconds) but still maintaining clinically acceptable response times for decision support applications.

9.5.2 *Evaluation Using the Real-World Dataset*

Validation results employing the real-world dataset are presented in the following sections.

9.5.2.1 *GPT-4o Performance Analysis*

Using GPT-4o as the underlying language model, HELIOT demonstrated robust performance on the real-world dataset, achieving an overall accuracy of 99.38% with macro precision of 100.00%, recall of 99.28%, and F1-score of 99.63% (see Table 9.18). The system correctly classified all 114 patients without documented reactions and achieved perfect performance for drug class cross-reactivity with documented tolerance cases. For patients with no reactiv-

Table 9.18: GPT-4o Classification Results per Category on Real-World Dataset.

Case Category	Precisio	Recall	F1	Cases
No documented reactions or intolerances	1.0000	1.0000	1.0000	114
Drug class cross-reactivity with documented tolerance	1.0000	1.0000	1.0000	2
Drug class cross-reactivity without documented tolerance	0.0000	0.0000	0.0000	1
No reactivity to prescribed drug's ingredients or excipients	1.0000	0.9783	0.989c	45
Macro Average	**1.0000**	**0.9928**	**0.9963**	**162**

ity to prescribed drug ingredients or excipients, GPT-4o achieved

97.83% recall with one misclassification out of 46 cases. This single error resulted in a conservative over-classification where the system predicted drug class cross-reactivity without documented tolerance, representing a safety-first approach that avoids missed reactions. For reaction type classification (see Table 9.19), HELIOT

Table 9.19: Classification Results per Reaction Type for GPT-4o on Real-World Dataset.

Reaction Type	Precision	Recall	F1	Cases
None	1.0000	0.9938	0.9969	161
Non life-threatening immune-mediated	0.0000	0.0000	0.0000	1
Macro Average	**1.0000**	**0.9938**	**0.9969**	**162**

achieved similar high performance, with an overall accuracy of 99.38%. The system correctly identified that the vast majority of cases (161 out of 162) required no reaction alerts, with only one case being conservatively over-classified as requiring a non life-threatening immune-mediated reaction alert. The alert type classification results (see Table 9.20) mirror the reaction type performance, with 99.38% accuracy in determining appropriate alert levels. HELIOT correctly identified that 161 out of 162 cases required no alert, with one case being conservatively flagged for an interruptive alert. This conservative approach aligns with clinical safety requirements, where false positives are preferable to missed potential adverse reactions.

Table 9.20: Classification Results per Alert Type for GPT-4o on Real-World Dataset.

Alert Type	Ground Truth (%)	Heliot (%)	Traditional Systems (%)
No Alert Needed	162 (100%)	161 (99.4%)	160 (98.76%)
Interruptive Alert	0 (0%)	1 (0.6%)	2 (1.2%)
Non-Interruptive Alert	0 (0%)	0 (0%)	0 (0%)

In terms of response time, the evaluation corroborates the initial results, with 2.3 seconds on average per clinical case classification.

This real-world evaluation with GPT-4o confirms several critical aspects of the proposed CDSS clinical deployment readiness.

GPT-4o successfully processed Italian clinical notes through HELIOT's linguistic standardization approach, demonstrating effectiveness across different languages and medical terminology systems. The model's natural language processing capabilities handled the documentation styles, abbreviations, and clinical terminology in actual patient records, showing strong adaptability to real-world documentation practices.

9.5.2.2 *Gemma 3 Performance Analysis*

The deployment of Google's Gemma 3 (12B parameters) within the HELIOT framework yielded strong performance metrics on the real-world dataset (see Table 9.21), with an overall accuracy of 98.77% and macro-averaged metrics of precision 100.00%, recall 98.58%, and F1-score 99.28%. While maintaining high overall performance, Gemma 3 exhibited distinct behavioral patterns compared to GPT-4o in handling edge cases and classification boundaries. Gemma 3 demonstrated perfect precision across all popu-

Table 9.21: Classification Results per Category - Gemma 3 Real-World Evaluation.

Case Category	Precisio	Recall	F1	Cases
Drug class cross-reactivity with documented tolerance	1.0000	1.0000	1.0000	2
No documented reactions or intolerances	1.0000	1.0000	1.0000	113
No reactivity to prescribed drug's ingredients or excipients	1.0000	0.9574	0.9783	45
Drug class cross-reactivity without documented tolerance	0	0	0	2
Macro Average	**1.0000**	**0.9858**	**0.9928**	**162**

lated categories, correctly identifying all 113 patients with no documented reactions and both cases of drug class cross-reactivity with documented tolerance. However, the model showed a more conservative classification approach, with 2 cases from the "no reactivity to prescribed ingredients or excipients" category being over-classified as "drug class cross-reactivity without documented tolerance." This behavior resulted in a recall of 95.74% for the former category, representing a more cautious stance that prioritizes potential risk identification over classification precision.

For reaction type classification, Gemma 3 achieved 98.77% accuracy with perfect precision (100.00%) and recall of 98.77% (see Table 9.22). The model correctly identified most cases with no ad-

verse reactions (160 out of 162 cases) but conservatively classified 2 cases as life-threatening reactions. This represents a more alert-prone behavior compared to GPT-4o, reflecting Gemma 3's tendency toward risk-averse classification in ambiguous scenarios.

Table 9.22: Classification Results per Reaction Type - Gemma 3 Real-World Evaluation.

Reaction Type	Precision	Recall	F1	Cases
None	1.0000	0.9877	0.9938	160
Life-threatening	0	0	0	2
Macro Average	**1.0000**	**0.9877**	**0.9938**	**162**

From a clinical safety perspective, Gemma 3's conservative approach generated two interruptive alerts (1.2% of cases) compared to GPT-4o's single alert, indicating a slightly more alert-prone behavior (see Table 9.23). The alert type classification mirrors the reaction classification performance, with 98.77% accuracy and perfect precision for correctly identified cases requiring no alert (160 out of 162). While this result led to no reduction in alerts compared to traditional CDSSs for this specific dataset, the model maintained zero false negatives for critical reactions, thereby preserving the essential safety profile required for clinical deployment.

Table 9.23: Classification Results per Alert Type - Gemma 3 Real-World Evaluation.

Alert Type	Ground Truth (%)	Heliot (%)	Traditional Systems (%)
No Alert Needed	162 (100%)	160 (98.8%)	160 (98.8%)
Interruptive Alert	0 (0%)	2 (1.2%)	2 (1.2%)
Non-Interruptive Alert	0 (0%)	0 (0%)	0 (0%)

Processing efficiency with Gemma 3 was comparable to GPT-4o, averaging 2.51 seconds per response. The model benefited from the dataset's characteristics, where many patients reported no allergies, enabling more rapid processing of straightforward cases. Gemma 3's performance profile suggests a model architecture that errs on the side of caution, potentially valuable in clinical

contexts where false positives are preferable to missed reactions, albeit at the cost of slightly increased alert frequency compared to GPT-4o's more balanced approach.

9.5.2.3 *Claude Sonnet Performance Analysis*

Claude Sonnet's on the real-world dataset achieved excellent results, with 99.38% accuracy, perfect macro precision (100.00%), and robust recall (99.29%) and F1-score (99.64%) (see Table 9.24).

Table 9.24: Classification Results per Category - Claude Sonnet Real-World Evaluation.

Case Category	Precisio	Recall	F1	Cases
Drug class cross-reactivity with documented tolerance	1.0000	1.0000	1.0000	2
Drug class cross-reactivity without documented tolerance	0.0000	0.0000	0.0000	1
No documented reactions or intolerances	1.0000	1.0000	1.0000	113
No reactivity to prescribed drug's ingredients or excipients	1.0000	0.9787	0.9892	46
Macro Average	**1.0000**	**0.9929**	**0.9964**	**162**

The model exhibited a single but consistent error pattern while maintaining perfect accuracy for the majority classes—correctly identifying all 113 patients with no documented reactions and both cases of drug class cross-reactivity with documented tolerance. One case that should have been classified as "No Reactivity to Prescribed Drug's Ingredients or Excipients" was instead categorized as "Drug Class Cross-Reactivity Without Documented Tolerance," triggering an unnecessary interruptive alert.

Claude Sonnet attained 99.38% accuracy for reaction type classification, with perfect precision (100.00%) and recall of 99.38% (see Table 9.25). The model correctly identified most cases with no adverse reactions (161 out of 162 cases) but classified one case as non life-threatening immune-mediated reaction. This conservative approach mirrors the pattern observed in the category classification, demonstrating consistent risk-averse behavior.

Nonetheless, the model obtained the same GPT-4o results in terms of alert distribution: 161 cases (99.4%) requiring no alerts and 1 case (0.6%) generating an interruptive alert (see Table 9.26).

From a computational efficiency perspective, Claude Sonnet averaged 1.94 seconds per clinical case compared to GPT-4o's 2.3-

Table 9.25: Classification Results per Reaction Type - Claude Sonnet Real-World Evaluation.

Reaction Type	Precision	Recall	F1	Cases
None	1.0000	0.9938	0.9969	161
Non Life-threatening Immune-Mediated	0	0	0	1
Macro Average	**1.0000**	**0.9938**	**0.9969**	**162**

Table 9.26: Classification Results per Alert Type - Claude Sonnet Real-World Evaluation.

Alert Type	Ground Truth (%)	Heliot (%)	Traditional Systems (%)
No Alert Needed	162 (100%)	161 (99.4%)	160 (98.76%)
Interruptive Alert	0 (0%)	1 (0.6%)	2 (1.2%)
Non-Interruptive Alert	0 (0%)	0 (0%)	0 (0%)

second average on this dataset. However, this contrasts with synthetic dataset evaluation where Claude was approximately 57% slower than GPT-4o, suggesting variable performance depending on case complexity.

The model's effective handling of Italian clinical documentation, with performance nearly identical to GPT-4o, establishes it as a reliable alternative for clinical decision support while maintaining the same conservative safety profile.

9.5.3 *Clinical Usability Evaluation Results*

The clinical usability evaluation, conducted with nine healthcare professionals, demonstrated a positive reception of HELIOT's contextual decision support approach. Clinical appropriateness received a mean score of 4.97 out of 5 across all seven clinical cases, with risk assessment accuracy achieving unanimous perfect ratings of 5.0, indicating complete agreement with the system's clinical reasoning and safety classifications. This high clinical acceptance translated into a unanimous preference for HELIOT's alert approach over traditional CDSS across all scenarios. The system's ability to reduce alert fatigue was particularly well-received, with 67% of participants rating the impact as "significantly reduces

alert fatigue" and 33% as "reduces alert fatigue." Supporting this effectiveness, participants considered the system's contextual understanding as "excellent" (78%) or "good" (22%), while information clarity received "very clear and easy to understand" ratings from 89% of participants. The system's transparency and trustworthiness further reinforced clinical acceptance. Clinical reasoning transparency received consistently high ratings, with 89% rating it as "excellent explanation of reasoning" and 11% rating it as "good explanation of reasoning." This transparency contributed to high confidence levels, with 44% of participants rating themselves as "very confident" and 56% as "confident" in HELIOT's recommendations. These assessments directly influenced adoption intentions and perceived workflow benefits. Participants judged HELIOT's clinical decision support value highly, with 44% considering it "extremely valuable" and 56% considering it "very valuable." Correspondingly, the adoption likelihood was positive, with 78% indicating they would be "very likely" to adopt the system and 22% rating it as "likely." Time efficiency expectations aligned with these intentions, as 78% indicated that HELIOT would "significantly reduce time spent," and 22% noted it would "moderately reduce time spent."

Table 9.27 summarizes the results across all evaluated dimensions, demonstrating that HELIOT's contextual approach addresses key limitations of traditional CDSS systems while maintaining appropriate clinical safety standards.

Table 9.27: Clinical Usability Evaluation Results Summary.

Evaluation Dimension	Scale	Mean Rating
Clinical Appropriateness	1-5 (Inappropriate-Appropriate)	4.97
Risk Assessment Accuracy	1-5 (Inaccurate-Accurate)	5.0
Alert Preference	HELIOT vs Traditional	100% HELIOT
Alert Fatigue Impact	Increase-Significantly Reduce	Significantly Reduce (67%)
Alert Contextuality	No-Excellent Understanding	Excellent (78%)
Information Clarity	1-5 (Unclear-Clear)	5.0
Clinical Reasoning Transparency	1-5 (No-Excellent Explanation)	4.89
Time Efficiency Impact	Increase-Significantly Reduce	Significantly Reduce (78%)
Trust and Confidence	1-5 (Not-Very Confident)	4.44
Clinical Decision Support Value	1-5 (Not-Extremely Valuable)	4.44
Likelihood of Adoption	1-5 (Unlikely-Very Likely)	4.78

9.5.4 *Safety Analysis and Risk Assessment*

Safety analysis revealed remarkable profiles across both datasets. For the synthetic dataset, GPT-4o correctly identified all 133 life-threatening cases, maintaining optimal sensitivity and specificity parameters (1.0000) for critical events and detection scenarios. Gemma 3 achieved comparable results with zero undetected potential critical reactions. Despite a modest increase in false alarm rate (3.10%) compared to GPT-4o's flawless performance, it preserved the essential safety threshold with perfect sensitivity (1.0000) for dangerous reactions and high specificity (0.9982). Claude Sonnet reached 100% across all metrics, eliminating false negatives and positives.

The analysis using the real-world dataset confirmed HELIOT robustness in authentic clinical scenarios while maintaining zero unidentified critical events. GPT-4o correctly classified 99.4% of cases as not requiring intervention, with a single false positive generating a conservative false alarm rate of 1.23% and a perfect negative predictive value (1.0000). Gemma 3 showed analogous characteristics with 98.8% correct classifications. It presented a slightly higher false alarm rate (2.47%) with two false positives while preserving the fundamental profile of zero critical omissions. Claude Sonnet matched GPT-4o's performance with 99.4% of cases appropriately classified and a single false positive, maintaining perfect negative predictive value and zero missed critical events. The consistency of results across different architectures confirms the robustness of the HELIOT framework (see Figure 9.9). In both evaluations, all LLMs met the indispensable requirement of zero omissions for critical events. Minimal variations in false positive rates (0.00%- 3.10%) reflect different degrees of caution without compromising patient safety. Alert distribution showed uniformity, with all systems identifying most real cases as not requiring intervention (98.8%-99.4%), demonstrating that HELIOT's approach to alert fatigue reduction preserves safety standards regardless of the underlying architecture. This multi-model validation establishes reference parameters for clinical implementation. The consistent rate of zero false negatives supports the adoption of automated decisions for "no intervention required" classifications. Acceptable rates for false positives provide a prudential

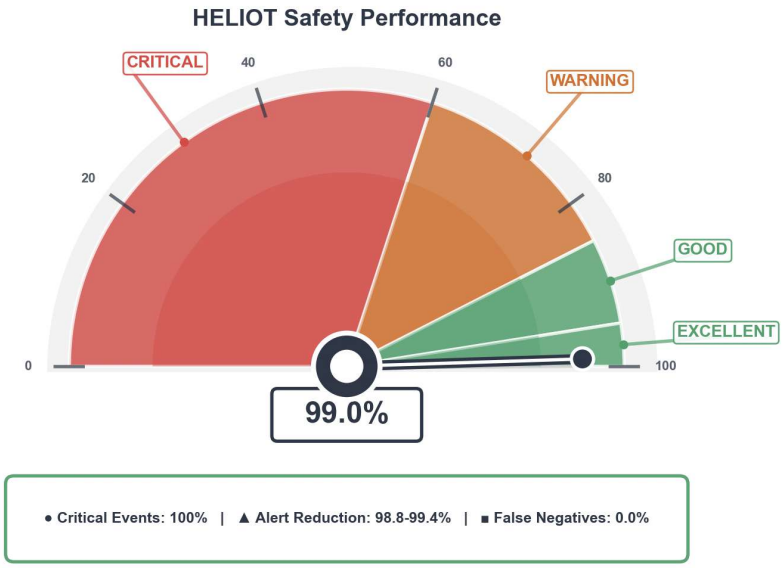


Figure 9.9: HELIOT Safety Performance Dashboard.

margin that ensures appropriate clinical attention to potentially problematic cases while significantly reducing the burden of superfluous notifications.

9.5.5 *Confidence Analysis*

The confidence scoring analysis (see Table 9.28) across all three LLMs revealed distinct patterns that provide valuable insights for risk-stratified clinical deployment. Each model exhibits unique confidence characteristics while maintaining consistent safety profiles.

GPT-4o demonstrated robust confidence levels across both evaluation scenarios. In the synthetic dataset, the model exhibited high mean confidence scores: overall confidence (0.855), analysis confidence (0.791), case confidence (0.999), and reaction confidence (0.994). The two observed misclassifications maintained similar confidence patterns (overall: 0.853, analysis: 0.788, case: 0.999, reaction: 0.994), occurring between clinically similar categories with non-life-threatening profiles. For real-world validation, GPT-4o showed comparable patterns with correctly classi-

Table 9.28: Confidence Analysis Results Across LLMs and Datasets.

Model	Dataset	Classificatio	Overall Conf.	Analysis Conf.	Case Conf.	Reaction Conf.	Cases (n)
GPT-4o	Synthetic	Correct	0.855	0.791	0.999	0.994	998
		Incorrect	0.853	0.788	0.999	0.994	2
	Real-world	Correct	0.870	0.745	0.999	0.999	161
		Incorrect	0.819	0.747	0.975	0.966	1
Gemma 3	Synthetic	Correct	0.910	0.732	0.999	0.997	998
		Incorrect	0.901	0.725	0.979	0.999	2
	Real-world	Correct	0.899	0.721	0.986	0.997	160
		Incorrect	0.789	0.615	0.766	0.997	2
Claude Sonnet	Synthetic	Correct	0.815	0.786	0.990	0.950	998
		Incorrect	0.797	0.786	0.999	0.999	2
	Real-world	Correct	0.873	0.767	0.988	0.999	161
		Incorrect	0.796	0.786	0.999	0.999	1

fied cases achieving mean confidence scores of: overall confidence (0.870), analysis confidence (0.745), case confidence (0.999), and reaction confidence (0.999). The single misclassification revealed a notable confidence drop: overall confidence (0.819), analysis confidence (0.747), case confidence (0.975), and reaction confidence (0.966). Critically, the analysis confidence of 0.747 falls within the medium-risk threshold range, which would appropriately trigger clinical review protocols.

Gemma 3 exhibited the highest overall confidence levels in synthetic evaluation with mean scores of overall confidence (0.910), analysis confidence (0.732), case confidence (0.999), and reaction confidence (0.997). Misclassifications showed confidence degradation patterns, where overall confidence (0.897-0.905), analysis confidence (0.713-0.736), case confidence (0.979), and reaction confidence maintained near-perfect levels (0.999). In real-world scenarios, Gemma 3 demonstrated consistent confidence patterns with mean scores of overall confidence (0.899), analysis confidence (0.721), case confidence (0.986), and reaction confidence (0.997). The model's two misclassifications revealed more pronounced confidence drops: overall confidence (0.772-0.806), analysis confidence (0.604-0.626), and notably lower case confidence (0.715-0.816), indicating the model's uncertainty recognition capabilities.

Claude Sonnet displayed unique confidence patterns with moderate overall confidence (0.815) but strong analysis confidence (0.786) in synthetic evaluation. Case confidence (0.990) and reac-

tion confidence (0.950) remained high. Notably, the model's rare misclassifications maintained high case confidence (0.999) and reaction confidence (0.999), suggesting different uncertainty expression mechanisms compared to other models. Real-world evaluation showed improved confidence levels: overall confidence (0.873), analysis confidence (0.767), case confidence (0.988), and reaction confidence (0.999). The single misclassification maintained high confidence across most metrics (overall: 0.796, analysis: 0.786, case: 0.999, reaction: 0.999), indicating the model's consistent decision-making approach even in uncertain scenarios. The multi-model analysis validates a risk threshold framework that accounts for varying confidence expression patterns across different architectures. While GPT-4o shows apparent confidence degradation in uncertain cases, Gemma 3 demonstrates more pronounced uncertainty signals, and Claude Sonnet maintains consistent high confidence even during misclassifications.

Based on the confidence analysis across all models and datasets, we propose implementing the risk threshold framework detailed in Table 9.29. This framework establishes four risk levels based on confidence thresholds: cases with case confidence above 0.95 and analysis confidence above 0.75 qualify for automated implementation, case confidence between 0.80-0.95 or analysis confidence between 0.60-0.75 trigger non-interruptive notifications for clinical awareness, case confidence below 0.80 or analysis confidence below 0.60 requires mandatory clinical review, and any confidence metric below 0.50 demands immediate clinical evaluation with override prevention.

This multi-dimensional confidence assessment approach accommodates the distinct uncertainty expression patterns observed across models while maintaining the demonstrated alert reduction capabilities. The framework ensures appropriate clinical oversight for uncertain decisions regardless of the underlying language model architecture, supporting robust clinical deployment across different institutional preferences for model selection.

9.5.6 *Research Question Assessment*

The evaluation highlights that HELIOT effectively addresses the core challenges of clinical decision support for drug administra-

Table 9.29: Risk Threshold Framework for Clinical Decision Support.

Confidence Range	Risk Level	Clinical Action	System Response
Case Confidence > 0.95	Low Risk	Automated decision implementation	Direct alert delivery or no alert as determined
Analysis Confidence > 0.75		Proceed without interruption	Normal workflow continuation
Case Confidence 0.80-0.95	Medium Risk	Non-interruptive notification	Background alert with rationale
Analysis Confidence 0.60-0.75		Clinical awareness recommended	Passive information display
Case Confidence < 0.80	High Risk	Mandatory clinical review	Interruptive alert requiring acknowledgment
Analysis Confidence < 0.60		Human verification required	System recommendation with uncertainty flag
Any confidence metric < 0.50	Critical Risk	Immediate clinical evaluation	Override prevention until review
Multiple low confidence scores		Expert consultation recommended	Escalation to senior clinician

tion. Across all tested language models, the system obtained high classification accuracy (98.77%- 99.80%) while maintaining a perfect safety profile with zero false negatives for life-threatening reactions. Most significantly, HELIOT reduced unnecessary interruptive alerts by 50-53% compared to traditional CDSSs, directly addressing the alert fatigue problem that compromises patient safety. The system's contextual approach enables precise risk stratification, supported by a confidence-based framework that ensures appropriate clinical oversight for uncertain cases. A clinical usability evaluation confirmed strong acceptance among healthcare professionals, with unanimous preference for HELIOT's alerts over traditional systems. Response times (1.94-6.73 seconds) remain clinically acceptable for real-time decision support. These results align with Co et al. [57], who demonstrated that traditional CDSSs achieve performance improvements through overalerting, creating alert fatigue that undermines clinical effectiveness. HELIOT's contextual understanding successfully distinguishes between necessary and unnecessary alerts, addressing this fundamental trade-off.

Based on the evaluation results presented above, we can address our research question:

RQ9.1: How effective is HELIOT CDSS in identifying potential drug reactions and reducing alert fatigue?

HELIOT can provide effective support for drug prescription through accurate identification of potential adverse drug reactions while reducing alert fatigue. Clinical validation in operational environments remains necessary to confirm practical deployment effectiveness.

9.6 IMPLICATIONS, LIMITATIONS AND FUTURE WORK

This section discusses the practical implications of our findings and outlines limitations and future research directions.

9.6.1 *Practical Implications and Future Work*

The initial results of HELIOT CDSS suggest advancements in addressing key challenges in adverse drug reaction management, particularly in healthcare settings where clinical information exists primarily in unstructured formats. By leveraging LLMs to interpret clinical narratives and generate contextual alerts, HELIOT demonstrates an approach to reduce alert fatigue while maintaining safety, a significant improvement over traditional rule-based systems that typically generate non-specific alerts for all potentially cross-reactive medications.

From an implementation perspective, HELIOT's ability to process unstructured clinical notes and generate appropriate alerts makes it particularly valuable across diverse healthcare settings. While some environments have advanced EHR systems that could feed structured data into HELIOT's database, many healthcare facilities worldwide still operate with basic or no EHR infrastructure, relying primarily on unstructured clinical notes. HELIOT's flexible architecture accommodates both scenarios: it can process data from existing EHR systems where available while functioning independently using clinical narratives in settings with limited technological infrastructure. This versatility and its microservices architecture and streaming response mechanism facilitate adaptation to different clinical workflows, from specialized hospitals to primary care practices. Moreover, the system's capability

to maintain patient clinical histories could enhance continuity of care across different healthcare settings.

The impact on patient safety and healthcare costs may be significant. By improving the accuracy of adverse drug reaction alerts and reducing alert fatigue, HELIOT could help prevent ADEs while ensuring that critical warnings are not overlooked. All this leads to fewer medication errors, reduced hospital readmissions, and shorter hospital stays. The economic implications of these improvements could be substantial, as medication errors and ADEs impose significant costs on healthcare systems. The resources saved could be redirected to other critical aspects of patient care.

Looking ahead, several crucial directions for future work emerge. Large-scale clinical validation represents our immediate priority. While our pilot study with 162 real-world cases provides encouraging initial evidence, comprehensive validation requires larger datasets across diverse healthcare settings, different languages, and varied clinical documentation practices. To fully assess its clinical utility, future studies must evaluate HELIOT's performance with higher rates of contraindicated prescriptions and more complex adverse reaction scenarios. Additionally, conducting prospective clinical trials will be essential to measure the system's impact on clinical workflow, alert override rates, and patient outcomes in live healthcare environments.

Technical enhancement and optimization efforts will focus on improving system performance and deployment flexibility. We plan to systematically evaluate alternative LLM architectures, including open-source models such as LLaMA and FLAN-T5 and specialized medical LLMs like ClinicalBERT and BioBERT. This evaluation will assess accuracy metrics and factors critical for clinical deployment, including response consistency, computational requirements, and cost-effectiveness. Parallel development will implement intelligent caching mechanisms for frequently queried drug-reaction combinations and explore model distillation techniques to reduce latency while maintaining clinical decision support quality.

Extended clinical applications will leverage HELIOT's LLM-based architecture to address broader healthcare challenges beyond adverse drug reactions. Future research will explore expanding into diagnostic assistance, treatment protocol recommendations, and

medication dosing optimization. We will investigate integrating patient-reported outcomes to enhance personalized recommendations by processing patient narratives about medication effectiveness and impact on quality of life. Additionally, implementing continuous learning mechanisms through real-world clinical feedback could enable HELIOT to adapt to evolving clinical practices and institution-specific prescribing patterns, creating a self-improving system that learns from clinical outcomes and prescriber experiences.

9.6.2 *Limitations*

Despite the promising results, HELIOT presents a few limitations that we have addressed through various mitigation strategies. The primary limitation of our evaluation concerns the scale of real-world validation. While we expanded beyond synthetic data to include 162 real-world patient cases, this represents a limited sample size for comprehensive clinical validation. To address this, we conducted rigorous validation and demonstrated consistent performance across three different LLM architectures (GPT-4o, Gemma 3, and Claude Sonnet), showing the robustness of our approach. Additionally, we achieved perfect safety profiles with zero false negatives for critical reactions across all models, establishing a foundation for larger-scale studies. We plan to conduct extensive multi-center validation studies with larger patient cohorts to validate these findings further.

Another problem relates to the geographic and linguistic scope of our evaluation, which focused primarily on Italian clinical documentation and European pharmaceutical standards. We managed this limitation by demonstrating that our framework successfully processes different documentation styles and clinical terminologies, and by designing the system architecture to support multiple languages and drug databases. The modular design enables easy adaptation to various healthcare systems and regulatory frameworks.

The dependency on external LLM services for cloud-based models (GPT-4o and Claude Sonnet) presents potential concerns regarding data privacy and service availability. We addressed this aspect by implementing and validating a local deployment op-

tion using Gemma 3, which achieved comparable performance (99.80% accuracy) while maintaining data sovereignty. This dual deployment approach provides healthcare institutions with flexibility to choose between cloud-based efficiency and local privacy controls based on their specific requirements.

The management of drug leaflets containing information for multiple pharmaceutical forms remains a consideration; however, our evaluation showed that this had minimal impact on performance. We tackled this issue by providing specific prompts to extract information relevant only to the pharmaceutical form of interest, with healthcare professionals validating the extraction process. The high precision scores (>98%) across all models demonstrate the effectiveness of this approach.

Regarding the non-deterministic nature of LLMs, our comprehensive evaluation across five runs for each model revealed perfect consistency (Fleiss kappa = 100%) across all tested architectures, effectively eliminating this concern. We employed best practices in prompt engineering, structured templates, and clear instructions, resulting in deterministic behavior despite the theoretical potential for variability. The confidence scoring framework we developed provides additional safeguards by flagging uncertain decisions for clinical review.

9.6.3 *Deployment Challenges and Mitigation Strategies*

The transition from research prototype to clinical implementation presents several challenges that must be addressed to ensure the successful deployment of HELIOT in real-world healthcare settings. HELIOT's modular architecture supports two primary deployment scenarios, each with distinct challenges and requirements. In integrated deployment, HELIOT operates as a background service within existing EHR systems, requiring seamless technical integration without disrupting established clinical workflows. Healthcare institutions operate diverse EHR platforms including Epic, Cerner, and Allscripts, each with unique data formats and APIs. To address this challenge, we implement FHIR-compliant APIs providing standardized data exchange formats and HL7 messaging standards for real-time data communication. The key advantage of integrated deployment is that healthcare

professionals continue using their familiar EHR interfaces with no additional training requirements, as HELIOT's recommendations appear within existing alert systems and decision support modules.

Standalone deployment presents different challenges, primarily centered on user adoption and training. Healthcare professionals must learn a new interface and integrate it into their existing workflows, requiring training programs tailored for physicians, pharmacists, and nurses. Our mitigation strategy focuses on developing intuitive interfaces that mirror familiar clinical workflows and implementing a clinical champion program to provide peer support during adoption. A gradual rollout strategy beginning with specific departments allows progressive adaptation while demonstrating measurable value.

Both deployment scenarios face common technical infrastructure challenges. Cloud-native deployment options accommodate institutions with limited IT capabilities, while auto-scaling handles varying workloads without manual management. As an open-source solution, HELIOT eliminates software licensing costs, with infrastructure expenses being the primary financial consideration. This significantly reduces deployment barriers, allowing institutions to focus resources on server infrastructure rather than proprietary software licenses. Technical support including monitoring and maintenance ensures the reliability required in clinical environments, while pilot implementations allow institutions to evaluate benefits and infrastructure requirements before full-scale deployment.

9.6.4 *Ethical Considerations and Regulatory Compliance*

The integration of AI into clinical workflows raises fundamental questions about accountability, privacy, and patient safety. HELIOT addresses these concerns through a human-centered design philosophy that positions healthcare professionals at the center of all clinical decisions. By analyzing clinical notes and drug codes, the system generates evidence-based recommendations with transparent explanations (see Section 9.3.1.1)—but these remain advisory insights rather than prescriptive directives. This approach creates a collaborative dynamic between human expertise and ar-

tificial intelligence, where professionals retain full authority to evaluate and accept or reject recommendations based on their clinical judgment. Such oversight preserves clinical autonomy while simultaneously acting as a natural barrier against potential algorithmic bias.

Building on this foundation, HELIOT's technical architecture prioritizes data protection through deliberate design choices. The system processes only anonymized clinical notes and standardized pharmaceutical codes, creating an inherent barrier against privacy breaches while maintaining analytical effectiveness. The platform's modular architecture offers flexible deployment options: standalone mode operates entirely within institutional boundaries, while EHR integration utilizes secure APIs to maintain consistent privacy standards without compromising functionality.

These technical safeguards represent only part of the compliance equation. HELIOT's current position in early validation phases reflects a methodical approach to regulatory approval, with comprehensive clinical validation studies planned under rigorous ethical supervision. Each phase must meet established compliance standards before it can be advanced. Nevertheless, technology alone cannot guarantee ethical implementation. Healthcare institutions bear responsibility for establishing governance frameworks—including staff training, outcome monitoring, and workflow integration. This shared responsibility model ensures that AI enhancements strengthen clinical judgment rather than replace the human expertise essential to quality patient care.

9.7 CONCLUSION

This chapter presented HELIOT CDSS, a novel approach to adverse drug reaction management that leverages LLMs to interpret unstructured clinical narratives. While traditional CDSSs often struggle with free-text clinical notes and generate excessive alerts due to their rule-based nature, HELIOT demonstrated promise in processing natural language descriptions of adverse reactions and generating more contextual alerts. Our evaluation across multiple LLMs and datasets validated HELIOT's effectiveness in clinical decision support. The system successfully balances alert reduction with patient safety requirements. The implemented risk

management framework provides structured approaches to handle classification uncertainty, supporting clinical deployment.

The potential impact of this approach could be considerable, particularly in healthcare environments with varying levels of EHR integration. By improving the interpretation of clinical narratives and providing more contextual alerts, HELIOT could help reduce alert fatigue while maintaining patient safety. All this could lead to better clinical outcomes and cost savings by preventing adverse drug events and reducing unnecessary alert overrides.

Future work will focus on validating the system in operational clinical environments to confirm practical deployment effectiveness and workflow integration. We will also enhance the system's capabilities for complex clinical scenarios and evaluate its workflow impact across diverse healthcare settings.

10

LLMS FOR DRUG-DRUG INTERACTION PREDICTION

10.1 INTRODUCTION

Drug-drug interactions (DDIs) are a serious issue for clinicians as they can affect the desired response in patients taking multiple medications at the same time, and produce adverse effects or inadequate levels of treatment effectiveness [236]. The consequences of these interactions range from adverse drug reactions to reduced therapeutic efficacy and, in the worst case, can even be life-threatening [71, 236]. This problem is getting worse as polypharmacy becomes more common [88], given the rise in the population's average age.

All this is confirmed by recent studies [71, 173]. On the one hand, the percentage of older adults taking more than one drug is increasing and varies from 40-50% in Western countries (United States, Ireland, Sweden) to over 80% in some Asian regions (South Korea, Taiwan) [173]. On the other hand, DDIs cost U.S. healthcare about $5.4B annually [71]. Therefore, there is a clear need to predict potential DDIs before clinical use for patient safety and for efficient drug development.

The detailed mechanisms underlying DDIs are complex and various molecular pathways may be involved, such as competition for drug-metabolizing enzymes (e.g., cytochrome P450), alterations in drug transport proteins, changes in drug absorption due to pH variations, and interference with receptor binding. Perhaps more importantly, drug administration order can significantly influence interaction outcomes, making it an inherently directional problem [164, 316, 327]. For example, the effect of drug A on the metabolism of drug B may differ from the effect of drug B on the metabolism of drug A.

Traditional approaches to identifying DDIs rely on experimental methods, including in vitro and in vivo studies [253]. However, given the large number of possible drug combinations, these

approaches are time-consuming, expensive, and often impractical. In addition, direct testing of potentially harmful interactions raises significant ethical concerns due to the risk of adverse effects on human subjects [34].

The advances in computational biology and the increasing availability of molecular data have stimulated the development of in silico methods for DDI prediction, in order to complement traditional experimental approaches and improve interpretability and clinical validation [197]. Current machine learning-based prediction methods [109, 160, 198, 331], although their success, often require integration of heterogeneous data sources and complex preprocessing, making them challenging to scale and adapt to new drug combinations [236].

This chapter presents the first investigation of LLMs for DDI prediction, leveraging their ability to process multiple drug information simultaneously (SMILES notation, target organisms, and gene interactions) as text. We explore a complementary approach to established methods (e.g., graph neural network methods), introducing a fundamentally different paradigm that does not require complex feature engineering or structured chemical representations: We enable LLMs to directly interpret and reason about drug interactions using purely textual representations.

To this aim, we design specific prompts to instruct LLMs to use the multiple textual drug information to predict DDIs. We evaluate this approach using zero-shot, few-shot, and fine-tuning techniques across 18 different LLMs, ranging from state-of-the-art models to smaller, more efficient variants. The analyses reveal several key findings. First, while LLMs demonstrate limited effectiveness in zero-shot DDI prediction (average sensitivity of 0.5463), and few-shot approaches show improved but variable performance (with some models reaching an accuracy up to 0.8376), fine-tuning significantly improve their performance. Surprisingly, smaller models such as Phi-3.5 (2.7B parameters) achieves the best results, with a sensitivity of 0.978 and accuracy of 0.919, significantly outperforming the l2-regularized logistic regression [198] and MSDAFL deep learning model [135] baselines and larger LLMs. This performance advantage of smaller models is consistent across 13 external validation datasets, suggesting that model size is not the determining factor for DDI prediction tasks.

The main contributions of the paper are the following:

- We present the first investigation of LLMs for DDI prediction, evaluating 18 models ranging from 1.5B to over 250B parameters.
- We introduce a text input representation that combines SMILES notation, target organisms, and gene interactions, enabling LLMs to leverage multiple aspects of drug information simultaneously.
- We demonstrate a performance progression across prediction approaches, from limited zero-shot performance to improved few-shot learning, with fine-tuned smaller models ultimately achieving state-of-the-art results that outperform larger LLMs and traditional machine learning baselines.
- We validate the findings through extensive analyses across 13 external datasets, confirming the robustness and generalizability of the fine-tuned LLM-based DDI prediction method.

10.2 RELATED WORK

In this section, we review existing approaches for DDI identification and prediction. We then conclude with recent applications of LLMs in drug discovery and the research gap our work addresses.

10.2.1 *Machine learning-based prediction methods*

Various computational methods for DDI prediction have been proposed, which mainly explored structured molecular data and involved extensive feature engineering.

Early approaches, like those by Ferdousi et al. [109] and Kastrin et al. [160], were based on the idea that drugs similar in properties may interact [301] and therefore used several drug similarity metrics combined with traditional ML algorithms to identify DDIs.

Techniques based on matrix factorization approached DDI prediction from a different angle by decomposing interaction matrices to help discover latent patterns [249, 261]. Rohani et al. used

Integrated Similarity-Constrained Matrix Factorization (ISCMF) [249] to integrate multiple similarity types. Shtar et al. [261] framed DDI prediction as a link prediction problem using Adjacency Matrix Factorization (AMF) with factor propagation over graph nodes.

Traditional machine-learning strategies have also been explored to improve DDI prediction. Mei et al. [198] used drug target profiles (i.e., genes) with L2-regularized logistic regression, achieving competitive results and demonstrating that simpler approaches could be effective in this field.

Graph neural networks (GNNs) marked a significant paradigm shift, facilitating DDI predictions by allowing for the direct modeling of molecular structure and network interactions. Using unified architectures, the initial GNN techniques, including GCN-BMP [53] and DeepDrug [44], combined structural graph information with sequential SMILES representations. However, these methods failed to capture the intricate relationships within interaction networks sufficiently and instead focused mainly on individual drug characteristics. Wang et al. [306] attempted to address this limitation by using multi-view learning, which simultaneously processes both molecular structures and DDI network topology through contrastive learning. Despite the fact that this technique enabled more comprehensive interaction modeling, the researchers recognized the need for a more granular understanding of interaction.

Nyamabo et al. [211] pioneered substructure-substructure interaction modeling using co-attention mechanisms (SSI-DDI) – a concept that has been successively advanced by Li et al. [182] with dual-view learning, which couples intra-drug and inter-drug message passing at the atomic level. Zhao et al. [337] tackled the generalization challenge by incorporating contrastive self-supervised learning with deep mix-hop architectures to capture high-order dependencies.

Complementary to these molecule-centric models, Mohamed et al. [203] surveyed and empirically evaluated knowledge-graph embedding methods on biomedical knowledge graphs, showing that low-rank embeddings can accurately predict drug–target links and polypharmacy side-effects (i.e., DDIs).

Current state-of-the-art methods have tried to address the challenge of combining diverse drug information sources. Chao et al.

introduced MSDAFL [135], a dual attention framework that captures both intra-drug substructure patterns and inter-drug substructure interactions through self-attention and interactive attention mechanisms. On the other hand, Su et al. proposed KGCNN [274], which advances spatial modeling by incorporating a spatial-aware capsule aggregator that captures relationships between neighboring molecules in biomedical knowledge graphs.

Although these methods achieved good performance, they have similar limitations: they require complex feature engineering, structured chemical representations (such as graphs or matrices), and extensive preprocessing of heterogeneous data sources. Perhaps most importantly, no current model can process multiple drug characteristics (action type, property, pathway, etc.) simultaneously without time-consuming, sophisticated preprocessing or integration frameworks.

10.2.2 *LLMs in Drug Discovery*

Recently, LLMs have been successfully used in drug discovery applications [5, 217], especially in drug-target interaction (DTI) prediction. Ahmed et al. developed DTI-LM [5], an innovative framework for DTI prediction that processes protein amino acid sequences and drug SMILES format using LLMs. The authors found that while LLMs can identify similarities and interactions between proteins, they are unable to represent the similarities between drugs adequately. Oniani et al. [217] proposed a method that utilized both protein language models (ESM-2) and chemical language models (ChemBERTa) for predicting DTIs, further corroborating the observations of Ahmed et al. on the limitations of chemical language models and the importance of larger datasets. Recent developments have also introduced specialized biomedical LLMs, such as BioClinical ModernBERT [269] for long-context biomedical text processing, MedGemma [259] for medical applications, Me-LLama citexie2024me for foundation medical models, and OpenBioLLMs [15] for healthcare and life sciences. These domain-specific models show promise for biomedical tasks, but their application to DDI prediction has not been systematically evaluated. To the best of our knowledge, only a few studies have employed general-purpose LLMs [324] in the context of DDI pre-

diction. Xu et al. presented DDI-GPT [324], a model that integrates LLMs with knowledge graphs for DDI prediction. DDI-GPT provides interpretable predictions through gene importance scoring and network analyses. However, it relies on a curated knowledge graph and specific feature engineering techniques (sentence-tree injection and visibility masks), which limit its scalability compared to LLMs that operate directly on textual drug information. Although these studies indicate a growing opportunity for LLMs in drug discovery, the direct application of LLMs for DDI prediction remains an emerging field, presenting opportunities for further research.

10.2.3 *Research Gap*

The review highlights several limitations of DDI prediction approaches. First, machine learning methods need complex feature engineering and face challenges in integrating multiple data sources seamlessly [236]. Second, in order to attain optimal performance, deep learning techniques require large-scale training data, meticulous architecture design, and intensive hyperparameter tuning [236]. In addition, as we have seen in the section above, LLM applications in this context have primarily focused on specific tasks or aspects, such as DTIs [5, 217] or knowledge graph integration [324]. Therefore, the potential of LLMs to process multiple textual drug characteristics (e.g., SMILES notation, target organisms, and gene interactions) for DDI prediction remains unexplored. Our goal is to bridge this gap by determining whether (and how) LLMs can efficiently process only textual drug information to forecast DDIs as a complementary approach to established methods (e.g., graph neural network methods) without the need for intricate feature engineering or organized chemical representations. We aim at exploring various adaptation techniques in order to pinpoint the advantages and disadvantages of LLMs in identifying DDIs and, if applicable, establish a more accessible approach in this field.

10.3 DATASETS AND DATA PROCESSING

This section describes the datasets and data processing steps implemented to achieve this objective. Data and source codes are publicly accessible in our online repository [73].

10.3.1 *Datasets*

We used two primary data sources to assess LLM performance. The first one is DrugBank [318], a database containing 16,581 drugs, 3,921 unique target genes, and 1,420,072 known drug-drug interactions. This rich dataset was the primary source for zero-shot, few-shot, and fine-tuning analyses, and we also used it to generate negative examples (non-interacting drug pairs) for all datasets. The second data source is the DDI repository provided by Ayvaz

Table 10.1: Drugs and Initial DDI For Datasets.

Category	Dataset	Drugs	DDI
Primary Dataset	Drugbank	16,581	1,420,072
Clinical knowledge bases	CredibleMeds	63	83
	HEP	557	11,194
	HIV	556	19,198
Annotated corpora	DDI Corpus 2011	244	334
	DDI Corpus 2013	410	787
	NLM Corpus	131	238
	PK DDI Corpus	85	146
Healthcare systems	OSCAR EMR	227	10,325
	WorldVista	378	13,693
Reference resources	French DDI Referrals	854	62,047
	KEGG	1,033	52,104
	NDF-RT	425	1,876
Clinical guidelines	ONC High Priority DDI List	123	193
	ONC Non-Interruptive DDI List	187	2,101

et al. [22], which we employed for external validation, as it aggregates drug interactions from 14 distinct sources from clinical knowledge bases, annotated corpora, healthcare systems, reference resources, and clinical guidelines.

The initial number of drugs and DDIs for each dataset is reported in Table 10.1.

10.3.2 *Data Processing*

We prepared the datasets for the LLM empirical analyses following a four-step data processing pipeline (see Figure 10.1).

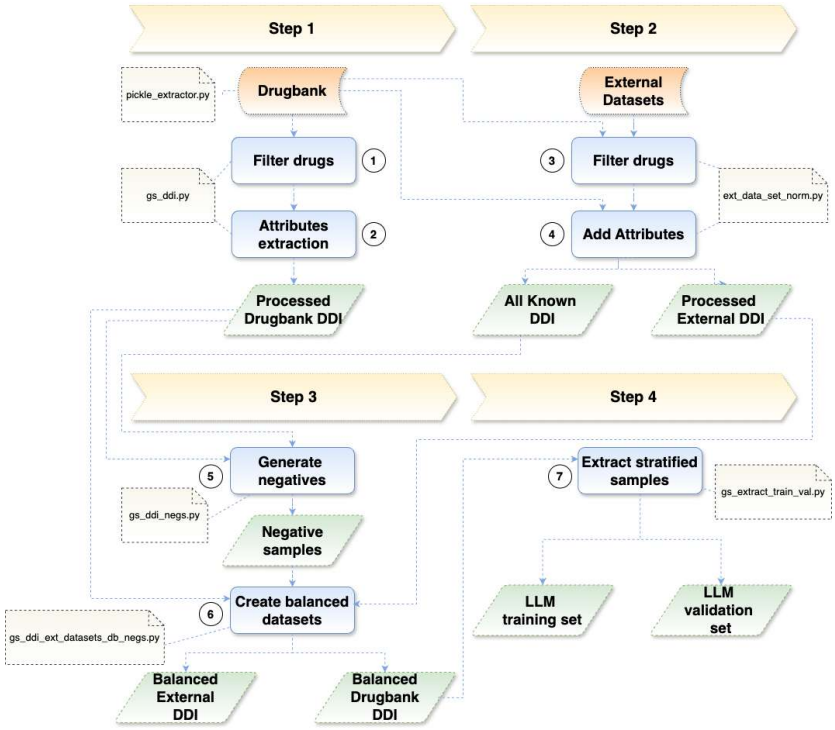


Figure 10.1: Data preprocessing pipeline.

Following the principle that "two drugs potentially interact when a drug alters the other drug's therapeutic effects through targeted genes or signaling pathways" [198], we focus on drugs targeting at least one gene.

In the first step, we filtered the DrugBank database to include only drugs that were approved or experimental and excluded withdrawn or illicit drugs. Furthermore, we selected only drug pairs where both drugs targeted at least one gene. For each drug pair, we extracted DrugBank IDs, SMILES notations, target organisms, and two binary vectors (length 3,921) representing gene targets, where each position corresponded to a gene in DrugBank's lexicographically ordered gene list. At the end of this process, we collected 1,035,150 positive DDIs. In the second step, we pro-

cessed the 14 datasets from Ayvaz et al. [22], filtering out drugs not present in the processed DrugBank dataset and removing any interactions already present in the DrugBank dataset. We then enriched the remaining drug pairs with the necessary features from DrugBank, such as SMILES, target organisms, and gene vectors. We obtained 13 usable datasets, due to the fact that the ONC High Priority DDI dataset contained no valid interactions after filtering, totaling 21,947 additional unique DDIs. In the third step, we combined all known interactions from DrugBank and external datasets to generate 1,057,097 negative examples, making sure that none of them overlapped with any existing interactions. We used 1,035,150 negative examples to balance the processed DrugBank dataset and the remaining 21,947 examples to balance the external datasets. Finally, we randomly selected 1,000 examples from the DrugBank balanced dataset for training and 1,090 for validation using stratified sampling, as supported by recent research on LLMs that demonstrated that performance plateaus beyond the 1,000-example threshold [27, 216, 341, 344]. The data preprocessing pipeline and the processed datasets (with the exception of the DrugBank dataset, which cannot be publicly distributed) are available in our online repository [73] for reproducibility.

The final distribution of positive and negative drug pairs for each dataset is illustrated in Table 10.2.

10.4 METHODS

This section describes the methodological approaches used to adapt LLMs for DDI prediction. More in detail, we implemented three increasingly sophisticated adaptation approaches using the preprocessed datasets described in Section 10.3.2 (see Figure 10.2): (1) zero-shot learning leveraging models' inherent knowledge, (2) few-shot learning with exemplar drug interactions, and (3) fine-tuning for task-specific optimization.

In the following, we outline the model selection criteria and then describe the implementation of each adaptation approach.

Table 10.2: Datasets with Positive and Negatives Drug Pairs.

Dataset	Drug Pairs		
	Positive	Negative	Total
Drugbank	1,035,150	1,035,150	2,070,300
CredibleMeds	5	5	10
HEP	1,271	1,271	2,542
HIV	5,651	5,651	11,302
DDI Corpus 2011	32	32	64
DDI Corpus 2013	74	74	148
NLM Corpus	9	9	18
PK DDI Corpus	2	2	4
OSCAR EMR	2,052	2,052	4,104
WorldVista	1,513	1,513	3,026
French DDI Referrals	4,297	4,297	8,594
KEGG	6,631	6,631	13,262
NDF-RT	119	119	238
ONC High Priority DDI List	0	0	0
ONC Non-Interruptive DDI List	291	291	582
LLMs training set	500	500	1,000
LLMs validation set	545	545	1,090

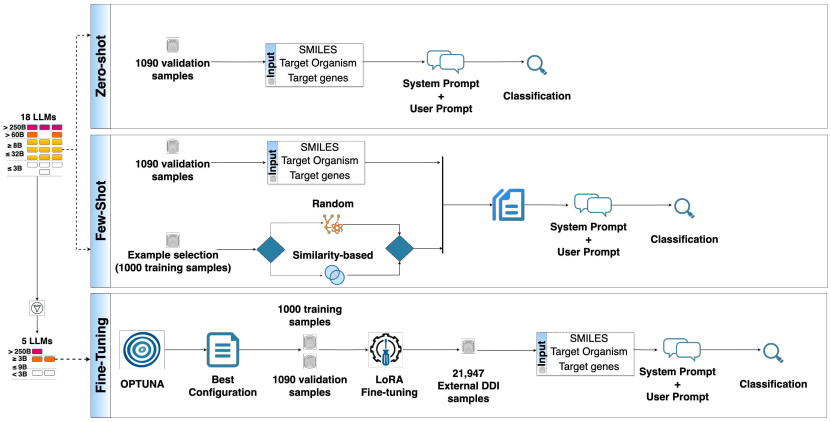


Figure 10.2: Methods overview showing the three-tier LLM approach for DDI prediction: zero-shot, few-shot, and fine-tuning workflows with drug pair input processing and classification output.

10.4.1 *Selected LLMs*

We evaluated 18 different LLMs, ranging from 1.5B to over 250B parameters, which can be categorized into four groups:

- Proprietary models: GPT-4o [219], Claude 2.1 [17], and Gemini 1.5 Pro [125], representing state-of-the-art commercial LLMs accessed through their respective APIs.
- Open-weights large models: LLaMA 3.3-70B [97] and Qwen2 72B [326], which are publicly available models with architectures and parameter count comparable to commercial solutions.
- Open-weights middle-range models: Granite 3.1 8B [202], LLaMa 3.1 8B [97], Gemma 2 9B [285], Falcon 3 10B [283], Mistral-Nemo 12B [347], Qwen 2.5 14B [326], Gemma2 27B [285], Aya-expanse-32b [70], and Qwen 2.5 32B [326], representing a balance between computational efficiency and model capacity.
- Open-weights efficient models: Phi-3.5 (2.7B) [1], Qwen2.5 3B [326], LLaMa3.2 3B [97], and DeepSeek R1 distilled Qwen 1.5B [128], representing recent advances in efficient model architectures.

There were several considerations that guided our selection. First, we aimed for model diversity across different scales, including both large-scale architectures and smaller models (1.5B-7B parameters) to investigate the relationship between model size and performance and assess practical deployment scenarios. Second, we took into account LLM accessibility, selecting both proprietary and open-weight models in order to evaluate the feasibility of DDI prediction across different deployment scenarios. Finally, we included the latest models, such as Phi-3, Gemma 2, and DeepSeek R1, to evaluate cutting-edge architectures.

We accessed the proprietary models by using their respective API; whereas, for local deployment of open-weights models, we utilized LM Studio [100]. LM Studio provides a unified interface for model deployment and inference, and supports various model

architectures and configurations. This platform enabled us to maintain consistent experimental conditions across all open-weight models, offering an OpenAI-compatible REST API.

10.4.2 *Zero-shot Approach*

The first adaptation approach we implemented is zero-shot learning, which aims to assess LLMs' inherent ability to predict DDIs without task-specific training and without providing examples of interactions.

For prompt design, we paid attention to three key aspects: i) providing drug information in a structured format, ii) clearly specifying the binary classification task, and iii) implementing explicit output constraints.

To this aim, we conceived the prompt structure reported in Figure 10.3.

System prompt

You are an expert in drug-drug interaction. Given two drugs, where the order of administration counts, the genes and organisms targeted by the two drugs and the SMILES formulas of the two drugs, classify whether their administration causes 'interaction' or 'no interaction.' Answer only with the classification ('interaction' or 'no interaction'), nothing else.

User prompt: Zero-shot setting

Drug1: drug1
SMILES for drug1: smiles1
Organism targeted by drug1: org1
Genes targeted by drug1: genes1
Drug2: drug2
SMILES for drug2: smiles2
Organism targeted by drug2: org2
Genes targeted by drug2: genes2
CLASSIFICATION:

Figure 10.3: Zero-shot prompt template

On the one hand, the system prompt emphasizes the importance of drug administration order and constrains the model's output to a binary classification. On the other hand, the user prompt explicitly includes all available drug characteristics: drug names,

molecular structures (SMILES notation, such as "CC(C)(C)OC=O"), target organisms, and gene interactions (i.e., Glucosamine-targeted gene "IFNG").

This information is inserted directly as formatted text strings, i.e., with SMILES notation "as is", genes as comma-separated names, and organisms as text labels. LLM tokenizers process it as ordinary sequences of characters or words, without any intrinsic chemical understanding; this means that the model treats it as textual patterns, and any domain knowledge emerges from patterns learned during pretraining, where such notations co-occurred with discussions of drug interactions.

Although the instructions requested only the classification term, some models may provide additional explanations to support their classification. For these cases, we used a pattern-matching approach that searched the response text for one of the classification keywords ("interaction" or "no interaction"), giving priority to exact matches that followed the "CLASSIFICATION:" marker when present.

10.4.3 *Few-shot Approach*

The second adaptation approach is few-shot learning, which allows teaching the LLM with respect to the DDI prediction task through as few as 2-5 examples of input-output pairs. We implemented and compared two strategies for example selection, namely random and similarity-based, in order to assess whether the inclusion of curated examples improves LLM performance in DDI prediction. For both approaches, we used a balanced set of 10 examples, comprising 50% positive and 50% negative interactions, extracted from the training dataset (see Section 10.3.2), to maintain an equal representation of both interaction classes. We limited examples to 10 due to context length constraints in some LLMs (i.e., a 8192 token limit) while maintaining sufficient information for effective inference.

In the first strategy, we used random selection, where examples were chosen arbitrarily from the training set. While this approach introduces models to the general task format and diverse interaction patterns, it does not ensure that examples have similarities to the query pair.

We developed a second strategy using an embedding-based similarity search to test whether contextually relevant examples would yield better performance. We computed a composite similarity measure for each test drug pair based on six embedding vectors: SMILES notation, target organisms, and target genes for each drug in the pair. Using the OpenAI text-embedding-3-small model, we calculated the cosine similarity between the query pair and all training examples.

User prompt: Few-shot settings

```
## INSTRUCTIONS
Analyze the provided drug information and consider the following examples to answer correctly:
** Examples **

Example 1:
Drug1: example_drug1_name
SMILES for drug1: example_smiles1
Organism targeted by drug1: example_org1
Genes targeted by drug1: example_genes1
Drug2: example_drug2_name
SMILES for drug2: example_smiles2
Organism targeted by drug2: example_org2
Genes targeted by drug2: example_genes2
CLASSIFICATION: interaction [or no interaction]
...

Example 10:
...

## CURRENT TASK INPUT DATA ##
Drug1: {drug1}
SMILES for drug1: {smiles1}
Organism targeted by drug1: {org1}
Genes targeted by drug1: {genes1}
Drug2: {drug2}
SMILES for drug2: {smiles2}
Organism targeted by drug2: {org2}
Genes targeted by drug2: {genes2}
CLASSIFICATION:
```

Figure 10.4: Few-shot prompt template

More specifically, we calculated embeddings for six different components: SMILES notation, target organisms, and gene interactions for each drug. The similarity score was calculated as the arithmetic mean of the six individual cosine similarity scores: $similarity_score = (sim_smiles1 + sim_smiles2 + sim_org1 + sim_org2 + sim_genes1 + sim_genes2)/6$.

We then identified the top matches while retaining the balance of positive and negative examples. This equally weighted approach allows the drug characteristics to have the same influence on similarity assessment without manual feature engineering efforts.

We used the same system prompt as in the zero-shot setting (see Section 10.4.2) and augmented the user prompt with the selected examples as shown in Figure 10.4. In order to extract classifications from LLM outputs, we exploited the pattern-matching strategy described in Section 10.4.2.

10.4.4 *Fine-tuning Approach*

As the final level in the adaptation approach, we implemented a fine-tuning technique. Fine-tuning helps specialize LLMs for specific tasks by adjusting their parameters through supervised learning on task-specific data. In our context, we fine-tuned selected models to optimize their DDI prediction capabilities using the same textual drug information and prompt structure described in Section 10.4.2. To implement this approach, we leveraged the LLM training and validation sets described in Section 10.3.2, and created JSONL files containing conversational sequences structured with system prompts, user prompts (as presented in Section 10.4.2), and assistant responses.

Among proprietary models, we chose GPT-4o as the state-of-the-art representative for fine-tuning analysis. Indeed, Google and Anthropic do not offer fine-tuning capabilities for Gemini-1.5-pro and Claude Sonnet. For fine-tuning GPT-4o, we utilized OpenAI's API and set the following hyperparameter values, as suggested by the OpenAI documentation [218]: 3 epochs, batch size 3, and learning rate multiplier 0.3.

We chose four representatives for open-weights models: Phi-3.5 2.7B, Qwen2.5 3B, Deepseek R1 Distilled Qwen 1.5B, and Gemma2 9B. Our interest in these LLMs was motivated by exploring how

smaller models performed as compared to state-of-the-art models like GPT-4o, as well as some practical considerations about computational resources and deployment options.

To determine the fine-tuning strategy, we considered multiple alternatives. In preliminary experiments with Phi-3.5 2.7B, full fine-tuning consistently led to catastrophic forgetting – i.e., a phenomenon where a neural network loses previously acquired knowledge when learning new information – regardless of learning rate or number of epochs. The model converged to degenerate solutions, predicting only one class with high confidence while ignoring the other. Reducing trainable parameters by fine-tuning only a subset of layers improved discrimination but resulted in suboptimal performance. For detailed results, see B. We also considered QLoRA [89] but opted against it since it trades accuracy for memory efficiency through quantization, and our focus was on maximizing prediction performance.

Finally, we employed Low-Rank Adaptation (LoRA) [137] for fine-tuning open-weight LLMs. To optimize the fine-tuning parameters, we implemented a hyperparameter search using Optuna [8], aiming to minimize validation loss while avoiding overfitting. The search space included learning rate (log-uniform distribution in the range [1.8e-4, 2.8e-4]), number of LoRA layers (16, 18, 20, 22, 24, 26, 28 for Deepseek R1 with 30 linear layers, extended to 32 for models with more layers), LoRA rank (16 or 32), LoRA alpha scaling (1 or 2, where one indicates alpha equals rank and 2 indicates alpha doubles rank), LoRA dropout (uniform distribution in [0.0, 0.02] with 0.001 steps), and LoRA scale (uniform distribution in [3.8, 4.4] with 0.1 steps). We used the Adam optimizer [162] with a cosine decay learning scheduler across 1,000 trials. We monitored validation loss and stopped training when it ceased to improve for two consecutive validation iterations. During fine-tuning, model adapters were saved every 100 iterations and merged with the base model layers at the end of training. For the final evaluation, we picked the checkpoint with the lowest validation loss for each model.

10.5 EMPIRICAL EVALUATION

In this section, we present the research questions and outline the empirical evaluation design used to assess the performance of LLMs in predicting DDIs. All experiments were performed using a MacBook Pro M3 Max, equipped with 96GB of RAM, 14 cores, and a Metal GPU.

10.5.1 *Research Questions*

Four research questions drove the evaluation:

RQ10.1 How effectively do LLMs predict DDIs in a zero-shot setting?

This research question evaluates LLMs' ability to leverage SMILES notation, target organisms, and gene interactions to predict drug-drug interactions without task-specific training.

RQ10.2 How effectively do LLMs predict DDIs in a few-shot setting?

This research question explores whether providing example interactions improves LLMs' DDI prediction performance through in-context learning.

RQ10.3 What is the impact of fine-tuning on LLMs' DDI prediction performance?

This question investigates how fine-tuning affects LLMs' ability to process multiple drug representations for DDI prediction, with particular attention to the relationship between model size and performance.

RQ10.4 How do fine-tuned LLMs compare with traditional approaches?

This question assesses the performance of fine-tuned LLMs against baselines — L2-regularized logistic regression and deep learning — across multiple external datasets, evaluating prediction effectiveness and generalizability.

10.5.2 *Metrics*

We assessed the performance of the zero-shot, few-shot, and fine-tuning methods using various complementary metrics, namely sensitivity, precision, F1-score, and accuracy. 'Sensitivity' measures the model's ability to correctly identify positive interactions, which is particularly crucial in drug safety applications. 'Precision' and 'F1-score' respectively quantify the proportion of true positive predictions and the harmonic mean of precision and sensitivity. We also calculated the 'accuracy' to measure general performance across both classes. Since the datasets are fully balanced regarding the 'target' (i.e., the same number of positive and negative data instances), 'accuracy' is a reliable performance metric. We did not use ROC-AUC since the LLMs do not output probabilities.

10.5.3 *Research methods for $RQ_{10.1}$*

To address RQ10.1, we systematically evaluated the 18 LLMs described in Section 10.4.1 through the zero-shot approach. Each model processed the same 1,090 balanced drug pairs from the validation dataset (see Section 10.3.2) using the zero-shot prompt structure detailed in Section 10.4.2. We set the temperature to 0.0 for all evaluations and models to mitigate non-deterministic outputs, and accessed proprietary models through their respective APIs, and open-weight models through LM Studio's REST API. Each model generated a binary classification ("interaction" or "no interaction") for each drug pair, which we compared against ground truth labels. We then calculated the four metrics defined in Section 10.5.2 (namely, Accuracy, Precision, Sensitivity, and F1-score) in order to establish a baseline for LLMs' intrinsic DDI prediction capabilities as a comparison with the other adaptation approaches.

10.5.4 *Research methods for $RQ_{10.2}$*

To answer RQ10.2, we applied and assessed the two few-shot strategies described in Section 10.4.3: random selection and similarity-

based selection of examples. We followed the procedure reported in RQ10.1 for both strategies, using the same inference configurations, LLMs, validation dataset, and performance metrics, to maintain consistent conditions and enable direct comparison between zero-shot and few-shot approaches.

10.5.5 *Research methods for $RQ_{10.3}$*

RQ10.3 aims to assess the impact of fine-tuning on the capabilities of LLMs in DDI prediction. We used the five fine-tuned models described in Section 10.4.4 (i.e., GPT-4o, Phi-3.5 2.7B, Qwen2.5 3B, Deepseek R1 Distilled Qwen 1.5B, and Gemma2 9B), picking the resulting fine-tuned version of GPT-4o and the best fine-tuned versions for the open-weight models in terms of validation loss. We adopted the same setup and performance metrics used in RQ10.1 and RQ10.2, and to account for LLMs' non-deterministic nature and ensure reliable results, we performed five repeated classifications on the validation set described in Section 10.3.2, computing the average of the metrics for each repetition. We treated each of the five iterations as a separate annotator and used Fleiss' Kappa score [114] to measure the inter-rater reliability of the classifications, a metric which has been used in Software Engineering studies [334]. This score ranges from 0 (random agreement) to 1 (perfect agreement) [171], accounting for chance-based agreements.

10.5.6 *Research methods for $RQ_{10.4}$*

RQ10.4 aims to assess the generalizability of fine-tuned LLMs (i.e., GPT-4o, Phi-3.5 2.7B, Qwen2.5 3B, Deepseek R1 Distilled Qwen 1.5B, and Gemma2 9B) and evaluate their performance against baselines. We selected the L2-regularized logistic regression model [198] and the MSDAFL deep-learning model [135] as baselines. The baseline comparisons served two purposes: first, to reproduce the original results reported in the published works [135, 198], and second, to provide a direct performance comparison between our LLM-based approaches and established state-of-the-art methods on the same datasets. The L2-regularized logistic re-

gression model was selected due to the impressive performance metrics, achieving an 'accuracy' of 0.9479, 'sensitivity' of 0.9556, 'specificity' of 0.948, and 'ROC-AUC' of 0.9884, the authors claimed in their original work. First of all, we created new training and validation sets (95% and 5%, respectively) from the balanced DrugBank dataset described in Section 10.3.2. Then we tuned the model using a stratified 10-fold cross validation and exploring the regularization parameter C in the range $[2^{-16}, 2^{16}]$, as specified in the original paper [198].

On the other hand, we chose MSDAFL as the second baseline because the authors compared the performance to ten leading methods (such as GCN-BMP, DeepDrug, and SSI-DDI) and demonstrated superior performance across multiple metrics in their 2024 study. We utilized the authors' replication package and followed their methodology to train the model. Specifically, we extracted 21,000 examples from the balanced DrugBank dataset described in Section 10.3.2 using stratified sampling for creating the training set, and used it to train the model for 300 epochs with a learning rate schedule of 0.001 for the first 150 epochs and 0.0001 for the subsequent 150 epochs. As stated in the original work [135], we used the Xavier initialization [122] to initialize the model and the Adam optimizer [162] to optimize it, and we also kept the original hyperparameter settings, namely: a batch size of 512, a node embedding dimension of 128, 60 representative vectors (M), and five layers in the Graph Isomorphism Network [325] structure.

In order to evaluate the fine-tuned LLMs and trained baselines, we used the 13 external balanced datasets detailed in Section 10.3.2 and the metrics described in Section 10.5.2. For LLMs, we adopted the same setup as in RQ10.1 and RQ10.2, and performed five repeated classifications on all the datasets, computing the average of the metrics for each repetition. We finally compared the LLM's performance against the baselines.

10.6 RESULTS AND DISCUSSION

This section presents and discusses the findings around the four research questions.

10.6.1 *RQ10.1: Zero-shot Learning Analysis*

The zero-shot evaluation (see Figure 10.5 and Table B.5 for details) revealed several interesting patterns across different models, sizes, and architectures. Proprietary models (i.e., GPT-4o, Claude Sonnet, and Gemini Pro) generally demonstrated superior performance, with Claude Sonnet achieving the highest 'accuracy' (0.7358) and 'precision' (0.8859) among all models; nonetheless, even these state-of-the-art LLMs achieved relatively modest 'sensitivity' scores (0.5413-0.5927), which indicates that they have limitations in identifying positive interactions.

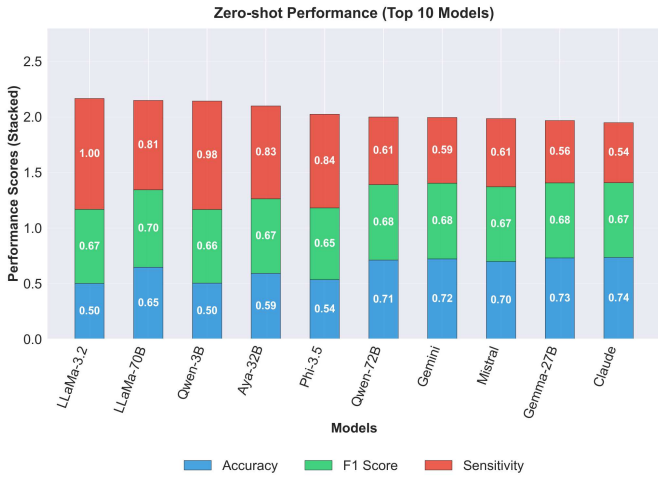


Figure 10.5: Zero-shot performance of the top 10 LLMs on DDI prediction: Validation Dataset.

The results also showed variance in DDI prediction performance across different open-weight models, although model size did not consistently correlate with performance. On the one hand, smaller models (2-3B parameters) exhibited high sensitivity (e.g., LLaMA-3.2 3B: 0.9982), despite having lower overall accuracy, which suggests a bias toward positive predictions rather than true discriminative ability. On the other hand, mid-range LLMs, such as Gemma2 27B (accuracy: 0.7303) and Mistral-Nemo 12B (accuracy: 0.6991), performed similarly to proprietary models, whereas larger models like Qwen2 72B (accuracy: 0.7119) did not significantly outperform their smaller counterparts. The findings indi-

cate that although LLMs can use their pre-trained knowledge for DDI prediction, their zero-shot performance is limited, especially in terms of sensitivity.

To find out if sophisticated reasoning capabilities could overcome the drawbacks of conventional LLMs, we ran additional experiments using more recent reasoning-enhanced models. To this aim, we selected five additional LLMs, such as OpenAI O3, Anthropic Opus-4, Qwen3 4B thinking, Phi-4 4B reasoning, and NVIDIA Nemotron 14B.

Table 10.3: Zero-shot performance of reasoning-enhanced LLMs on DDI prediction.

# Params	Model	Acc.	Prec.	Sens.	F1
>250B	OpenAI O3	**0.7009**	0.8789	0.4661	0.6091
	Anthropic Opus-4	0.6881	**0.8839**	0.4330	0.5813
4B	Phi-4 4B reasoning	0.4853	0.4646	0.1927	0.2724
	Qwen3 4B thinking	0.5569	0.8780	0.1321	0.2297
14B	Nemotron 14B	0.5138	0.9412	0.0294	0.0569

Table 10.3 reports the results of these experiments. With an accuracy of 0.701 and an F1-score of 0.609, O3 performed the best, but it still fell short of the best-performing conventional LLMs, such as Claude 3.5 Sonnet. Notably, O3 showed a high precision (0.879) but a low sensitivity (0.466), which may suggest a conservative approach to positive predictions. Anthropic Opus-4 exhibited similar patterns with slightly lower performance (accuracy: 0.688, Sensitivity: 0.433, F1: 0.581), while the smaller reasoning models showed concerning limitations. Phi-4 4B reasoning, Qwen3 4B thinking, and Nemotron 14B all demonstrated poor sensitivity scores (0.193, 0.132, and 0.029, respectively), indicating severe difficulty in identifying positive drug interactions despite their advanced reasoning capabilities. These findings imply that reasoning-enhanced models do not address the issues of DDI prediction in zero-shot settings, even though they can offer structured analytical processes.

All this leads us to answer RQ10.1:

> ***RQ$_{10.1}$ Zero-shot Learning Analysis:*** *LLMs show limited effectiveness in zero-shot DDI prediction, with even the best mod-*

> *els achieving only moderate accuracy and modest sensitivity. The variable performance across different model sizes suggests that pre-trained knowledge alone is insufficient for reliable DDI prediction.*

10.6.2 *RQ10.2: Few-shot Learning Analysis*

Figures 10.6 and 10.7 (see Tables B.6 and B.7 for details) report the results of random and similarity-based few-shot strategies. BBoth approaches demonstrated some effectiveness, as they improved LLM performance compared to the zero-shot setting (see Section 10.6.1), although there is still room for improvement.

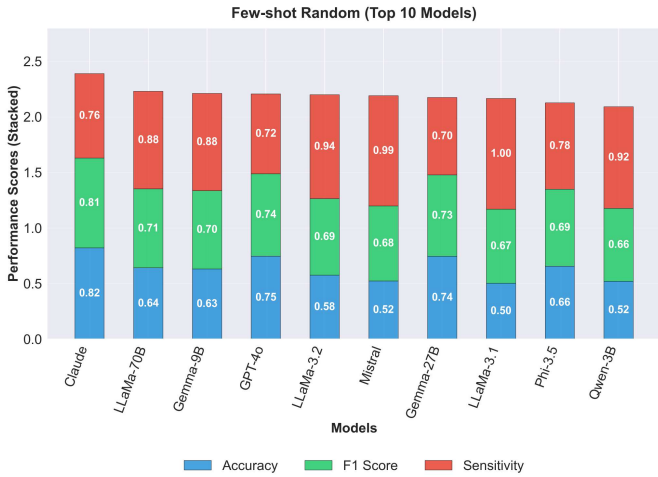


Figure 10.6: Random Few-shot performance of the top 10 LLMs on DDI prediction: Validation Dataset.

In general, the similarity-based strategy outperformed the random one across most models, confirming our hypothesis that providing contextually relevant examples enhances the ability of LLMs to make accurate predictions. This improvement was particularly evident in proprietary and several mid-sized models, with GPT-4o showing substantial gains (F1: 0.7403→0.8014) and Claude 3.5 Sonnet maintaining its leadership (F1: 0.8384).

Distinctive performance patterns emerged across model families. LLaMA models (3.1 8B, 3.2 3B, and 3.3 70B) consistently exhibited extremely high sensitivity (>97%) but lower precision, indicating a tendency to ”overpredict” positive interactions. In

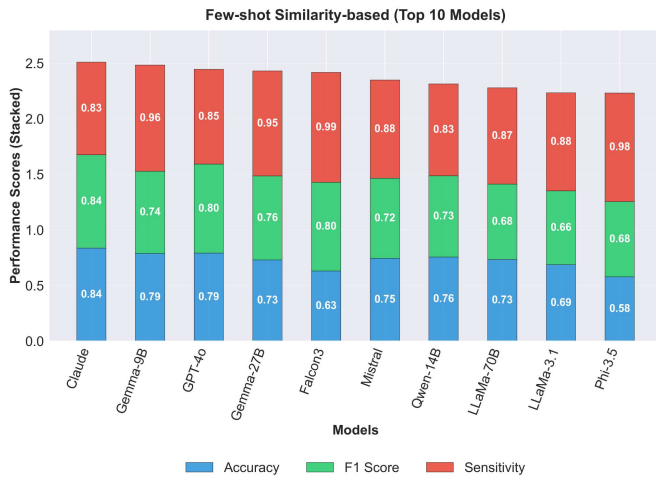


Figure 10.7: Similarity-based Few-shot performance of the top 10 LLMs on DDI prediction: Validation Dataset.

contrast, Gemini and Qwen models showed the opposite pattern, suggesting a more conservative approach to identifying DDIs.

As with the zero-shot approach, model size did not consistently predict the performance benefits of few-shot learning. Although proprietary models performed better overall, a few mid-sized models – Falcon 3 10B (F1: 0.7975), Aya-Expanse 32B (F1: 0.7625), Gemma2 27B (F1: 0.7556), and Qwen2 72B (F1: 0.7629) – performed exceptionally well with similarity-based selection. Surprisingly, the largest open models did not consistently outperform their smaller counterparts. LLaMA-3.3 70B achieved only a 0.6789 F1 score with similarity-based selection, underperforming compared to several 8-14B models. All this suggests that for highly specialized tasks such as identifying DDIs, it is more important to have a model specifically optimized for that domain than a model that is simply larger but generic.

These results lead us to answer RQ10.2 as follows:

> ***RQ$_{10.2}$ Few-shot Learning Analysis:*** *LLMs show improvement with few-shot learning for DDI prediction compared to the zero-shot setting, with similarity-based selection outperforming random selection. Despite this approach narrowing the gap between open-weight and proprietary models, even the best performers, like Claude*

> *3.5 Sonnet (F1=0.8384), achieve results that remain suboptimal for reliable pharmaceutical applications, indicating DDI prediction is challenging even with in-context examples.*

10.6.3 *RQ10.3: Impact of Fine-tuning*

Table 10.4 reports the best parameters found for each fine-tuned model along with their corresponding training epochs.

Table 10.4: Optimal LoRA parameters for fine-tuned models obtained through hyperparameter search.

Model	Epochs	Layers	LR	Rank	Alpha	Scale	Dropout
Phi-3.5 2.7B	3	16	2e-4	16	16	4.0	0.0
Qwen2.5 3B	4	16	2e-4	16	16	4.0	0.0
Deepseek R1 1.5B	4	20	2.2e-4	32	64	4.0	0.009
Gemma2 9B	5	16	1e-5	16	16	4.0	0.1

The Fleiss' Kappa scores across five repetitions (as described in Section 10.4.4) were 1.0 for all models except Qwen 2.5 3B and Gemma2 9B, which scored 0.99, confirming the consistency of the fine-tuned models. Specifically, in a single iteration out of five, Qwen 2.5 3B showed different classifications for four drug pairs (one from each of the datasets DDI Corpus 2013, HEP, KEGG, and OSCAR), while Gemma2 9B misclassified three pairs (one from each of the three datasets French DDI Referrals, HEP, and OSCAR).

Figure 10.8 (see Table B.8 for details) presents the performance of LLMs after fine-tuning.

Fine-tuning significantly improved the performance of all models on the validation set. Significant gains were seen in smaller models, with Qwen2.5 3B demonstrating strong performance (accuracy: 0.878, sensitivity: 0.969) and Phi-3.5 (2.7B parameters) performing on par with GPT-4o (accuracy: 0.913 vs. 0.926) and even outperforming it in sensitivity (0.960 vs. 0.930). It is interesting to note that Deepseek R1 distilled Qwen 1.5B (i.e., the smallest model in our study) produced competitive results (accuracy: 0.895, sensitivity: 0.919) after fine-tuning, indicating that model size has no bearing on DDI prediction performance.

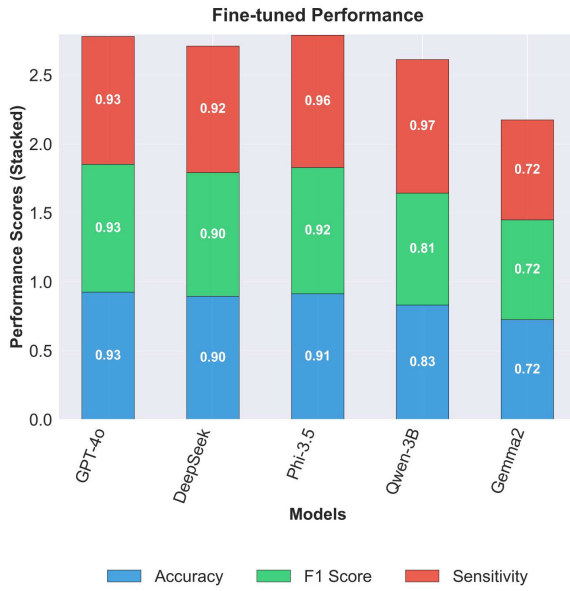


Figure 10.8: Fine-tuning results of different LLMs on DDI prediction: Validation Dataset.

Given the results and the above analysis, we can answer RQ10.3 as follows:

> ***RQ$_{10.3}$ Impact of Fine-tuning:*** *Fine-tuning dramatically improves LLMs' DDI prediction capabilities, with even small models achieving performance comparable to or exceeding larger models. All this suggests that model size is less critical than task-specific adaptation for effective DDI prediction.*

10.6.4 *RQ10.4: Comparative Analysis*

Several noteworthy findings emerge after comparing replicated machine learning models to LLM-based approaches across external datasets. Although slightly worse than but comparable to the results in the original paper (accuracy: 0.948, sensitivity: 0.948), the replication of the L2-regularized logistic regression model performed competitively (accuracy: 0.918, sensitivity: 0.925, f1-score: 0.919). Similarly, the MSDAFL model implementation showed good results (accuracy: 0.916, sensitivity: 0.913, f1-score: 0.918), but it

fell short of the performance ranges reported in the original paper (accuracy: 0.953-0.987, f1-score: 0.900-0.993).

When examining the fine-tuned LLM performance, Phi-3.5 3B consistently demonstrated superior sensitivity (0.978 average) across the 13 external datasets, outperforming traditional machine-learning approaches and other LLMs (see Figure 10.9 and Table B.9 for details). This model achieved perfect sensitivity (1.000) in 7 of the 13 datasets, including CredibleMeds, HEP, HIV, PK Corpus, and NDF-RT. Qwen2.5 3B followed closely with an average sensitivity of 0.975, while Deepseek Qwen 1.5B achieved 0.953, surpassing traditional machine learning models.

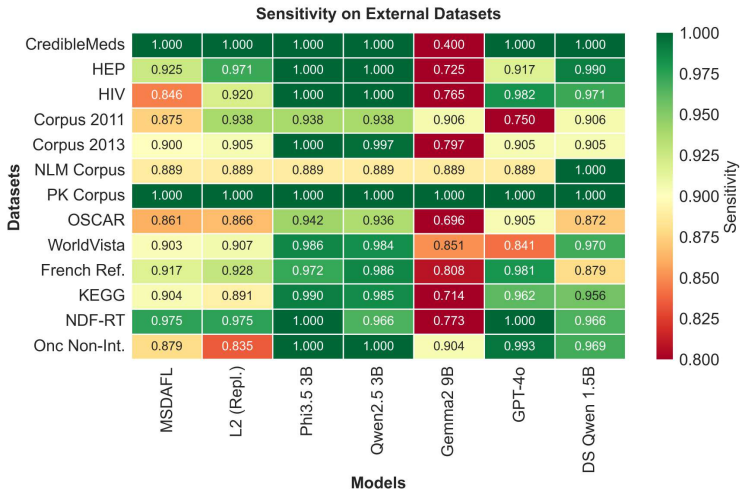


Figure 10.9: Sensitivity comparison across external datasets.

Interestingly, the traditional machine learning approaches showed slightly better overall performance when examining accuracy and F1-score (for details, see Figures 10.10 and 10.11, and Tables B.10 and B.11). The L2-regularized model had the highest F1-score (0.919) and average accuracy (0.918), with MSDAFL coming in second (accuracy: 0.916, F1-score: 0.918). With the best average accuracy (0.902) and F1-score (0.906) among LLMs, GPT-4o excelled on the HIV, OSCAR, French DDI Referrals, KEGG, NDF-RT, and ONC Non-Interruptive datasets. These results suggest that, although fine-tuned LLMs are excellent at detecting true positive interactions (high sensitivity), they may generate more false positives than conventional machine learning techniques. Even so, Phi-3.5

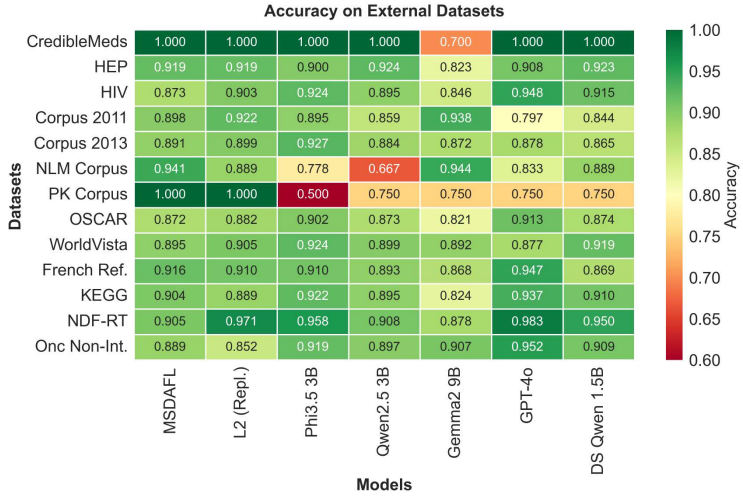


Figure 10.10: Accuracy comparison across external datasets.

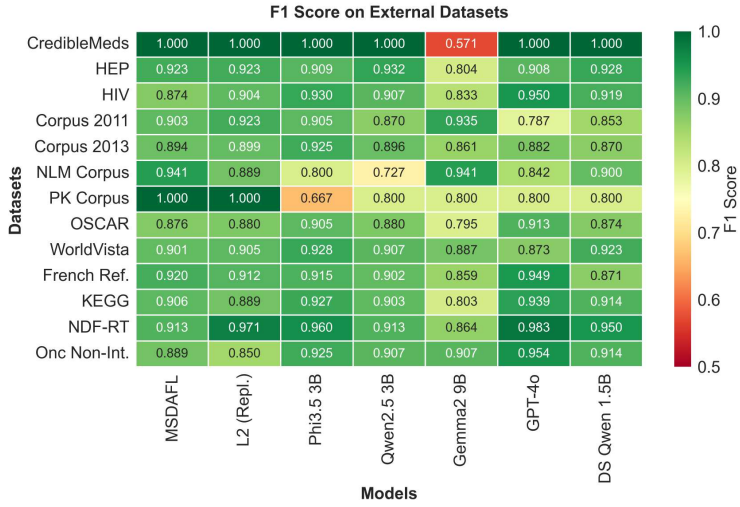


Figure 10.11: F1 comparison across external datasets.

3B's remarkable sensitivity (0.978) is a big plus in DDI prediction scenarios where it is usually riskier to miss a possible interaction (false negative) than to mistakenly flag a non-interaction (false positive).

To understand if there are failure patterns, we conducted a qualitative error analysis across external datasets in the top-performing fine-tuned models, Phi-3.5 2.7B and GPT-4o.

Phi-3.5 2.7B showed a strong ability to identify true DDIs but tended to overestimate interactions. False positives were associated with higher molecular complexity (i.e., SMILES string length) in 50% of datasets (e.g., KEGG: 125.1±104 SMILES characters for false positives vs. 114.2±71 for false negatives) and with a higher number of target genes in 60% of cases (e.g., HIV: 8.9±17.0 number of target genes for false positives vs. 4.9±9.6 for false negatives). Some drugs were systematically misclassified as interacting in certain datasets (e.g., capecitabine, insulin degludec, and fexofenadine in the KEGG dataset), although the pattern is not generalizable.

GPT-4o exhibited a more conservative behavior, but with the opposite bias, i.e., difficulty identifying true interactions. False negatives were linked to higher SMILES complexity in 60% of datasets (e.g., KEGG: 197.7±227.8 for false negatives vs. 124.7±106.9 for false positives), while no consistent patterns emerged for the number of target genes. Only four drugs showed recurring errors (i.e., metformin, thiamine, pseudoephedrine, eslicarbazepine acetate) in more than one dataset.

Despite opposite biases (overestimation vs. underestimation), the models shared 18 problematic drugs (e.g., methazolamide, pseudoephedrine) that were misclassified by both LLMs. This aspect, which is independent of the model architecture, may suggest that certain drugs have features (e.g., structural or pharmacological) that current LLMs cannot capture, likely tied to complex mechanisms of action or atypical physicochemical properties that are difficult to encode linguistically.

Given the results, we can answer RQ10.4 as follows:

> ***$RQ_{10.4}$ Comparative Analysis:*** *Fine-tuned LLMs demonstrate performance comparable to the l2-regularized logistic regression and MSDAFL models on both the validation set and external datasets, while not requiring complex feature engineering, structured chemical preprocessing, or large-scale training data. Notably, smaller models like Phi-3.5 and Qwen2.5-3B achieve superior sensitivity scores while maintaining competitive accuracy and F1-scores.*

10.7 IMPLICATIONS, LIMITATIONS AND FUTURE WORK

This section discusses the practical implications of the findings and outlines limitations and future research directions.

10.7.1 *Practical Implications*

The findings have several implications for both the deployment of DDI prediction systems and their clinical applications. First of all, the excellent performance of smaller models (2–3 billion parameters) represents an important breakthrough in accessibility, because these models can run efficiently in various healthcare settings without requiring a heavy computational load or specialized infrastructure. The ability to deploy them locally also addresses important privacy concerns, since sensitive medical data can be processed on-premises rather than via cloud APIs, and their lightweight nature makes it easier to integrate them into existing healthcare information systems. Second, the high sensitivity achieved by the models in identifying DDIs across multiple datasets – an ability that is particularly valuable in cases of polypharmacy where multiple drugs are administered at the same time – makes them suitable as screening tools in clinical practice. Third, the models' ability to process multiple types of drug information simultaneously could offer opportunities to uncover emerging patterns and hidden correlations that are unexplored to date.

Furthermore, they could support clinical decision-making at various levels:

- Pre-prescription screening to identify potential interactions before medication is prescribed
- Support for pharmacists in medication review processes
- Aid in clinical research for identifying potential drug combinations for investigation
- Real-time decision support in emergency medicine where rapid assessment of drug interactions is crucial

The combination of all these aspects implies that, given their computational efficiency, smaller LLMs could be used as practical tools in everyday clinical workflows to improve patient safety.

10.7.2 *Future Directions*

Our study demonstrated the potential of LLMs for DDI prediction, but it also outlined several research directions that warrant further investigation. The strong performance of smaller models suggests exploring even more efficient architectures and biomedical LLMs designed explicitly for processing molecular and biological data; for example, by developing specialized pre-training strategies that incorporate domain-specific knowledge about DDIs and biological pathways. A second interesting direction is including additional drug-related information into the LLM prompt, which could enable more precise predictions regarding the timing and severity of potential interactions. Indeed, although our approach combines several types of drug characteristics, future work could explore other information, such as pharmacokinetic properties, metabolic pathways, and the temporal aspects of drug administration. Additionally, future work could explore more explicit representations of interaction directionality, such as incorporating directional markers or comparing predictions for reversed drug pairs (A,B) versus (B,A) to validate the LLMs' understanding of asymmetric interactions and strengthen the modeling of directional drug-drug relationships. A third possible direction to investigate is the interpretability of model predictions, which would help build trust in these tools and provide valuable suggestions to healthcare practitioners. Future research could develop approaches to explain why some drug combinations were flagged as dangerous, for example, by exploring the attention weights or generating importance scores to help delineate the relative importance of each model input in prediction.

Finally, to improve the usefulness of LLMs in managing novel medications and rare interactions, future studies could investigate methods for continuously updating models' knowledge base as soon as new information on drug interactions becomes available.

10.7.3 *Limitations*

Our study faced a few potential limitations, which we systematically addressed through our methodology. Regarding experimental design, the selection of fine-tuning parameters could have influenced model performance; we mitigated this through comprehensive hyperparameter optimization using Optuna. The potential bias from random training/validation splits was addressed by validating the findings across 13 external datasets. Following the methodology in [198], we used balanced datasets to prevent learning bias. Another limitation relates to the negative sampling strategy we used, i.e., random generation of negative drug pairs, which carries the theoretical risk of mislabeling future true positives as negatives. To mitigate this issue, we generated negatives exclusively from drug pairs absent in our extensive interaction database (over 1.4 million known interactions from 14 sources) and constrained selection to drugs targeting at least one gene (according to established DDI prediction principles [198]). The validation across 13 different external datasets and the excellent results show that this method did not introduce significant noise, even though more complex sampling techniques (such as structural dissimilarity sampling or pharmacological class separation) could theoretically increase robustness. Concerning generalizability, while the fine-tuning analysis included only five selected models (one proprietary and four open-weights), we ensured diversity by choosing models with different architectures and sizes (ranging from 1.5B to over 250B parameters). The robustness of our approach was further validated across 13 diverse external datasets representing various domains and complexity levels. To ensure the reliability of the results, we performed five repeated classifications for fine-tuned models (RQ10.3 and RQ10.4), which demonstrated consistent performance with no variability in the outcomes. Beyond methodological issues, LLM architectures present ethical and regulatory obstacles to real-world clinical implementation. LLMs' black-box nature makes clinical decision-making more difficult to interpret, and obtaining regulatory approval as medical decision support tools (such as FDA approval for medical devices) would necessitate thorough clinical validation studies and adherence to healthcare regulations. Given these factors,

it would be more appropriate to use LLM-based systems for DDI prediction as screening tools under clinical supervision as opposed to independent decision-makers.

10.8 CONCLUSION

This chapter presents the first investigation of LLMs for DDI prediction, introducing a novel approach that leverages these models' ability to process multiple drug characteristics (i.e., SMILES, target organisms, and genes) simultaneously as text input through carefully designed prompts. While the findings across 13 external datasets demonstrates that LLMs show limited effectiveness in zero-shot settings and improvement but variable performance in few-shot settings, fine-tuned models achieved remarkable performance, with smaller models like Phi-3.5 (2.7B parameters) performing comparably to or better than larger models and state-of-the-art machine-learning approaches. All this suggests that model size is less critical than task-specific adaptation for effective performance and provides evidence that LLMs can be powerful tools for DDI prediction. These findings has several practical implications, making DDI prediction more accessible for clinical applications without requiring extensive computational resources. Future work could develop more efficient LLM architectures, incorporate additional drug-related information into the LLM's prompt, and improve model interpretability for clinical applications.

Part IV

CONCLUSION AND FURTHER RESEARCH DIRECTIONS

This part presents the key insights derived from our contributions across software engineering and healthcare domains. The chapters distill cross-cutting lessons learned that challenge conventional assumptions about model performance, adaptation strategies, and deployment practices. Furthermore, we highlight important unresolved problems and suggest promising avenues for future research that may improve the workflow integration of LLM-based systems in practical applications.

11

LESSONS LEARNED AND OPEN ISSUES

This chapter outlines the main insights we derived from examining LLM-based applications across software engineering and healthcare domains, besides the ones specific of each contribution, and discusses the lessons learned and open issues that help inform both current practice and future research.

11.1 LESSONS LEARNED

Based on the results described in Part II and Part III, we can confidently assert that LLMs, when properly adapted, can effectively support various software engineering tasks (e.g., use case quality enhancement, COSMIC functional size measurement automation, and issue classification) as well as healthcare tasks (drug reaction management and drug-drug interaction prediction).

Beyond the specific results reported in each chapter, the extensive analysis conducted in this dissertation across various domains and task types allows us to extract further and more detailed lessons learned, which we report below and summarize in Table 11.1 along with their associated research goals and the chapters from which they have been derived.

- **Lesson 1.** *The Surprising Reality of Model Performance.* One of the main findings of this thesis indicates that the assumption that larger models with more parameters perform better does not always hold in all contexts, but there is a more complex reality. For example, in classification tasks [86], Qwen-2.5 obtained better results than GPT-3.5, despite having significantly fewer parameters (32B vs. 175B). This was not an isolated case. In drug-drug interaction prediction [79], Phi-3.5, with only 2.7 billion parameters, performed better than GPT-4o, while Gemma 3, with 9 billion parameters, was competitive with proprietary models, such as Anthropic Sonnet, in drug reaction management [77], even though it

is significantly smaller. All this suggests that the architectural design of LLMs counts more than their raw computation scale. The way a model processes and integrates information, how it handles attention mechanisms, and how its layers interact seem to be more influential than the number of parameters the model has. This insight has huge implications for practitioners and researchers. Instead of simply defaulting to the largest model as a matter of principle, evidence suggests that a thoughtful selection based on task characteristics may yield better performance. But also, for organizations with limited computational resources, this finding is astounding, because organizations do not need to invest substantially in infrastructure to be performant. This lesson goes beyond performance.

Smaller and better-designed models often offer many additional benefits, especially when one considers speed of inference, computational cost, and the possibility of deploying models locally for privacy-sensitive applications. The experiments we have carried out in healthcare have also shown that these smaller models had the same safety profile as their larger counterparts and complied with institutional requirements.

- **Lesson 2.** *Task-Specific Adaptation: The Great Performance Equalizer.* Model architecture is the first building block, but our experiments also revealed that how a model is expected to be adapted to specific tasks matters more than its initial capabilities. We clearly showed this principle in our work on DDI prediction [79], where fine-tuning transformed average performers into effective specialized tools.

  The results were nothing short of remarkable. Models that performed poorly in zero-shot settings, with an accuracy range of 50-60%, after fine-tuning reached performance levels above 90%. By way of example only and without limitation, our smallest model, 1.5 billion parameter "Deepseek R1 Distilled" Qwen, achieved results competitive with systems many times its scale once it had adapted to the domain. Similar effects were observable also in other applications. For instance, in functional size measurement [75], few-shot

learning gained up to 99% accuracy, and RAG coupled with prompt engineering patterns obtained high classification accuracy (98.77%- 99.80%) maintaining a perfect safety profile for drug administration management [77].

The commonality among all was that domain adaptation technique, whether it was fine-tuning, few-shot learning, or specialized prompts, was a performance multiplier. This lesson is particularly interesting because it also represents a democratization opportunity for AI deployment. Organizations do not have to pay for premium models to get excellent performance in their specific domain. A well-adapted model can outperform flagship, expensive models.

The implications for practical use are clear. If we begin to treat models as adaptive projects to be shaped for an intended use as opposed to static artifacts, we can deploy powerful LLM-based application. This approach represents a paradigm shift for achieving optimal performance, from simply acquiring powerful (but costly) models to developing expertise in adaptation techniques.

- **Lesson 3.** *Context Preservation: Why How You Handle Information Matters More Than How Much You Have.* We made a counterintuitive discovery during our experiments to handle long, complicated inputs: smart-processing information often leads to better outputs than "just" giving the model all that information at once. The lesson learned fundamentally changed how we think about information-rich tasks. The issue report classification experiments we performed [86] are the most transparent and surprising examples of this aspect. The proposed approach for managing lengthy issue reports (e.g., MapReduce-like strategy), decomposing them into smaller pieces and recursively processing them, always outperformed both simple truncation and direct ingestion of full-length content across different model architectures. This pattern was certainly not limited to text classification.

  In measuring functional size as well [75], breaking down complex use case scenarios into manageable steps and then synthesizing the results yielded more accurate estimates than attempting to process everything simultaneously. The smart

contract generation [74], likewise, worked well with an iterative, chunking process, as understanding occurred cumulatively, rather than requiring an immediate, comprehensive analysis and code generation. What may be most confounding and interesting about this discovery is that it was not an intuitive dimension, as we typically think that "more context can only be better". Even when models theoretically could handle longer inputs, structured decomposition approaches delivered superior results.

The evidence suggests that models, like humans, can become overwhelmed by information density and benefit from organized, sequential processing. This lesson implies that effective AI deployment requires rethinking how we present information to models, moving from "dump everything and hope for the best" approaches toward carefully structured information flows and organized and presented in digestible, contextually relevant segments.

- **Lesson 4.** *Method-Model Interactions: The End of One-Size-Fits-All Solutions.* Another humbling outcome that emerged from our evaluations was that successful AI deployment requires matching specific methods to specific models. There are no universal best practices that are equally effective across all models (and vice versa), and we learned this repeatedly when we tried the same methods across different model architectures, only to get very different results. This discovery implies that successful deployment requires systematic experimentation with model-method combinations for each specific use case. The one-size-fits-all mentality that dominates much of current practice is anything but effective – it can be actively counterproductive.

  These implications go beyond the technical dimensions and extend to organizational strategy, because teams need to become experts not just at models or at methods (which is itself difficult enough), but also at learning the varied ways in which those models and methods interrelate. Implementation of AI is no longer a relatively simple set of steps regarding using an established (best) method, and has migrated to a separate practice of rigorous empirical validation.

- **Lesson 5.** *Ensemble Methods: When More Votes Don't Equal Better Decisions.* According to traditional machine learning knowledge, ensemble methods can improve performance by leveraging diversity among individual learners. However, the analysis revealed an arguably more complex result: ensemble methods may perpetuate errors rather than reduce them when the diversity assumption does not hold. In particular, we found that, when multiple models make the same mistake for the same reason (e.g., similar misconceptions or similar biases from training data), aggregating predictions simply replicates the model error, as expected from ensemble theory when base learners lack sufficient diversity [86].

  This interesting finding has serious implications for mission-critical applications (e.g., healthcare diagnoses, financial risk assessments) that rely on ensemble strategies to increase reliability based on the assumption that "more opinions" mitigates catastrophic error – without guaranteeing that those "opinions" are actually diverse. All this does not invalidate ensemble methods but just points out their limitations in systems with correlated failure modes, especially in LLM-based applications. In LLM-based systems, more advanced methods capable of recognizing and compensating for systematic patterns in model failures are a need, not a choice.

- **Lesson 6.** *Safety-First Design: Perfect Protection with Improved Efficiency.* Healthcare applications require an indispensable commitment to patient safety, but this requirement must not come at the expense of system usability. Our work on clinical decision support has shown that it is entirely possible to maintain high levels of safety while improving performance.

  We developed approaches [77] capable of distinguishing between truly dangerous situations and routine situations that posed minimal risk, through the contextual processing of clinical notes and patients' medical histories at the same time, without relying on simple rule-based matching. The strong results and validation across synthetic and real-world datasets and across different model architectures prove

that more intelligent, context-aware approaches can improve both dimensions, safety and effectiveness.

Safety-first design, properly executed, becomes an enabler of effectiveness rather than an obstacle to it.

- **Lesson 7.** *Confidence-Based Frameworks: The Bridge to Clinical Trust.* The accuracy of predictions in clinical AI systems is not enough to earn the trust of healthcare professionals; mechanisms are needed that measure the uncertainty of these predictions and guide appropriate human oversight. Our work in healthcare has shown that assigning multidimensional confidence scores to LLM responses creates the missing bridge between model recommendations and clinical judgment [77]. The framework we designed differentiates types of uncertainty, which we then analyzed to identify confidence patterns in LLM responses and define a sophisticated risk stratification (i.e., thresholds, ranges, and actions): cases with high confidence across all dimensions can proceed with automated recommendations, while those showing uncertainty in any metric trigger progressive levels of human review.

  Doing so, the system never prevents clinical oversight but instead guides when such oversight is most needed, creating a collaborative rather than competitive relationship between AI and clinical expertise. Healthcare professionals involved in the assessment of our approach responded positively to this clarity about uncertainty, proving that they value understanding not just what the system recommends but how certain it is about those recommendations.

- **Lesson 8.** *User Acceptance Through Transparency and Control.* The acceptance of AI systems does not depend only on the capabilities they offer or their effectiveness. Our evaluations revealed that, whatever the domain or task, the transparency in decision-making processes and the preservation of human experts' autonomy were the key factors determining whether users embraced or rejected AI assistance.

  On the one hand, users can follow the logic, spot possible problems, and remain confident in the outcomes when sys-

tems make their reasoning explicit, drastically altering the paradigm: a "transparent colleague" appears from the "black box's" opacity.

On the other hand, the success of LLM-based systems depends on a careful strategic placement, specifically the use of advanced support tools as opposed to substitute decision-makers.

- **Lesson 9.** *Modular Architectures: Flexibility as a Core Design Principle.* Real-world production environments are heterogeneous and often incompatible with rigid LLM-based solutions. Our research has shown that modular architectures capable of adapting to different infrastructure constraints and organizational requirements were necessary to implement AI successfully in different contexts. This flexibility manifested across several dimensions.

  First, cloud versus local deployment was perhaps the most important divide. Some organizations required cloud-based solutions for scalability and maintenance simplicity, while others demanded local deployment for data sovereignty and security compliance or resource-constrained environments. Our healthcare systems operated successfully in both modes without performance degradation, using the same functional logic with different deployment "wrappers."

  Second, different organizations had very different needs when it came to integration. Some needed stand-alone applications that users could use on their own, while others needed complete integration with workflow management tools, development environments, or electronic health record systems.

  Third, privacy and compliance constraints represent another dimension to consider. Navigating through different regulatory contexts was made possible primarily thanks to the ability to replace system components – from adopting local models instead of cloud APIs, to implementing alternative protocols for data management, down to regulating audit logging mechanisms.

Such requirements and constraints lead us to reconsider the role of architectural flexibility; it must be a core design principle in order to transform AI systems from specialized tools that require specific environments into adaptable platforms capable of conforming to existing organizational structures and constraints rather than imposing radical infrastructure changes.

- **Lesson 10.** *Context-Appropriate Industry Engagement: Adapting the Depth of Collaboration to Domain Requirements.* The nature of collaboration with professionals and companies has varied across our contributions. In safety-critical healthcare applications (i.e., HELIOT [77]) and highly specialized software engineering domains (i.e., CosMet [75]), professionals were involved throughout the entire research process. Medical professionals contributed to synthetic dataset creation, clinical decision-making process definition, risk threshold specification, and final validation.

  Similarly, COSMIC experts participated in identifying few-shot examples, prompt refinement iterations, and accuracy criteria definition. In both cases, research questions, solution architectures, and evaluation criteria emerged from collaborative knowledge co-creation rather than being academically predefined and subsequently validated. Other contributions followed a different model where problem identification emerged from systematic literature analysis and professionals contributed primarily during the validation phase (e.g., controlled experiments, usability evaluations, and feedback on practical deployment). This approach enabled both exploring LLM capabilities and creating solutions that met professional quality standards.

  Both engagement models proved valid.

  On one hand, end-to-end collaboration produced solutions tightly aligned with professional practice and regulatory requirements, while validation-focused engagement enabled broader exploration of LLM adaptation strategies across diversified tasks. The key insight is that the depth of engagement should be determined by domain and task characteristics rather than following a one-size-fits-all model.

Table 11.1: Summary of Lessons Learned and their relationship with the Research Goals indicated in the Chapter 1.

Lesson Learned	Chapter	Cross-domain
The Surprising Reality of Model Performance [RG4]	8 (Software Engineering) 9 (Healthcare) 10 (Healthcare)	✓
Task-Specific Adaptation: The Great Performance Equalizer [RG1]	3 (Software Engineering) 4 (Software Engineering) 5 (Software Engineering) 6 (Software Engineering) 7 (Software Engineering) 8 (Software Engineering) 9 (Healthcare) 10 (Healthcare)	✓
Context Preservation: Why How You Handle Information Matters More Than How Much You Have [RG1, RG2]	4 (Software Engineering) 5 (Software Engineering) 7 (Software Engineering) 8 (Software Engineering) 9 (Healthcare) 10 (Healthcare)	✓
Method-Model Interactions: The End of One-Size-Fits-All Solutions [RG1, RG4]	8 (Software Engineering) 9 (Healthcare) 10 (Healthcare)	✓
Ensemble Methods: When More Votes Don't Equal Better Decisions [RG1, RG2]	8 (Software Engineering)	
Safety-First Design: Perfect Protection with Improved Efficiency [RG2, RG3]	9 (Healthcare) 10 (Healthcare)	
Confidence-Based Frameworks: The Bridge to Clinical Trust [RG1, RG2]	9 (Healthcare)	
User Acceptance Through Transparency and Control [RG2, RG3]	3 (Software Engineering) 4 (Software Engineering) 5 (Software Engineering) 6 (Software Engineering) 7 (Software Engineering) 8 (Software Engineering) 9 (Healthcare) 10 (Healthcare)	✓
Modular Architectures: Flexibility as a Core Design Principle [RG1, RG3]	5 (Software Engineering) 6 (Software Engineering) 9 (Healthcare)	✓
Context-Appropriate Industry Engagement: Adapting the Depth of Collaboration to Domain Requirements [RG1, RG2, RG3, RG4]	3 (Software Engineering) 4 (Software Engineering) 5 (Software Engineering) 6 (Software Engineering) 7 (Software Engineering) 8 (Software Engineering) 9 (Healthcare) 10 (Healthcare)	✓

11.2 CROSS-CONTEXT GENERALIZABILITY AND BOUNDARY CONDITIONS

Table 11.1 reports the lessons that showed cross-domain applicability within our research. However, understanding whether and how these results extend beyond the specific cases and companies involved requires a deeper analysis of the mechanisms underlying the empirical observations.

Three elements suggest that the results transcend the specific contexts studied. First, we observed the same causal mechanisms manifesting independently in situations with profoundly different requirements. Task-specific adaptation (Lesson 2) produced performance improvements of the same magnitude (typically 20-40 percentage points) in both classification and measurement tasks, and in contexts where the priority was efficiency as well as patient safety. These contexts share neither the cognitive nature of the task, nor the type of knowledge required, nor the operational constraints. When the same causal principle produces quantitatively similar effects in such heterogeneous situations, it is reasonable to conclude that we are observing a property of how LLMs acquire and apply specialized knowledge, rather than an artifact of particular cases.

Second, we varied not only the application contexts but also the model architectures, essentially creating a cross-validation matrix (cross-LLM adaptation). The finding that smaller models can compete with larger ones after appropriate adaptation (Lesson 1) held consistently across different LLM families, both proprietary and open-weight, with architectures ranging from 1.5B to over 175B parameters and spanning fundamentally different tasks. This convergence indicates that we are not observing peculiarities of specific implementations, but intrinsic properties of how architectural efficiency interacts with domain specialization. The fact that this pattern persists regardless of the model's manufacturer, license, or size implies that architectural alignment with the task is more determinative than the LLM's computational capacity–a principle that should hold even outside the specific cases studied in the dissertation.

Third, the validity of cross-cutting patterns is reinforced by the fact that, where we observed differences between domains, these

were not random but linked to objective characteristics of the contexts. For example, confidence thresholds and safety mechanisms (Lessons 6 and 7) required different calibrations in healthcare compared to software engineering, but this variation was predictable. Indeed, in healthcare, a false negative can be fatal for the patient, whereas in software engineering, errors typically have less severe consequences. The fact that differences in results correspond exactly to objective differences in the safety requirements of the two domains suggests that we are capturing real causal relationships.

Explicitly recognizing the boundaries of generalizability is equally important. The companies involved in our studies have relatively structured processes and formalized artifacts, which facilitated the application of the proposed approaches. Organizations with less mature processes might require an additional layer of formalization. Similarly, the professionals we involved possessed significant expertise; acceptance of the solutions might differ with less experienced users. Another aspect to consider concerns the linguistic context. Although we managed non-English datasets, the need to translate artifacts into English (due to LLMs' linguistic bias) poses a constraint that could limit applicability in domains where language conveys important cultural nuances.

A further element of generalizability emerges from the methodology itself. The framework based on failure modes and adaptation levers is not specific to the domains studied, but provides a template for approaching LLM integration in any specialized domain. An organization in a different domain can identify its own failure modes and select the most appropriate levers. This aspect perhaps represents the most general contribution, namely, not only solutions to specific tasks but also an approach for building LLM-based solutions in specialized domains.

11.3 OPEN ISSUES

Although our investigation has shed light on several aspects of LLM implementation, it has also highlighted some unaddressed challenges. These open issues reflect not only technical challenges but also opportunities that could guide the evolution of our approaches to LLM-based system integration.

- **Open issue 1.** *Predictive Model-Method Compatibility Framework.* According to our research, method-model interactions are extremely unpredictable, and the same approach can yield radically different results depending on the architecture. To find the best combinations, practitioners currently have to rely on thorough empirical testing, which becomes unaffordable as the number of models and techniques available increases. The question of whether there are underlying training paradigms or architectural features that could predict compatibility patterns and help practitioners narrow their search before spending a lot of money on costly experimentation is still unanswered.

- **Open issue 2.** *Dynamic Confidence Calibration Across Domains.* Although we have created confidence-based frameworks for particular uses, it is still difficult to automatically adjust confidence thresholds across various risk profiles and domains. Our healthcare implementations required manual threshold setting based on clinical expertise. However, this approach does not scale to domains where such expertise may not be available or where risk tolerances vary significantly. The unanswered question is whether confidence calibration can be automated through domain-independent mechanisms, or whether each implementation context requires bespoke calibration by domain experts. The challenge becomes even more pressing for organizations seeking to integrate AI solutions across different functional areas with different risk profiles and decision-making processes.

- **Open issue 3.** *Systematic Bias Detection and Mitigation in Ensemble Failures.* The discovery that an ensemble of models with common systematic biases can compound errors instead of fixing them reveals a more serious problem: traditional validation may not detect LLMs' systematic errors when they are trained on similar data or share correlated architectures. Therefore, addressing shared failure modes and devising strategies for mitigating these persistent biases, remain open needs.

- **Open issue 4.** *Longitudinal Performance Stability and Adaptation.* Though real-world implementation necessitates sustained performance over months or years as input distributions change, user behaviors evolve, and organizational contexts shift, performance evaluations of LLM-based systems are typically "point-in-time" (i.e., conducted at a single moment). How to preserve performance stability in this situation while allowing for suitable adaptation to shifting circumstances is unknown. This problem involves both organizational and technical components, such as managing ongoing validation, stakeholder communication, and system evolution, as well as identifying when models need to be retrained or recalibrated. The goal of the research is to determine whether AI systems can be designed with built-in adaptation mechanisms that react to environmental changes appropriately without needing to be completely redeveloped or if replacing the entire system on a regular basis is a more sustainable long-term approach.

Among the four open issues, the main goal is to devise techniques and prioritization approaches, which would be a direct follow-up of the work presented in this thesis.

12

FURTHER RESEARCH DIRECTIONS

The results gathered across software engineering and healthcare domains points toward several promising research trajectories that could fundamentally reshape how LLM-based applications integrate into established workflows. Some directions address current technical limitations, while others explore entirely new approaches for human-AI collaboration, especially within domains characterized by high criticality and complex decision-making processes.

12.1 ARCHITECTURAL INTELLIGENCE AND PREDICTIVE COMPATIBILITY

The fact that LLMs' architectural design is more significant than their size challenges conventional wisdom regarding model performance and opens up new avenues for computational efficiency in specialized fields.

More complex assessment frameworks that methodically characterize the connection between model architectures and adaptation method performance may be developed in future studies. These frameworks could reveal measurable patterns in LLM architectures (e.g., in layer interactions, parameter distributions, and attention mechanisms) that are associated with the success of adaptation techniques across multiple domains. Instead of relying solely on empirical trial-and-error methods, practitioners could utilize these findings to inform the development of model selection guidelines, which would aid in selecting appropriate models and adaptation methods based on task characteristics and resource constraints.

The ultimate goal of this research trajectory is to develop predictive frameworks that can recommend optimal model-method combinations based on domain requirements, computational constraints, and performance objectives. Such frameworks could drastically cut down on the experimental overhead currently needed

for every new deployment scenario, but they would still need empirical validation.

12.2 CONTEXTUAL INTELLIGENCE AND DYNAMIC INFORMATION ARCHITECTURE

Current context management methods in language models are insufficient for information orchestration for real-world applications.

A first research direction in this area could be the development of adaptive context filtering mechanisms capable of dynamically identifying and prioritizing information based on task requirements and application domain. These systems, optimized with reinforcement learning, could automatically focus computational resources on information that most likely influences correct decisions by using learned attention patterns that separate signal from noise in complex documents.

Another area that researchers may investigate is hierarchical context architectures, which can process data at multiple abstraction levels simultaneously without consuming excessive computational resources. In such a hierarchy, low-level processors could handle specific technical information (like code syntax, medical measurements, and legal citations), while higher-level systems could focus on conceptual relationships, causal patterns, and reasoning structures used by human experts.

Temporal context management and context validation systems follow the same direction. The former have the potential to progressively integrate new information streams with the goal of building a cumulative understanding of the domain over time –unlike current approaches that handle each interaction separately. For instance, software engineering applications can keep track of how a project is changing and what architectural choices are being made during the development cycle. Healthcare applications can keep track of changing patient histories by combining new clinical observations with old patterns.

The latter –context validation mechanisms– could evaluate the reliability of information in real-time during inference by using consistency checking and multi-source verification to find potentially contradictory or unreliable information before it affects im-

portant decisions. Context validation mechanisms could use cross-reference verification and source reliability scoring to identify potentially contradictory information before it leads to incorrect responses.

12.3 ADAPTIVE LEARNING AND CONTINUOUS EVOLUTION

Maintaining LLM-based systems over time requires continuous learning approaches that go well beyond current incremental training methods. Such systems must evolve with their domains while preserving the reliability standards required by mission-critical applications.

Two research lines lead in this direction.

The first is represented by modular learning architectures that employ specialized memory modules for different types of knowledge –foundational principles, emerging practices, experimental approaches– with validation mechanisms to ensure consistency between modules, and knowledge graphs to incorporate new information while maintaining logical coherence across the entire knowledge base.

The second line of research is represented by collaborative learning systems that enable co-evolution of LLM-based systems and human expertise through ongoing expert collaboration. Not only can these systems take direct instructions, but they can also learn from expert reasoning patterns, adaptation techniques, and decision-making processes. Such designs would create productive human-AI partnerships where systems become more effective through use while experts develop better ways to guide and collaborate with AI tools over time.

12.4 CONFIDENCE ARCHITECTURE AND EXPLAINABLE INTELLIGENCE

Confidence-based frameworks increase the trust of users in LLM-based solutions, as we observed in Section 11. However, existing confidence metrics are not enough on their own if they are not combined with expert-level explanations of LLM-based system responses.

A crucial research direction involves developing solutions that dynamically adjust explanation depth based on confidence levels. When confidence is high, explanations can be brief and action-focused; when confidence is low, systems should provide comprehensive justifications that include decision points, alternative interpretations, and supporting evidence. That is to say, transparency will now be calibrated to the level of certainty within the system, rather than uniform details for each prediction.

A significant advancement would involve LLM-based systems that can distinguish between different types of uncertainty, recognizing whether uncertainty stems from training data gaps or intrinsic ambiguities within the specific domain. In healthcare settings, this distinction becomes crucial when differentiating uncertain diagnoses due to missing patient information from those reflecting case complexity requiring specialist consultation.

Context-aware explanation systems represent another promising research direction, offering personalized transparency while maintaining computational efficiency. A healthcare system might offer different explanatory levels between emergency physicians and specialists, while software development tools could modulate the granularity of information provided based on programmer experience and code complexity.

Dual-path reasoning architectures represent a more advanced research direction. These systems employ parallel processing streams: one path handles decision-making while another simultaneously generates and monitors the reasoning process. Instead of relying on ad hoc reconstructions, this method produces explanations based on genuine reasoning by creating audit trails that domain experts can verify.

12.5 ENSEMBLE INTELLIGENCE AND COLLABORATIVE ARCHITECTURES

When ensemble approaches consolidate errors rather than correct them, a serious issue occurs: different models may have systematic biases brought on by similar training datasets or convergent development approaches.

The development of diagnostic methods to uncover hidden correlations between the errors of seemingly independent models

should be the main focus of future research. It is still difficult to distinguish between accurate convergence and shared systematic biases, since both generate ensemble consensus but have drastically different implications for system reliability.

Future studies could investigate methods for creating training datasets that expose models to complementary representations of the same phenomena. This approach acknowledges that variety in data perspectives, not just architectural differences, is the source of true diversity.

Another interesting research direction is the development of ensemble systems that can detect when their agreement derives from shared errors rather than accurate mutual validation, using, for instance, "sentinel" models, which are specifically trained to detect systematic errors that lead to a false consensus.

New evaluation frameworks that go beyond conventional metrics are also needed. Adversarial datasets may reveal flaws that are not visible in routine testing because they are designed to reveal latent biases that only manifest through ensemble aggregation.

Part V

APPENDIX

A

APPENDIX HELIOT

A.1 USABILITY QUESTIONNAIRE

You will be presented with 7 clinical cases, grouped in scenarios, where the CDSS HELIOT provides decision support for medication prescribing. For each case, please review the patient information, clinical notes, and both HELIOT's output and a traditional CDSS output, then answer the questions below.

In these scenarios, traditional CDSSs raise **Interruptive Alert**. Interruptive Alert stops the prescribing workflow; you cannot proceed with the prescription unless you actively override the alert by providing a justification, or you choose to cancel the prescription altogether. Alert fatigue arises when such interruptive alerts are triggered frequently, even for known patient tolerances or expected side effects [64, 200, 237, 288, 289, 297, 298, 313]. HELIOT addresses this by using **Non-Interruptive Alert** in these cases, allowing clinicians to continue their workflow without unnecessary interruptions. Therefore, it is not necessary to block the prescription workflow.

* *Indicates mandatory questions*

Scenario: Cross-reactivity with documented tolerance

The following cases involve patients who have had reactions to drugs or drug classes but report tolerance for the medication to be prescribed. In addition, there are cases where patients have had side effects.

Case 1

Patient History: Patient shows Anaphylaxis reaction within Opioids class but has documented tolerance to fentanyl since 2016
Prescribed drug: Fentanil

HELIOT Output: Patient has anaphylaxis to opioids but documented tolerance to fentanyl since 2016. Fentanyl is the active ingredient in the prescribed drug. Drug class cross-reactivity exists but with documented tolerance to fentanyl. Therefore the drug can be prescribed. Non-Interruptive Alert
Traditional CDSS Output: Interruptive Alert

Case 2

Patient History: Urticarial reaction after taking pyrazolones (Zepelin, Zerinol, Novalgina) and sulfonamides (Bactrim); itching after taking Danzen and opioids; tremor after taking Plasil. Food intolerances to pepper, onion, salami, sausage. Avoids shellfish and mollusks and consumes foods rich in histamine (spinach, tuna, eggplant, pineapple, strawberries, kiwi) with caution.
Prescribed drug: Morphine
HELIOT Output: The patient has documented reactions to opioids, such as itching. There is no direct reactivity to the active ingredient morphine hydrochloride, nor to its excipients. There is no documented tolerance to morphine in patient's medical history. Despite the side effect, the medication can still be prescribed with caution. Non-Interruptive Alert
Traditional CDSS Output: Interruptive Alert

Case 3

Patient History: Reports an allergy to Amoxicillin and beta-lactamase inhibitors, tolerates Meropenem.
Prescribed drug: Meropenem
HELIOT Output: The patient reports an allergy to amoxicillin and beta-lactamase inhibitors but tolerates meropenem. Although the patient is allergic to beta-lactam antibiotics, they tolerate meropenem, which is also a beta-lactam. This case is classified as drug class cross-reactivity with documented tolerance, meaning that the drug can be prescribed. Non-Interruptive Alert
Traditional CDSS Output: Interruptive Alert

Scenario: Cross-reactivity without documented tolerance

The following case involves patients who have had reactions to drugs or drug classes without reporting tolerance to the medication to be prescribed. In this scenario, CDSSs should raise interruptive alerts, because prescribing the specific drug may cause severe reaction to patients.

Case 4

Patient History: Reports anaphylaxis after taking opioids in 2021
Prescribed drug: Morphine
HELIOT Output: The patient reports anaphylaxis to opioids without a specific tolerance to morphine. This case is classified as drug class cross-reactivity without documented tolerance, meaning that the drug should not be prescribed because it may cause life-threatening reactions. Interruptive Alert
Traditional CDSS Output: Interruptive Alert

Scenario: Side effects, non-immune and non-life-threatening

The following cases involve patients who have experienced side effects from medications or classes of medications, for which the prescribing physician should be informed, but it is not necessary to block the prescription workflow and override alerts to move on prescribing the drug.

Case 5

Patient History: Patient showed Mild shortness of breath side effect, not indicative of hypersensitivity, with possible cross-reactivity within Opioids class, specifically to tramadol and dexketoprofen since 2014.
Prescribed drug: oxycodone
HELIOT Output: The patient has mild shortness of breath with opioids, specifically tramadol, which indicates a non-immune-mediated side effect. While oxycodone also belongs to the opioid class, sug-

gesting potential cross-reactivity, it can be prescribed with caution as there is no immune-mediated reaction or documented intolerance. Non-Interruptive Alert
Traditional CDSS Output: Interruptive Alert

Case 6

Patient History: Patient showed Vomiting side effect, not indicative of hypersensitivity, to INDOMETACINA (NSAID) in 2000.
Prescribed drug: indometacin
HELIOT Output: The patient experienced vomiting as a documented side effect to indometacin, which is the same active ingredient as that in the prescribed drug Indomethacin, indicating direct active ingredient reactivity. This reaction is non-life-threatening, as it is a side effect and not due to hypersensitivity, and should be considered non-immune and non-life-threatening. Non-Interruptive Alert
Traditional CDSS Output: Interruptive Alert

Case 7

Patient History: The patient reports itching after taking codeine and ibuprofen in 2012.
Prescribed drug: codeine
HELIOT Output: The patient reports itching after taking Codeine, an active ingredient in the prescribed drug, indicating a direct active ingredient reactivity. Given that this is a non-immune, non-life-threatening side effect, the medication can be prescribed but with caution. Non-Interruptive Alert
Traditional CDSS Output: Interruptive Alert The following cases involve patients who have had reactions to drugs or drug classes but report tolerance for the medication to be prescribed. In addition, there are cases where patients have had side effects.

Section A: Clinical Accuracy Assessment

For each scenario (1-7), please rate:

A1. Clinical Appropriateness of HELIOT's response:

Rate from 1 (completely inappropriate) to 5 (completely appropriate) *

	1	2	3	4	5
Case 1	○	○	○	○	○
Case 2	○	○	○	○	○
Case 3	○	○	○	○	○
Case 4	○	○	○	○	○
Case 5	○	○	○	○	○
Case 6	○	○	○	○	○
Case 7	○	○	○	○	○

A2. Risk Assessment Accuracy

How accurately does HELIOT identify and categorize clinical risks (i.e. Life-threatening situations, where drugs may cause severe reactions)? Rate from 1 (Very inaccurate) to 5 (Very accurate) *

	1	2	3	4	5
Case 1	○	○	○	○	○
Case 2	○	○	○	○	○
Case 3	○	○	○	○	○
Case 4	○	○	○	○	○
Case 5	○	○	○	○	○
Case 6	○	○	○	○	○
Case 7	○	○	○	○	○

Section B: Alert Fatigue and Appropriateness

B1. Alert Preference *

	HELIOT's Alert	Traditional CDSS Alert	No Preference
Case 1	◦	◦	◦
Case 2	◦	◦	◦
Case 3	◦	◦	◦
Case 4	◦	◦	◦
Case 5	◦	◦	◦
Case 6	◦	◦	◦
Case 7	◦	◦	◦

B2. Alert Fatigue Impact *

Rate how much HELIOT's approach would reduce alert fatigue compared to traditional CDSS *

◦ Increase alert fatigue
◦ No difference
◦ Slightly reduces alert fatigue
◦ Reduces alert fatigue
◦ Significantly reduces alert fatigue

B3. Alert Contextuality *

How well does HELIOT consider clinical context when generating alerts? *

◦ No contextual understanding
◦ Poor contextual understanding
◦ Adequate contextual understanding
◦ Good contextual understanding
◦ Excellent contextual understanding

Section C: Usability and Clinical Integration

C1. Information Clarity

How clear and understandable are HELIOT's recommendations? *

◦ Very unclear
◦ Unclear
◦ Somewhat clear
◦ Clear and understandable
◦ Very clear and easy to understand

C2. Clinical Reasoning Transparency

How well does HELIOT explain its clinical reasoning? *

◦ No explanation provided
◦ Poor explanation of reasoning
◦ Adequate explanation of reasoning
◦ Good explanation of reasoning
◦ Excellent explanation of reasoning

C3. Time Efficiency

How would HELIOT impact the time you spend reviewing medication alerts? *

◦ Increases time spent
◦ No change in time spent
◦ Slightly reduces time spent
◦ Moderately reduces time spent
◦ Significantly reduces time spent

C4. Trust and Confidence

How confident would you feel considering HELIOT's recommendations in your decisions? *

◦ Not confident at all
◦ Not very confident
◦ Somewhat confident
◦ Confident
◦ Very confident

Section D: Clinical Workflow Integration

D1. Clinical Decision Support Value

Overall, how valuable would HELIOT be for your clinical decision-making support? *

◦ Not valuable
◦ Slightly valuable
◦ Somewhat valuable
◦ Very valuable
◦ Extremely valuable

D2. Likelihood of Adoption

How likely would you be to use HELIOT in your practice if available? *

◦ Very unlikely
◦ Unlikely
◦ Somewhat likely
◦ Likely
◦ Very likely

Section E: Demographic Information

E1. Medical Speciality *

◦ Internal Medicine
◦ Emergency Medicine
◦ Primary Care/Family Medicine

◦ Clinical Pharmacology
◦ Anesthesiology
◦ Other: _____

E2. Years of clinical experience *

◦ < 5 years
◦ 5-10 years
◦ 11-20 years
◦ > 20 years

E3. Current practice setting *

◦ Hospital (inpatient)
◦ Outpatient clinic
◦ Emergency department
◦ Academic medical center
◦ Private practice
◦ Other: _____

B

APPENDIX LLMDDI

B.1 EXPERIMENTS WITH FULL FINE-TUNING

Tables B.1 and B.2 report the best results (average and class) achieved training 50% of the trainable parameters and using different fine-tuning configurations: learning rate (1e-5, 5e-6, 2e-5) and training duration (1, 1.5, 2, 3 epochs).

Table B.1: Full fine-tuning results for Phi-3.5 2.7B with half trainable parameters.

Model	Acc.	Prec.	Sens.	F1
Phi-3.5 2.7B (16 layers)	0.871	0.847	0.905	0.875

Table B.2: Detailed classification metrics for Phi-3.5 2.7B full fine-tuning with half trainable parameters.

Class	Precision	Recall	F1-score
No Interaction (0)	0.90	0.84	0.87
Interaction (1)	0.85	0.90	0.87

Tables B.3 and B.4 report the best results (average and class) achieved training four layers and using different fine-tuning configurations: learning rate (1e-5, 5e-6, 2e-5) and training duration (1, 1.5, 2, 3 epochs).

Table B.3: Full fine-tuning results for Phi-3.5 2.7B with 4 trainable layers.

Model Configuration	Acc.	Prec.	Sens.	F1
Phi-3.5 2.7B (4 layers)	0.691	0.647	0.840	0.731

B.2 ADDITIONAL RESULTS

Table B.4: Detailed classification metrics for Phi-3.5 2.7B full fine-tuning with 4 trainable layers.

Class	Precision	Recall	F1-score
No Interaction (0)	0.77	0.54	0.64
Interaction (1)	0.65	0.84	0.73

Table B.5: Zero-shot performance of different LLMs on DDI prediction.

# Params	Model	Acc.	Prec.	Sens.	F1
>250B	Claude3.5 Sonnet	**0.7358**	0.8859	0.5413	0.6720
	Gemini 1.5	0.7220	0.7995	0.5927	0.6807
	GPT-4o	0.6459	0.6710	0.5725	0.6178
< 2B	Deepseek Qwen1.5B	0.4807	0.4677	0.2789	0.3494
2-3B	LLaMa-3.2 3B	0.5009	0.5005	**0.9982**	0.6667
	Qwen2.5 3B	0.5037	0.5019	0.9761	0.6629
	Phi-3.5 2.7B	0.5358	0.5221	0.8440	0.6452
8-9B	Granite 3.1 8B	0.4734	0.3535	0.0642	0.1087
	LLaMa-3.1 8B	0.5294	0.5656	0.2532	0.3498
	Gemma2 9B	0.6376	0.8178	0.3541	0.4942
10-14B	Falcon 3 10B	0.5853	0.8394	0.2110	0.3372
	Mistral-Nemo 12B	0.6991	0.7395	0.6147	0.6713
	Qwen 2.5 14B	0.6569	**0.9212**	0.3431	0.5000
27-32B	Gemma2 27B	0.7303	0.8476	0.5615	0.6755
	Aya-Expanse 32B	0.5917	0.5617	0.8349	0.6716
	Qwen 2.5 32B	0.5982	0.6754	0.3780	0.4847
70-72B	LLaMa-3.3 70B	0.6477	0.6123	0.8055	**0.6957**
	Qwen2 72B	0.7119	0.7667	0.6092	0.6789

Table B.6: Random Few-shot performance of different LLMs on DDI prediction.

# Params	Model	Acc.	Prec.	Sens.	F1
>250B	Claude3.5 Sonnet	**0.8211**	**0.8661**	0.7596	**0.8094**
	Gemini	0.7394	0.8595	0.5725	0.6872
	GPT-4o	0.7477	0.7626	0.7193	0.7403
< 2B	Deepseek Qwen1.5B	0.5156	0.5209	0.3890	0.4454
2-3B	LLaMa-3.2 3B	0.5761	0.5443	0.9358	0.6883
	Qwen2.5 3B	0.5202	0.5113	0.9156	0.6561
	Phi-3.5 2.7B	0.6560	0.6254	0.7780	0.6934
8-9B	Granite 3.1 8B	0.6248	0.6104	0.6899	0.6477
	LLaMa-3.1 8B	0.5028	0.5014	**0.9963**	0.6671
	Gemma2 9B	0.6321	0.5889	0.8752	0.7041
10-14B	Falcon 3 10B	0.7128	0.7900	0.5798	0.6688
	Mistral-Nemo 12B	0.5229	0.5118	0.9927	0.6754
	Qwen 2.5 14B	0.6807	0.8530	0.4367	0.5777
27-32B	Gemma2 27B	0.7450	0,7708	0.6972	0.7322
	Aya-Expanse 32B	0.6771	0.7468	0.5358	0.6239
	Qwen 2.5 32B	0.6963	0.8365	0.4881	0.6165
70-72B	LLaMa-3.3 70B	0.6413	0.5958	0.8789	0.7102
	Qwen2 72B	0.7018	0.7209	0.6587	0.6884

Table B.7: Similarity-based Few-shot performance of different LLMs on DDI prediction.

# Params	Model	Acc.	Prec.	Sens.	F1
>250B	Claude3.5 Sonnet	**0.8376**	0.8345	0.8422	**0.8384**
	Gemini	0.7257	**0.8534**	0.5450	0.6652
	GPT-4o	0.7917	0.7659	0.8404	0.8014
< 2B	Deepseek Qwen1.5B	0.5046	0.5052	0.4422	0.4716
2-3B	LLaMa-3.2 3B	0.5431	0.5231	0.9780	0.6816
	Qwen2.5 3B	0.6394	0.6271	0.6881	0.6562
	Phi-3.5 2.7B	0.5789	0.5491	0.8826	0.6770
8-9B	Granite 3.1 8B	0.6138	0.5675	0.9560	0.7122
	LLaMa-3.1 8B	0.5009	0.5005	**0.9890**	0.6646
	Gemma2 9B	0.6881	0.6350	0.8844	0.7393
10-14B	Falcon 3 10B	0.7899	0.7696	0.8275	0.7975
	Mistral-Nemo 12B	0.6321	0.5811	0.9468	0.7202
	Qwen 2.5 14B	0.7459	0.7780	0.6881	0.7303
27-32B	Gemma2 27B	0.7578	0.7626	0.7486	0.7556
	Aya-Expanse 32B	0.7303	0.6811	0.8661	0.7625
	Qwen 2.5 32B	0.7046	0.8159	0.5284	0.6414
70-72B	LLaMa-3.3 70B	0.5330	0.5173	0.9872	0.6789
	Qwen2 72B	0.7349	0.6899	0.8532	0.7629

Table B.8: Fine-tuning results of different LLMs on DDI prediction.

Model	Acc.	Prec.	Sens.	F1
GPT-4o	**0.926**	0.922	0.930	**0.926**
Deepseek Qwen1.5B	0.895	0.877	0.919	0.898
Phi-3.5 2.7B	0.913	0.878	0.960	0.917
Qwen2.5 3B	0.878	0.820	**0.969**	0.888
Gemma2 9B	0.832	**0.923**	0.725	0.812

Table B.9: Sensitivity comparison across external datasets.

	Sensitivity						
Dataset	Msdafl (Repl.)	L2 (Repl.)	Phi3.5 3b	Qwen2.5 3B	Gemma2 9B	GPT-4o +250B	Deepseek Qwen 1.5B
CredibleMeds	**1,000**	**1.000**	**1.000**	**1.000**	**0.400**	**1.000**	**1.000**
HEP	0,925	0.971	**1.000**	**1.000**	0.725	0.917	0.990
HIV	0,846	0.920	**1.000**	**1.000**	0.765	0.982	0.971
Corpus 2011	0,875	**0.938**	**0.938**	**0.938**	0.906	0.750	0.906
Corpus 2013	0,900	0.905	**1.000**	0.997	0.797	0.905	0.905
NLM Corpus	0,889	0.889	0.889	0.889	0.889	0.889	**1.000**
PK Corpus	**1,000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
OSCAR	0,861	0.866	**0.942**	0.936	0.696	0.905	0.872
WorldVista	0,903	0.907	**0.986**	0.984	0.851	0.841	0.970
French Ref.	0,917	0.928	0.972	**0.986**	0.808	0.981	0.879
KEGG	0,904	0.891	**0.990**	0.985	0.714	0.962	0.956
NDF-RT	0,975	0.975	**1.000**	0.966	0.773	**1.000**	0.966
Onc Non-Int.	0,879	0.835	**1.000**	**1.000**	0.904	0.993	0.969
AVG	0,913	0.925	**0.978**	0.975	0.787	0.933	0.953

Table B.10: Accuracy comparison across external datasets.

	Accuracy						
Dataset	Msdafl (Repl.)	L2 (Repl.)	Phi3.5 3b	Qwen2.5 3B	Gemma2 9B	GPT-4o +250B	Deepseek Qwen 1.5B
CredibleMeds	**1,000**	**1.000**	**1.000**	0.900	0.700	**1.000**	**1.000**
HEP	0,919	0.919	**0.924**	0.895	0.823	0.908	0.923
HIV	0,873	0.903	0.927	0.898	0.846	**0.948**	0.915
Corpus 2011	0,898	0.922	0.875	0.859	**0.938**	0.797	0.844
Corpus 2013	0,891	0.899	**0.919**	0.884	0.872	0.878	0.865
NLM Corpus	0,941	0.889	0.778	0.667	**0.944**	0.833	0.889
PK Corpus	**1.000**	**1.000**	0.500	0.750	0.750	0.750	0.750
OSCAR	0,872	0.882	0.902	0.873	0.821	**0.913**	0.874
WorldVista	0,895	0.905	**0.924**	0.899	0.892	0.877	0.919
French Ref.	0,916	0.910	0.910	0.893	0.868	**0.947**	0.869
KEGG	0,904	0.889	0.922	0.895	0.824	**0.937**	0.910
NDF-RT	0,905	0.971	0.958	0.908	0.878	**0.983**	0.950
Onc Non-Int.	0,889	0.852	0.919	0.897	0.907	**0.952**	0.909
AVG	0,916	**0.918**	0.881	0.863	0.851	0.902	0.894

Table B.11: F1 comparison across external datasets.

	F1-score						
Dataset	Msdafl (Repl.)	L2 (Repl.)	Phi3.5 3b	Qwen2.5 3B	Gemma2 9B	GPT-4o +250B	Deepseek Qwen 1.5B
CredibleMeds	**1.000**	**1.000**	**1.000**	0.909	0.571	**1.000**	**1.000**
HEP	0,923	0.923	**0.930**	0.905	0.804	0.908	0.928
HIV	0,874	0.904	0.932	0.907	0.833	**0.950**	0.919
Corpus 2011	0,903	0.923	0.882	0.870	**0.935**	0.787	0.853
Corpus 2013	0,894	0.899	**0.925**	0.896	0.861	0.882	0.870
NLM Corpus	**0,941**	0.889	0.800	0.727	**0.941**	0.842	0.900
PK Corpus	**1.000**	**1.000**	0.667	0.800	0.800	0.800	0.800
OSCAR	0,876	0.880	0.905	0.880	0.795	**0.913**	0.874
WorldVista	0,901	0.905	**0.928**	0.907	0.887	0.873	0.923
French Ref.	0,920	0.912	0.915	0.902	0.859	**0.949**	0.871
KEGG	0,906	0.889	0.927	0.903	0.803	**0.939**	0.914
NDF-RT	0,913	0.971	0.960	0.913	0.864	**0.983**	0.950
Onc Non-Int.	0,889	0.850	0.925	0.907	0.907	**0.954**	0.914
AVG	0,918	**0.919**	0.900	0.879	0.835	0.906	0.901

PUBLICATIONS

This dissertation includes design and findings of research that appeared in the following publications.

[1] Gabriele De Vito. "Assessing healthcare software built using IoT and LLM technologies." In: *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. EASE '24. Salerno, Italy: Association for Computing Machinery, 2024, pp. 476–481. ISBN: 9798400717017. DOI: 10.1145/3661167.3661202. URL: https://doi.org/10.1145/3661167.3661202.

[2] Gabriele De Vito, Damiano D'Amici, Fabiano Izzo, Filomena Ferrucci, and Dario Di Nucci. "LLM-Based Generation of Solidity Smart Contracts from System Requirements in Natural Language: The AstraKode Case." In: *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2025, pp. 170–180. DOI: 10.1109/SANER64311.2025.00024.

[3] Gabriele De Vito, Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, and Fabio Palomba. "LLM-Based Automation of COSMIC Functional Size Measurement From Use Cases." In: *IEEE Transactions on Software Engineering* 51.5 (2025), pp. 1500–1523. DOI: 10.1109/TSE.2025.3554562.

[4] Gabriele De Vito, Filomena Ferrucci, and Athanasios Angelakis. "HELIOT: LLM-Based CDSS for Adverse Drug Reaction Management." In: *Knowledge-Based Systems* (2025), p. 114184. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2025.114184. URL: https://www.sciencedirect.com/science/article/pii/S0950705125012250.

[5] Gabriele De Vito, Filomena Ferrucci, and Athanasios Angelakis. "LLMs for drug-drug interaction prediction: A comprehensive comparison." In: *Knowledge-Based Systems - Under Review (II round)* (2025).

[6] Gabriele De Vito, Stefano Lambiase, Fabio Palomba, and Filomena Ferrucci. "Meet C4SE: Your New Collaborator for Software Engineering Tasks." In: *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2023, pp. 235–238. DOI: 10.1109/SEAA60479.2023.00044.

[7] Gabriele De Vito, Fabio Palomba, and Filomena Ferrucci. "SecLLM: Enhancing Security Smell Detection in IaC with Large Language Models." In: *IEEE Access - Under Review* (2025).

[8] Gabriele De Vito, Fabio Palomba, and Filomena Ferrucci. "The role of Large Language Models in addressing IoT challenges: A systematic literature review." In: *Future Generation Computer Systems* 171 (2025), p. 107829. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2025.107829. URL: https://www.sciencedirect.com/science/article/pii/S0167739X25001244.

[9] Gabriele De Vito, Fabio Palomba, Carmine Gravino, Sergio Di Martino, and Filomena Ferrucci. "ECHO: An Approach to Enhance Use Case Quality Exploiting Large Language Models." In: *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2023, pp. 53–60. DOI: 10.1109/SEAA60479.2023.00017.

[10] Gabriele De Vito, Luigi Libero Lucio Starace, SD Martino, Filomena Ferrucci, and Fabio Palomba. "Large language models in software engineering: A focus on issue report classification and user acceptance test generation." In: *Proceedings of the Ital-IA Intelligenza Artificiale-Thematic Workshops co-located with the 4th CINI National Lab AIIS Conference on Artificial Intelligence (Ital-IA 2024), Naples, Italy*. 2024, pp. 48–53.

[11] Gabriele De Vito, Luigi Libero Lucio Storace, Fabio Palomba, and Sergio Di Martino. "Advancing LLM-Based Issue Report Classification with Explained Few-Shot Learning, Intent Extraction, Ensemble, and Summarization." In: *TOSEM - Under Review (II round)* (2025).

[12] Gabriele De Vito, Gabriele Vassallo, Fabio Palomba, and Filomena Ferrucci. "AGORA: An Approach for Generating Acceptance Test Cases from Use Cases." In: *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2024, pp. 126–133. DOI: 10.1109/SEAA64295.2024.00027.

BIBLIOGRAPHY

[1] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. "Phi-3 technical report: A highly capable language model locally on your phone." In: *arXiv preprint arXiv:2404.14219* (2024).

[2] Harold Abelson and Gerald Jay Sussman. *Structure and interpretation of computer programs*. The MIT Press, 1996.

[3] Silvia Abrahão, Lucia De Marco, Filomena Ferrucci, Jaime Gomez, Carmine Gravino, and Federica Sarro. "Definition and evaluation of a COSMIC measurement procedure for sizing Web applications in a model-driven development environment." In: *Inf. and Soft. Technology* 104 (2018), pp. 144–161.

[4] A. Abran and P. N. Robillard. "Function points analysis: an empirical study of its measurement processes." In: *IEEE Transactions on Software Engineering* 22.12 (1996), pp. 895–910.

[5] Khandakar Tanvir Ahmed, Md Istiaq Ansari, and Wei Zhang. "DTI-LM: language model powered drug–target interaction prediction." In: *Bioinformatics* 40.9 (2024), btae533.

[6] Ehsan Ahsani-Estahbanati, Vladimir Sergeevich Gordeev, and Leila Doshmangir. "Interventions to reduce the incidence of medical error and its financial burden in health care systems: A systematic review of systematic reviews." In: *Frontiers in medicine* 9 (2022), p. 875426.

[7] Ahmet Ata Akca and Ayca Tarhan. "Run-time measurement of COSMIC functional size for Java business applications: Is it worth the cost?" In: *IWSM-Mensura*. IEEE. 2013, pp. 54–59.

[8] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. "Optuna: A next-generation hyperparameter optimization framework." In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[9] Allan J Albrecht. "Measuring application development productivity." In: *Proc. joint share, guide, and ibm application development symposium*. 1979, pp. 83–92.

[10] Waleed Alhindi et al. "Issue-Labeler: an ALBERT-based Jira Plugin for Issue Classification." In: *2023 IEEE/ACM 10th Intern. Conf. on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE. 2023, pp. 40–43.

[11] Vahid Alizadeh, Mohamed Amine Ouali, Marouane Kessentini, and Meriem Chater. "RefBot: intelligent software refactoring bot." In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2019, pp. 823–834.

[12] Sai Chaithra Allala et al. "Towards Transforming User Requirements to Test Cases Using MDE and NLP." In: *IEEE COMPSAC*. Vol. 2. 2019, pp. 350–355. DOI: 10.1109/COMPSAC.2019.10231.

[13] Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. "L-Eval: Instituting Standardized Evaluation for Long Context Language Models." In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 14388–14411.

[14] Bente Anda and Dag IK Sjøberg. "Towards an inspection technique for use case models." In: *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. 2002, pp. 127–134.

[15] Pal Ankit. *OpenBioLLMs: Advancing OpenSource Large Language Models for Healthcare and Life Sciences. Hugging Face Repository*. accessed on 14 February 2025. 2025. URL: https://huggingface.co/aaditya/Llama3-OpenBioLLM-70B.

[16] Yadagiri Annepaka and Partha Pakray. "Large language models: a survey of their development, capabilities, and applications." In: *Knowledge and Information Systems* 67.3 (2025), pp. 2967–3022.

[17] Anthropic. *Claude 3.5 Sonnet* (*Version 3.5*). [Online; accessed 15 November 2024]. 2024. URL: https://claude.ai/.

[18] Giuliano Antoniol et al. "Is it a bug or an enhancement? A text-based approach to classify change requests." In: *Proc. of the 2008 Conf. of the Center for Advanced Studies on Collaborative Research*. 2008, pp. 304–318.

[19] Gabriel Aracena, Kyle Luster, Fabio Santos, Igor Steinmacher, and Marco Aurelio Gerosa. "Applying Large Language Models to Issue Classification." In: *Proceedings of the Third ACM/IEEE International Workshop on NL-Based Software Engineering*. NLBSE '24. Lisbon, Portugal: Association for Computing Machinery, 2024, pp. 57–60. ISBN: 9798400705762. DOI: 10.1145/3643787.3648043. URL: https://doi.org/10.1145/3643787.3648043.

[20] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "Automated checking of conformance to requirements templates using natural language processing." In: *IEEE transactions on Software Engineering* 41.10 (2015), pp. 944–968.

[21] Elham Asgari, Nina Montaña-Brown, Magda Dubois, Saleh Khalil, Jasmine Balloch, Joshua Au Yeung, and Dominic Pimenta. "A framework to assess clinical safety and hallucination rates of LLMs for medical text summarisation." In: *npj Digital Medicine* 8.1 (2025), p. 274.

[22] Serkan Ayvaz, John Horn, Oktie Hassanzadeh, Qian Zhu, Johann Stan, Nicholas P Tatonetti, Santiago Vilar, Mathias Brochhausen, Matthias Samwald, Majid Rastegar-Mojarad, et al. "Toward a complete dataset of drug–drug interaction information from publicly available sources." In: *Journal of biomedical informatics* 55 (2015), pp. 206–217.

[23] Mohammad Azzeh, Ali Bou Nassif, and Imtinan Basem Attili. "Predicting software effort from use case points: A

systematic review." In: *Science of Computer Programming* 204 (2021), p. 102596.

[24] Saadi Azzouz and Alain Abran. "A proposed measurement role in the rational unified process and its implementation with ISO 19761: COSMIC-FFP." In: *Software Measurement European Forum, Italy*. 2004.

[25] Selami Bagriyanik and Adem Karahoca. "Automated COSMIC Function Point measurement using a requirements engineering ontology." In: *Information and Software Technology* 72 (2016), pp. 189–203.

[26] Aleksandar Bajceta, Miguel Leon, Wasif Afzal, Pernilla Lindberg, and Markus Bohlin. "Using NLP Tools to Detect Ambiguities in System Requirements-A Comparison Study." In: *REFSQ Workshops*. 2022.

[27] Scott Barnett, Zac Brannelly, Stefanus Kurniawan, and Sheng Wong. "Fine-tuning or fine-failing? debunking performance myths in large language models." In: *arXiv preprint arXiv:2406.11201* (2024).

[28] Olga Baysal et al. "Situational awareness: personalizing issue tracking systems." In: *2013 35th Intern. Conf. on Software Engineering (ICSE)*. IEEE. 2013, pp. 1185–1188.

[29] Emily M. Bender, Timnit Gebru, et al. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *ACM Conference on Fairness, Accountability, and Transparency*. Virtual Event, Canada, 2021, pp. 610–623. ISBN: 9781450383097.

[30] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. "What makes a good bug report?" In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 2008, pp. 308–318.

[31] Alan W Biermann. "The inference of regular LISP programs from examples." In: *IEEE transactions on Systems, Man, and Cybernetics* 8.8 (1978), pp. 585–600.

[32] William Brace and Kalevi Ekman. "CORAMOD: a checklist-oriented model-based requirements analysis approach." In: *Requirements Engineering* 19 (2014), pp. 1–26.

[33] Nick C Bradley, Thomas Fritz, and Reid Holmes. "Context-aware conversational developer assistants." In: *Proceedings of the 40th International Conference on Software Engineering*. 2018, pp. 993–1003.

[34] Alasdair Breckenridge, Klaus Lindpaintner, Peter Lipton, Howard McLeod, Mark Rothstein, and Helen Wallace. "Pharmacogenetics: ethical problems and solutions." In: *Nature Reviews Genetics* 5.9 (2004), pp. 676–680.

[35] Chris Brown and Chris Parnin. "Sorry to Bother You: Designing Bots for Effective Recommendations." In: *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 2019, pp. 54–58. DOI: 10.1109/BotSE.2019.00021.

[36] Tom Brown et al. "Language Models are Few-Shot Learners." In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 1877–1901.

[37] Bernd Bruegge and Allen H Dutoit. "Object–oriented software engineering. using uml, patterns, and java." In: *Learning* 5.6 (2009), p. 442.

[38] Maria A Bruusgaard-Mouritsen, Shuaib Nasser, Lene H Garvey, Matthew S Krantz, and Cosby A Stone. "Anaphylaxis to excipients in current clinical practice: evaluation and management." In: *Immunology and Allergy Clinics* 42.2 (2022), pp. 239–267.

[39] COSMIC. *Automatic Line Switching v.1.1*. URL: https://cosmic-sizing.org/publications/automatic-line-switching-v1-1/.

[40] COSMIC. *Course Registration ('C-REG') System case study*. URL: https://cosmic-sizing.org/wp-content/uploads/2020/04/C-Reg-Case-Study.

[41] COSMIC. *Rise cooker v.2.0.1 case study*. URL: https://cosmic-sizing.org/wp-content/uploads/2018/08/Rice-Cooker-2.0.1-GoogleDocs.pdf.

[42] COSMIC. *The COSMIC Funcional Size Measurement Method, Version 5. Measurement Manual*. URL: https://cosmic-sizing.org/measurement-manual/.

[43] Elena Calvo-Cidoncha, Concepción Camacho-Hernando, Faust Feu, Xavier Pastor-Duran, Carles Codina-Jané, and Raimundo Lozano-Rubí. "OntoPharma: ontology based clinical decision support system to reduce medication prescribing errors." In: *BMC Medical Informatics and Decision Making* 22.1 (2022), p. 238.

[44] Xusheng Cao, Rui Fan, and Wanwen Zeng. "Deepdrug: a general graph-based deep learning framework for drug relation prediction." In: *biorxiv* (2020).

[45] Nathan Carlson and Phil Laplante. "The NASA automated requirements measurement tool: a reconstruction." In: *Innovations in Systems and Software Engineering* 10.2 (2014), pp. 77–91.

[46] Gustavo Carvalho et al. "NAT2TEST Tool: From Natural Language Requirements to Test Cases Based on CSP." In: *Software Engineering and Formal Methods*. Cham: Springer International Publishing, 2015, pp. 283–290. ISBN: 978-3-319-22969-0.

[47] Miguel Castro, Barbara Liskov, et al. "Practical byzantine fault tolerance." In: *OsDI*. Vol. 99. 1999. 1999, pp. 173–186.

[48] Denis Čeke and Boris Milašinović. "Automated web application functional size estimation based on a conceptual model." In: *23rd Int. Conf. on Software, Telecommunications and Computer Networks*. IEEE. 2015, pp. 234–241.

[49] Jhonny Cerezo, Juraj Kubelka, Romain Robbes, and Alexandre Bergel. "Building an Expert Recommender Chatbot." In: *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 2019, pp. 59–63. DOI: 10.1109/BotSE.2019.00022.

[50] N. Chamkha, A. Sellami, and A. Abran. "Automated COSMIC Measurement of Java Swing Applications throughout their Development Life Cycle." In: *IWSM-Mensura*. 2018, pp. 20–33.

[51] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. "Assessing the quality of the steps to reproduce in bug reports." In: *Proceedings of*

the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering. 2019, pp. 86–96.

[52] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. "Stay professional and efficient: automatically generate titles for your bug reports." In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 2020, pp. 385–397.

[53] Xin Chen, Xien Liu, and Ji Wu. "GCN-BMP: investigating graph representation learning for DDI prediction task." In: *Methods* 179 (2020), pp. 47–54.

[54] Y. Cheung, R. Willis, and B. Milne. "Software benchmarks using function point analysis." In: *Benchmarking: An International Journal* 6.3 (1999), pp. 269–276.

[55] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, et al. "Scaling Instruction-Finetuned Language Models." In: *arXiv:2210.11416* (2022).

[56] Cloudflare, Inc. *Cloudflare Workers AI Pricing*. Accessed: 2025-07-30. 2025. URL: https://developers.cloudflare.com/workers-ai/platform/pricing/.

[57] Zoe Co, A Jay Holmgren, David C Classen, Lisa Newmark, Diane L Seger, Melissa Danforth, and David W Bates. "The tradeoffs between safety and alert fatigue: Data from a national evaluation of hospital medication-related clinical decision support." In: *Journal of the American Medical Informatics Association* 27.8 (2020), pp. 1252–1258.

[58] Alistair Cockburn and Lord Cockburn. *Writing effective use cases*. Pearson Education India, 2008.

[59] Jacob Cohen. "A coefficient of agreement for nominal scales." In: *Educational and psychological measurement* 20.1 (1960), pp. 37–46.

[60] Mike Cohn. *Agile estimating and planning*. Pearson Education, 2005.

[61] Giuseppe Colavito, Filippo Lanubile, Nicole Novielli, and Luigi Quaranta. "Leveraging GPT-like LLMs to Automate Issue Labeling." In: (2024).

[62] Giuseppe Colavito et al. "Issue report classification using pre-trained language models." In: *Proc. 1st Int. Workshop on Nat. Lang.-based Softw. Eng.* 2022, pp. 29–32.

[63] Giuseppe Colavito et al. "Few-Shot Learning for Issue Report Classification." In: *Proc. of the 2023 IEEE/ACM 2nd Int. Workshop on NLBSE*. IEEE. 2023, pp. 16–19.

[64] Tiago K Colicchio and James J Cimino. "Beyond the override: Using evidence of previous drug tolerance to suppress drug allergy alerts; a retrospective study of opioid alerts." In: *Journal of Biomedical Informatics* 147 (2023), p. 104508.

[65] Yann Collet and Murray Kucherawy. *Zstandard Compression and the application/zstd Media Type*. Tech. rep. 2018.

[66] C. Commeyne, A. Abran, and R. Djouab. "Effort estimation with story points and cosmic function points-an industry case study." In: *Software Measurement News* 21.1 (2016), pp. 25–36.

[67] Jennifer Corny, Asok Rajkumar, Olivier Martin, Xavier Dode, Jean-Patrick Lajonchère, Olivier Billuart, Yvonnick Bézie, and Anne Buronfosse. "A machine learning–based clinical decision support system to identify prescriptions with a high risk of medication error." In: *Journal of the American Medical Informatics Association* 27.11 (2020), pp. 1688–1694.

[68] Karl Cox, Aybuke Aurum, and Ross Jeffery. "An experiment in inspecting the quality of use case descriptions." In: *Journal of Research and Practice in Information Technology* 36.4 (2004), pp. 211–229.

[69] Simon Curty, Felix Härer, and Hans-Georg Fill. "Design of blockchain-based applications using model-driven engineering and low-code/no-code platforms: a structured literature review." In: *Software and Systems Modeling* 22.6 (2023), pp. 1857–1895.

[70] John Dang, Shivalika Singh, Daniel D'souza, Arash Ahmadian, Alejandro Salamanca, Madeline Smith, Aidan Peppin, Sungjin Hong, Manoj Govindassamy, Terrence Zhao, et al. "Aya expanse: Combining research breakthroughs

for a new multilingual frontier." In: *arXiv preprint arXiv:2412.04261* (2024).

[71] Randy E David, Kelsy Gibson Ferrara, Joshua Schrecker, David Paculdo, Steven Johnson, Rhonda Bentley-Lewis, Rebecca Heltsley, and John W Peabody. "Impact of medication nonadherence and drug-drug interaction testing on the management of primary care patients with polypharmacy: a randomized controlled trial." In: *BMC medicine* 22.1 (2024), p. 540.

[72] Gabriele De Vito. "Assessing healthcare software built using IoT and LLM technologies." In: *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. 2024, pp. 476–481.

[73] Gabriele De Vito and Athanasios Angelakis. *LLM DDI GitHub Repository*. URL: https://github.com/gadevito/LLMDDI.

[74] Gabriele De Vito, Damiano D'Amici, Fabiano Izzo, Filomena Ferrucci, and Dario Di Nucci. "LLM-Based Generation of Solidity Smart Contracts from System Requirements in Natural Language: The AstraKode Case." In: *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2025, pp. 170–180. DOI: 10.1109/SANER64311.2025.00024.

[75] Gabriele De Vito, Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, and Fabio Palomba. "LLM-Based Automation of COSMIC Functional Size Measurement From Use Cases." In: *IEEE Transactions on Software Engineering* 51.5 (2025), pp. 1500–1523. DOI: 10.1109/TSE.2025.3554562.

[76] Gabriele De Vito and Filomena Ferrucci. "Approximate COSMIC Size: The Quick/Early Method." In: *EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, Italy, August 27-29, 2014*. IEEE, 2014, pp. 69–76.

[77] Gabriele De Vito, Filomena Ferrucci, and Athanasios Angelakis. "HELIOT: LLM-Based CDSS for Adverse Drug Reaction Management." In: *Knowledge-Based Systems* (2025), p. 114184. ISSN: 0950-7051. DOI: https://doi.org/10.

1016/j.knosys.2025.114184. URL: https://www.sciencedirect.com/science/article/pii/S0950705125012250.

[78] Gabriele De Vito, Filomena Ferrucci, and Athanasios Angelakis. "LLMs for Drug-Drug Interaction Prediction: A Comprehensive Comparison." In: *arXiv preprint arXiv:2502.06890* (2025).

[79] Gabriele De Vito, Filomena Ferrucci, and Athanasios Angelakis. "LLMs for drug-drug interaction prediction: A comprehensive comparison." In: *Knowledge-Based Systems - Under Review (II round)* (2025).

[80] Gabriele De Vito, Filomena Ferrucci, and Carmine Gravino. "Design and automation of a COSMIC measurement procedure based on UML models." In: *Software and Systems Modeling* 19 (2020), pp. 171–198.

[81] Gabriele De Vito, Stefano Lambiase, Fabio Palomba, and Filomena Ferrucci. "Meet C4SE: Your New Collaborator for Software Engineering Tasks." In: *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2023, pp. 235–238. DOI: 10.1109/SEAA60479.2023.00044.

[82] Gabriele De Vito, Fabio Palomba, and Filomena Ferrucci. "The role of Large Language Models in addressing IoT challenges: A systematic literature review." In: *Future Generation Computer Systems* 171 (2025), p. 107829. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2025.107829. URL: https://www.sciencedirect.com/science/article/pii/S0167739X25001244.

[83] Gabriele De Vito, Fabio Palomba, Carmine Gravino, Sergio Di Martino, and Filomena Ferrucci. "ECHO: An Approach to Enhance Use Case Quality Exploiting Large Language Models." In: *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2023, pp. 53–60. DOI: 10.1109/SEAA60479.2023.00017.

[84] Gabriele De Vito, Fabio Palomba, Carmine Gravino, Sergio Di Martino, and Filomena Ferrucci. "Echo: An approach to enhance use case quality exploiting large language models." In: *2023 49th Euromicro Conference on Software Engi-*

neering and Advanced Applications (SEAA). IEEE. 2023, pp. 53–60.

[85] Gabriele De Vito, Luigi Libero Lucio Starace, Sergio Di Martino, Filomena Ferrucci, and Fabio Palomba. *Replication Package for the paper titled "Beyond Few-Shot Learning: Advancing LLM-Based Issue Report Classification through Intent Extraction, Ensemble Learning, and Summarization"*. Available at the private link: https://figshare.com/s/ebef53fca65adcedfd24. Public DOI will be made available upon acceptance. 2025.

[86] Gabriele De Vito, Luigi Libero Lucio Storace, Fabio Palomba, and Sergio Di Martino. "Advancing LLM-Based Issue Report Classification with Explained Few-Shot Learning, Intent Extraction, Ensemble, and Summarization." In: *TOSEM - Under Review (II round)* (2025).

[87] Gabriele De Vito, Gabriele Vassallo, Fabio Palomba, and Filomena Ferrucci. "AGORA: An Approach for Generating Acceptance Test Cases from Use Cases." In: *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2024, pp. 126–133. DOI: 10.1109/SEAA64295.2024.00027.

[88] Mahin Delara, Lauren Murray, Behnaz Jafari, Anees Bahji, Zahra Goodarzi, Julia Kirkham, Mohammad Chowdhury, and Dallas P Seitz. "Prevalence and factors associated with polypharmacy: a systematic review and meta-analysis." In: *BMC geriatrics* 22.1 (2022), p. 601.

[89] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. "Qlora: Efficient finetuning of quantized llms." In: *Advances in neural information processing systems* 36 (2023), pp. 10088–10115.

[90] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Proc. naacL-HLT*. 2019, pp. 4171–4186.

[91] Bhuwan Dhingra, Jeremy R Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W Cohen. "Time-aware language models as temporal knowledge bases." In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 257–273.

[92] Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, and Federica Sarro. "Web effort estimation: function point analysis vs. COSMIC." In: *Information and Software Technology* 72 (2016), pp. 90–109.

[93] Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, and Federica Sarro. "Assessing the effectiveness of approximate functional sizing approaches for effort estimation." In: *Inf. and Soft. Technology* 123 (2020), p. 106308.

[94] H Diab, F Koukane, M Frappier, and R St-Denis. "μcROSE: automated measurement of COSMIC-FFP for Rational Rose RealTime." In: *Inf. and Soft. Technology* 47.3 (2005), pp. 151–166.

[95] Yutao Dou, Yuwei Huang, Xiongjun Zhao, Haitao Zou, Jiandong Shang, Ying Lu, Xiaolin Yang, Jian Xiao, and Shaoliang Peng. "ShennongMGS: An LLM-based chinese medication guidance system." In: *ACM Transactions on Management Information Systems* 16.2 (2025), pp. 1–14.

[96] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. "Improving factuality and reasoning in language models through multiagent debate." In: *Forty-first International Conference on Machine Learning*. 2023.

[97] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. "The llama 3 herd of models." In: *arXiv preprint arXiv:2407.21783* (2024).

[98] Miguel Ecar, Fabio Natanael Kepler, and João Pablo S da Silva. "AutoCosmic: COSMIC automated estimation and management tool." In: *Brazilian Symposium on Information Systems*. 2018, pp. 1–8.

[99] Sabit Ekin. "Prompt Engineering For ChatGPT: A Quick Guide To Techniques, Tips, And Best Practices." In: (2023). DOI: 10.36227/techrxiv.22683919.v2. URL: https://www.techrxiv.org/articles/preprint/Prompt_Engineering_For_ChatGPT_A_Quick_Guide_To_Techniques_Tips_And_Best_Practices/22683919.

[100] Inc. Element Labs. *LM Studio website*. [Online; accessed 15 November 2024]. URL: https://lmstudio.ai/.

[101] Rachel Ann Elliott, Elizabeth Camacho, Dina Jankovic, Mark J Sculpher, and Rita Faria. "Economic analysis of the prevalence and clinical and economic burden of medication error in England." In: *BMJ Quality & Safety* 30.2 (2021), pp. 96–105.

[102] Önder Eren, Baris Özkan, and Onur Demirörs. "PL FSM: An Approach and a Tool for the Application of FSM in SPL Environments." In: *Euromicro Conference on Software Engineering and Advanced Applications*. IEEE. 2015, pp. 119–126.

[103] Linda Erlenhov, Francisco Gomes de Oliveira Neto, and Philipp Leitner. "An Empirical Study of Bots in Software Development: Characteristics and Challenges from a Practitioner's Perspective." In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2020. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 445–455. ISBN: 9781450370431. DOI: 10.1145/3368089.3409680. URL: https://doi.org/10.1145/3368089.3409680.

[104] Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. "Out of the bleu: how should we assess quality of the code generation models?" In: *Journal of Systems and Software* 203 (2023), p. 111741.

[105] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. "TAPHSIR: towards AnaPHoric ambiguity detection and ReSolution in requirements." In: *Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering*. 2022, pp. 1677–1681.

[106] Angela Fan, Mike Lewis, and Yann Dauphin. "Hierarchical Neural Story Generation." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 889–898.

[107] Alessandro Fantechi, Stefania Gnesi, Giuseppe Lami, and Alessandro Maccari. "Applications of linguistic techniques for use case analysis." In: *Requirements Engineering* 8 (2003), pp. 161–170.

[108] Zhangyin Feng, Daya Guo, et al. "Codebert: A pre-trained model for programming and natural languages." In: *arXiv preprint arXiv:2002.08155* (2020).

[109] Reza Ferdousi, Reza Safdari, and Yadollah Omidi. "Computational prediction of drug-drug interactions based on drugs functional similarities." In: *Journal of biomedical informatics* 70 (2017), pp. 54–64.

[110] Alessio Ferrari, Gloria Gori, Benedetta Rosadini, Iacopo Trotta, Stefano Bacherini, Alessandro Fantechi, and Stefania Gnesi. "Detecting requirements defects with NLP patterns: an industrial experience in the railway domain." In: *Empirical Software Engineering* 23.6 (2018), pp. 3684–3733.

[111] Andrea Filippo Ferraris, Davide Audrito, Luigi Di Caro, and Cristina Poncibò. "The architecture of language: Understanding the mechanics behind LLMs." In: *Cambridge Forum on AI: Law and Governance*. Vol. 1. Cambridge University Press. 2025, e11.

[112] Jannik Fischbach et al. "SPECMATE: Automated Creation of Test Cases from Acceptance Criteria." In: *IEEE ICST*. 2020, pp. 321–331. DOI: 10.1109/ICST46399.2020.00040.

[113] Antoine Flahault, Michel Cadilhac, and Guy Thomas. "Sample size calculation should be performed for design accuracy in diagnostic test studies." In: *Journal of clinical epidemiology* 58.8 (2005), pp. 859–862.

[114] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. *Statistical methods for rates and proportions*. john wiley & sons, 2013.

[115] Xavier Franch, Cristina Palomares, Carme Quer, Panagiota Chatzipetrou, and Tony Gorschek. "The state-of-practice in requirements specification: an extended interview study at 12 companies." In: *Requirements Engineering* 28.3 (2023), pp. 377–409.

[116] Julian Frattini, Lloyd Montgomery, Jannik Fischbach, Daniel Mendez, Davide Fucci, and Michael Unterkalmsteiner. "Requirements quality research: a harmonized theory, evaluation, and roadmap." In: *Requirements engineering* 28.4 (2023), pp. 507–520.

[117] Julian Frattini, Lloyd Montgomery, Jannik Fischbach, Michael Unterkalmsteiner, Daniel Mendez, and Davide Fucci. "A live extensible ontology of quality factors for textual requirements." In: *2022 IEEE 30th international requirements engineering conference (RE)*. IEEE. 2022, pp. 274–280.

[118] C. Gencel and O. Demirors. "Functional size measurement revisited." In: *ACM Transactions on Software Engineering and Methodology* 17.3 (2008), pp. 1–36.

[119] Cigdem Gencel. "How to use cosmic functional size in effort estimation models?" In: *Int. Conf. on Software Process and Product Measurement*. Springer. 2008, pp. 196–207.

[120] Alex Genov. "Iterative usability testing as continuous feedback: A control systems perspective." In: *Journal of Usability Studies* 1.1 (2005), pp. 18–27.

[121] Mauro Giuffrè and Dennis L Shung. "Harnessing the power of synthetic data in healthcare: innovation, application, and privacy." In: *NPJ digital medicine* 6.1 (2023), p. 186.

[122] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[123] Alberto Goffi et al. "Automatic Generation of Oracles for Exceptional Behaviors." In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 2016, pp. 213–224. ISBN: 9781450343909. DOI: 10.1145/2931037.2931061.

[124] Rana Gonultas and Ayca Tarhan. "Run-time calculation of COSMIC functional size via automatic installment of measurement code into Java business applications." In: *Euromicro Conference on Software Engineering and Advanced Applications*. IEEE. 2015, pp. 112–118.

[125] Google. *Google Gemini 1.5 Pro*. [Online; accessed 15 November 2024]. 2024. URL: https://ai.google.dev/.

[126] Cordell Green. "Application of theorem proving to problem solving." In: *Readings in Artificial Intelligence*. Elsevier, 1981, pp. 202–222.

[127] Sumit Gulwani, Oleksandr Polozov, et al. "Program synthesis." In: *Foundations and Trends® in Programming Languages* 4.1-2 (2017), pp. 1–119.

[128] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning." In: *arXiv preprint arXiv:2501.12948* (2025).

[129] Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy S Liang. "A retrieve-and-edit framework for predicting structured outputs." In: *Advances in Neural Information Processing Systems* 31 (2018).

[130] Brian Henderson-Sellers. "Identifying internal and external characteristics of classes likely to be useful as structural complexity metrics." In: *OOIS'94: 1994 International Conference on Object Oriented Information Systems 19–21 December 1994, London*. Springer. 1995, pp. 227–230.

[131] Kim Herzig et al. "It's not a bug, it's a feature: how misclassification impacts bug prediction." In: *2013 35th intern. conf. on software engineering*. IEEE. 2013, pp. 392–401.

[132] Ari Holtzman, Jan Buys, Li Du, et al. "The curious case of neural text degeneration." In: *arXiv:1904.09751* (2019).

[133] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. "MetaGPT: Meta programming for a multi-agent collaborative framework." In: International Conference on Learning Representations, ICLR. 2024.

[134] Kasper Hornbæk. "Current practice in measuring usability: Challenges to usability studies and research." In: *International journal of human-computer studies* 64.2 (2006), pp. 79–102.

[135] Chao Hou, Guihua Duan, and Cheng Yan. "MSDAFL: molecular substructure-based dual attention feature learning framework for predicting drug–drug interactions." In: *Bioinformatics* 40.10 (Oct. 2024), btae596. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btae596. eprint: https://academic.oup.com/bioinformatics/article-pdf/40/10/btae596/59835317/btae596_supplementary_data.pdf. URL: https://doi.org/10.1093/bioinformatics/btae596.

[136] Tyken C Hsieh, Gilad J Kuperman, Tonushree Jaggi, Patricia Hojnowski-Diaz, Julie Fiskio, Deborah H Williams, David W Bates, and Tejal K Gandhi. "Characteristics and consequences of drug allergy alert overrides in a computerized physician order entry system." In: *Journal of the American Medical Informatics Association* 11.6 (2004), pp. 482–491.

[137] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. "Lora: Low-rank adaptation of large language models." In: *ICLR* 1.2 (2022), p. 3.

[138] H. Huijgens, A. Van Deursen, and R. Van Solingen. "The effects of perceived value and stakeholder satisfaction on software project impact." In: *Inf. and Soft. Technology* 89 (2017), pp. 19–36.

[139] I. Hussain, O. Ormandjieva, and L. Kosseim. "Mining and Clustering Textual Requirements to Measure Functional Size of Software with COSMIC." In: *Soft. Eng. Research and Practice*. 2009, pp. 599–605.

[140] Ishrar Hussain, Leila Kosseim, and Olga Ormandjieva. "Towards approximating COSMIC functional size from user requirements in agile development processes using text mining." In: *Int. Conf. on Applications of Natural Language to Information Systems*. Springer. 2010, pp. 80–91.

[141] Ishrar Hussain, Leila Kosseim, and Olga Ormandjieva. "Approximation of COSMIC functional size to support early effort estimation in Agile." In: *Data & Knowledge Engineering* 85 (2013), pp. 2–14.

[142] IFPUG. *International Function Point Users Group (IFPUG) Function Point Counting Practices Manual*. 2000.

[143] ISO/IEC. *ISO/IEC 14143-1: 2007. Information Technology-Software Measurement-Functional Size Measurement—Part 1: Definition of Concepts*. 2007.

[144] Maliheh Izadi et al. "Predicting the objective and priority of issue reports in software repositories." In: *Empirical Software Engineering* 27.2 (2022), p. 50.

[145] Fabiano Izzo and Damiano D'Amici. "Harnessing No-Code Blockchain for DeFi: A Microcredit Case Study on AstraKode Blockchain." In: *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. 2024, pp. 433–436.

[146] Ivar Jacobson. *Object-oriented software engineering: a use case driven approach*. Pearson Education India, 1993.

[147] Sajed Jalil, Suzzana Rafi, et al. "Chatgpt and software testing education: Promises & perils." In: *2023 IEEE ICSTW*. IEEE. 2023, pp. 4130–4137.

[148] K. R. Jayakumar and A. Abran. "Estimation models for software functional test effort." In: *Journal of Software Engineering and Applications* 10.4 (2017), pp. 338–353.

[149] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. "Perplexity—a measure of the difficulty of speech recognition tasks." In: *The Journal of the Acoustical Society of America* 62.S1 (1977), S63–S63.

[150] MS Jenner. "COSMIC-FFP and UML: Estimation of the Size of a System Specified in UML–Problems of Granularity." In: *Fourth European Conference Soft. Measurement and ICT Control*. 2001, pp. 173–184.

[151] Malcolm S Jenner. "5.1 Automation of Counting of Functional Size Using COSMIC FFP in UML." In: *COSMIC Function Points: Theory and Advanced Practices* 276 (2016).

[152] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. "Survey of hallucination in natural language generation." In: *ACM computing surveys* 55.12 (2023), pp. 1–38.

[153] Rafael Kallis, Giuseppe Colavito, Ali Al-Kaswan, Luca Pascarella, Oscar Chaparro, and Pooja Rani. "The NLBSE'24 Tool Competition." In: *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*. 2024, pp. 33–40.

[154] Rafael Kallis et al. "Ticket Tagger: Machine Learning Driven Issue Classification." In: *Proc. of the IEEE Int. Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 406–409.

[155] Rafael Kallis et al. "Predicting issue types on GitHub." In: *Science of Computer Programming* 205 (2021), p. 102598. ISSN: 0167-6423.

[156] Rafael Kallis et al. "The NLBSE'23 Tool Competition." In: *Proceedings of The 2nd Intern. Workshop on Natural Language-based Software Engineering (NLBSE'23)*. 2023.

[157] Rabimba Karanjai, Edward Li, Lei Xu, and Weidong Shi. "Who is smarter? an empirical study of ai-based smart contract creation." In: *2023 5th Conference on BRAINS*. IEEE. 2023, pp. 1–8.

[158] Syaeful Karim, Suryadiputra Liawatimena, et al. "Automating functional and structural software size measurement based on XML structure of UML sequence diagram." In: *IEEE Int. Conf. on Cybernetics and Computational Intelligence*. 2017, pp. 24–28.

[159] G Karner. "Metrics for Objectory [Diploma thesis]." In: *University of Linköping, Sweden* (1993).

[160] Andrej Kastrin, Polonca Ferk, and Brane Leskošek. "Predicting potential drug-drug interactions on topological and semantic similarity features using statistical learning." In: *PloS one* 13.5 (2018), e0196865.

[161] Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. "Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 284–294.

[162] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[163] Barbara A Kitchenham and Shari L Pfleeger. "Personal Opinion Surveys." In: *Guide to advanced empirical software engineering*. Springer, 2008, pp. 63–92.

[164] Wasaburo Koizumi, Minoru Kurihara, Koichi Hasegawa, Akimichi Chonan, Yasuhiko Kubo, Ryuichiro Maekawa, Ryozo Iwasaki, Tadashi Sasai, Yoshio Fukuyama, Kunitsugu Ishikawa, et al. "Sequence-dependence of cisplatin and 5-fluorouracil in advanced and recurrent gastric cancer." In: *Oncology reports* 12.3 (2004), pp. 557–561.

[165] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. "Large language models are zero-shot reasoners." In: *Advances in neural information processing systems* 35 (2022), pp. 22199–22213.

[166] Hadas Kotek, Rikker Dockum, and David Sun. "Gender bias and stereotypes in large language models." In: *Proceedings of the ACM collective intelligence conference*. 2023, pp. 12–24.

[167] Christos Kotronis, Mara Nikolaidou, Anargyros Tsadimas, Christos Michalakelis, and Dimosthenis Anagnostopoulos. "Extending SysML to Integrate Cost Analysis Into Model-Based Systems Engineering." In: *IEEE Transactions on Engineering Management* 71 (2022), pp. 2865–2880.

[168] Lisa Kühnel, Julian Schneider, Ines Perrar, Tim Adams, Sobhan Moazemi, Fabian Prasser, Ute Nöthlings, Holger Fröhlich, and Juliane Fluck. "Synthetic data generation for a longitudinal cohort study–evaluation, method extension and reproduction of published data analysis results." In: *Scientific Reports* 14.1 (2024), p. 14412.

[169] Pranjal Kumar. "Large language models (LLMs): survey, technical frameworks, and future challenges." In: *Artificial Intelligence Review* 57.10 (2024), p. 260.

[170] Janice L Kwan, Lisha Lo, Jacob Ferguson, Hanna Goldberg, Juan Pablo Diaz-Martinez, George Tomlinson, Jeremy M Grimshaw, and Kaveh G Shojania. "Computerised clinical decision support systems and absolute improvements in care: meta-analysis of controlled clinical trials." In: *Bmj* 370 (2020).

[171] JR Landis. "The Measurement of Observer Agreement for Categorical Data." In: *Biometrics* (1977).

[172] Luiz A. Laranjeira. "Software size estimation of object-oriented systems." In: *IEEE Transactions on software engineering* 16.5 (1990), pp. 510–522.

[173] Hyesung Lee, Yeon-Hee Baek, Ju Hwan Kim, Tzu-Chi Liao, Wallis CY Lau, Kenneth KC Man, Xiwen Qin, Stephen Wood, Jenni Ilomäki, J Simon Bell, et al. "Trends of polypharmacy among older people in Asia, Australia and the United Kingdom: a multinational population-based study." In: *Age and ageing* 52.2 (2023), afad014.

[174] Min Hun Lee, Daniel P Siewiorek, Asim Smailagic, Alexandre Bernardino, and Sergi Bermúdez i Badia. "Towards efficient annotations for a human-ai collaborative, clinical decision support system: A case study on physical stroke rehabilitation assessment." In: *Proceedings of the 27th International Conference on Intelligent User Interfaces*. 2022, pp. 4–14.

[175] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7871–7880.

[176] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. "Retrieval-

augmented generation for knowledge-intensive nlp tasks." In: *Advances in neural information processing systems* 33 (2020), pp. 9459–9474.

[177] Liwu Li. "Translating use cases to sequence diagrams." In: *Fifteenth IEEE Int. Conf. on Automated Software Engineering*. 2000, pp. 293–296.

[178] Ming Li, Jian Weng, et al. "CrowdBC: A blockchain-based decentralized framework for crowdsourcing." In: *IEEE transactions on parallel and distributed systems* 30.6 (2018), pp. 1251–1266.

[179] Wei Li and Sallie Henry. "Object-oriented metrics that predict maintainability." In: *Journal of systems and software* 23.2 (1993), pp. 111–122.

[180] Yujia Li, David Choi, et al. "Competition-level code generation with alphacode." In: *Science* 378.6624 (2022), pp. 1092–1097.

[181] Zhuoyan Li, Hangxiao Zhu, Zhuoran Lu, and Ming Yin. "Synthetic Data Generation with Large Language Models for Text Classification: Potential and Limitations." In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 10443–10461. DOI: 10.18653/v1/2023.emnlp-main.647. URL: https://aclanthology.org/2023.emnlp-main.647/.

[182] Zimeng Li, Shichao Zhu, Bin Shao, Xiangxiang Zeng, Tong Wang, and Tie-Yan Liu. "DSN-DDI: an accurate and generalized framework for drug–drug interaction prediction by dual-view representation learning." In: *Briefings in Bioinformatics* 24.1 (2023), bbac597.

[183] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. "Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate." In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 2024, pp. 17889–17904.

[184] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries." In: *ACL*. 2004.

[185] Chin-Yew Lin and FJ Och. "Looking for a few good metrics: ROUGE and its evaluation." In: *Ntcir workshop*. 2004.

[186] Kenneth Lind and Rogardt Heldal. "A model-based and automated approach to size estimation of embedded software components." In: *ACM/IEEE The 14th Int. Conf. on Model Driven Engineering Languages and Systems*. Springer. 2011, pp. 334–348.

[187] Kenneth Lind and Rogardt Heldal. "A Practical Approach to Size Estimation of Embedded Software Components." In: *IEEE Transactions on Software Engineering* 38.5 (2012), pp. 993–1007. DOI: 10.1109/TSE.2011.86.

[188] Jiachang Liu, Dinghan Shen, Yizhe Zhang, William B Dolan, Lawrence Carin, and Weizhu Chen. "What Makes Good In-Context Examples for GPT-3?" In: *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. 2022, pp. 100–114.

[189] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing." In: *ACM computing surveys* 55.9 (2023), pp. 1–35.

[190] Mark Lorenz and Jeff Kidd. *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc., 1994.

[191] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. "Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity." In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 8086–8098.

[192] Yao Lu et al. "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity." In: *arXiv* (2021).

[193] Marta Luri, Leire Leache, Gabriel Gastaminza, Antonio Idoate, and Ana Ortega. "A systematic review of drug allergy alert systems." In: *International journal of medical informatics* 159 (2022), p. 104673.

[194] Zohar Manna and Richard J Waldinger. "Toward automatic program synthesis." In: *Communications of the ACM* 14.3 (1971), pp. 151–165.

[195] Christopher D Manning. "Human language understanding & reasoning." In: *Daedalus* 151.2 (2022), pp. 127–138.

[196] Yulong Mao, Kaiyu Huang, Changhao Guan, Ganglin Bao, Fengran Mo, and Jinan Xu. "DoRA: Enhancing Parameter-Efficient Fine-Tuning with Dynamic Rank Distribution." In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 11662–11675.

[197] José L Medina-Franco, Karina Martinez-Mayorga, Eli Fernández-de Gortari, Johannes Kirchmair, and Jürgen Bajorath. "Rationality over fashion and hype in drug design." In: *F1000Research* 10 (2021).

[198] Suyu Mei and Kun Zhang. "A machine learning framework for predicting drug–drug interactions." In: *Scientific Reports* 11.1 (2021), p. 17619.

[199] Glaucia Melo, Edith Law, Paulo Alencar, and Donald Cowan. "Understanding User Understanding: What do Developers Expect from a Cognitive Assistant?" In: *2020 IEEE International Conference on Big Data (Big Data)*. 2020, pp. 3165–3172. DOI: 10.1109/BigData50022.2020.9378140.

[200] Pierre-Yves Meunier, Camille Raynaud, Emmanuelle Guimaraes, François Gueyffier, and Laurent Letrilliart. "Barriers and facilitators to the use of clinical decision support systems in primary care: a mixed-methods systematic review." In: *The Annals of Family Medicine* 21.1 (2023), pp. 57–69.

[201] Microsoft Corporation. *Azure OpenAI Service Pricing*. Accessed: 2025-07-30. 2025. URL: https://azure.microsoft.com/en-us/pricing/details/cognitive-services/openai-service/#pricing.

[202] Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, et al. "Granite code models: A family of open foundation models for code intelligence." In: *arXiv preprint arXiv:2405.04324* (2024).

[203] Sameh K Mohamed, Aayah Nounu, and Vít Nováček. "Biological applications of knowledge graph embedding models." In: *Briefings in bioinformatics* 22.2 (2021), pp. 1679–1693.

[204] Martin Monperrus. "Explainable Software Bot Contributions: Case Study of Automated Bug Fixes." In: *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 2019, pp. 12–15. DOI: 10.1109/BotSE.2019.00010.

[205] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system." In: (2008).

[206] Emanuele Antonio Napoli et al. "Leveraging Large Language Models for Automatic Smart Contract Generation." In: *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2024, pp. 701–710. DOI: 10.1109/COMPSAC61105.2024.00100.

[207] Roberto Navigli, Simone Conia, and Björn Ross. "Biases in large language models: origins, inventory, and discussion." In: *ACM Journal of Data and Information Quality* 15.2 (2023), pp. 1–21.

[208] C. Nebut et al. "Automatic test generation: a use case driven approach." In: *IEEE TSE* 32.3 (2006), pp. 140–155. DOI: 10.1109/TSE.2006.22.

[209] Zeyu Ni, Beijun Shen, Yuting Chen, Zhangyuan Meng, and Junming Cao. "CrowDevBot: A Task-Oriented Conversational Bot for Software Crowdsourcing Platform (S)." In: *SEKE*. 2019, pp. 410–522.

[210] Frank Nielsen and Frank Nielsen. "The MapReduce Paradigm." In: *Introduction to HPC with MPI for Data Science* (2016), pp. 147–160.

[211] Arnold K Nyamabo, Hui Yu, and Jian-Yu Shi. "SSI–DDI: substructure–substructure interactions for drug–drug interaction prediction." In: *Briefings in Bioinformatics* 22.6 (2021), bbab133.

[212] Mirosław Ochodek. "Approximation of COSMIC functional size of scenario-based requirements in Agile based on syntactic linguistic features—a replication study." In: *IWSM-Mensura*. IEEE. 2016, pp. 201–211.

[213] Mirosław Ochodek. "Functional size approximation based on use-case names." In: *Information and Software Technology* 80 (2016), pp. 73–88.

[214] Mirosław Ochodek, Sylwia Kopczyńska, and Miroslaw Staron. "Deep learning model for end-to-end approximation of COSMIC functional size based on use-case names." In: *Information and Software Technology* 123 (2020), p. 106310.

[215] Dušan Okanović, Samuel Beck, Lasse Merz, Christoph Zorn, Leonel Merino, André van Hoorn, and Fabian Beck. "Can a Chatbot Support Software Engineers with Load Testing? Approach and Experiences." In: *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. ICPE '20. Edmonton AB, Canada: Association for Computing Machinery, 2020, pp. 120–129. ISBN: 9781450369916. DOI: 10.1145/3358960.3375792. URL: https://doi.org/10.1145/3358960.3375792.

[216] Michael Oliver and Guan Wang. "Crafting efficient fine-tuning strategies for large language models." In: *arXiv preprint arXiv:2407.13906* (2024).

[217] David Oniani, Jordan Hilsman, Chengxi Zang, Junmei Wang, Lianjin Cai, Jan Zawala, and Yanshan Wang. "Emerging opportunities of using large language models for translation between drug molecules and indications." In: *Scientific Reports* 14.1 (2024), p. 10738.

[218] OpenAI. *GPT models fine-tuning*. [Online; accessed 15 November 2024]. URL: https://platform.openai.com/docs/guides/fine-tuning.

[219] OpenAI. "GPT-4 Technical Report." In: *arXiv:2303.08774* (2023).

[220] OpenAI. *Prompt Engineering Guide*. https://platform.openai.com/docs/guides/prompt-engineering. Accessed: 2023-10-17. 2023.

[221] A. Oriou, E. Bronca, B. Bouzid, et al. "Manage the automotive embedded software development cost & productivity with the automation of a Functional Size Measurement Method (COSMIC)." In: *IWSM-Mensura*. IEEE. 2014, pp. 1–4.

[222] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. "Training language models to follow instructions with human feedback." In: *Advances in neural information processing systems* 35 (2022), pp. 27730–27744.

[223] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. "An Empirical Study of the Non-determinism of ChatGPT in Code Generation." In: *ACM Transactions on Software Engineering and Methodology* (2023).

[224] B. Ozkan, O. Turetken, and O. Demirors. "Software functional size: For cost estimation and more." In: *European Conference, EuroSPI 2008*. Springer. 2008, pp. 59–69.

[225] Cristobal Pais, Jianfeng Liu, Robert Voigt, Vin Gupta, Elizabeth Wade, and Mohsen Bayati. "Large language models for preventing medication direction errors in online pharmacies." In: *Nature Medicine* (2024), pp. 1–9.

[226] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. "The TileDB array data storage manager." In: *Proceedings of the VLDB Endowment* 10.4 (2016), pp. 349–360.

[227] Kishore Papineni, Salim Roukos, et al. "Bleu: a method for automatic evaluation of machine translation." In: *Annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[228] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. "Generative agents: Interactive simulacra of human be-

havior." In: *Proceedings of the 36th annual acm symposium on user interface software and technology*. 2023, pp. 1–22.

[229] Robert E Park et al. *Software size measurement: A framework for counting source statements*. Citeseer, 1992.

[230] Eugenio Parra, Christos Dimou, Juan Llorens, Valentín Moreno, and Anabel Fraga. "A methodology for the classification of quality of requirements using machine learning techniques." In: *Information and Software Technology* 67 (2015), pp. 180–195.

[231] F. Pedregosa, G. Varoquaux, et al. "Scikit-learn: Machine learning in Python." In: *the Journal of machine Learning research* (2011).

[232] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. "Flexible modelling using conversational agents." In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2019, pp. 478–482.

[233] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. "Language Models as Knowledge Bases?" In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 2463–2473.

[234] Keith Thomas Phalp, Jonathan Vincent, and Karl Cox. "Improving the quality of use case descriptions: empirical assessment of writing guidelines." In: *Software Quality Journal* 15 (2007), pp. 383–399.

[235] Ilham Qasse, Shailesh Mishra, and Mohammad Hamdaqa. "iContractBot: a chatbot for smart contracts' specification and code generation." In: *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. IEEE. 2021, pp. 35–38.

[236] Yang Qiu, Yang Zhang, Yifan Deng, Shichao Liu, and Wen Zhang. "A comprehensive review of computational methods for drug-drug interaction detection." In: *IEEE/ACM*

transactions on computational biology and bioinformatics 19.4 (2021), pp. 1968–1985.

[237] Paola Leonor Quan, Sergio Sánchez-Fernández, Lucía Parrado Gil, Alfonso Calvo Alonso, José Miguel Bodero Sánchez, Ana Ortega Eslava, Marta Luri, and Gabriel Gastaminza Lasarte. "Usefulness of Drug Allergy Alert Systems: Present and Future." In: *Current Treatment Options in Allergy* 10.4 (2023), pp. 413–427.

[238] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[239] Qwen et al. *Qwen2.5 Technical Report*. 2025. arXiv: 2412.15115 [cs.CL]. URL: https://arxiv.org/abs/2412.15115.

[240] Alec Radford. "Improving language understanding by generative pre-training." In: (2018).

[241] Alejandro Rago, Claudia Marcos, and J Andrés Diaz-Pace. "Uncovering quality-attribute concerns in use case specifications via early aspect mining." In: *Requirements Engineering* 18 (2013), pp. 67–84.

[242] Aidin Rasti, Amal Ahmed Anda, et al. "Automated generation of smart contract code from legal contract specifications with Symboleo2sc." In: *Software and Systems Modeling* (2024), pp. 1–30.

[243] Gokul Rejithkumar, Preethu Rose Anish, and Smita Ghaisas. "Text-To-Text Generation for Issue Report Classification." In: *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*. 2024, pp. 53–56.

[244] Adel Remadi, Karim El Hage, Yasmina Hobeika, and Francesca Bugiotti. "To prompt or not to prompt: Navigating the use of large language models for integrating and modeling heterogeneous data." In: *Data & Knowledge Engineering* 152 (2024), p. 102313.

[245] Ranci Ren, John W Castro, Silvia T Acuña, and Juan de Lara. "Evaluation techniques for chatbot usability: A systematic mapping study." In: *International Journal of Software Engineering and Knowledge Engineering* 29.11n12 (2019), pp. 1673–1702.

[246] L. Reynolds and K. McDonell. "Prompt programming for large language models: Beyond the few-shot paradigm." In: *Conference on Human Factors in Computing Systems*. 2021, pp. 1–7.

[247] Alejandro Ríos-Hoyo, Naing Lin Shan, Anran Li, Alexander T Pearson, Lajos Pusztai, and Frederick M Howard. "Evaluation of large language models as a diagnostic aid for complex medical cases." In: *Frontiers in Medicine* 11 (2024), p. 1380148.

[248] D. Rodríguez, MA Sicilia, E García, and R. Harrison. "Empirical findings on team size and productivity in software development." In: *J. of Systems and Software* 85.3 (2012), pp. 562–570.

[249] Narjes Rohani, Changiz Eslahchi, and Ali Katanforoush. "ISCMF: Integrated similarity-constrained matrix factorization for drug–drug interaction prediction." In: *Network Modeling Analysis in Health Informatics and Bioinformatics* 9 (2020), pp. 1–8.

[250] Colette Rolland and Camille Ben Achour. "Guiding the construction of textual use case specifications." In: *Data & Knowledge Engineering* 25.1-2 (1998), pp. 125–160.

[251] Romy Roy, Shamsudheen Marakkar, Munawar P Vayalil, Alisha Shahanaz, Athira Panicker Anil, Shameer Kunnathpeedikayil, Ishaan Rawal, Kavya Shetty, Zahrah Shameer, Saraswathi Sathees, et al. "Drug-food interactions in the era of molecular big data, machine intelligence, and personalized health." In: *Recent Advances in Food Nutrition & Agriculture* 13.1 (2022), pp. 27–50.

[252] Ronen Rozenblum, Rosa Rodriguez-Monguio, Lynn A Volk, Katherine J Forsythe, Sara Myers, Maria McGurrin, Deborah H Williams, David W Bates, Gordon Schiff, and Enrique Seoane-Vazquez. "Using a machine learning system to identify and prevent medication prescribing errors: a clinical and cost analysis evaluation." In: *The Joint Commission Journal on Quality and Patient Safety* 46.1 (2020), pp. 3–10.

[253] Jennifer E Sager, Sasmita Tripathy, Lauren SL Price, Abhinav Nath, Justine Chang, Alyssa Stephenson-Famy, and Nina Isoherranen. "In vitro to in vivo extrapolation of the complex drug-drug interaction of bupropion and its metabolites with CYP2D6; simultaneous reversible inhibition and CYP2D6 downregulation." In: *Biochemical pharmacology* 123 (2017), pp. 85–96.

[254] June Sallou, Thomas Durieux, and Annibale Panichella. "Breaking the silence: the threats of using llms in software engineering." In: *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 2024, pp. 102–106.

[255] Urmimala Sarkar and Lipika Samal. *How effective are clinical decision support systems?* 2020.

[256] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. "Toolformer: Language models can teach themselves to use tools." In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 68539–68551.

[257] G Schneider and JP Winters. "Applied use cases." In: *A Practical Guide* (2001).

[258] Asma Sellami, Mariem Haoues, and Hanêne Ben-Abdallah. "Automated COSMIC-based analysis and consistency verification of UML activity and component diagrams." In: *8th Int. Conf., Revised Selected Papers 8*. Springer. 2013, pp. 48–63.

[259] Andrew Sellergren, Sahar Kazemzadeh, Tiam Jaroensri, Atilla Kiraly, Madeleine Traverse, Timo Kohlberger, Shawn Xu, Fayaz Jamil, Cían Hughes, Charles Lau, et al. "Medgemma technical report." In: *arXiv preprint arXiv:2507.05201* (2025).

[260] David E Shaw, William R Swartout, and C Cordell Green. "Inferring LISP Programs From Examples." In: *IJCAI*. Vol. 75. 1975, pp. 260–267.

[261] Guy Shtar, Lior Rokach, and Bracha Shapira. "Detecting drug-drug interactions using artificial neural networks and classic graph similarity measures." In: *PloS one* 14.8 (2019), e0219796.

[262] Bruno Cesar F. Silva, Gustavo Carvalho, and Augusto Sampaio. "Test Case Generation from Natural Language Requirements Using CPN Simulation." In: *Formal Methods: Foundations and Applications*. Ed. by Márcio Cornélio and Bill Roscoe. Cham: Springer International Publishing, 2016, pp. 178–193. ISBN: 978-3-319-29473-5.

[263] Avik Sinha, Stanley M Sutton, and Amit Paradkar. "Text2Test: Automated inspection of natural language use cases." In: *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE. 2010, pp. 155–164.

[264] David Canfield Smith. *Pygmalion: a creative programming environment*. Stanford University, 1975.

[265] Mozhan Soltani, Felienne Hermans, and Thomas Bäck. "The significance of bug report elements." In: *Empirical Software Engineering* 25.6 (2020), pp. 5255–5294.

[266] H. Soubra, A. Abran, and A. Ramdane-Cherif. "A refined functional size measurement procedure for real-time embedded software requirements expressed using the simulink model." In: *IWSM-Mensura*. IEEE. 2012, pp. 70–77.

[267] H. Soubra, A. Abran, S. Stern, and A. Ramdan-Cherif. "Design of a functional size measurement procedure for real-time embedded software requirements expressed using the simulink model." In: *IWSM-Mensura*. IEEE. 2011, pp. 76–85.

[268] H. Soubra, Y. Abufrikha, A. Abran, et al. "Towards Universal COSMIC Size Measurement Automation." In: *IWSM-Mensura*. 2020.

[269] Thomas Sounack, Joshua Davis, Brigitte Durieux, Antoine Chaffin, Tom J Pollard, Eric Lehman, Alistair EW Johnson, Matthew McDermott, Tristan Naumann, and Charlotta Lindvall. "BioClinical ModernBERT: A State-of-the-Art Long-Context Encoder for Biomedical and Clinical NLP." In: *arXiv preprint arXiv:2506.10896* (2025).

[270] Saurabh Srivastava and TV Prabhakar. "A Reference Architecture for Applications with Conversational Components." In: *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE. 2019, pp. 1–5.

[271] Ion Stoica, Matei Zaharia, Joseph Gonzalez, Ken Goldberg, Koushik Sen, Hao Zhang, Anastasios Angelopoulos, Shishir G Patil, Lingjiao Chen, Wei-Lin Chiang, et al. "Specifications: The missing link to making the development of LLM systems an engineering discipline." In: *CoRR* (2024).

[272] Cosby A Stone Jr, Yiwei Liu, Mary V Relling, Matthew S Krantz, Amanda L Pratt, Andrew Abreo, Jonathan A Hemler, and Elizabeth J Phillips. "Immediate hypersensitivity to polyethylene glycols and polysorbates: more common than we have recognized." In: *The Journal of Allergy and Clinical Immunology: In Practice* 7.5 (2019), pp. 1533–1540.

[273] Margaret-Anne Storey and Alexey Zagalsky. "Disrupting Developer Productivity One Bot at a Time." In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: Association for Computing Machinery, 2016, pp. 928–931. ISBN: 9781450342186. DOI: 10.1145/2950290.2983989. URL: https://doi.org/10.1145/2950290.2983989.

[274] Xiaorui Su, Bowei Zhao, Guodong Li, Jun Zhang, Pengwei Hu, Zhuhong You, and Lun Hu. "Knowledge Graph Neural Network With Spatial-Aware Capsule for Drug-Drug Interaction Prediction." In: *IEEE Journal of Biomedical and Health Informatics* 29.3 (2025), pp. 1771–1781. DOI: 10.1109/JBHI.2024.3419015.

[275] Phillip D Summers. "A methodology for LISP program construction from examples." In: *Journal of the ACM (JACM)* 24.1 (1977), pp. 161–175.

[276] Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. "A grammar-based structural cnn decoder for

code generation." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 7055–7062.

[277] Nigar M Shafiq Surameery and Mohammed Y Shakor. "Use chat gpt to solve programming bugs." In: *International Journal of Information Technology and Computer Engineering* 31 (2023), pp. 17–22.

[278] Reed T Sutton, David Pincock, Daniel C Baumgart, Daniel C Sadowski, Richard N Fedorak, and Karen I Kroeker. "An overview of clinical decision support systems: benefits, risks, and strategies for success." In: *NPJ digital medicine* 3.1 (2020), p. 17.

[279] Charles R Symons. *Software sizing and estimating: Mk II FPA (function point analysis)*. John Wiley & Sons, Inc., 1991.

[280] Thomas Yu Chow Tam, Sonish Sivarajkumar, Sumit Kapoor, Alisa V Stolyar, Katelyn Polanska, Karleigh R McCarthy, Hunter Osterhoudt, Xizhi Wu, Shyam Visweswaran, Sunyang Fu, et al. *A framework for human evaluation of large language models in healthcare derived from literature review. npj Digital Medicine, 7 (1)*. 2024.

[281] Antony Tang and Man F Lau. "Software architecture review by association." In: *Journal of systems and software* 88 (2014), pp. 87–101.

[282] Ayça Tarhan and Muhammet Ali SAĞ. "Cosmic solver: A tool for functional sizing of java business applications." In: *Balkan Journal of Electrical and Computer Engineering* 6.1 (2018), pp. 1–8.

[283] Falcon-LLM Team. *The Falcon 3 Family of Open Models*. 2024. URL: https://huggingface.co/blog/falcon3.

[284] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. "Gemma 3 technical report." In: *arXiv preprint arXiv:2503.19786* (2025).

[285] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. "Gemma 2: Improving open language models at a practical size." In: *arXiv preprint arXiv:2408.00118* (2024).

[286] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. "Large language models in medicine." In: *Nature medicine* 29.8 (2023), pp. 1930–1940.

[287] Saurabh Tiwari and Atul Gupta. "Use case specifications: How complete are they?" In: *Journal of Software: Evolution and Process* 32.1 (2020), e2218.

[288] Maxim Topaz, Diane L Seger, Kenneth Lai, Paige G Wickner, Foster Goss, Neil Dhopeshwarkar, Frank Chang, David W Bates, and Li Zhou. "High override rate for opioid drug-allergy interaction alerts: current trends and recommendations for future." In: *MEDINFO 2015: eHealth-enabled Health*. IOS Press, 2015, pp. 242–246.

[289] Maxim Topaz, Diane L Seger, Sarah P Slight, Foster Goss, Kenneth Lai, Paige G Wickner, Kimberly Blumenthal, Neil Dhopeshwarkar, Frank Chang, David W Bates, et al. "Rising drug allergy alert overrides in electronic health records: an observational retrospective study of a decade of experience." In: *Journal of the American Medical Informatics Association* 23.3 (2016), pp. 601–608.

[290] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[291] Satvik Tripathi, Rithvik Sukumaran, and Tessa S Cook. "Efficient healthcare with large language models: optimizing clinical workflow and enhancing patient care." In: *Journal of the American Medical Informatics Association* 31.6 (2024), pp. 1436–1440.

[292] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. "Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompt-

ing." In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 74952–74965.

[293] E. Ungan, C. Hammond, and A. Abran. "Automated COSMIC Measurement and Requirement Quality Improvement Through ScopeMaster® Tool." In: *IWSM-Mensura*. 2018, pp. 1–13.

[294] Erdir Ungan and Onur Demirörs. "A Functional Software Measurement Approach to Bridge the Gap Between Problem and Solution Domains." In: *Software Measurement*. Ed. by Andrzej Kobyliński, Beata Czarnacka-Chrobot, and Jaroslaw Świerczek. Springer, 2015, pp. 176–191. ISBN: 978-3-319-24285-9.

[295] Chirag Usdadiya, Saurabh Tiwari, and Asim Banerjee. "An empirical study on assessing the quality of use case metrics." In: *Proceedings of the 12th Innovations in Software Engineering Conference (formerly known as India Software Engineering Conference)*. 2019, pp. 1–11.

[296] Francisco Valente, Simão Paredes, Jorge Henriques, Teresa Rocha, Paulo de Carvalho, and João Morais. "Interpretability, personalization and reliability of a machine learning based clinical decision support system." In: *Data Mining and Knowledge Discovery* 36.3 (2022), pp. 1140–1173.

[297] Greet Van De Sijpe, Charlotte Quintens, Karolien Walgraeve, Eva Van Laer, Jens Penny, Greet De Vlieger, Rik Schrijvers, Paul De Munter, Veerle Foulon, Minne Casteels, et al. "Overall performance of a drug–drug interaction clinical decision support system: quantitative evaluation and end-user survey." In: *BMC Medical Informatics and Decision Making* 22.1 (2022), p. 48.

[298] Bethany A Van Dort, Wu Yi Zheng, Vivek Sundar, and Melissa T Baysari. "Optimizing clinical decision support alerts in electronic medical records: a systematic review of reported strategies adopted by hospitals." In: *Journal of the American Medical Informatics Association* 28.1 (2021), pp. 177–183.

[299] Gabriele Vassallo. *AGORA GitHub Repository*. URL: https://github.com/GabrieleVassallo/AutomaticUAT.

[300] A. Vaswani, N. Shazeer, N. Parmar, et al. "Attention is all you need." In: *Advances in neural information processing systems*. Vol. 30. 2017.

[301] Santiago Vilar, Rave Harpaz, Eugenio Uriarte, Lourdes Santana, Raul Rabadan, and Carol Friedman. "Drug—drug interaction through molecular structure similarity analysis." In: *Journal of the American Medical Informatics Association* 19.6 (Nov. 2012), pp. 1066–1074. ISSN: 1067-5027. DOI: 10.1136/amiajnl-2012-000935. eprint: https://academic.oup.com/jamia/article-pdf/19/6/1066/17374173/19-6-1066.pdf. URL: https://doi.org/10.1136/amiajnl-2012-000935.

[302] Chonghua Wang, Haodong Duan, Songyang Zhang, Dahua Lin, and Kai Chen. "Ada-LEval: Evaluating long-context LLMs with length-adaptable benchmarks." In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 2024, pp. 3712–3724.

[303] Chunhui Wang et al. "Automatic Generation of Acceptance Test Cases From Use Case Specifications: An NLP-Based Approach." In: *IEEE Transactions on Software Engineering* 48.2 (2022), pp. 585–616.

[304] Fangxu Wang, Futan Ma, Guoxing Song, and Jianqiang Zhang. "An Improved COSMIC Software Evaluation Method." In: *IEEE Conference on Telecommunications, Optics and Computer Science*. 2022, pp. 222–226.

[305] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. "A survey on large language model based autonomous agents." In: *Frontiers of Computer Science* 18.6 (2024), p. 186345.

[306] Yingheng Wang, Yaosen Min, Xin Chen, and Ji Wu. "Multi-view graph contrastive representation learning for drug-drug interaction prediction." In: *Proceedings of the web conference 2021*. 2021, pp. 2921–2933.

[307] Bolin Wei, Ge Li, et al. "Code generation as a dual task of code summarization." In: *Advances in neural information processing systems* 32 (2019).

[308] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. "Chain-of-thought prompting elicits reasoning in large language models." In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.

[309] Chris Wendler, Veniamin Veselovsky, Giovanni Monea, and Robert West. "Do llamas work in english? on the latent language of multilingual transformers." In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 15366–15394.

[310] Mairieli Wessel, Igor Steinmacher, Igor Wiese, and Marco A Gerosa. "Should i stale or should i close? an analysis of a bot that closes abandoned issues and pull requests." In: *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE. 2019, pp. 38–42.

[311] Mairieli Wessel, Igor Wiese, Igor Steinmacher, and Marco A. Gerosa. *Don't Disturb Me: Challenges of Interacting with Software Bots on Open Source Software Projects*. 2021. arXiv: 2103.13950 [cs.SE].

[312] Mairieli Wessel, Igor Wiese, Igor Steinmacher, and Marco Aurelio Gerosa. "Don't Disturb Me: Challenges of Interacting with Software Bots on Open Source Software Projects." In: *Proceedings of the ACM on Human-Computer Interaction* 5.CSCW2 (2021), pp. 1–21.

[313] Leonie Westerbeek, Kimberley J Ploegmakers, Gert-Jan De Bruijn, Annemiek J Linn, Julia CM van Weert, Joost G Daams, Nathalie van der Velde, Henk C van Weert, Ameen Abu-Hanna, and Stephanie Medlock. "Barriers and facilitators influencing medication-related CDSS acceptance according to clinicians: a systematic review." In: *International Journal of Medical Informatics* 152 (2021), p. 104506.

[314] Jules White, Quchen Fu, et al. "A prompt pattern catalog to enhance prompt engineering with chatgpt." In: *arXiv preprint arXiv:2302.11382* (2023).

[315] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design." In: *Generative ai for effective software development*. Springer, 2024, pp. 71–108.

[316] Sebastian G Wicha, Chunli Chen, Oskar Clewe, and Ulrika SH Simonsson. "A general pharmacodynamic interaction model identifies perpetrators and victims in drug interactions." In: *Nature communications* 8.1 (2017), p. 2129.

[317] Noam Wies, Yoav Levine, and Amnon Shashua. "The learnability of in-context learning." In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 36637–36651.

[318] David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, et al. "DrugBank 5.0: a major update to the DrugBank database for 2018." In: *Nucleic acids research* 46.D1 (2018), pp. D1074–D1082.

[319] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. *Experimentation in software engineering*. Vol. 236. Springer, 2012.

[320] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. "Autogen: Enabling next-gen LLM applications via multi-agent conversations." In: *First Conference on Language Modeling*. 2024.

[321] Marvin Wyrich., Regina Hebig., Stefan Wagner., and Riccardo Scandariato. "Perception and Acceptance of an Autonomous Refactoring Bot." In: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,* INSTICC. SciTePress, 2020, pp. 303–310. ISBN: 978-989-758-395-7. DOI: 10.5220/0009168803030310.

[322] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. "The rise and potential of large language model based agents: A survey." In: *Science China Information Sciences* 68.2 (2025), p. 121101.

[323] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. "An Explanation of In-context Learning as Implicit Bayesian Inference." In: *International Conference on Learning Representations*. 2021.

[324] Chengqi Xu, Olivier Elemento, Krishna C Bulusu, and Heng Pan. "DDI-GPT: Explainable Prediction of Drug-Drug Interactions using Large Language Models enhanced with Knowledge Graphs." In: *bioRxiv* (2024), pp. 2024–12.

[325] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826* (2018).

[326] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. "Qwen2. 5 technical report." In: *arXiv preprint arXiv:2412.15115* (2024).

[327] Boyuan Yang, Huijuan Liang, Jiahao Xu, Yanchi Liu, Sha Ma, Yuqiu Li, and Chengxiao Wang. "Multi-drug sequential release systems: Construction and application for synergistic tumor treatment." In: *International Journal of Pharmaceutics* 670 (2025), p. 125156.

[328] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. "React: Synergizing reasoning and acting in language models." In: *International Conference on Learning Representations (ICLR)*. 2023.

[329] Ming Ye, Jicheng Cao, and Shengyu Cheng. "Rebot: An automatic multi-modal requirements review bot." In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2022, pp. 777–781.

[330] Pengcheng Yin and Graham Neubig. "Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation." In: *arXiv preprint arXiv:1810.02720* (2018).

[331] Hui Yu, ShiYu Zhao, and JianYu Shi. "Stnn-ddi: a substructure-aware tensor neural network to predict drug–drug interactions." In: *Briefings in Bioinformatics* 23.4 (2022), bbac209.

[332] Tao Yue, Shaukat Ali, and Man Zhang. "RTCM: A Natural Language Based, Automated, and Practical Test Case Generation Framework." In: *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. Association for Computing Machinery, 2015, pp. 397–408. ISBN: 9781450336208. DOI: 10.1145/2771783.2771799.

[333] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. "Big bird: Transformers for longer sequences." In: *Advances in neural information processing systems* 33 (2020), pp. 17283–17297.

[334] Fiorella Zampetti, Ritu Kapur, Massimiliano Di Penta, and Sebastiano Panichella. "An empirical characterization of software bugs in open-source cyber–physical systems." In: *Journal of Systems and Software* 192 (2022), p. 111425.

[335] Thandar Zaw, Swe Zin Hlaing, Myint Myint Lwin, et al. "An Automated Software Size Measurement Tool based on Generation Model using COSMIC Function Size Measurement." In: *Int. Conf. on Advanced Information Technologies*. IEEE. 2019, pp. 268–273.

[336] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. "Bertscore: Evaluating text generation with bert." In: *arXiv preprint arXiv:1904.09675* (2019).

[337] Chengshuai Zhao, Shuai Liu, Feng Huang, Shichao Liu, and Wen Zhang. "CSGNN: Contrastive Self-Supervised Graph Neural Network for Molecular Interaction Prediction." In: *IJCAI*. 2021, pp. 3756–3763.

[338] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. "Explainability for large language models: A survey." In: *ACM Transactions on Intelligent Systems and Technology* 15.2 (2024), pp. 1–38.

[339] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J Letsholo, Muideen A Ajagbe, Erol-Valeriu Chioasca, and Riza T Batista-Navarro. "Natural language processing for requirements engineering: A systematic mapping study." In: *ACM Computing Surveys (CSUR)* 54.3 (2021), pp. 1–41.

[340] Wayne Xin Zhao et al. "A Survey of Large Language Models." In: *arXiv* (2023).

[341] Xiaoyong Zhao, Xingxin Leng, Lei Wang, and Ningning Wang. "Research on Fine-Tuning Optimization Strategies for Large Language Models in Tabular Data Processing." In: *Biomimetics* 9.11 (2024), p. 708.

[342] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. "Calibrate before use: Improving few-shot performance of language models." In: *International conference on machine learning*. PMLR. 2021, pp. 12697–12706.

[343] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. "Blockchain challenges and opportunities: A survey." In: *International journal of web and grid services* 14.4 (2018), pp. 352–375.

[344] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. "Lima: Less is more for alignment." In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 55006–55021.

[345] Yu Zhou et al. "Combining text mining and data mining for bug report classification." In: *Journal of Software: Evolution and Process* 28.3 (2016), pp. 150–176.

[346] Bohan Zhuang, Jing Liu, Zizheng Pan, Haoyu He, Yuetian Weng, and Chunhua Shen. "A survey on efficient training of transformers." In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 2023, pp. 6823–6831.

[347] Mistral AI team. *Mistral NeMo*. [Online; accessed 15 November 2024]. 2024. URL: https://mistral.ai/en/news/mistral-nemo.