Università degli Studi di Salerno

Dipartimento di Informatica

Dottorato di Ricerca in Informatica
Curriculum Computer Science and Information Technology
XXXVIII Ciclo

Tesi di Dottorato / Ph.D. Thesis

Beyond the Surface: Navigating Complex Systems via ABMs and Hypergraphs

Daniele De Vinco

Supervisor: **Prof. Vittorio Scarano**

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

A.A 2024/2025

Nessun *limite* eccetto il cielo.

— Miguel de Cervantes

Qua siamo in due e mi sembra che l'unico, fra noi due, che sta facendo uno sforzo per evitare che io ti meni sono sempre io, la stessa persona che, prima o poi, ti menerà.

— Mariano Giusti

Free Palestine

ACKNOWLEDGMENTS

A Ph.D. is an intensive journey, but not an impossible one. You have to remember every day what you are striving for and the people who made it possible.

I cannot begin to thank my family. I am not the easiest person to deal with, and yet they have always been the most supportive relatives.

A big thank you also goes to all my friends who have been by my side during these hard years. As above, thank you for your patience with me.

Thank you to all my colleagues who have made the long hours in the lab a bit less frustrating.

I did not want to mention any names in these acknowledgments, but I have to write a special thank you to Pauline. I am so glad to have you in my life. Thank you for everything.

ABSTRACT

Complex systems, including natural, social, and artificial structures such as traffic control, weather forecasting, policy-making, and epidemic dynamics, consist of interconnected components that exhibit unintelligible behavior when examined in isolation. To analyze these systems, researchers often collaborate across fields to develop models that replicate real-world behavior. The study of complex systems is marked by challenges such as data access and complexity, difficulties in modeling nonlinear and dynamic relationships, high computational demands, and the need for interdisciplinary cooperation. Researchers typically adopt either a bottom-up or a top-down approach, or a mix of both, to unravel human behavior within these systems. Agent-based models (ABMs) can serve as powerful tools for simulating interactions at the component level when adopting a bottom-up approach. Similarly, hypergraphs can accurately model systems to investigate higher-order interactions. Recent technological advancements have further enhanced this analytical capacity, notably with large language models (LLMs) that, after extensive training on large datasets, exhibit context-awareness and superior semantic understanding. These LLMs have proven particularly effective in tasks such as stance detection, showing adaptability across different datasets and outperforming traditional supervised methods with fewer resources. All these topics naturally give rise to a wide range of theoretical, methodological, and practical challenges. Working at the intersection of complex systems, online social dynamics, ABMs, hypergraph-based modeling, and LLM-driven analysis requires navigating heterogeneous data sources, integrating multiple formalisms, and balancing computational scalability with conceptual expressiveness. These challenges arise from both the intrinsic complexity of the systems under study and the limitations of existing analytical tools. This dissertation fits within this broader context and adopts an integrated perspective. Rather than pursuing two distinct research pathways, it brings together complementary approaches to advance the analysis of complex systems. On the one hand, the work contributes to improving

ABM frameworks, with particular attention to challenges such as scalability and modeling large, heterogeneous populations.

On the other hand, it investigates online social interactions through a combined methodological lens: group interactions are represented and analyzed using hypergraphs, while LLMs are employed to capture their semantic dimension. These components are not treated independently; instead, they try to form a unified framework that leverages structural and semantic information simultaneously.

The primary objective of this thesis is to investigate and analyze complex systems by leveraging ABMs, hypergraph-based network models, and LLMs. The contribution described in this thesis can be grouped into these three macro areas.

Agent-based modelling. In the context of ABMs, our goal is to enhance their functionality to advance the state of the art in ABM engines, particularly by addressing their computational limits, as the most widely used ABMs prioritize either performance or usability. Hence, my primary focus is to develop an approach that effectively integrates both features. The main research question this thesis aims to answer is:

> *How can advances in parallel and distributed computing technologies be harnessed to improve the scalability and efficiency of ABMs for simulating large-scale, real-time, and highly dynamic systems?*

This question led us to: (i) conduct an in-depth review of existing ABM tools to identify current trends, challenges, and opportunities for innovation; (ii) develop a stable version of krABMaga, an ABM simulation engine, and various simulation models; (iii) extend the features and capabilities of krABMaga to include distributed computation and GIS data integration. Additionally, we focused on improving the system's usability by creating a user-friendly GUI.

Hypergraphs. Another topic in my Ph.D. project is analyzing group interactions and the emergence of collective behavior in online social media through the lens of hypergraphs. In this regard, the main research questions my project aims to answer are:

> *Are specific structural patterns in hypergraphs dependent on the communities involved, or do they highlight specific*

user behaviors? Can we predict the formation of these patterns?

Motivated by this question we: (i) developed a community-driven open-source platform to store hypergraph datasets; (ii) examined interactions in online social media groups through the lenses of hypergraphs (contributing to H^1_{HG}); (iii) investigated the application of modern techniques, such as graph neural networks, to analyze high-order interactions (contributing to H^2_{HG}).

Large Language Models. Regarding LLMs, my project's goal is to assess the feasibility of using these tools to extract meaningful insights from text and enhance our understanding of how user interact and exchange their opinions in online social contexts. Specifically, the main research question my project aims to answer is:

Can LLMs improve our understanding of online social interactions and opinion dynamics?

The investigation on this topic led us to: (i) analyze the semantic information within the text linked to each node in a hypergraph using LLMs; (ii) develop a model to embed LLM-generated information into a stance detection task.

ITALIAN ABSTRACT

I sistemi complessi, che comprendono strutture naturali, sociali e artificiali come il controllo del traffico, le previsioni meteorologiche, l'elaborazione di politiche pubbliche e le dinamiche epidemiche, sono costituiti da componenti interconnesse che mostrano comportamenti di difficile interpretazione se esaminate in isolamento. Per analizzare questi sistemi, i ricercatori spesso collaborano tra diverse discipline per sviluppare modelli in grado di replicare il comportamento osservato nel mondo reale. Lo studio dei sistemi complessi è caratterizzato da sfide quali l'accesso ai dati e la loro complessità, le difficoltà nel modellare relazioni non lineari e dinamiche, gli elevati requisiti computazionali e la necessità di una cooperazione interdisciplinare. In genere, i ricercatori adottano un approccio bottom-up o top-down, oppure

una combinazione di entrambi, per comprendere il comportamento umano all'interno di questi sistemi. I modelli ad agenti (Agent-Based Models, ABM) possono costituire strumenti potenti per simulare le interazioni a livello di singolo componente in un'ottica bottom-up. Analogamente, gli ipergrafi permettono di modellare con precisione i sistemi per indagare le interazioni di ordine superiore. I recenti progressi tecnologici hanno ulteriormente potenziato questa capacità di analisi, in particolare grazie ai Large Language Models (LLM) che, dopo un esteso addestramento su grandi insiemi di dati, mostrano consapevolezza del contesto e una comprensione semantica superiore. Questi LLM si sono dimostrati particolarmente efficaci in compiti come la stance detection, mostrando adattabilità a diversi dataset e superando i metodi supervisionati tradizionali con un minore impiego di risorse. Tutti questi temi danno naturalmente origine a un'ampia gamma di sfide teoriche, metodologiche e pratiche. Lavorare all'intersezione tra sistemi complessi, dinamiche sociali online, ABM, modellazione basata su ipergrafi e analisi guidata da LLM implica affrontare fonti di dati eterogenee, integrare molteplici formalismi e bilanciare la scalabilità computazionale con l'espressività concettuale. Queste sfide derivano sia dall'intrinseca complessità dei sistemi oggetto di studio, sia dai limiti degli attuali strumenti di analisi. Questa tesi si colloca all'interno di questo quadro più ampio e adotta una prospettiva integrata. Invece di perseguire due direttrici di ricerca distinte, essa riunisce approcci complementari per avviare l'analisi dei sistemi complessi. Da un lato, il lavoro contribuisce al miglioramento dei framework ABM, con particolare attenzione a sfide quali la scalabilità e la modellazione di popolazioni ampie ed eterogenee.

Dall'altro lato, la tesi indaga le interazioni sociali online attraverso una lente metodologica combinata: le interazioni di gruppo sono rappresentate e analizzate tramite ipergrafi, mentre gli LLM vengono impiegati per catturarne la dimensione semantica. Questi componenti non sono trattati in modo indipendente; al contrario, mirano a formare un quadro unitario che sfrutti simultaneamente le informazioni strutturali e semantiche.

L'obiettivo principale di questa tesi è investigare e analizzare i sistemi complessi sfruttando ABM, modelli di rete basati su ipergrafi e LLM. I contributi descritti in questa tesi possono essere raggruppati in tre macro-aree.

Agent-based modelling. Nel contesto dei ABM, il nostro obiettivo è potenziarne le funzionalità per avanzare lo stato dell'arte dei motori ABM, in particolare affrontandone i limiti computazionali, dal momento che gli ABM più diffusi privilegiano o le prestazioni o l'usabilità. Di conseguenza, il mio obiettivo principale è sviluppare un approccio capace di integrare efficacemente entrambe le caratteristiche. La principale domanda di ricerca a cui questa tesi mira a rispondere è:

> *In che modo i progressi nelle tecnologie di calcolo parallelo e distribuito possono essere sfruttati per migliorare la scalabilità e l'efficienza degli ABM nella simulazione di sistemi su larga scala, in tempo reale e altamente dinamici?*

Questa domanda ci ha portato a: (i) condurre una revisione approfondita degli strumenti ABM esistenti per identificare i trend attuali, le sfide e le opportunità di innovazione; (ii) sviluppare una versione stabile di krABMaga, un motore di simulazione ABM, e diversi modelli di simulazione; (iii) estendere le funzionalità e le capacità di krABMaga per includere il calcolo distribuito e l'integrazione con dati GIS. Inoltre, ci siamo concentrati sul miglioramento dell'usabilità del sistema creando una GUI intuitiva per l'utente.

Hypergraphs. Un altro tema del mio progetto di Ph.D. è l'analisi delle interazioni di gruppo e dell'emergere di comportamenti collettivi nei social media online attraverso la lente degli ipergrafi. A questo proposito, le principali domande di ricerca a cui il mio progetto mira a rispondere sono:

> *Determinati pattern strutturali negli ipergrafi dipendono dalle comunità coinvolte oppure mettono in evidenza specifici comportamenti degli utenti? Possiamo prevedere la formazione di questi pattern?*

Motivati da questa domanda, abbiamo: (i) sviluppato una piattaforma open-source guidata dalla comunità per archiviare dataset di ipergrafi; (ii) esaminato le interazioni nei gruppi dei social media online attraverso la lente degli ipergrafi (contribuendo a H^1_{HG}); (iii) investigato l'applicazione di tecniche moderne, come le graph neural networks, per analizzare interazioni di ordine superiore (contribuendo a H^2_{HG}).

Large Language Models. Per quanto riguarda gli LLM, l'obiettivo del mio progetto è valutare la fattibilità di utilizzare questi stru-

menti per estrarre informazioni significative dal testo e migliorare la nostra comprensione di come gli utenti interagiscono e scambiano opinioni nei contesti sociali online. In particolare, la principale domanda di ricerca a cui il mio progetto mira a rispondere è:

> *Gli LLM possono migliorare la nostra comprensione delle interazioni sociali online e delle dinamiche delle opinioni?*

L'indagine su questo tema ci ha portato a: (i) analizzare le informazioni semantiche contenute nel testo associato a ciascun nodo in un ipergrafo utilizzando gli LLM; (ii) sviluppare un modello per incorporare le informazioni generate dagli LLM in un compito di stance detection.

CONTENTS

1 Introduction 1
  1.1 Background 6
  1.2 Contributions 10
    1.2.1 Goals 10
    1.2.2 Challenges 12
    1.2.3 Results 13
  1.3 Too Long; Dont Read (TL;DR) 16
2 A bottom-up approach to model complex systems 19
  2.1 Agent-based models and simulations analysis 21
    2.1.1 ABM tools description 24
    2.1.2 ABM tools comparison 30
    2.1.3 ABM tools evaluation 38
  2.2 A brief introduction to Rust 43
  2.3 A new tool for reliable agent-based modelling 44
    2.3.1 The krABMaga architecture 46
    2.3.2 Programming with krABMaga: the Wolf, Sheep, and Grass Model 61
    2.3.3 Performance evaluation 71
  2.4 Expanding on distributed ABM 78
  2.5 Scaling ABM visualization 84
  2.6 Conclusion 94
3 A top-down perspective on complex system analysis 97
  3.1 Analysis of the state-of-the-art 98
  3.2 Data collection and distribution 101
  3.3 Modelling social networks 115
    3.3.1 The shape of developer communities 116
    3.3.2 Deeper analysis on communities 129
  3.4 Exploiting LLMs to understand semantics 137
  3.5 Machine learning techniques to better understand network 140
  3.6 Conclusion 162
4 Conclusion 165
  4.1 Directions 166

I Appendix
A Appendix 171

A.1 Other papers 171
A.1.1 Hyperlink Prediction on Hypergraphs of Text 171
A.1.2 HyperBench: A Python library for Hyperlink prediction based on Pytorch and Pytorch Geometric 172
A.1.3 The Metaverse through the Eyes of University Students. 172
A.1.4 CrossWarp: A Framework for the Integration of Virtual Worlds and Augmented Reality 172

Bibliography 175

LIST OF FIGURES

Figure 1.1 Execution performance comparison: (a) absolute execution times and (b) relative speedup. 14

Figure 1.2 Similarity matrix of CPs of 22 subreddits on different topics. 15

Figure 2.1 Running times of each framework on the four ABMs, varying the computational load by increasing the number of agents while preserving the agent density. 43

Figure 2.2 The krABMaga architecture. 47

Figure 2.3 The krABMaga field taxonomy. 50

Figure 2.4 krABMaga scheduling flow. 55

Figure 2.5 Simulation results of the Wolf–Sheep model showing (a) population trends and (b) birth and death dynamics. 70

Figure 2.6 The GUI interface for the Wolf, Sheep & Grass simulation. 71

Figure 2.7 Running times of each framework on the four ABMs, varying the computational load by increasing the number of agents, while preserving the agent density. 73

Figure 2.8 Simulation steps per second required by each framework on the four ABMs, varying the computational load by increasing the number of agents while preserving the agent density. 74

Figure 2.9 SIR calibration results. 77

Figure 2.10 Partitioning of the field when there are 7 processors. Each block has a unique ID, the height and width of its perimeter, an origin point, and is assigned to a different processor (or machine). 81

Figure 2.11 An example of a halo region. In this example, the central square represents the main actor. The red-highlighted areas are the Halo regions of its neighboring agents, while the green areas indicate the portions of the field shared with those neighbors. Agents located within these green regions are locally stored by the processor managing them. At the end of each step, processors sharing borders exchange information about the agents in the red halo regions and, if needed, transfer agents between processors. 82
Figure 2.12 Speedup and execution times of the experiments. 84
Figure 2.13 Entities are designed to be independent and point to the relevant data. The data associated with an entity is stored in components, while systems are responsible for processing and updating them. 85
Figure 2.14 Performance of the WSG model along different implementations on 2,048,000 agents varying the number of threads. 92
Figure 2.15 Performance of the WSG model across different implementations on 2,048,000 agents varying the number of threads. 93
Figure 3.1 Graph *vs* hypergraph models. 98
Figure 3.2 Workflow of the dataset management pipeline. 108
Figure 3.3 PR template for adding or modifying a dataset. 109
Figure 3.4 CI/CD pipeline for validating PR integrity during dataset addition or modification, along with the assignment of at least one reviewer. 110
Figure 3.5 The HypergraphRepository's website. 112
Figure 3.6 Example of *filtering* and *group by* operations. 113
Figure 3.7 A snapshot of the web page reporting the details of a hypergraph. 114
Figure 3.8 Examples of node-related interactive charts. 115
Figure 3.9 Number of users and questions per trimester. 120

Figure 3.10 Comparison of the number of answers and durability of conversations in Stack Overflow and Reddit. 121
Figure 3.11 Proportion of the number of each language-related questions in Stack Overflow and Reddit. 123
Figure 3.12 Vertex degree distribution, first trimester. 124
Figure 3.13 Community evolution. 129
Figure 3.14 Example of a hypergraph with three hyperedges and 4 nodes (on the left). From this configuration, it is possible to extract an instance of h-motif (on the right), given by the intersections: (1) $he_1 \cap he_3 \setminus he_2$; (2) $he_1 \cap he_2 \setminus he_3$; (3) $he_2 \cap he_3 \setminus he_1$; (4) $he_3 \cap he_2 \setminus he_1$. 131
Figure 3.15 CPs of the subreddits analyzed in this work. 133
Figure 3.16 Clustering-based similarity. 134
Figure 3.17 Cosine-based similarity. 135
Figure 3.18 a) Three subreddits that have similar CPs using clustering. b) Three subreddits that have similar CPs using cosine similarity. c) Three subreddits with similar domains. 136
Figure 3.19 Overview of the pipeline. 140
Figure 3.20 A high-level description of the h-motif prediction problem. 144
Figure 3.21 Overview of the positive and negative sampling strategy along with the training-test data splitting procedure. 147
Figure 3.22 The overall architecture of the proposed HM learning framework. The input hypergraph $\mathcal{H}$ and the initial node embeddings are refined and combined using hypergraph convolution to generate hyperedge embeddings. These representations are further refined using graph convolution over the line graph derived from $\mathcal{H}$. Finally, the representations are aggregated to construct the h-motif embeddings. 150

Figure 3.23 Datasets statistics—Number of nodes $|V|$, hyperedges $|E|$, clique expansion edges $|\mathcal{E}|$, and h-motifs (y-log scale). 153
Figure 3.24 Negative sampling algorithm. 158
Figure 3.25 t-SNE visualization of h-motif embeddings learned by *HM VL* for type **t=2**, comparing different negative sampling strategies. Positive h-motifs are shown in violet and negative h-motifs in yellow. 160
Figure 3.26 Confusion matrices comparing the classification performance of **JC** and **HMVL** methods for h-motif type **t=2** on the Email-Enron dataset. 161
Figure 3.27 AUROC results of the **N2V** and **VL** against our methods **HMN2V** and **HMVL** to predict h-motifs instances on the *Email-Enron* hypergraph by varying the number of training motif instances for a specific type **t=2** averaged results over **10** experiments. 162

LIST OF TABLES

Table 1.1 List of published works. ▲ bottom-up works, ▼ top-down works 18
Table 2.1 Description of the desirable features for ABM platforms. 31
Table 2.2 Comparison of ABM tools based on their features. 32
Table 2.3 Comparison of ABM tools based on testing, CI/CD, and HPC capabilities. 33
Table 2.4 Trade-off between the declared ease of use and efficiency of the ABM platforms. 35
Table 2.5 Comparison of ABM framework/software features. 36

Table 2.6 Evaluation of ABM tools based on the perceived ease of installation and usage. N/V stands for "Not Verifiable" due to the inability to install or use the corresponding tool. 39
Table 2.7 Summary of whether each model was available for a given platform (●) or developed from scratch (○. Only the ABM tools that could be installed and used are considered. 41
Table 2.8 Experiment configurations for evaluating ABM tools' performance. 42
Table 2.9 Agent count and corresponding field size configurations. 72
Table 2.10 Evaluation of krABMaga's scalability on an Amazon EC2 virtual cluster machine. 78
Table 2.11 Size of the examples. 83
Table 2.12 Numerical results of the experiments. 84
Table 2.13 Sequential krABMaga execution time vs the ECS implementation runned with 1 thread (time in seconds). 90
Table 2.14 Relative speedup of WSG model on 2048000 agents (time in seconds). 91
Table 2.15 Relative speedup of Boids model on 2048000 agents (time in seconds). 91
Table 3.1 Currently available statistics. 111
Table 3.2 Community and user distributions. 125
Table 3.3 Percentage of users per programming language within the top 5% biggest communities. 126
Table 3.4 Distribution of the size of Stack Overflow and Reddit communities over each trimester. 127
Table 3.5 Percentage of users per programming language within the top 5% biggest communities per trimester. 128
Table 3.6 Performance comparison of prediction methods across all **26** h-motif types. 155

Table 3.7 Performance comparison of h-motif prediction across different motif types using ACC and F1 scores on the Email-Enron dataset. 157
Table 3.8 AUROC scores when varying the negative sampling node-substitution mechanisms on the Email-Enron dataset. 159

LISTINGS

2.1 Wolf agent properties. 62
2.2 Sheep agent properties. 62
2.3 Agents' moving behavior. 62
2.4 Agents' reproducing behavior. 63
2.5 Eating behavior of sheep agents. 63
2.6 Eating behavior of wolf agents. 64
2.7 WSG Simulation State. 66
2.8 main.rs file. 67
2.9 Setting the krABMaga TUI interface. 69

ACRONYMS

Ph.D. Philosophiæ Doctor
ABM Agent based model
CP Characteristic Profile
LLM Large Language Models
QA Question & Answer
HIF Hypergraph Interchange Format
GUI Graphical User Interface
TUI Terminal User Interface
GIS Geographic Information Systems

OOP Object-Oriented Programming

OSN Online Social Networks

ECS Entity-Component-System

AS Applied Science

CHI Computer Human Interaction

JASSS Journal of Artificial Societies and Social Simulation

WAW Workshop on Modelling and Mining Networks

WebSCI Web Science

ASONAM Advances in Social Network Analysis and Mining

ECML-PKDD European Conference on Machine Learning and Data Mining

KDD Knowledge Discovery and Data Mining

WWW International World Wide Web Conference

AIVR International Conference on Artificial Intelligence and Virtual Reality

BIGHPC Special Track on Big Data and High-Performance Computing

ITADATA Italian Conference on Big Data and Data Science

1

INTRODUCTION

Many closely interconnected components constitute various natural, social, and artificial structures, giving rise to unintelligible behaviors if we focus on a single component at a time [179]. Such organizations are commonly referred to as complex systems, and as examples, we can name traffic control, weather forecast, policy-making or epidemic dynamics. Usually, the analysis of complex systems involves creating a model that can replicate the real system. These models are typically the result of the collaborative efforts of multiple experts from various fields. Complex systems are characterized by key features such as non-linearity (when small changes in one part of the system can lead to disproportionately large and unpredictable effects in other parts), self-organization, and adaptation [58]. Investigating these phenomena means dealing with the following challenges:

- *Data complexity and access*: Complex systems often require and generate vast amounts of data, making analysis and interpretation challenging. Further, online platforms frequently deny or hide their data behind paywalls, making data acquisition difficult.
- *Modeling complexity*: Developing accurate mathematical or computational models for complex systems can be difficult due to their non-linear and dynamic nature.
- *Computational resources*: Simulating or analyzing complex systems may require significant computational resources regarding time, money, and equipment.
- *Interdisciplinary nature*: Understanding complex systems often requires knowledge and cooperation from multiple domain experts.

Researchers can use top-down, bottom-up, or a combination of both strategies to understand the intricacies of human behavior. In this thesis, we identified these categories as a conceptual guideline for structuring the investigation of complex

social phenomena. These categories were developed following a thorough review of the existing literature and methodologies in computational social science and complex systems analysis. Top-down strategies allow for the examination of global patterns and emergent properties, focusing on system-level behaviors before analyzing individual components. Bottom-up strategies emphasize analyzing individual agents and their interactions to build an understanding of overall system behavior from the ground up. Hybrid strategies recognize that both perspectives are often needed to fully grasp the multifaceted nature of human behavior and social systems. These acknowledge the interplay between individual-level actions and system-wide dynamics.

By establishing these categories, I provided a systematic framework for comparing methods, interpreting results, and guiding model construction throughout the research process. This reflects the diverse methodological options available and serves as a foundation for integrating different perspectives in this work's study of complex systems.

The bottom-up approach involves examining individual components within the complex system and understanding its behavior starting from its basic elements. Agent-based models (Agent based models (ABMs)) are a robust modeling technique that uses this approach [92]. Specifically, in ABMs, modelers define agents and environments to replicate specific aspects or properties of the underlying reality. These models provide a detailed understanding of how local interactions give rise to system-level behavior, offering fine-grained control and manipulation of individual components. This approach is particularly useful when the behavior of individual elements within the system is well understood.

The top-down approach involves examining the system's global behavior and properties, then deconstructing and understanding its underlying components and interactions. One of the most expressive mathematical structures capable of capturing high-order interactions is a hypergraph, a generalization of a graph where a (hyper)edge connects an arbitrary number of nodes [61]. Hypergraphs are useful for modeling complex systems because they can represent relationships involving multiple entities simultaneously, unlike graphs, which model only pairwise interactions. Hypergraphs are particularly valuable for analyzing social networks, as interactions often occur in groups of varying sizes. For instance, users might collaborate in teams, participate in

group activities, form communities, or work together towards a common goal. All these activities involve more complex connections than simple pairwise relationships. By using hypergraphs, we can more accurately capture and analyze the dynamics of these high-order interactions, leading to better insights into the system's behavior and structure [322].

For example, social interactions on online platforms frequently involve written communications. Textual messages can carry multiple meanings, and the context in which they are located can significantly affect their interpretation and nuance. As a result, understanding the semantics of these interactions is crucial and cannot be ignored when examining online user behavior. As technology advances, recent research has shown that the rapid development of Large Language Models (LLM) has enabled researchers to perform various tasks across multiple fields [307]. After being pre-trained on extensive text corpora, LLMs demonstrated extensive context-aware knowledge and exceptional semantic comprehension [80]. For instance, LLMs have demonstrated their effectiveness in stance detection tasks, showing adaptability across datasets with different prompting schemes and even outperforming supervised models while using fewer resources [93].

My Ph.D. thesis fits within this broader context and adopts an integrated perspective. Rather than pursuing two distinct research pathways, it brings together complementary approaches to advance the analysis of complex systems. On the one hand, the work contributes to improving ABM frameworks, with particular attention to challenges such as scalability and modeling large, heterogeneous populations. On the other hand, it investigates online social interactions through a combined methodological lens: group interactions are represented and analyzed using hypergraphs, while LLMs are employed to capture their semantic dimension. These components are not treated independently; instead, they try to form a unified framework that leverages structural and semantic information simultaneously. This approach aims to deepen our understanding of real-world social dynamics and supports the research questions introduced in Section 1.2.1. Ultimately, this project seeks to produce insights that reflect—and contribute to—an increasingly interdisciplinary research landscape.

Organization. This section provides an overview of the dissertation's structure and guidance on navigating the various chapters and their internal structure.

General structure. Each chapter is based on one of the peer-reviewed or accepted papers produced during my Ph.D. research. Despite addressing different aspects of the overall problem, all chapters follow a consistent organizational schema designed to ensure clarity and comparability across contributions:

- A presentation of the motivation behind the work.
- An introduction to the relevant state of the art.
- A description of the core contribution.
- A summary of the main results obtained.
- A concluding synthesis highlighting the contribution of the chapter to the broader Ph.D. project.

To provide additional context or insight where needed, supplementary elements are included throughout the manuscript. These may take the form of short graffiti notes , or highlighted commentary boxes,

An illustrative example of a graffito.

Tip

An example of commentary box.

which serve to highlight important nuances, methodological considerations, or broader reflections.

Detailed description of the chapters. Chapter 1 presents an extended and updated version of the work initially submitted to a Ph.D. forum track under the title *Beyond the Surface: Navigating Complex Systems via ABMs and Hypergraphs*. This chapter outlines the dissertation's course, detailing the research questions that guided its development and framing the methodological and conceptual choices made along the way.

Chapter 2 explores the bottom-up modeling paradigm, with a particular focus on the design, implementation, and evaluation of tools for ABM. Through three complementary works, *Experimenting with Agent-Based Model Simulation Tools*, *Reliable and Efficient Agent-Based Modeling and Simulation*, and *High-performance Computation on a Rust-based Distributed ABM Engine*, this chapter

examines the practical, methodological, and computational challenges of simulating large-scale complex systems. It discusses the limitations of existing ABMs platforms, introduces innovations that improve reliability and reproducibility, and presents a high-performance Rust-based distributed engine designed to scale ABM simulations beyond traditional boundaries. By integrating these contributions, the chapter provides both conceptual insight and technical advancements for researchers seeking to leverage ABMs for complex, data-driven investigations.

Chapter 3 presents a set of top-down analytical studies that investigate large-scale patterns of behavior, communication, and knowledge exchange in online communities. The first section focuses on the construction of usable data resources. At its core is the project *Hypergraph Repository: A Community-driven and Interactive Hypernetwork Data Collection*, which introduces an open, collaborative platform for gathering, documenting, and standardizing hypergraph and hypernetwork datasets. This chapter describes the motivation for building such a repository, the design principles guiding its architecture, and the role of community contributions in ensuring its growth and long-term sustainability. It also details the technical infrastructure, data formats, and interactive tools that facilitate exploration, visualization, and reuse, thereby positioning the repository as a valuable resource for researchers across computational social science, network science, and complex systems modeling. After, *The Age of Snippet Programming: Toward Understanding Developer Communities in Stack Overflow and Reddit* examines how code snippets circulate across platforms and what these patterns reveal about collective problem-solving and knowledge transfer in developer ecosystems. The second study, *Deciphering Conversational Networks: Stance Detection via Hypergraphs and LLMs*, introduces a novel framework that blends hypergraph representations with large language models to identify and classify stances within complex conversational threads, highlighting how structural and semantic signals converge in online discourse. The final work, *Fifty Shapes of Reddit: Do Prolife Activists Have the Same Interaction Patterns of Gun Fanatics?*, applies network-level analysis to politically and ideologically charged communities, comparing their interaction structures to uncover similarities and divergences in how polarized groups organize and engage. Together, these studies demonstrate how top-down computational methods can

extract high-level behavioral signatures from online social networks.

Finally, Chapter 4 synthesizes the insights gained throughout the dissertation, discusses the broader implications of the work, and outlines potential avenues for future research.

1.1 BACKGROUND

ABM. Among the techniques used to analyze real-world systems, ABMs are one of the most effective [35]. From modeling an epidemic spread to an economic fluctuation of the market, ABMs have been used to develop an accurate representation of reality [79, 94]. The main element of an ABM is an *agent*, a basic autonomous entity that the modeler can define with different characteristics and behaviors. Agents can interact with each other and the environment according to a set of rules defined by the modeler. These interactions result in a model of the world being studied, which the modeler can analyze to learn relevant lessons from its emerging behaviors. Structures, patterns, and behaviors emerge through interactions rather than being purposefully coded within the model. ABMs also help researchers determine how a system's micro-level properties, constraints, and laws affect its macroscopic behavior. An agent-based simulation is composed of a three-component structure:

- *Agents.* Each agent is an individual that forms the population under consideration. Agents may act independently or in relation to one another, executing complex functions through simple interactions.
- *Relationships.* The relationships define how agents interact with each other.
- *Rules.* Rules define the agents' behavior, determining the outcomes of their interactions.

Using ABM simulations makes analyzing complex models easier, decomposing the problem into smaller components. However, their implementation can be complex for domain experts with little experience in software development [20]. The solution to this type of problem lies in multiple frameworks that make it easy to develop and run simulations, eliminating the need to

know all the necessary technical details. Specifically, simulations share common implementation aspects, and these frameworks effectively circumvent the need to reinvent solutions for well-known challenges. Good examples of successful ABM engines are Mason, a fast, modular, discrete event multi-agent simulation toolkit written in Java [196], and Netlogo, designed to be used by researchers and educators through a simple interface and language [286].

ABM simulations can be time-consuming because of their exceptionally high computational load. In addition, their execution time can significantly affect the final result, since some simulations achieve greater accuracy over time, as in weather forecasting models. The distributed computing paradigm alleviates this problem by reducing the runtime of long-running simulations without affecting the final outcomes [90]. While distributed engines offer many advantages, managing communication and workload across machines requires additional effort. In a distributed environment, agents may be simulated on different machines at each time step, so the workload should be dynamically adjusted during the simulation. These operations introduce communication overhead, which is compounded by the synchronization step. Further, the entire computation proceeds at the pace of the slowest computer, which can be a barrier to the system's overall performance. Another unresolved issue in this context pertains to the reliability of distributed processes, meaning that one should expect functions to either complete successfully or terminate with errors without losing information, even though this may not always be theoretically possible [146]. Frameworks like Mason already provide algorithms and tools to help their users implement distributed ABMs. This design choice allows them to focus solely on simulation-related details without directly addressing the technicalities of distributed computation.

Hypergraphs. A hypergraph is a generalization of a graph in which a hyperedge connects an arbitrary number of nodes. Formally, a hypergraph is an ordered pair $H = (V, E)$ where V is a set of vertices and E is a set of hyperedges. Each hyperedge is a non-empty subset of vertices; that is, $E \subseteq 2^V \setminus \emptyset$, where 2^V is the power set of V [61].

Hypergraphs model extremely non-linear interactions among collections of three or more nodes, thereby accounting for additional information [216]. For instance, hypergraphs can abstract

social systems in which people interact in groups of any size. One example is the co-authorship collaboration network, where a hyperedge represents an article and links together the authors who have collaborated on it. Other examples can be found in biology, ecology, and neuroscience [38]. One of the few downsides of hypergraphs' strong expressiveness is that studying these structures requires specialized algorithms due to their complexity [162, 185]. Because of this, hypergraphs have not been utilized as much in past literature as their graph equivalent. The use of hypergraphs has been arising recently as a result of numerous systematic studies showing how converting a hypergraph to a classical graph either results in an information loss that cannot be avoided or generates a significant number of additional nodes and edges that increase the amount of space and time needed for analytic tasks [27].

The literature has often studied an entire graph's topology by observing its substructures, commonly known as motifs. In particular, triangular subgraphs appear to be a powerful signal for identifying high-density communities or local trends within a graph [299]. This concept can be generalized to hypergraphs, where h-motifs describe the connectivity patterns of connected hyperedges [180]. Considering an h-motif involving three hyperedges, it is possible to enumerate 26 instances of these substructures. For example, given a set $E1, E2, E3$ of three connected hyperedges, an h-motif describes its connectivity pattern by the emptiness of the sets of the intersections. This definition can be generalized to an arbitrary number of hyperedges and expanded by including a temporal layer to control the evolution of these structures along time [184].

H-motifs can be used to build a hypergraph's characteristic profile (Characteristic Profiles (CPs)) to distinguish the structural design principles of real-world hypergraphs [180]. The significance of each h-motif in a hypergraph is measured by comparing the count of its instances against their count in randomized hypergraphs. Specifically, the significance of a h-motif t in a hypergraph H is defined as

$$\Delta_t := \frac{Motif[t] - Motif_{random}[t]}{Motif[t] + Motif_{random}[t] + \epsilon}$$

where ϵ is a parameter fixed to 1 to avoid a possible division by zero when a specific motif does not exist. The CP of a hypergraph H, which summarizes the local structural patterns, is a vector of size 26, where each t-th element is

$$CP_t := \frac{\Delta_t}{\sqrt{\sum_{t=1}^{n=26} \Delta_t^2}}$$

The information conveyed by h-motifs can be leveraged for tasks like hyperedge prediction. According to Lee et al. [180], different motifs produce varying results in this task, possibly because different motifs hold varying levels of importance in different application domains. This finding suggests that further efforts should focus on exploring how selecting the appropriate motifs across various datasets can provide the most valuable insights for the task at hand.

Large Language Models. Language models are artificial intelligence models designed to understand, generate, and manipulate human language [138]. These models are used in various applications, such as text generation, translation, and sentiment analysis, to name a few. Language models with hundreds of billions (or more) of parameters, known as Large Language Models (LLMs), are trained on enormous amounts of text data. As an example, we can include GPT [115], LLaMA [288], and Gemma [281]. LLMs generally try to forecast the likelihood of future (or absent) tokens by modeling the generative likelihood of word sequences [320]. These models have been evolving rapidly, from basic language model interfaces to problem-solving agents, demonstrating strong abilities to comprehend natural language and complete challenging tasks via text generation. Prompting has emerged as the most popular method for using LLMs for various tasks based on the natural language interface. Context learning imbues LLMs with the capacity to perform well on unknown tasks, sometimes surpassing fine-tuned models, by merging task descriptions and demonstration examples into prompts. Advanced prompting approaches, such as the chain-of-thought strategy that incorporates the intermediate reasoning steps into prompts, have been proposed to improve the ability to use complicated reasoning. LLMs enable the development of more intelligent systems (e.g., autonomous AI agents) to tackle various complex tasks in real-world scenarios. With the industry's most recent

developments, these tools have many applications, and new ones are found daily [159]. In my Ph.D. project, I leverage LLMs to tackle a stance detection task. Specifically, this task combines hypergraphs for abstracting conversational networks with the analytical power of LLMs to examine their dynamics.

1.2 CONTRIBUTIONS

1.2.1 *Goals*

The primary objective of my Ph.D. project is to investigate and analyze complex systems by leveraging ABMs, hypergraph-based network models, and LLMs. The specific goals for each of these research areas are outlined below.

Agent-based modelling. In the context of ABMs, my goal is to enhance their functionalities to advance the state of the art in ABM engines, particularly regarding their computational limits, as the most widely used ABMs prioritize either performance or usability. Hence, a primary focus of my research is to develop an approach that effectively integrates both features. The main research question my project aims to answer is:

> *How can advances in parallel and distributed computing technologies be harnessed to improve the scalability and efficiency of ABMs for simulating large-scale, real-time, and highly dynamic systems?*

Answering such a question implies validating the following hypotheses:

- **Q1_ABM:** Feasibility of developing an efficient and reliable distributed ABM engine.
- **Q2_ABM:** Addressing the potential increase in model development complexity through a comprehensive study of system usability from a user perspective.

Hypergraphs. Another main line of work of my Ph.D. project is analyzing group interactions and the rise of collective behaviors in online social media through the lens of hypergraphs. In this regard, the main research questions my project aims to answer are:

> *Are specific structural patterns in hypergraphs dependent on the communities involved, or do they highlight specific user behaviors? Can we predict the formation of these patterns?*

Answering this question leaves room for several hypotheses, such as:

- **Q1_HG:**When analyzing data from the same domain (e.g., conversations about general purpose vs. conversations on domain-specific programming languages), users' activity patterns should differ.
- **Q2_HG:** Tracking users across communities and observing their influence can reveal a correlation between structural patterns and specific user behaviors.
- **Q3_HG:** When a community of the same users persists over time, its structural pattern should remain consistent.
- **Q4_HG:** Incorporating higher-order relational information from hypergraph structures should improve link prediction by capturing collective interaction patterns.

Large Language Models. Regarding LLMs, my project's goal is to assess the feasibility of using these tools to extract meaningful insights from text and enhance our understanding of how user interact and exchange their opinions in online social contexts. Specifically, the main research question my project aims to answer is:

> *Can LLMs improve our understanding of online social interactions and opinion dynamics?*

Answering this question leaves space for different hypotheses, such as:

- **Q4_LLM:** An LLM can serve as a viable ground truth generator in the absence of annotated conversational datasets.
- **Q2_LLM:** Extracting information from text and structures using LLMs can enhance the performance of existing models.

- **Q3_LLM:** For stance identification tasks, incorporating high-order interactions that capture the subtle local context of conversations can improve model performance.
- **Q4_LLM:** Integrating semantic information extracted by LLMs with higher-order relational structures can improve link prediction performance in complex interaction networks.

1.2.2 *Challenges*

The themes explored in this dissertation naturally give rise to a range of theoretical, methodological, and practical challenges. Working at the intersection of complex systems, online social dynamics, ABMs, hypergraph-based modeling, and LLMs-driven analysis requires navigating heterogeneous data sources, integrating diverse formalisms, and balancing computational scalability with conceptual expressiveness. These challenges stem from both the inherent complexity of the systems under study and the limitations of current analytical tools. The following paragraphs outline the main challenges encountered during this work.

ABMs. Regarding ABM, there are two main problems to deal with. First, ABMs can be computationally intensive, especially when simulating large numbers of agents or complex interactions between agents [231]. This can limit the scale and scope of simulations that can be conducted, particularly on basic computing hardware (e.g., personal laptops). Furthermore, ABMs often require substantial parameterization and calibration to accurately represent real-world systems. Identifying appropriate parameter values and validating the model against real-world data can be challenging and may introduce uncertainties in the model [236].

Hypergraphs. Hypergraphs are a growing field, but there is a lack of standardization regarding definitions, models, and algorithms. Due to the arbitrary sizes of node relationships, dealing with hypergraphs requires interacting with a more complex network than a traditional graph. One example of this increased complexity is the visual representation of hypergraphs, which is often depicted using Venn diagrams. However, drawing such representations becomes nearly impossible or extremely chaotic as the number of nodes and hyperedges grows. While graphs can

be easily represented with points and lines, hypergraphs require circles or other techniques that do not always provide intuitive insights, necessitating the introduction of valid alternatives [114].

Another challenging task on hypergraphs is link prediction. In a graph, link prediction involves predicting whether a specific relationship exists between two nodes. This definition does not translate well to hypergraphs, where a hyperedge may include an arbitrary number of nodes [27]. In this context, predicting the appearance of an h-motif is even more challenging, as it requires understanding not only if a link will connect certain nodes but also if it will form a specific substructure with other existing links.

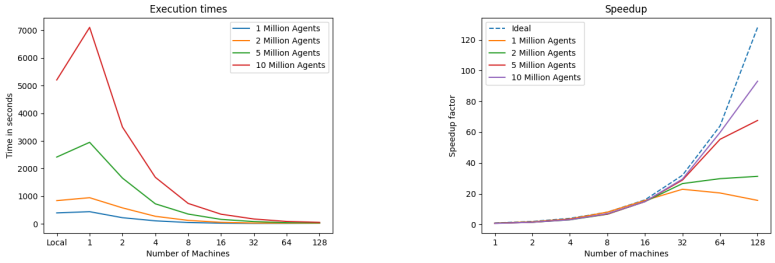
LLMs. Although LLMs offer numerous advantages, they also present several drawbacks. One of the most notable issues is bias. Training LLMs on massive datasets can inadvertently reinforce preexisting inequities or omit important perspectives due to biases in the data. Moreover, LLMs are often seen as black box models, making it difficult to understand how they reach their conclusions. This lack of interpretability can hinder trust and acceptance of their analyses, especially in scientific contexts where transparency and reproducibility are crucial. Finally, LLMs may struggle with understanding the context of a task, particularly in specialized domains where terminology and conventions vary significantly. Misinterpreting context can lead to inaccurate analyses.

1.2.3 *Results*

Over the past decade, the demand for more complex, computation-intensive models has led to the development of numerous frameworks and tools for running ABM simulations. To offer a tool integrating reliability, performance, and ease of use, we developed krABMaga [20, 24], an ABM simulation engine written in Rust.

One of the main goals of krABMaga is to make large-scale simulation available without expensive equipment while still offering efficiency and reliability [24]. Leveraging the Rust programming language has been instrumental in achieving this objective thanks to its performance akin to C, enabled by its memory model, and its expressiveness and usability typical of high-level languages

like Python [158]. To further support distributed computation, we developed a custom implementation of a K-Dimensional (K-D) Tree to partition the simulation field and evenly distribute the workload across processes. Unlike traditional K-D Trees, which maintain references to each child in the parent node, our modified approach ensures that all nodes retain references to one another throughout the tree. This adaptation simplifies neighbor search and synchronization operations, crucial for KrABMaga simulations.



(a) Execution times.

(b) Relative speedup.

Figure 1.1: Execution performance comparison: (a) absolute execution times and (b) relative speedup.

Figure 1.1 shows the performance of our framework running the Flocker simulation, commonly known as Boids [244], on a Microsoft Azure cluster with 128 nodes using the K-D Tree. Specifically, Figure 1.1a highlights the execution times for simulations with varying numbers of agents (1M, 2M, 5M, and 10M), while Figure 1.1b showcases the speedup achieved in each experiment. This second plot shows a significant speedup as the number of agents increases, attributed to the simulation processes' effective management of agents. This efficiency becomes more pronounced when the computational workload exceeds the communication overhead. The results closely follow the ideal performance curve, where speedup scales linearly with the number of processors or machines (e.g., 2 processors yield a speedup of 2, 4 processors yield a speedup of 4, and so on).

Online social networks generate vast amounts of data. Websites like Facebook, YouTube, or Reddit have millions or billions of users who interact daily. My focus in this context is on analyzing group interactions and the structural and semantic patterns of

user communities in online social media through the lens of hypergraphs.

In a first study, we analyzed developers' behavior and the formation of communities around 20 popular programming languages, emphasizing the significant differences in how users engage with different Question & Answer (QA) platforms, such as Stack Overflow and Reddit [23]. QA platforms, and social networks in general, usually consist of discussions in which users reply to each other on the same topic. Hence, conversational networks provide opportunities to identify strongly connected communities and examine whether specific user behavior depends upon the characteristics of the given platform. Examining the CP of such conversations could yield deeper insights into user communication patterns and their similarities across topics, communities, and platforms. Figure 1.2 shows the CP of 22 different subreddits as an example of this analysis. Preliminary results indicate that conversations on similar topics do not necessarily exhibit similar CPs.

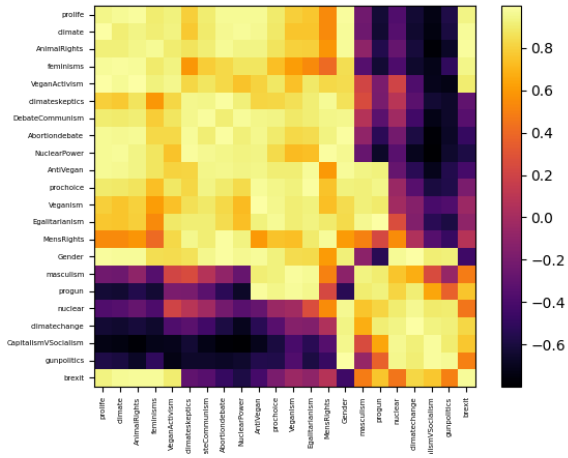


Figure 1.2: Similarity matrix of CPs of 22 subreddits on different topics.

Unfortunately, access to the APIs or raw data of many online social platforms is often limited or behind a paywall, posing a barrier to researchers interested in studying online behavior. As a consequence, having a ready-to-use dataset for specific analyses is not always straightforward. To alleviate this problem, we pub-

lished an open-source online repository to enable researchers to compare and download data tailored for high-order analysis [30].

Another interesting line of inquiry that has emerged recently is the use of LLMs to gain insights into both structural and linguistic attributes of conversational networks. Our initial attempt combined hypergraphs (to learn the conversational hypernetwork's structure) and LLMs (to understand the text's semantics) to perform stance detection [103]. Specifically, stance detection involves automatically classifying a user's attitude toward a target into one of three categories: *Favor, Against, Neither*.

In our experiments, we focused on the Reddit discussion platform, selecting three subreddits centered around the climate topic (i.e., *climate, climatechange, climateskeptics*), and retrieved all users' conversations over a two-year period. To obtain an initial training set, we used Gemma [281], a recently published LLM, to label a real-world dataset from Reddit, annotating the stance of all the messages, while the results obtained will be validated against a social network dataset annotated by humans for the same task [255].

1.3 TOO LONG; DONT READ (TL;DR)

Studying complex systems encompasses a variety of open problems, ranging from examining emergent behaviors to algorithmically analyzing how to prevent the spread of misinformation. This thesis aims to enhance our understanding of social dynamics in online social media by considering group interactions and to provide tools for easily and efficiently studying complex dynamics. Specifically, the contributions of my Ph.D. research can be summarized as follows:

- **ABMs.** We (i) conducted an in-depth review of existing ABM tools to identify current trends, challenges, and opportunities for innovation; (ii) developed a stable version of krABMaga, an ABM simulation engine, and various simulation models (contributing to **Q1_ABM**); (iii) extended the features and capabilities of krABMaga to include distributed computation and GIS data integration. Additionally, we focused on enhancing the system's usability by creating an easy-to-use GUI (addressing **Q2_ABM**).

- **Hypergraphs.** We (i) developed a community-driven open-source platform to store hypergraph datasets; (ii) examined interactions in online social media groups through the lenses of hypergraphs (contributing to **Q1_HG**); (iii) investigated the application of modern techniques, such as graph neural networks, to analyze high-order interactions (contributing to **Q2_HG**).

- **LLMs.** We (i) analyzed the semantic information within the text linked to each node in a hypergraph using LLMs (contributing to **Q1_LLM**); (ii) developed a model to embed LLM-generated information into a stance detection task (addressing **Q2_LLM**).

Table 1.1 provides a consolidated overview of the main contributions presented throughout this dissertation, along with the peer-reviewed venues in which each corresponding work has been published or accepted. This table is intended to help the reader quickly identify the relationships between the manuscript's chapters and their associated outputs. Appendix A.1 further presents the set of manuscripts that, at the time of writing, were submitted and under peer review, together with additional complementary works. These additional works are not central to the arguments and results developed in this dissertation, but have been included as an output of this Ph.D. project. In the interest of transparency and reproducibility, all code developed for the models, simulations, and experiments described in this thesis is openly accessible on GitHub or external tools, such as Google Drive for large datasets or websites. This ensures that the results can be independently verified, extended, or integrated into future research.

Table 1.1: List of published works. ▲ bottom-up works, ▼ top-down works

Venue	Type	Title	Reference	Year
AS	Journal	Experimenting with Agent-Based Model Simulation Tools [24]	▲	2022
WWW	Conference	The Age of Snippet Programming: Toward Understanding Developer Communities in Stack Overflow and Reddit [23]	▼	2023
JASSS	Journal	Reliable and Efficient Agent-Based Modeling and Simulation [20]	▲	2024
ITADATA	Conference	High-performance computation on a Rust-based distributed ABM engine [104]	▲	2024
WAW	Conference	Hypergraph Repository: A Community-driven and Interactive Hypernetwork Data Collection [31]	▼	2024
WebSCI	Conference	Deciphering Conversational Networks: Stance Detection via Hypergraphs and LLMs [103]	▼	2024
ASONAM	Conference	Beyond the surface: navigating complex systems via ABMs and Hypergraphs [101]	▼▲	2024
ASONAM	Conference	Fifty Shades of Reddit: Do Prolife Activists Have the Same Interaction Patterns of Gun Fanatics? [102]	▼	2025
ECML-PKDD	Conference	Conversational data and LLMs	▼	2025
BIGHPC	Conference	The impact of ECS logic on parallel performance in agent-based model simulations	▲	2025
KDD	Conference	Hypergraph Motifs Representation Learning [22]	▼	2025
Other Works				
CHI	Conference	The Metaverse through the Eyes of University Students [95]		2023
AIVR	Conference	CrossWarp: A Framework for the Integration of Virtual Worlds and Augmented Reality		2025

2

A BOTTOM-UP APPROACH TO MODEL COMPLEX SYSTEMS

Simulation models are one of the most effective and successful methods for studying real-world phenomena [106]. The term model refers to an abstract, simplified representation of the object of study that considers only the aspects relevant to the investigation; the concept of simulation indicates a model's manifestation, realized through software, for reproducing its dynamics and providing analyzable results. Among simulation models, ABMs are among the most widely adopted techniques for representing reality using a bottom-up approach [199]. Starting from a simple independent entity called an agent, the modeler can shape the global behavior of a complex system. An agent is an autonomous, independent entity that acts within an environment and interacts with it and other agents, according to a set of rules the modeler defines. The combination of these interactions creates a reproduction of the reality under investigation that the modeler can evaluate to extract valuable data and meaningful information from its emergent behaviors. The resulting model will exhibit patterns, structures, and behaviors that were not explicitly programmed but arise through agents' interactions [200]. For these reasons, ABMs are a valuable tool for studying, explaining, and predicting complex phenomena, supporting researchers in investigating how the macroscopic behavior of a system depends on its micro-level properties, constraints, and rules [1, 200, 262].

ABMs are extremely powerful for studying problems centered on an individual's interactions with other individuals or the surrounding environment [262]. This intrinsic characteristic makes ABMs particularly suited for application across diverse research fields. Nowadays, ABMs are widely used by researchers across various disciplines, allowing them to test and assess new theories and observe mechanisms previously unconsidered. Applications domains for ABMs include social science [125], economics [46, 111], climate change [112, 140], epidemiology [28, 295], transportation and logistics [163, 261], and many others [1, 6, 199]. The widespread use of ABMs across various application fields created

a need for tools that allow easy, quick simulation development without requiring significant coding skills. Several ABM tools and platforms were developed to address this requirement by abstracting the complexity of simulation implementation, allowing modelers to focus on the reality under investigation rather than the coding.

Existing ABMs reviews. The plethora of existing tools, frameworks, and libraries for developing ABM simulations makes it difficult for modelers to choose the right tool. Over the years, several works analyzed and compared the available ABM software to identify their pros and cons from different perspectives and support modelers towards the right choice for their needs.

The first review about general-purpose ABM software dates back to 2006 when Railsback et al. [239] compared the execution speed and the functionalities offered by some pillar ABM platforms, including MASON, NetLogo, Repast, and Swarm, and identified development priorities for future ABM platforms. In 2010, Allan et al. [6] provided a comprehensive survey of the ABM software packages available at the time, describing the use of ABMs across different domains and highlighting some of the most significant works. Five years later, Kravari et al. [172] supplied a comparative review of ABM platforms based on given evaluation criteria to classify ABM software and characterize their ideal usage situation. In 2016, Rousset et al. [250] provided a detailed description of ABM platforms running simulations on high-performance architectures; specifically, the authors focused on parallel/distributed multi-agent simulation tools and on the performance assessment of HPC-compliant platforms. A year later, Abar et al. [1] presented a concise characterization of all the ABM frameworks and tools available at the time, providing a valuable reference for researchers and developers, and identified future priorities for adopting ABM platforms. In 2020, Pal et al. [227] recently presented a review of existing ABM platforms, including active platforms and those no longer under development or with unclear status, providing a historical perspective on this domain.

2.1 AGENT-BASED MODELS AND SIMULATIONS ANALYSIS

First, we characterize ABMs by their key components; then, we focus on ABM tools, outlining their main features and the desiderata each tool should meet.

Although no single formal definition of ABM exists in literature, we can easily identify some key components that ABMs share: agents, environment, and rules [199]. Agents model the living population, the environment determines the setting where the agents act and the rules define the potential agent-to-agent and agent-to-environment interactions [73].

The main aspect of an agent is its ability to act autonomously in response to the surrounding environment while making decisions to achieve its internal goals. The modeler must define this decision-making process and specify the agent's behavior, which determines how the agent interacts with other agents and environmental factors [62]. The agent's behavior includes simple actions such as moving and communicating, as well as more complex operations that allow the population to evolve. Each agent maintains its attributes and the information it acquires during the simulation in its state, which may change over its life cycle. Agents live and act within an environment that defines how they can move and how they are connected to other agents. An environment has a well-defined topology, defined by fields such as spatial grids, continuous spaces, and networks. When the simulation environment must reproduce real-world places, ABMs can exploit GIS data to replicate existing locations, such as buildings or towns [62, 200, 279]. All other information related to fields and other non-active objects is included in the environment state. The collection of all agents' and the environment's states constitutes the simulation state, which holds all the information about the model and delineates its status at a specific time in the simulation.

The modeler must be able to identify, model, and code agents, environment, and behavioral rules to realize an ABM.

The effectiveness of ABMs clashes with the difficulty of developing models for researchers with limited computer science expertise. Therefore, the ABM community has made considerable efforts to provide standardized software platforms for designing, building, and executing ABMs. ABM tools remove many of the complexities of model implementation, allowing the user to

focus on simulation outcomes rather than on the development process [55, 73, 129]. ABM tools usually come in the form of frameworks and libraries, providing developers with *(i)* a framework consisting of a set of standard concepts for designing and describing a model and *(ii)* a library for implementing the framework, containing tools for the execution and the analysis of the simulation [6, 239].

Over the years, numerous ABM platforms have been developed with different objectives and targets. The first discriminant factor is identifiable in the platform's purpose. A tool can be either general or special-purpose [172, 227]. In the former case, this characteristic denotes the user can use the ABM platform to model any system of interest. In the latter case, the term special purpose implies that the system is oriented to a specific domain, including functionalities to address the peculiar situations of that field. A further differentiation concerns the tool's main development objective, usually identifiable as better ease of use or improved efficiency. Some ABM platforms emphasize easy-to-use interfaces with a reasonable learning curve that allows non-experienced programmers to produce models quickly [1, 218, 250]. The drawback of this approach is its low scalability, since graphical tools and domain-specific programming languages are not optimized for performance. Several ABM toolkits for mainstream programming languages also exist. In this case, they offer high-performance capabilities but require technical skills to use them appropriately [73, 129].

Although there is a lack of standardized criteria for analyzing ABM tools, we can enumerate a set of desiderata derived from existing work. In the following, we discuss the desirable requirements either explicitly mentioned in the literature by previous ABM-related articles or implicitly addressed and implemented by current ABM tools.

Researchers use ABMs to investigate and analyze complex phenomena, understanding how each component and its interactions with other elements affect their emergent behavior [1]. Conducting this kind of experiment often requires building elaborate models, given the number of agents and the parameters that regulate their interactions. The need to implement and run such models implies the first two fundamental desiderata of an ABM tool: efficiency and ease of use [1, 73, 172, 277, 282]. These two aspects are heavily influenced by the design of the ABM platform

and its programming model and are often two conflicting objectives. Some ABM tools expose their functionality via a Graphical User Interface (GUI) to enhance the user experience and provide high ease of use, while limiting achievable model complexity by preventing the user from personalizing or adding more complex dynamics. Conversely, frameworks and libraries based on standard programming languages provide a basis for developing complex ABM by offering generic facilities. As a downside, these tools require adequate technical knowledge, which may result in a higher perceived difficulty [239], a difficulty that is usually mitigated by proper documentation, examples, and tutorials [73, 218, 287].

Developing an ABM involves defining common patterns for agents' behavior, environment, and interactions. ABM tools should provide ready-to-use methods and interface covering those patterns, allowing the modeler to easily and quickly implement standard actions like movement and communication, agents' lifecycle and internal state management, environments creation using grids, continuous spaces, and networks, and interaction with the environment [73, 133]. Defining a realistic simulation field is a critical feature to accommodate since the simulation environment may provide a rich set of information influencing the agents' behavior. As 80% of the data have a spatial or geographical nature, the ability to work with GIS data is a fundamental requirement for any ABM tool. GIS-based systems use multiple spatial data models to represent and store information about phenomena with spatial location and extent [62, 279]. In the ABM context, GIS data are particularly crucial, especially for the extensive use of simulation in transportation, urban mobility [36, 324], epidemic, and pandemic models [71, 108] that incorporate spatial and network topologies to model, for instance, people's realistic activity [323]. Additionally, ABM should transparently handle multiple agent and field types within the same simulation without requiring developer intervention.

Other desiderata for simulation analysis include facilities for graphical model visualization, tools for statistical and non-statistical analysis, real-time monitoring, and data visualization [1, 73, 129, 133, 200, 239]. Among tools for statistical analysis, random number generation is of tremendous importance, since ABMs often include stochastic processes [33, 34, 200, 239], i.e., processes influenced by a specific random component, leading to volatile

simulation results. Researchers need to handle this stochasticity using random number generators to reliably reproduce a model's behavior and investigate how random distributions affect it [73, 143, 239]. The intrinsic stochastic nature of most ABMs also affects the reproducibility of such models. In this context, the automated validation process becomes a critical step in ABM development as simulations must be run several times and their results aggregated [34, 35, 63].

As the last desideratum, we include model exploration and optimization capabilities since modelers need to explore the model's parameter space, experimenting with how the simulation behaves when given parameters vary [34, 35, 73, 129].

The increasing adoption of ABMs in various research fields led to the emergence of numerous platforms to guide and facilitate the design and development of ABMs.

We focus our analysis on open-source general-purpose platforms supported by peer-reviewed academic literature and still actively maintained. To compile an initial list of candidates, we gathered data from the bibliographic database Scopus[1] because it is comprehensive yet accurate, covering peer-reviewed research on ABM-related topics. From this source, we collected all articles describing ABM frameworks, platforms, or tools. We then filtered out all articles related to commercial or proprietary platforms or discussing domain-specific ABM tools from this list. We finally included additional ABM frameworks collected from the code hosting service GitHub to consider other free, open-source works described in peer-reviewed articles not indexed in Scopus.

Scopus is an abstract and citation database of peer-reviewed literature including scientific journals, books, and conference proceedings.

2.1.1 *ABM tools description*

ActressMAS [186] is an agent-based framework written in .NET with the primary objective of being simple to learn and easy to use. ActressMAS is designed to allow the user to focus on the model logic rather than learning the framework, enhancing its accessibility at the expense of performance. According to its developers, ActressMAS should be used for applications that do not require fast execution speed or do not include a considerable amount of agents.

1 https://www.scopus.com/

AgentPy [120] is an open-source Python library for developing and analyzing ABMs integrated with IPython and Jupyter Notebooks, a web-based interactive development environment. AgentPy is designed for scientific applications and provides features for model exploration, numeric experiments, and advanced data analysis. The library provides functionality for easily creating models and their visualizations that can be embedded in Jupyter notebooks. Moreover, AgentPy allows the modeler to run simulations in a parallel environment without writing parallel code.

Agents.jl [100] is a recent framework for agent-based simulations that implements, runs, and visualizes models in the Julia programming language. This framework is mainly centered on granting efficiency and ease of use by exposing methods that allow the user to develop models with a few lines of code. Agents.jl is available as a Julia library and can be easily used with the plethora of analytical tools in the Julia ecosystem. It offers the most common ABM-related features, including different environments, support for GIS data, and model exploration capabilities. Agents.jl also supports parallel and distributed computing to empower simulation execution.

Care HPS [56] is a C++ tool for modeling and executing ABMs on high-performance architectures while hiding the complexity of parallel and distributed programming. The tool abstracts the modeler from crucial and complex tasks such as agent distribution, load balancing, and synchronization. Still, Care HPS remains easily extensible by expert developers.

Cormas (Common-Pool Resources and Multi-Agent Systems) [54] is a simulation platform based on the VisualWorks programming environment and the Smalltalk language. This platform is mainly dedicated to non-computer scientists and offers facilities to build, design, and analyze ABMs; however, it exchanges this ease of use with limited efficiency e scalability. Cormas editor allows the user to define agent behaviors through activity diagrams, without including sophisticated features, keeping its interface as simple as possible.

CppyABM [222] is a library for ABM development that combines the efficiency of C++ with the availability of Python libraries and exploits CMake to be platform-free. CppyABM provides all functionality in both languages, enabling users to choose their

preferred language. CppyABM relies on third-party packages to provide additional functionality while remaining lightweight. Other Python or C++ libraries can be installed separately and integrated into CppyABM.

EcoLab [271] is a framework for developing ABMs in C++ and executing them using TCL (Tool Command Language). This tool provides a GUI using the Tk toolkit and supports parallel and distributed processing by exposing utilities for managing communication. The user has to handle synchronization and partitioning manually.

Evoplex [70] is a platform for developing ABM applications in C++, using CMake scripts to facilitate compilation and setup, making it cross-platform. Evoplex adopts a fully modular approach, separating the core library from the GUI and visualization tools. The APIs exposed by the core library allow the user to develop the model. At the same time, additional components are available to improve ease of use with an interactive GUI and a web visualization tool.

FLAME (FLexible Agent Modeling Environment) [145] is an agent-based modeling system for creating models runnable on most computing systems, ranging from laptops to HPC supercomputers. FLAME provides a formal framework for creating models in the XXML language, a dialect of XML, which it uses to automatically generate the simulation source code in C. The FLAME engine automatically generates parallel code with no effort from the modeler, using a new, easy-to-understand programming language.

FLAME GPU [83] is an extended version of the FLAME framework to write ABMs for Graphics Processing Units (GPUs) using the FLAME standard formal XXML language. Thanks to FLAME GPU, the user does not need to explicitly understand GPU programming languages or optimization strategies since the API available uses the FLAME template to generate the simulation program in CUDA for target GPU devices. Visualization of the simulation is available even for a massive agent population without suffering performance loss.

GAMA (Gis & Agent-based Modelling Architecture) [279] is an agent-oriented generic modeling and simulation platform. GAMA provides high ease of use by offering a simple agent-

based programming language, GAML, that provides a formalism for describing the characteristics of entities in an ABM. Moreover, the platform can manage simulations with hundreds of thousands of agents with good performance. The modular architecture, separating each aspect of the model into a specific component, and the facilities provided make GAMA an accessible tool for non-expert developers with minimum learning requirements. This characteristic is enhanced by the full integration of GAMA with the Eclipse IDE, which provides convenient features such as auto-compilation, auto-completion, and template support. Finally, the platform supports the integration of external modules to introduce additional functionality, such as GAMAR [139], which enables the analysis of simulation results with R.

Insight Maker [122] is a graphical modeling and simulation tool focused on accessibility and availability of features rather than performance. This tool is a web application accessible in a standard web browser and includes features specific to the web environment, such as user management, model searching, and sharing. The main advantage of Insight Maker lies in its VPL (Visual Programming Language), which provides access to all its functionalities, including tools for data analysis, model exploration, and validation. The simulator also includes API to build models and analyze their results programmatically.

JADE (Java Agent Development Framework) [42] is an industry-driven Java FIPA-compliant framework aiming to simplify the implementation of multi-agent systems. Thanks to its features, this tool has established itself as one of the most popular platforms in academic and industrial communities [172]. JADE provides a powerful, useful GUI that enables the user to control and configure the simulation during execution and also supports debugging and development tasks. Moreover, this tool is designed to work on distributed systems, abstracting most of the inherent complexities from the modeler. The core characteristics of JADE make the tool highly scalable, robust, easy to learn, and compatible with most Java-based platforms. Moreover, its popularity provides strong user support, with comprehensive documentation and many tutorials and examples available.

JAS-mine (Java Agent-based Simulation library - Modelling In a Networked Environment) [246] is a Java-based toolkit for discrete-

event simulations designed to aid ABM development. Specifically, this platform aims to accelerate model development, facilitate model documentation, and support model testing and sharing. The core capabilities of JAS-mine reside in integrating I/O communication functionalities in the form of embedded relational database management systems tools and automatic CSV table creation. The database explorer included in the platform enables the user to inspect the database using Structured Query Language (SQL)- style commands.

krABMaga [21] is a fast, reliable, discrete-event multi-agent simulation toolkit based on the Rust language for developing ABMs. Designed to be a ready-to-use tool for the ABM community, krABMaga leverages the architectural concepts of the widely adopted MASON simulation library to provide modelers with a familiar programming environment and reduce the learning curve of the framework. However, krABMaga re-engineered some aspects of the MASON architecture to exploit Rust's peculiarities and programming model. This framework comprises all the functionalities required to develop and execute a model, including a visualization component and a convenient UI. Additional functionalities relate to running model exploration jobs on parallel, distributed, and cloud architectures.

MaDKit (Multi-agent Development Kit) [136] is a lightweight Java library for designing and simulating agent systems. The tool follows an organization-centered rather than an agent-centered approach based on the AGR (Agent/Group/Role) model. MaDKit provides several functionalities via APIs, including agents' lifecycle management and distribution, and is primarily intended for users with some programming knowledge.

MASON [196] is a discrete-event simulation toolkit written in Java for designing, executing, and visualizing ABMs. MASON provides functionality and an API that support the most common needs of a modeler, including agent behavior, environment creation, and scheduling management. One of the main advantages of MASON is its snapshot system enabling the user to stop and save a simulation and resume it in another machine thanks to the compatibility provided by the Java Virtual Machine. Moreover, thanks to the existing extensions, additional features are available, including GIS data with GeoMASON [275], model exploration with ECJ [301], and the ability to execute simulations

on distributed systems and in the Cloud with DistributedMASON [90]. Further, MASON is well-suited for computationally intensive models or long-running simulations.

MASS (Multi-Agent Spatial Simulation) [81] is a multi-agent and spatial simulation library designed to address the need for parallel ABMs. The architecture is based on the coordinator-worker approach, in which the coordinator process spawns workers on different computing nodes to run parallel simulations. MASS automatically manages agent execution and migration, as well as the simulation space, via several APIs that facilitate model development (if the user has some basic knowledge of Java).

Mesa [166] is a Python-based ABM framework providing built-in core components to easily create, visualize, and analyze simulations. Mesa is one of the most widely used and actively supported ABM libraries, leveraging Python's popularity to provide ease of use and accessibility. One of the main advantages of Mesa is its extensibility, which allows users to develop and share components through an open-source ecosystem. This approach created a rich community that provided extensions for any need, including the ability to exploit a multi-processor system, support for GIS data, and advanced analysis.

NetLogo [303] is an agent-based modeling environment implemented in Java and Scala, and it is considered the standard platform for developing ABMs. The importance and popularity of NetLogo rose thanks to its community, which continuously provides extensions such as GIS data support, 3D visualization, and integration with other languages, such as Python with PyNetLogo [154] or Pylogo [3], and R with RNetLogo [284]. Other relevant extensions worth mentioning are HubNet [155] for creating participatory simulations and BehaviorSpace [238] for providing parameter-sweeping capabilities using distributed and parallel techniques. NetLogo allows modelers to develop models using a simple, dedicated modeling language while providing a VPL for creating and editing components to realize any simulation. However, its accessibility imposes significant limitations on model complexity.

Pandora [252] is an ABM framework for large-scale distributed simulation that provides two identical programming interfaces, exposing the same functionalities in two different programming languages. pyPandora enables non-expert developers to quickly

build models in Python. C++ Pandora offers a more efficient interface for implementing complex models, including automatic generation of parallel and distributed code. Pandora includes Cassandra, a GUI tool with functionalities to design and analyze a single model execution or to set up a model exploration process. This tool can run large-scale ABMs and deal with thousands of agents with complex behavior.

Repast (REcursive Porous Agent Simulation Toolkit) [220] is a family of agent-based modeling and simulation platforms available in several programming languages. Repast Simphony [220] is a Java-based modeling system that provides automated methods to perform all the common tasks required in a simulation and supports several crucial additional functionalities. The Simphony platform is based on a modular architecture adopting a plugin system that enables adding a wide range of external tools. Any other Repast version implements the core features of Repast Simphony. Repast4Py [86] is a Python-based framework that includes functionalities to develop distributed ABMs.

RepastHPC (Repast for High-Performance Computing) [85] is another member of the Repast suite; specifically, it is a C++-based modeling system designed for running on large computing clusters and supercomputers. This toolkit enables the execution of massive simulations containing hundreds of thousands of agents with very complex behavior, which require high computational power. Although some built-in functions are available for developing a model, RepastHPC still requires users to have a strong programming background, as they must manage the various aspects of parallel execution.

2.1.2 *ABM tools comparison*

The plethora of ABM tools available in the literature can easily overwhelm any scientists interested in adopting a platform to develop ABMs with less effort. In this section, we compare the leading open-source general-purpose platforms from a two-fold perspective: the available features and the trade-off between their declared ease of use and efficiency. It is worth noting that (as previously mentioned) we impartially gathered all information from each platform's related article(s), website, and documentation.

Table 2.1: Description of the desirable features for ABM platforms.

Feature	Description
Programming Language	Programming language adopted for model development.
GUI	Availability of a Graphical User Interface (GUI) to control the simulation execution.
Visual programming	Use of a Visual Programming Language (VSL) to support the model development phase.
Simulation environment	Fields and topologies available to create the simulation environment, including GIS data.
Visualization	Possibility to graphically visualize the model.
Snapshot and checkpoint	Presence of functionalities to pause and save the simulation state and resume it later.
Modularity and reusability*	Adoption of a modular design supporting the reuse of portions of simulation code that can be combined to realize other simulations.
Inspector	Facilities for retrieving the model's information during its execution.
Analysis tools*	Presence of tools for statistical and non-statistical analysis, creation of charts and plots.
Random number generator*	Support for the custom generation of random numbers.
Batch runner*	Facilities to automatically perform multiple runs of the same simulation to assess whether stochastic processes influence the model's output.
Continuous Integration (CI) / Continuous Development (CD)	Facilities to let developers automate code continuous integration and delivery while keeping track of the simulation versions.
Model exploration and optimization* (MEO)	Facilities for exploring the model's behavior when its input parameters vary and optimizing its outcomes. This feature also includes automated testing.
HPC	Exploitation of parallel and distributed computing as well as integration with cloud platforms for in-model execution or optimization.

* We only report whether the given feature is available without considering its comprehensiveness.

Every ABM tool should offer fundamental features to assist users in creating their models [239]. Table 2.1 lists these properties and includes facilities to write, run, analyze, and optimize ABMs. Tables 2.2 and 2.3 compare the tools based on whether they support each desirable feature.

We do not explicitly report the feature *Modularity and reusability* as a column since all frameworks support this property to some extent, primarily thanks to their model development language. In particular, most of the tools exploit an object-oriented programming (OOP) approach, natively benefitting from the concepts of inheritance and polymorphism. Further, OO programming languages, such as Java and Scala, serve as the building blocks for domain-specific languages (such as NetLogo and GAML),

Table 2.2: Comparison of ABM tools based on their features.

↓ Tool / Feature →	Programming language	GUI	VSL	Simulation environment	Visualization	Snapshot & checkpoint	Code
ActressMAS	C#	✓		Generic			[116]
AgentPy	Python	✓		Grid, continuous, network	2D		[118]
Agents.jl	Julia	✓		Grid, continuous, network, GIS	2D, 3D		[98]
Care HPS	C++						[56]
Cormas	VisualWorks	✓	✓	Generic	2D	✓	[53]
CppyABM	C++ and Python			Generic	2D, 3D		[221]
EcoLab	C++ and TCL	✓		Grid, continuous, network	2D	✓	[270]
EvoPlex	C++	✓		Grid, continuous, network	2D		[69]
FLAME	C++ and XML			Generic			[178]
FLAME GPU	C++ and Python			Generic	3D		[248]
GAMA	GAML	✓	✓	Grid, continuous, network, GIS	2D, 3D	✓	[278]
Insight Maker	JavaScript	✓	✓	Grid, network			[121]
Jade	Java	✓		Generic		✓	[41]
JAS-mine	Java	✓		Grid			[245]
krABMaga	Rust	✓		Grid, continuous, network	2D		[265]
MADKIT	Java	✓				✓	[135]
MASON	Java	✓		Grid, continuous, network, GIS with GeoMASON [275]	2D, 3D	✓	[259]
MASS	Java, C++, CUDA	✓	✓	Grid, network	2D		[213]
Mesa	Python	✓		Grid, continuous, network, GIS with Mesa-Geo [59]	2D, 3D with Mesa 3D [209]		[164]
NetLogo	NetLogo, Python [2, 153], R [283]	✓	✓	Grid, continuous, network, GIS [253]	2D, 3D [60]	✓	[293]
Pandora	C++ and Python	✓		Generic, GIS		✓	[251]
Repast	C++, Java, Python	✓		Grid, continuous, network, GIS	2D, 3D	✓	[88]
RepastHPC	XML			Generic	3D	✓	[87]

which directly inherit their OO characteristics. In this context, it is worth noting that GAMA is the first (and unique) tool to introduce the concept of modularity by design, thanks to its co-modeling mechanism [279], which enables users to incorporate previously defined agents or models into new models to create more complex simulations.

Ease of use and efficiency are two essential, often conflicting criteria commonly used to assess software platforms, and ABM tools are no exception [122]. In this work, the term *ease of use* refers to the effort required for installation and setup, the availability of examples, and the clarity of the documentation provided. The term *efficiency* refers to the capability of the ABM tool to handle large and complex models while maintaining low execution time.

We evaluated the ease of use of each ABM platform by adopting a five-level Likert scale. The scale is based on the criteria included in the work of Macal et al. [199], and it considers the programming model adopted by each ABM tool and the (number of) available functionalities as reported in the corresponding

Table 2.3: Comparison of ABM tools based on testing, CI/CD, and HPC capabilities.

↓ Tool / Feature →	Inspector	Analysis tools	Random number generator	Batch runner	CI/CD	MEO	HPC	
							In-model	MEO
ActressMAS	✓	✓	✓				Parallel, distributed	
AgentPy		✓	✓	✓	With extension [119]	✓		Parallel
Agents.jl	✓	✓	✓	✓	With extension [97]	✓		Distributed via `Distributed` [99]
Care HPS			✓				Parallel, distributed	
Cormas	✓			✓	With extension [52]	✓		
CppyABM		✓	✓	✓			Parallel	
EcoLab	✓		✓	✓	With extension [269]	✓	Parallel, distributed	
EvoPlex	✓	✓	✓	✓	With extension [68]		Parallel	
FLAME		✓	✓				Parallel, distributed	
FLAME GPU			✓	✓	With extension [247]		Parallel, distributed	
GAMA	✓	With Gamar [139]	✓	✓	✓	✓	Parallel	Parallel
Insight Maker	✓	✓	✓	✓		✓		
Jade	✓	✓			✓ [40]		Parallel, distributed	
JAS-mine	✓	✓	✓	✓		✓		
krABMaga		✓	✓	✓	With extension [264]	✓		Parallel, distributed, cloud
MADKIT	✓			✓	With extension [134]		Distributed	
MASON	✓	✓	✓	✓		With ECJ [128]	Distributed via DMASON [297]	Parallel, distributed
MASS	✓	✓	✓	✓	With extension [214]		Parallel, distributed, cloud	
Mesa		✓	✓	✓	With extension [165]	✓		Parallel
NetLogo	✓	With extension [268]	With BehaviorSpace [285]	With extension [325]	With extension [292]	With BehaviorSpace [285]		Parallel
Pandora	✓	✓	✓	✓		✓	Parallel, distributed	
Repast	✓	✓	✓	✓		With EMEWS [226]	Parallel, distributed	Distributed
RepastHPC		✓	✓	✓			Parallel	

documentation, websites, and articles (declared ease of use). A description of the scale follows.

- *Very low*: poor modeling/execution APIs are provided by the tool, and the developer is also responsible for many execution/technology-related tasks.
- *Low*: good modeling, but poor execution APIs are provided by the tool.
- *Medium*: comprehensive modeling/execution APIs.
- *High*: improves the previous level by adopting a well-known programming language, such as Python or Java, thereby expanding the pool of developers ready to use the tool.

- *Very high*: tools that offer the previous level by using a Domain-specific Language or a Visual Programming Language specifically developed for ABM.

Broadly stated, the ease of use of a tool strictly depends on the programming language used (higher-level languages are generally easier to approach), the number of available functionalities (the higher, the better), and the availability of a dedicated GUI or a VSL (which can further reduce the need for coding).

The classification of ABM tools by efficiency is based on how the authors position themselves within the state of the art regarding their potential to handle large-scale models and their execution efficiency. In this case, we used a five-level Likert scale ranging from *Very low* to *Very high* to evaluate the efficiency of each tool as a function of its underlying technology and HPC capabilities. Table 2.4 classifies all ABM tools based on their ease of use and efficiency.

We conclude this overview with **AnyLogic**, a proprietary software tool for developing ABM through a simple user interface, making it particularly suitable for non-expert developers. The AnyLogic platform includes APIs for modeling agents' behavior, supporting multiple environments (e.g., GIS space) and execution paradigms (e.g., distributed and parallel computation). AnyLogic offers visualization capabilities and a GUI to visually control any aspect of the simulation during execution. Moreover, a snapshot system allows the user to stop and restore the simulation later. AnyLogic Cloud is an extension of the main platform that permits running ABM on cloud computing resources.

Table 2.5 summarizes the ABM frameworks and tools described, emphasizing whether they provide support for the most relevant features. The last two rows of the table report the declared efficiency and ease of use for each tool as discussed before. The ease of use of a tool is closely tied to the programming language used (higher-level languages are generally easier to approach), the number of available functionalities (the higher, the better), and the availability of a dedicated GUI or a VSL (which can further reduce the need for coding). The term *efficiency* refers to the ABM tool's ability to handle large, complex models with low execution time. The classification of ABM tools by efficiency is based on how the authors position themselves within the state

Table 2.4: Trade-off between the declared ease of use and efficiency of the ABM platforms.

DECLARED EFFICIENCY \ DECLARED EASE OF USE	Very low	Low	Medium	High	Very high
Very low				Cormas Mesa	Insight Maker NetLogo GAMA
Low				AgentPy	
Medium			FLAME MADKIT Repast	ActressMAS Evoplex	
High		EcoLab	Agents.jl JADE krABMaga MASON MASS Pandora	CppyABM JAS-mine	
Very high	Care HPS FLAMEGPU	RepastHPC			

of the art regarding their potential to handle large-scale models and their execution efficiency.

The current state of the art in ABM software includes tools focused on ease of use, such as NetLogo and Mesa, and frameworks focused on performance and flexibility, such as MASON and Repast. With krABMaga, we aimed to address current open challenges in the ABM field, providing a valuable tool that delivers strong performance and reliable execution while still offering an accessible development experience. To this end, we provide extensive documentation, examples, and functionality to abstract the ABM development process from engine-specific mechanisms

Table 2.5: Comparison of ABM framework/software features.

ABM Tool / Features	MASON [196]	Agents.jl [100]	Mesa [166]	NetLogo [303]	Repast [220]	Flame [145]	AnyLogic [57]	krABMaga
Programming Language	Java	Julia	Python	Java and Scala, Python and R with extensions	C++, Java, and Python, based on version	C and XXML	Java	Rust
Supported environment fields	Grid, Continuous space, Network							
Visualization	2D/3D	2D/3D	2D/3D with extension	2D/3D	2D/3D	3D	2D/3D	2D (3D Experimental)
GUI	Yes, limited	No	Yes, limited	Yes	Yes	No	Yes	Yes, limited
GIS Data	Yes, with extension	No	Yes, with extension	Yes	Yes	No	Yes	No (Ongoing)
Model Exploration	Yes, with extension	Yes	Yes, with extension	Yes, with extension	Yes, with Repast Simphony	No	Yes	Yes, parallel, distributed and cloud computing are also supported
Checkpoint and Snapshot	Yes	No	No	Yes	Yes	No	Yes	No (Future Work)
Parallel In-Model Execution	No	No, delegated to the modeler	Yes, with extension	No	Yes, with Repast HPC	Yes	Yes	Yes, Experimental
Distributed In-Model Execution	Yes, with Distributed MASON	No, delegated to the modeler	No	Yes, with extension	Yes	Yes	Yes	Yes, Experimental
Declared efficiency	High	High	Low	Very low	Medium	Medium	High	High
Declared ease of use	Medium	Medium	High	Very high	Medium	Medium	Very high	Medium

and the specific nuances of the Rust programming language. While a basic familiarity with Rust is beneficial, it is worth noting that it is not a prerequisite for using our framework, even though some coding skills are required. Many implementation details remain transparent to the user, allowing individuals without extensive Rust expertise to utilize our tool effectively.

ECS. Implementing complex applications, such as ABM, using traditional OOP architectures can often be challenging. Large class hierarchies, complex inheritance structures, and the addition of new entity types can quickly lead to a large codebase. As a result, code flexibility and reusability decrease, and performance can suffer. The encapsulation of data inherent to OOP can also pose challenges when aiming for high performance, particularly with large numbers of entities. This encapsulation can make it more difficult to optimize memory usage and data access patterns, both of which are essential for efficiency in large-scale applications [302]. ECS architecture can provide a complementary framework for exploiting ABM's logic in parallel. The clear distinction between logic and data is the core of the ECS architecture. This separation is achieved by defining an object as an entity uniquely identified and associated with multiple dynamic components that represent its data. This data is stored in memory to optimize cache usage and can be added, removed, or iterated during program execution. Iteration is facilitated by systems that allow the selection of the specific data type to process for each entity. More specifically, systems work on a set of components called archetypes. This approach enables the development of reusable, extendable, and high-performance code, focusing only on the relevant data for a particular operation while maximizing cache locality. Currently, several frameworks offer efficient implementations of this architecture. Unity and Unreal Engine, two popular game development engines, provide an ECS framework, Unity DOTS [291] and Apparatus [107]. Other notable examples are entt [326] and Flecs [257]. Although our choice, better described in the next section, has fallen on Bevy ECS [48] to speed up the development process, there are other four major ECS implementations for the Rust programming language (Specs [10], Legion [9], hecs [240], Shipyard [188]). Bevy ECS is the ECS implementation used in the Bevy Engine, a game development engine. Bevy ECS was developed to provide a simpler alternative to other implementations that often require advanced Rust concepts such as lifetimes, macros, or the builder pattern to construct ECS elements. One of the most intriguing features offered by Bevy is the ability to run applications directly in the browser with minimal performance loss. The web platform leverages this capability to enable the execution and visualization of various simulations created with krABMaga directly in the browser. This is made

possible by integrating WebAssembly (WASM) and WebGL (Web Graphics Library).

2.1.3 *ABM tools evaluation*

Part of the remarkable popularity of ABM tools stems from the improved developer experience, which enables non-expert programmers and lay users to design and (potentially efficiently) run simulation models. In this section, we analyze the collected ABM tools from the developer's perspective, assessing each tool's experienced support and performance.

A critical quality of each ABM tool is the perceived hands-on experience. To evaluate this characteristic, we considered the following elements.

Installation and setup. Perceived difficulty in installing and configuring the tool.

- *N/A*: installation guide not available.
- *Easy*: an installation script or installer is provided, or the tool can be used as a standard library.
- *Hard*: information about the installation procedure are vague or requires technical expertise.

Documentation and examples. Extensiveness of the documentation and presence of examples and tutorials.

- *N/A*: no documentation and/or tutorials are available.
- *Basic*: most functionalities are not documented or only essential information is provided; few or no examples are given.
- *Good*: all functionalities are documented; some examples are given.
- *Extensive*: all functionalities are extensively documented; several comprehensive examples exist.

Effort. Effort required to implement a model, measured in hours.

Problems. Errors and issues that prevented the installation or the usage of the tool.

Table 2.6: Evaluation of ABM tools based on the perceived ease of installation and usage. N/V stands for "Not Verifiable" due to the inability to install or use the corresponding tool.

Tool	Installation & Setup	Documentation & Examples	Effort	Problems
ActressMAS	N/A	Basic	∼4h	No installation information provided; only works with Visual Studio.
AgentPy	Easy	Good	∼2h	
Agents.jl	Easy	Extensive	∼2h	
CppyABM	Easy	Basic	∼4h	
GAMA	Easy	Extensive	∼2h	
krABMaga	Easy	Good	∼3h	
MASON	Easy	Extensive	∼3h	
Mesa	Easy	Extensive	∼2h	
NetLogo	Easy	Extensive	∼3h (without VSL)	
Repast	Easy	Good	∼4h	
Care HPS	N/A	N/A	N/V	Only documented within the article.
Cormas	Easy	N/A	N/V	The application crashes after a few operations.
EcoLab	Easy	Basic	N/V	The installation script does not work.
EvoPlex	Easy	Basic	N/V	Examples fail to run; no usage information provided.
FLAME	Easy	Basic	N/V	Installation script requires fixes; no longer supported.
FLAME GPU	Easy	Good	N/V	Not included because of GPU involvement.
Insight Maker	Easy	Extensive	N/V	The web app cannot handle large workloads.
Jade	Easy	Extensive	N/V	Not included due to a different programming model.
JAS-mine	Easy	Basic	N/V	Installation fails due to configuration errors.
MADKIT	Easy	Basic	N/V	Installation fails due to missing JAR file.
MASS	Easy	Basic	N/V	The build process fails.
Pandora	Easy	Basic	N/V	No longer supported.
RepastHPC	Hard	Good	N/V	The installation script does not work.

Hands-on evaluation process. The hands-on programming experience was evaluated by two developers with strong backgrounds in computer science, relevant expertise in the ABM field, and experience with most of the ABM tools considered. Both developers independently installed each ABM tool reviewed in this work and attempted to resolve any issues that arose before moving on to the model development phase. At the end of this process, both developers rated each tool according to *(i)* its installation and setup procedures, *(ii)* the comprehensiveness of its documentation and examples, *(iii)* the effort required to develop a single model, and *(iv)* documented whether they encountered any issues in the process (as detailed above). Finally, both developers filled in Table 2.6, discussing any discrepancies until

agreement was reached. The upper part of the Table lists all tools correctly installed and used; the lower part lists the remaining tools, clarifying the problems encountered during installation or when running examples. The number of hours describing the effort required refers to the average time needed to develop a single model, since each model has its own peculiarities. In the case of ready-to-use ABM models (e.g., Flockers in NetLogo), the developers estimated the effort required to adapt them to the experiment's requirements.

To evaluate the efficiency and scalability of each ABM tool, we used four ABM models with different data structures, agent behaviors, and environment types. A brief description of each model follows:

- *Flockers*. Developed by Craig Reynolds, this is one of the most famous ABM simulating a flock's flying behavior. In this model, the agents move within a continuous toroidal space according to a simple set of rules.
- *Schelling*. This is a simple segregation model based on a 2D grid, in which agents decide whether to move to a new cell based on their neighbors' statuses.
- *Wolf, Sheep, and Grass*. This multi-agent model simulates the population dynamics of predators and prey coexisting in a shared environment.
- *ForestFire*. This stochastic spreading model is realized as a cellular automaton to reproduce the fire diffusion in a forest.

To perform a fair comparison, we developed a meta-model for each of the above models based on their NetLogo implementations, describing the agents' behavior, the simulation environment, and the possible interactions among agents and between agents and the environment. We used these meta-models to ensure that a specific model simulation would expose the same behavior in all tested platforms. Specifically, when we had to implement a given model for a specific platform from scratch, we followed the tool's suggested modeling choices and the meta-model's guidelines to guarantee the expected behavior. Conversely, if the model was already available in the platform's repository, we verified its behavior and applied small changes to

its implementation to match the meta-model, if needed. Despite our effort to reproduce the same model across ABM tools, some differences may persist due to differences in the frameworks, mainly because of the different architectures and programming languages involved. It is important to note that across all evaluated platforms, it is possible to reuse code portions, such as agent definitions and behaviors, in other models. For instance, by incrementally adapting its behavior, we included the Flocker agent model in the Wolf, Sheep, and Grass simulation. Specifically, we expanded a Flocker agent's basic movement behavior within a two-dimensional environment to a Wolf agent's more complex behavior, which also includes eating and reproducing.

Table 2.7 reports whether a given model was already available as an example on each functioning platform (see Section 2.1.3). The implementations of all models and the benchmark are freely available on a GitHub repository[2].

Table 2.7: Summary of whether each model was available for a given platform (●) or developed from scratch (○). Only the ABM tools that could be installed and used are considered.

↓ Tool / Model →	Flockers	Schelling	Wolf, Sheep, and Grass	ForestFire
ActressMAS	○	●	○	○
AgentPy	●	○	●	●
Agents.jl	●	●	●	●
CppyABM	○	●	○	○
GAMA	●	○	●	○
krABMaga	●	●	●	●
MASON	●	●	○	○
Mesa	●	●	●	●
NetLogo	●	●	●	●
Repast	●	●	●	○

Benchmark configurations. All experiments have been performed on the same Ubuntu 22.04 LTS x86_64 machine with kernel version 5.15.0-48-generic, and equipped with an Intel i7-8700T (12) @ 4.000GHz CPU, an NVIDIA GeForce GTX 1050 Mobile GPU, and 16GB RAM. The performance of each framework has been tested with different models configurations, starting with a field of size 100×100, 1000 agents, and 200 steps, while keeping an

2 https://github.com/isislab-unisa/ABM_Comparison

agent density of $\cong 10\%$, calculated as $\frac{\text{width}*\text{height}}{\text{number of agents}}$. We obtained the other configurations by doubling the number of agents and changing the field dimension to preserve the agent density. Table 2.8 lists all experiment configurations.

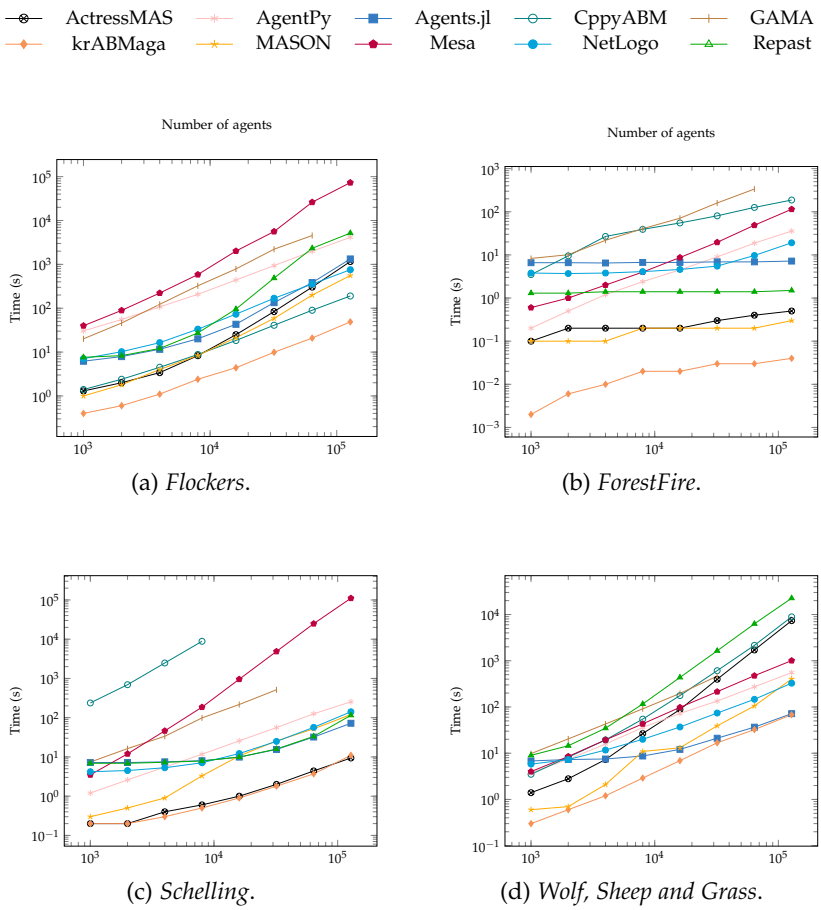
Benchmark configurations. All experiments have been performed on the same Ubuntu 22.04 LTS x86_64 machine with kernel version 5.15.0-48-generic, and equipped with an Intel i7-8700T (12) @ 4.000GHz CPU, an NVIDIA GeForce GTX 1050 Mobile GPU, and 16GB RAM.

Results. Figure 2.1 depicts the running times of all the benchmarked ABM tools varying the computational load over the four models. The missing data in the graphs indicate that the specific tool failed to perform the simulation with the given configuration.

Generally, most tools achieve the same performance stated in the referring articles. Still, we can note a few exceptions. Better-than-expected results from ActressMAS achieve execution times comparable to those of tools specifically designed for high performance. This outcome probably derives from the efficiency of C# in managing the simple data structures required to manage the simulation. The platform only struggles with the WSG model, which includes more complex data structures and requires multi-agent management. Likewise, NetLogo performs better than expected in all models, providing good efficiency and scalability. Despite not being designed for simulating complex models, NetLogo grants medium to low execution times and handles intensive workloads. Conversely, CppyABM struggled with heavy workloads, resulting in execution failures or high execution times. This behavior led to higher computational times

Table 2.8: Experiment configurations for evaluating ABM tools' performance.

#	Agents	Field size	#	Agents	Field size
1	1000	100×100	5	16000	400×400
2	2000	141×141	6	32000	565×565
3	4000	200×200	7	64000	800×800
4	8000	282×282	8	128000	1131×1131



(a) *Flockers*.

(b) *ForestFire*.

(c) *Schelling*.

(d) *Wolf, Sheep and Grass*.

Figure 2.1: Running times of each framework on the four ABMs, varying the computational load by increasing the number of agents while preserving the agent density.

than expected in all models but Flockers. The reasons for these results must be further investigated and may also be influenced by our inexperience with the platform and the need for more documentation and examples.

2.2 A BRIEF INTRODUCTION TO RUST

Rust is a multi-paradigm system programming language designed by the Mozilla Research Group in 2009 and released as a stable version in 2015. The Rust compiler is free and open-source dual-licensed under the MIT and Apache License 2.0.

Rust aims to provide developers safety, speed, and concurrency through its peculiar syntax, combining the expressiveness and usability of high-level languages, such as Python and Java, with the efficiency and performance of low-level languages, such as C and C++ [65, 204]. The Rust language adopts some Object-Oriented Paradigm principles and ensures safe concurrency and memory management thanks to its LLVM-based compiler, which further guarantees efficiency by automatically applying low-level optimizations. The memory management rules, these characteristics, and the lack of a garbage collector make Rust's performance comparable to C.

In recent years, Rust's popularity has risen among companies and developers thanks to its approach to software reliability and safety [66]. Companies like Firefox, Dropbox, and Cloudflare already use Rust in production, and several significant projects have adopted this language as their primary development tool. For instance, Linux developers added new features to the existing kernel infrastructure using Rust code, while both Google and Microsoft exploited Rust to reduce memory-related bugs and security flaws in Android and Windows systems. Rust's peculiarities also sparked academic interest, leading to several works that studied its characteristics through a theoretical lens. In particular, [157] provided the first formal and machine-checked proof of Rust safety properties by proving that *"a semantically well-typed program is memory and thread-safe: it will never perform any invalid memory access and will not have data races."*. Moreover, [234] formally demonstrated the validity of two basic concepts of Rust: reference lifetime and borrowing.

Throughout the manuscript, we may refer to technical elements of the Rust language. Although this work is self-contained, we refer the reader to the Rust official documentation for more details[3].

2.3 A NEW TOOL FOR RELIABLE AGENT-BASED MODELLING

A wide range of natural, social, and artificial organizations is characterized by many strongly interacting components, which give rise to behaviors that are incomprehensible when viewed in isolation [14]. Such organizations are commonly referred to as

3 https://www.rust-lang.org/learn

complex systems, and as examples, we can name traffic control, weather forecast, policy-making, or still epidemic dynamics [109]. Complex systems are analyzed by formulating models that imitate the real-world system and are usually the output of the scientific effort of numerous experts from different disciplines [263]. In this research field, ABMs constitute a robust modeling approach for designing the behavior of complex systems, using a bottom-up approach, as modelers can define agents and environments to reproduce particular aspects or properties of the underlying reality.

Over the last decade, the research community has developed many software frameworks and tools to support the development of ABM simulations [1]. These instruments are usually available as software libraries for different programming languages and may offer support for specific computational platforms (e.g., CPU, GPU, distributed computing [12, 250]) or particular application domains (e.g., traffic modeling [315], economics [7], or social sciences [242]). Generally, existing ABM simulation engines either prioritize ease of use, performance, or a trade-off between the two, depending on the modelers' needs and the computational requirements of the model to be developed [24]. In this context, it is worth noting that high-performance computing solutions are not only suited for large-scale/fine-grain ABM but are also convenient for small-scale ABM that require substantial computational support [11]. This statement becomes especially true when, for instance, a small-scale ABM simulation model undergoes multiple modeling phases, such as calibration, verification, validation, sensitivity and uncertainty analysis, and experimentation [72]. Furthermore, simulation runs typically require a large number of Monte Carlo repetitions. In a scenario of horizontal scaling, these massive Monte Carlo runs can be deployed to and accelerated by high-performance computing resources [280]. In a vertical-scaling scenario, the only suitable alternative to speed up the overall simulation process is to reduce the model's execution time. From this perspective, efficiently handling computation-intensive large- and small-scale models and analyses, and supporting the execution of long-running, reliable simulations, become even more critical requirements.

Current state-of-the-art efficiency-oriented ABM tools rely on C++-based solutions [24]. The major drawback of such solutions is their vulnerability to memory flaws, such as memory leaks or

stack overflows, which could unexpectedly cause a failure [317] in a single simulation run and eventually jeopardize the overall experiment or any of the cited modeling phases. In the context of ABM development, such scenarios are not uncommon given the simulation's inherent dynamics. In other words, the modeler is never guaranteed that everything will always work out just fine. To address the requirements of simultaneously offering efficiency, reliability, and safeness, we propose krABMaga, a novel tool for developing ABM simulations and supporting the modeler in handling their overall life cycle. krABMaga embraces these requirements as core development goals, thanks to Rust as the underlying implementation and model development language. The Rust language is characterized by performance comparable to C, which reduces the runtime of a single simulation, and a distinctive programming model that enhances simulation reliability by guaranteeing no memory-related errors in long-running experiments. In particular, our software library adheres to the *Safe Rust* specifications [254], which ensure the above reliability requirements. krABMaga particularly suits application scenarios where scaling up (both vertically and horizontally) is not viable or limited. Our simulation engine, therefore, targets small-scale, possibly long-running ABM simulations that require computationally intensive operations. Nonetheless, thanks to its architecture and programming model, krABMaga easily scales up to accommodate distributed computation environments (e.g., cloud-based platforms).

2.3.1 *The krABMaga architecture*

krABMaga is a fast, reliable, discrete-event multi-agent simulation toolkit based on the Rust language designed to be a ready-to-use tool for developing ABM simulations. Our selection of Rust as the framework's development language stems from its inherent principles of performance, reliability, and productivity, which harmonize seamlessly with the fundamental objectives of krABMaga. Still, it is crucial to emphasize that the true potential of our toolkit is unlocked by combining Rust with our expertise in ABM tools, which has empowered us to craft an efficient and resilient framework. Specifically, we carefully engineered all the underlying structures and components to maximize efficiency while ensuring ease of use and comprehensibility.

krABMaga embraces and re-engineers the architectural concepts of the wide-adopted MASON simulation library [196]. This design choice provides modelers with a familiar programming environment that decreases the learning curve of our framework while exploiting Rust's peculiarities and programming model. The name krABMaga is a portmanteau, i.e., a blend of two words from the combination of Krav Maga (a famous martial art) and ABM, which also sounds similar to the crustacean name *Crab*, the mascot of Rust. This name reflects krABMaga's original ambition to be effective and practical; we adopted the same principle in designing our tool.

krABMaga's architecture comprises two main software components: the *Engine* and the *Visualization*. The Engine component comprises all core functionalities to develop and run a simulation model, while the Visualization component exploits the Engine layer to visualize the simulation. Figure 2.2 depicts the architecture of krABMaga and highlights the dependencies and connections between its components.

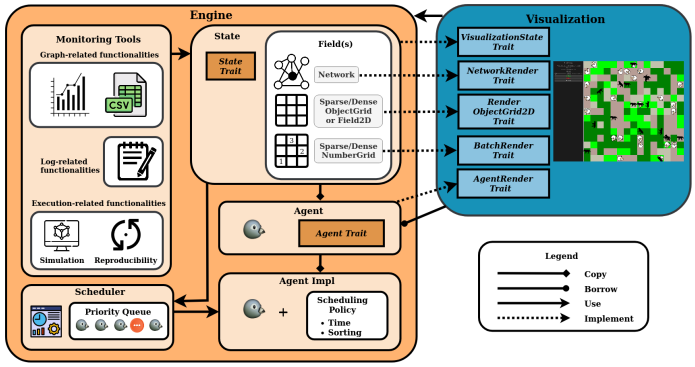


Figure 2.2: The krABMaga architecture.

The Engine component comprises the definition and implementation of the building blocks of every ABM simulation: *(i)* the agents, who model the entities of the simulation, *(ii)* the state of the simulation formed by all fields where the agents interact with each other and retrieve information, and *(iii)* the scheduler, which manages all simulation events. The absence of unsafe code blocks within the Engine component enhances the framework's memory and thread safety, thereby bolstering the reliability of krABMaga.

In krABMaga, an agent is any Rust object that implements the trait `Agent`. The use of traits helps the developer specify agents' behaviors and properties easily by enforcing the implementation of specific functions that define how agents should be appropriately scheduled (e.g., the method `step()`, which determines how agents act in each simulation step). The developer can further specify the implementation of other optional methods to improve agent control and complexity.

Tip

- A trait defines a specific type's functionalities, providing an abstract way to determine shared behavior. In other words, a trait is a valuable tool for establishing a set of behaviors that a group of objects must collectively exhibit.
- The trait `Agent` contains the behavior that krABMaga expects to be defined to consider a structure as an agent.
- An example of a specific set of traits could be *Eq + Hash + Clone + Copy*. This syntax means that the structure must implement (or derive) these traits; otherwise, the compiler would raise an error.

An overview of the methods included in the trait `Agent` follows.

- `function step(state)`, mandatory function defining the agent's behavior. The modeler can access the simulation state through the `State` input parameter and possibly modify the environment (e.g., simulation fields) or any other global structure;
- `function is_stopped(state)`, optional function defining the condition that determines whether the agent must be scheduled in the following simulation step or it can be removed from the simulation. This method allows the developer to manage the agent's lifecycle;
- `function before_step(state)`, optional function defining the agent's behavior that must be performed before the execution of each simulation step;

- `function after_step(state)`, optional function defining an agent's behavior that must be performed after the execution of each simulation step.

krABMaga supports the development of multi-agent simulations, enabling the specification of multiple agents within a single model. This feature is achieved by encapsulating the `Agent` object within the `AgentImpl` structure, which the scheduler actually uses.

Tip

- From the implementation perspective, we introduced a `Box<dyn Agent>` to satisfy the Rust compiler's requirement for explicit knowledge about the memory space required by each function's return type.
- The `dyn` keyword is used to highlight that calls to methods on the associated trait are dynamically dispatched. Such an approach allowed us to use the trait Agent as a function parameter, even if its real size is not known at compile time.

The simulation environment contains the *fields* where the agents live and act and represents the model's state. In krABMaga, the agents must update and read their location by interacting with the simulation state, even though they store their location internally. In this way, the state always contains the latest data, and the agents are guaranteed to access up-to-date information.

The modeler must implement the model's state, which defines the fields where agents are placed, any global properties of the model, and additional actions to perform during the simulation. Further, the modeler must define three mandatory methods, summarized below:

- `function init(schedule)`, function used to initialize the model, which should include the code for creating agents and fields and initializing the simulation properties at startup;
- `function update(stepnumber)`, function defining the simulation behavior to run at the end of each simulation step. It must define all the procedures for updating the different struc-

tures of the state (including the invocation of the update method for each field);

- `function reset()`, function used for resetting the simulation, usually containing the code for a clean initialization of the state structures.

The modeler can also define the state behavior at a specific simulation time via the following optional methods:

- `function before_step(schedule)`, function defining the state's behavior that must be performed before the execution of each simulation step;
- `function after_step(schedule)`, function defining the state's behavior that must be performed after the execution of each simulation step;
- `function end_condition(schedule)`, function defining the condition that determines when the simulation should end.

Simulation fields. A field is a data structure that represents the environment in which agents act and defines how they can move and interact within it. krABMaga provides three field classes[4], which cover the fundamental spaces required in an ABM, and specific functionalities for each of them, e.g., placing agents, retrieving agents' neighborhoods, and managing the simulation's static elements. Figure 2.3 depicts the krABMaga field taxonomy. A detailed description follows.

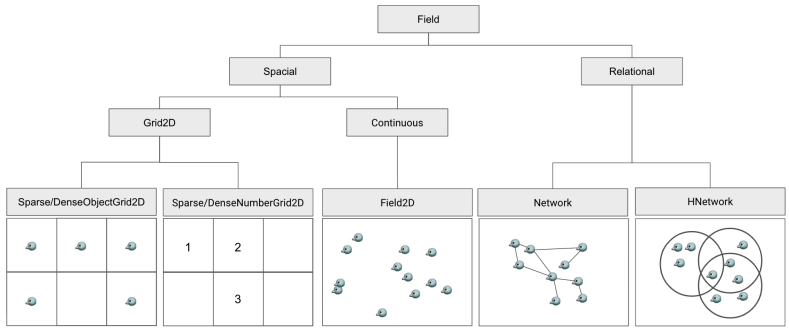


Figure 2.3: The krABMaga field taxonomy.

4 https://github.com/krABMaga/krABMaga/tree/main/src/engine/fields

`Grid2D` is a data structure based on a matrix design. In a `Grid2D` field, agents can move across the matrix cells, and a pair of coordinates identifies their location. krABMaga exposes two types of `Grid2D` fields: `SparseGrid2D` and `DenseGrid2D`. `SparseGrid2D` is appropriate for low-density fields, where most of the cells are empty, and exploits a HashMap to speed up read and search operations. The `DenseGrid2D` works better for high-density fields, where most of the cells contain a value, and is based on the Rust object matrix to make write operations as fast as possible. Both fields can be further specialized in:

- an `ObjectGrid2D` field, namely a `Grid2D` field for storing any Rust structures that implements a specific set of traits. Each cell of the field may contain more than one agent.
- a `NumberGrid2D`, a simpler `Grid2D` field for placing any Rust structure that implements a specific set of traits.

  This field is optimized to handle primitive types, such as numerical values. In this case, each cell contains only a single agent.

This design choice lets the developer optimize simulation performance by selecting the field that best suits the model.

`Field2D` is a bi-dimensional (possibly toroidal) continuous space where agents can be placed anywhere within the field.

The coordinates of an agent placed in a `Field2D` field are represented as a `Real2D` object consisting of a pair of floating point values.

`Network` is a graph-shaped field that defines non-spatial relationships between agents. This field permits the definition of directed, undirected, and weighted networks, where nodes represent agents and edges represent relationships between them. krABMaga also provides the `HNetwork` field, which models the underlying interaction network as a hypergraph. Specifically, hypergraphs are a generalization of graphs in which every (hyper)edge connects an arbitrary number of nodes.

Field updates. A crucial feature of krABMaga is the use of the *double-buffering* technique to manage memory for fields. Each

field uses two data structures to store its values: one structure is read-only (read state), and the other is write-only (write state). At the beginning of each simulation step, the read state contains up-to-date information, while the write state is empty. Any object performing a reading (resp. writing) procedure will thus access the read (resp. write) state. Hence, the two data structures are independent, but the engine automatically synchronizes them at the end of each step. Thanks to the double-buffering technique, krABMaga can perform concurrent read operations on the read-only data structure, thereby significantly improving simulation performance. On the contrary, writing operations, which change the simulation state, are performed sequentially to avoid data race and contention on the write state. In a nutshell, the double-buffering technique ensures that when an agent is scheduled, the correct information (e.g., its neighbors' locations) from the previous simulation step is used.

The modeler must define how each simulation field is updated using one of the following functions.

1. `function lazy_update()`. This efficient update operation swaps the read and write states and is recommended when the simulation data changes at each step. In this case, the swap approach significantly improves the performance. In practice, this function re-initializes the memory.

2. `function update()`. This update operation moves all the data from the write state to the read state. This function is computationally intensive and should be used when (part of) the data of the previous simulation step must be preserved.

The double-buffering mechanism of krABMaga is transparent to the modeler. Still, our framework offers additional methods that enable expert users to optimize their code for specific needs. These methods operate on the write state and are marked as `unbuffered`. For instance, a user may choose to design the simulation model to use the `lazy_update` function, thereby obtaining more efficient execution.

The krABMaga scheduler manages all the discrete events happening in the simulation. In other words, it controls when the agent's behavior (defined in the `Agent.step()` method) should run. The scheduler manages the simulation timeline via a priority queue that sorts agents by their scheduling time and priority.

At each simulation step, the scheduler selects all agents whose scheduling time matches the current simulation time, pushes them into a helper priority queue for the current step (which sorts agents by priority), and then calls their `step` functions. The modeler can access scheduling functionality to control the simulation pace through the `Scheduler`[5] object. Specifically, krABMaga provides two functions to schedule a new agent, i.e., insert it in the priority queue:

- `function schedule_once(agent, time, ordering)`. This function schedules a single-time agent, which is removed from the scheduling queue after its `step` function executes. The parameters include the agent's current step time and priority.
- `function schedule_repeating(agent, time, ordering)`. This function schedules an agent for each next simulation step. The parameters include the agent's current step time and priority.

The scheduler also handles the actions to perform before and after each simulation step for agents and for the simulation state. Further, the krABMaga scheduler can manage multiple agent types within the same model.

Scheduling process. The simulation process can be roughly split into three main phases. During the first phase, the scheduler initializes the number of simulation steps and its state to then invoke the state's behavior that must be performed before the execution of each simulation step (`State.before_step` function). Soon after, the scheduler selects and removes from the event queue $\mathbb{Q}$ all agents a whose scheduling time t_a matches the current simulation time T and inserts them into the priority queue $\mathbb{E}$ associated with the current simulation step. In the second phase, the scheduler handles the execution of each agent's behavior (`Agent.before_step`, `Agent.step`, `Agent.after_step` functions) contained in the queue $\mathbb{E}$ according to their priority. The agents are pushed again in the event queue $\mathbb{Q}$ if they need to be scheduled in the next simulation steps. When there are no more events in the queue $\mathbb{E}$, the scheduler moves to the next phase. In the third and last phase, the scheduler invokes the state's behavior that must be performed after the execution of each simulation

5 https://github.com/krABMaga/krABMaga/blob/main/src/engine/schedule.rs

step (`State.after_step` function) and at the end of each simulation step (`State.update` function) to update all the `State`'s data structures. The scheduler will continue processing the events until the maximum number of steps s is reached or an end condition occurs. Figure 2.4 summarizes the described scheduling process.

Developing an ABM is a complex process that requires the modeler to define each component properly to obtain an accurate simulation of the system under study. Further, ABM models usually include stochastic components, which give rise to patterns that can be analyzed statistically but not precisely predicted. For this reason, most models need to be run several times to analyze their behavior correctly. To alleviate these challenges, krABMaga offers a series of declarative macros for verifying the reproducibility of a model [318] and tracking simulation data. krABMaga also includes a convenient monitoring tool via a Terminal User Interface (TUI) that allows the modeler to monitor simulation events, create graphs from tracked data, and check simulation performance. The TUI provides default information about the simulation, including the step number, CPU and memory usage, and the number of steps per second. The user can add further information using a simple set of macro exposes by krABMaga. It is worth stressing that none of these features affect simulation performance or execution, as they are managed in a separate thread. A few details about the functions the user can manipulate to run a systematic suite of experiments and personalize the TUI follow.

- *Execution-related macros*
  - `simulate!(state, steps, reps)`. This macro runs an agent-based simulation for the specified number of `steps` and repeats the same simulation according to the number of repetitions `reps`.
  - `check_reproducibility!(state, steps, agent)`. This macro verifies whether two runs of the same simulation produce exactly the same results when a common seed is used. The definition of the agents must implement the trait `ReproducibilityEq` in order to perform the reproducibility control.
- *Visualization-related macros*

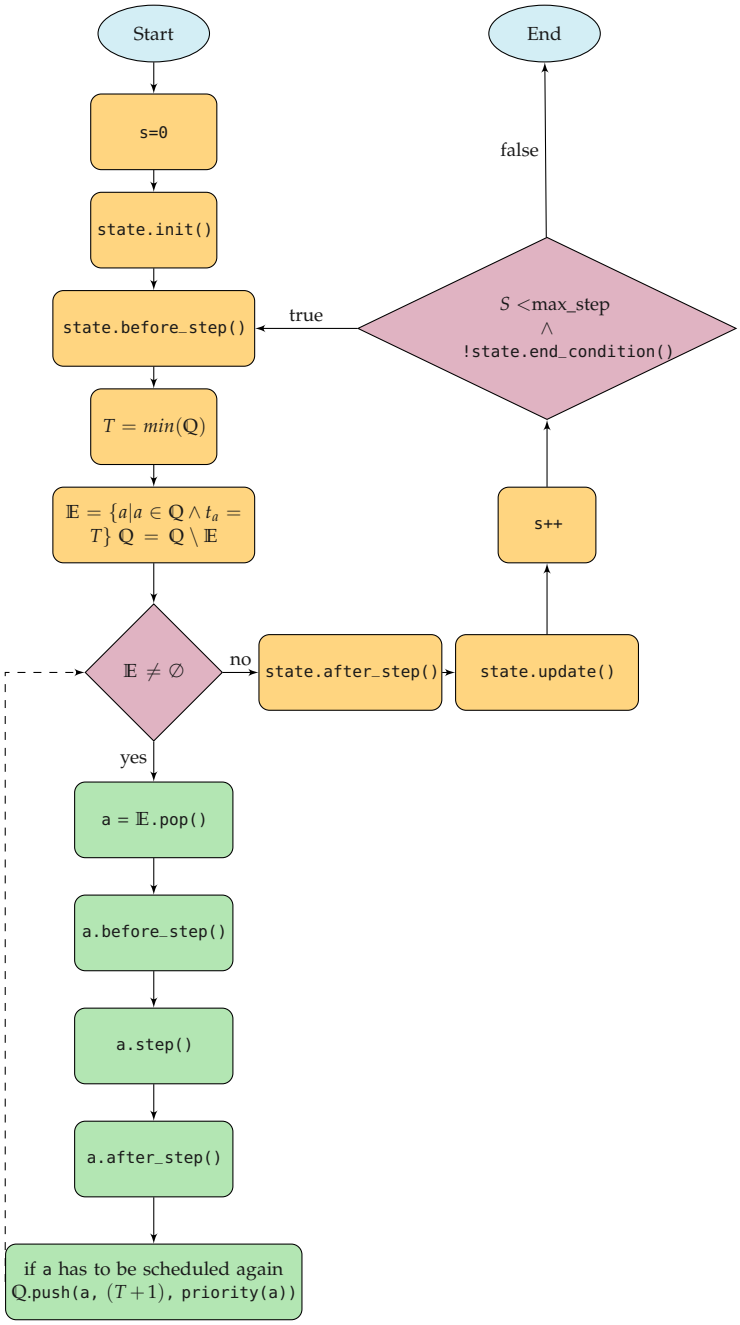


Figure 2.4: krABMaga scheduling flow.

  - `addplot!("Agents", "X_axis", "Y_axis", bool)`. This macro allows the modeler to create a new tab in the TUI to accommodate a new plot, which can be populated using the `plot!()` macro. The last input parameter determines whether the newly added plot is stored locally in CSV format.
  - `plot!("Agents", "Series", x, y)`. This macro adds a new point of the series `Series` located at coordinates `(x,y)` to the plot `Agents`.

- *Log-related macros*
  - `log!(Info, format!("STEP: {}", stepnumber)), bool)`. The `log!` macro allows the modeler to log any information within the TUI using different kinds of messages based on the event type, namely *Info*, *Warning*, *Critical*, and *Error*. An additional parameter enables saving the logs locally.

Tip

A macro is a code construct that generates additional code. Macros empower developers to define custom syntax for a specific use case. Unlike functions, macros accept an arbitrary number of parameters and are expanded before the compiler interprets the code's semantics. Consequently, a macro can be employed to implement a trait for a particular type.

An ABM framework that provides visual feedback on the simulation during execution results in a significant advantage for the modeler, allowing a better understanding of system behavior [171]. For this reason, krABMaga offers an efficient 2D visualization system based on the *Bevy* engine[6], which can be run locally or in a web browser thanks to WebAssembly[7]. Moreover, our framework provides a simple control panel to manage the simulation via the *egui*[8] library, enabling the user to pause, stop,

6 Bevy engine - `https://bevyengine.org/`
7 WebAssembly - `https://webassembly.org/`
8 egui - `https://www.egui.rs/`

and restart the simulation, and to manage its velocity (i.e., step rate). A few details about the Bevy engine and WebAssembly follow.

Bevy Engine. Rust's combination of low-level control, excellent performance, and modern build tools makes it an exciting choice for game developers and leads to the creation of several game engines. Among these, the Bevy engine stands out for its ease of use and efficiency. Bevy is a simple, open-source, data-driven game engine built in Rust that offers a complete 2D and 3D feature set. Bevy is simple to use for new developers but very flexible for experts, offering a data-oriented, modular architecture and high-performance parallelization.

WebAssembly. WebAssembly (Wasm) is a binary instruction format for stack-based virtual machines, designed as a portable compilation target for any programming language, enabling running applications in the browser. Wasm delivers excellent performance thanks to its conversion process, which does not require parsing or compilation to convert the source language into bytecode. The result is a small executable that runs in any browser with native performance. In krABMaga, we exploited *wasm-pack*[9] to generate WebAssembly code and *webpack*[10] to bundle the application for its release.

The krABMaga visualization component is entirely modular and acts as a separate system; thus, the user can either visualize the simulation or not without modifying the underlying model. This component serves as a wrapper for the simulation model: each primary trait defined in the Engine component has a counterpart in the Visualization subsystem (e.g., the trait `Agent` is paired with `AgentRender`). The Visualization component automatically manages model initialization and graphical updates at each step.

A detailed description of the traits exposed follows.

- `VisualizationState`, trait used to manage the visualization elements' startup and setup. The object implementing this trait must define how to initialize the graphics and retrieve the agent from the simulation state.

9 wasm-pack - `https://rustwasm.github.io/`
10 webpack - `https://webpack.js.org/`

- `AgentRender`, trait defining how agents should be drawn. The object implementing this trait must define the sprite representing the agent in the visualization, the agent's scale, location, and rotation, and how to manage information from the simulation state to update the agent's rendering.
- The Visualization component provides different traits defining how the field should be drawn based on its type:
  - `BatchRender`, a trait designed for fields containing numerical values that have to be visualized as a simple texture. The object implementing this trait must define how the visualization engine will convert the 2D points into pixels. It is worth noting that the simplicity of the data structure allows sending the entire structure to the GPU in a single batch, thereby improving overall performance.
  - `RenderObjectGrid2D`, trait designed for fields containing an object within each cell (not used for agents). The object implementing this trait must define how the engine will draw the object, including the sprite to use, its scale and rotation, and its update method.
  - `NetworkRender`, trait designed for the network field. The object implementing this trait must define how the visualization engine must draw the edges.

Calibrating, exploring, and validating ABM are crucial for obtaining reliable results. However, when the number of variables regulating an ABM's behavior jumps from a few tens to thousands or more, addressing these tasks may become computationally infeasible due to the high dimensionality of the search space.

[235] identified a community need for tools that facilitate model exploration and optimization. In response to this requirement, the authors developed the Simulation Automation Framework for Experiments (SAFE), which offers support from model construction to output data analysis. SAFE was designed to handle parallel execution on multi-core servers and distributed facilities. In a similar vein, SESSL [110] is a domain-specific language tailored for experiments and optimization processes, functioning as a distinct layer atop simulation systems. Similarly, the nlrx package [256] serves as an interface between R and the NetLogo framework, enabling self-contained, reproducible anal-

ysis of NetLogo models within R. It even supports the use of high-performance computing clusters, enabling multiprocessing and the execution of large-scale model simulations. In high-performance computing, OpenMOLE [243] is a workflow engine specifically designed for the distributed exploration of simulation models. It provides a domain-specific language that abstracts users from the technical details required for distributing experiments in a high-performance computing environment.

krABMaga effectively meets these needs by providing model exploration and optimization APIs via macros. These macros hide the complexity of the process and can be run in a parallel or distributed environment. Moreover, the framework seamlessly integrates with cloud computing platforms, employing the Function-as-a-Service (FaaS) model. In particular, the parallel mode leverages the Rayon library[11], enabling data parallelism, while the distributed mode harnesses the MPI (Message Passing Interface) protocol via the rsmpi crate [12], a Rust-based MPI binding. Additionally, krABMaga extends support to the Amazon AWS platform [13], facilitating the embedding of simulation execution within an independent function that can be deployed and run in parallel on the cloud.

In the following, we describe the krABMaga's model exploration and optimization macros in their sequential version. Each macro is also configurable to exploit different computing backends by specifying the optional enum parameter `ComputingMode`, which can assume the values `parallel`, `distributed`, or `cloud`. By default, each macro is run sequentially in the local environment. For further details, we refer the reader to the official krABMaga documentation[14].

This process, aka *parameter sweeping*, involves analyzing the model's sensitivity by varying input configurations. Listing 1 reports the pseudo-code of the procedure. For each input configuration $x_i \in \mathcal{X}$ (Line 1), the framework runs the simulation Φ over s computational steps, with a fixed random seed ϵ_j. This process is repeated R times (Lines $2-3$). The procedure returns the expected simulation values obtained from each input configuration (Lines $4-5$).

11 https://github.com/rayon-rs/rayon
12 https://github.com/rsmpi/rsmpi
13 https://aws.amazon.com/lambda
14 https://docs.rs/krabmaga/latest/krabmaga/#macros

Algorithm 1 `explore!`$(\mathcal{X}, \Phi(\cdot,\cdot,s)), R$

1: **for each** $x_i \in \mathcal{X}$ **do**
2: **for** $j \leftarrow 1$ to R **do**
3: $z_j \leftarrow \Phi(x_i, \epsilon_j, s)$
4: $y_i \leftarrow \mathbb{E}[z_1, z_2, \ldots, z_R]$
5: **return** $\mathcal{Y} \leftarrow \{y_1, y_2, \ldots, y_R\}$

krABMaga supports the user in generating random values for an input parameter by exposing the macro `gen_param!(type, min, max, n)`, which returns a vector of n uniformly distributed input values in the range `min` and `max`. We chose to implement uniform distributions as our first method, given their widespread use in simulation scenarios. We plan to incorporate additional distributions for creating random parameters in future updates. Further, the framework provides the output data $\mathcal{Y}$ in a `DataFrame` format, which enables straightforward analysis of the results and allows easy export to a CSV file.

This process, aka *simulation via optimization*, uses a search-based optimization algorithm to find the best input configuration for the simulation. In krABMaga, we implemented three parameter optimization approaches, accessible via as many Rust macros.

Random search [`random_search!(...)`] explores the parameter space by using a random searching algorithm, which consists of iteratively computing and evaluating a set of random input configurations until the maximum number of iterations is reached or a desired simulation score/error is achieved.

Evolutionary search [`evolutionary_search!(...)`] optimizes the simulation parameters by adopting an evolutionary searching strategy. Specifically, krABMaga provides a genetic algorithm-based approach [274].

Bayesian search [`bayesian_search!(...)`] performs a Bayesian optimization-based searching strategy [156] for parameter optimization similarly to the evolutionary strategy.

In krABMaga, each of the above macros implements a modified version of Algorithm 1. Specifically, each strategy defines how the set of input configurations $\mathcal{X}$ is defined; then, a goodness

or error value is evaluated for each set of expected simulation results to iteratively inform the parameter search space strategy.

2.3.2 *Programming with krABMaga: the Wolf, Sheep, and Grass Model*

This section describes the process of designing and implementing an ABM with krABMaga, using the Wolf, Sheep, and Grass (WSG) model as a use case. This model is a typical example of the effective use of ABM and has been widely studied in the literature [304, 305].

WSG is a model that simulates the population dynamics of predators and prey that coexist in a shared ecosystem. Wolves are the predators that eat sheep, their prey, which, in turn, eat the grass. Both wolves and sheep consume energy for each activity; thus, food availability determines their ability to replenish energy and survive. Specifically, sheep survival depends on the rate of grass growth. If the number of sheep is too high, the grass will not grow on time to be edible. Wolves' survival relies on the sheep's reproduction rate. If the number of wolves is too high, they will eat too many sheep, precluding their reproduction.

Given its dynamics, the WSG model requires a set of parameters that are properly calibrated to achieve stability. A system is called *stable* if the population of agents fluctuates over time but never reaches extinction. Conversely, the system is unstable if all the agents die at some point.

The WSG model has three fundamental concepts: wolves, sheep, and grass. Wolves and sheep move and actively interact with the environment, while the grass is stationary and only grows over time. For this reason, wolves and sheep can be represented as agents, while the grass is a numerical value. Although wolves and sheep act similarly, their interactions with the system differ; thus, these two entities are modeled as separate agents. Regardless of their nature, both agents need to carry the following essential information: a unique identifier (id), their current location in the field, and their previous location. Additionally, the WSG model needs specific properties for each agent, such as *(i)* its current energy, *(ii)* the energy gained from food, *(iii)* its reproduction probability, and *(iv)* its life state: dead or alive. The definition of the Wolf and Sheep agents is illustrated in Listing 2.1 and 2.2.

```rust
pub struct Wolf {
    pub id: u32,
    pub animal_state: LifeState,
    pub loc: Int2D,
    pub last: Option<Int2D>,
    pub energy: f64,
    pub gain_energy: f64,
    pub prob_reproduction: f64
}
```

Listing 2.1: Wolf agent properties.

```rust
pub struct Sheep {
pub id: u32,
pub animal_state: LifeState,
pub loc: Int2D,
pub last: Option<Int2D>,
pub energy: f64,
pub gain_energy: f64,
pub prob_reproduction: f64
}
```

Listing 2.2: Sheep agent properties.

In krABMaga, each entity representing an agent must implement the trait *Agent* and define its behavior in the function `step()`. Wolves and sheep perform three actions: *moving*, *eating*, and *reproducing*.

Moving. Wolves and sheep randomly move around the landscape. With a given probability α (`momentum_probability`), the agent moves in the same direction as the previous step; with probability $1 - \alpha$, the agent moves in a random position. Listing 2.3 depicts the code regulating how sheep move, but the same applies for wolves on the `wolf_grid`.

```rust
if self.last != None
&& rng.gen_bool(MOMENTUM_PROBABILITY) {
    [...]
    state.sheep_grid
    .set_object_location(*self, &self.loc);
```

Listing 2.3: Agents' moving behavior.

Survival and Reproduction. After moving, wolves and sheep simulate energy loss by subtracting a fixed value from their energy. If their energy drops below zero, the agent sets its `LifeState` to `Dead`. If the agent survives, it tries to reproduce. If it succeeds, it halves its energy and creates a new agent. It is worth noting that agents cannot add new entities to the scheduler

within their function `step()` as only the simulation State object can interact with the scheduler. The `new_sheep` and `new_wolves` arrays of the simulation State serve this purpose. Listing 2.4 depicts the code for the sheep agents, but the same applies for wolves using `Wolf::new()`.

```
self.energy -= ENERGY_CONSUME;
if self.energy <= 0.0 {
    self.animal_state = LifeState::Dead;
} else {
    if rng.gen_bool(self.prob_reproduction) {
        self.energy /= 2.0;
        let new_sheep = Sheep::new([...]);
        state.next_id += 1;
        state.new_sheep.push(new_sheep);
    }
}
```

Listing 2.4: Agents' reproducing behavior.

Eating. The system's stability depends on the possibility of wolves and sheep eating. After moving to a new location, agents will search for food and eat if certain conditions are met. The implementation of the eating behavior differs between wolves and sheep due to the different natures of their food, since wolves eat other agents (sheep) while sheep eat a simpler entity (grass).

Eating grass. A sheep can eat grass on its location if it is fully grown and has not been eaten by another sheep in that step. This last requirement is verified using the function `get_value_unbuffered()` that accesses the write state of the grass field. If the value obtained is *not null*, another sheep has already eaten the grass since it has been written in the write state structure. If the sheep successfully eats the grass, it sets the grass value on the field to 0 and gains some energy, as depicted in Listing 2.5.

```
if state.grass_field
.get_value_unbuffered(&self.loc).is_none() {
    if let Some(grass_val) =
    state.grass_field.get_value(&self.loc) {
        if grass_val >= FULLY_GROWN {
            state.grass_field
            .set_value_location(0, &self.loc);
            self.energy += self.gain_energy;
        }
    }
}
```

Listing 2.5: Eating behavior of sheep agents.

It is crucial to stress that using the function `get_value_unbuffered()` provides access to up-to-date information, since updates to the

data structures occur only at the end of the simulation step. This approach guarantees that if a sheep eats some grass at a location during a given step and, later in the same step, another sheep moves to the same location, it will correctly see the grass level as zero.

Eating sheep. A wolf can only eat a sheep on its location if another wolf has not already eaten the prey in that step. This requirement is checked using a dedicated data structure storing eaten sheep (see Listing 2.6). Wolf agents can modify the simulation field because their `step()` function has access to the simulation state. However, no agent can manipulate the scheduler due to the krABMaga safety policy. To overcome this limitation, we followed the same strategy used when introducing new agents: we added the array `killed_sheep` in the simulation State object to remove the eaten sheep from the scheduler.

```
if let Some(sheep) =
state.sheep_grid.get_objects(&self.loc) {
    for mut sheep in sheep {
        if state.killed_sheep.get(&sheep).is_none()
        && sheep.animal_state == LifeState::Alive{
            sheep.animal_state = LifeState::Dead;
            state.sheep_grid
            .remove_object_location(sheep, &sheep.loc);
            self.energy += self.gain_energy;
            state.killed_sheep.insert(sheep);
            break;
        }
    }
}
```

Listing 2.6: Eating behavior of wolf agents.

When a wolf eats a sheep, it sets the *LifeState* of its prey to *Dead*, removes the agent from the field, inserts it in the *killed_sheep* array, and gains some energy. Listing 2.6 shows the logic of this process.

In krABMaga, the simulation State object, which implements the trait `State`, initializes the model, defines and updates the data structures that regulate the model's logic, schedules the agents, and controls the simulation's pace. In this example, the `WsgState` object defines the *(i)* simulation fields, fields to manage the model's three entities step, *(iv)* a unique id counter to assign different identifiers to new agents, *(v)* the number of agents, i.e., sheep and wolves, and *(vi)* three data structures to support the

addition to and removal from the scheduler of agents. In more detail, the `WsgState` object initializes three different fields to manage the three entities of the model: the grass, wolf, and sheep fields. Specifically, the grass field is instantiated as a *NumberGrid2D* field, where each cell contains a numerical value representing the grass growth state, while the wolf and sheep fields are instantiated as two *DenseObjectGrid2D* structures. Using two separate fields to handle the behavior of wolves and sheep improves the simulation's performance when searching for a specific agent type; for instance, if a wolf tries to eat a sheep, the only structure queried will be the sheep field. Lines $1-12$ of Listing 2.7 list the above properties.

Initialization phase. Initialization of the simulation occurs in the function `init()` (Lines $15-20$). This function populates the three simulation fields by adding grass values, sheep, and wolves. Specifically, the method `generate_grass()` iterates over the `grass_field` cells and inserts a value representing the available grass in each cell (Lines $51-59$). The methods `generate_sheep()` (Lines $60-69$) and `generate_wolves()` (Lines $70-79$) work similarly: they create the respective agent, place it in a given location the corresponding fields, and add it to the scheduler's queue via the function `schedule_repeating()`, which notifies the scheduler that the agents need to be scheduled in all following simulation steps. Here, we emphasize the use of the `Box` structure, which wraps the newly created agent (either a wolf or a sheep) and enables krABMaga to handle multi-agent simulations.

Updating phase. The `WsgState` object updates all fields at each simulation step via the mandatory `update()` method. In particular, this method also implements the grass-growing process (Lines $22-29$), which increases the grass values of all cells in the `grass_field` via the function `apply_to _all_values`. The parameter `GridOption::READWRITE` ensures that the existing information is not overwritten. In more detail, this option allows us to check the WriteState structure before increasing the grass values since a sheep agent could have already written that field's cell (i.e., eating all grass). In Lines $30-33$, all simulation fields are updated.

In this use case, the `State` object also implements the `after_step` method, which defines the logic governing the agents' eating and reproducing behavior (Lines $35-50$). After each step, the

`WsgState` object iterates through the `new_sheep` and `new_wolves` structures to add agents in the scheduler using the method `schedule_repeating()` while it iterates over the array `killed_sheep` to remove dead agents from the scheduling process via the method `dequeue()`.

```
pub struct WsgState {
    pub wolves_grid: DenseGrid2D<Wolf>,
    pub sheep_grid: DenseGrid2D<Sheep>,
    pub grass_field: DenseNumberGrid2D<u16>,
    pub dim: (i32, i32),
    pub step: u64,
    pub next_id: u32,
    pub initial_animals: (u32, u32),
    pub new_sheep: Vec<Sheep>,
    pub new_wolves: Vec<Wolf>,
    pub killed_sheep: HashSet<Sheep>,
}
[...]
impl State for WsgState {
    fn init(&mut self, schedule: &mut Schedule) {
        [...]
        generate_grass(self);
        generate_wolves(self, schedule);
        generate_sheep(self, schedule);
    }
    fn update(&mut self, step: u64) {
        self.grass_field.apply_to_all_values(
            |grass| {
                [...]
                growth + 1
                [...]
            },
            GridOption::READWRITE,
        );}
        self.grass_field.lazy_update();
        self.sheep_grid.lazy_update();
        self.wolves_grid.lazy_update();
        self.step = step;
    }
    fn after_step(&mut self, schedule: &mut Schedule) {
        for sheep in self.new_sheep.iter() {
            schedule.schedule_repeating(
                Box::new(*sheep),schedule.time+1.0, 0);
        }
        for wolf in self.new_wolves.iter() {
            schedule.schedule_repeating(
                Box::new(*wolf),schedule.time+1.0, 0);
        }
        for sheep in self.killed_sheep.iter() {
            schedule.dequeue(
                Box::new(*sheep), sheep.id);
        }
        self.killed_sheep.clear();
    }
}
fn generate_grass(state: &mut WsgState) {
```

```
52      (0..state.dim.1).into_iter().for_each(|x| {
53          (0..state.dim.0).into_iter().for_each(|y| {
54              [...]
55              state.grass_field
56              .set_value_location(
57              grass_init_value, &Int2D { x, y });
58      })});
59  }
60  fn generate_sheep(state: &mut WsgState,
61      schedule: &mut Schedule) {
62      [...]
63      let sheep = Sheep::new([...]);
64          state.sheep_grid
65          .set_object_location(sheep, &loc);
66          schedule.schedule_repeating(
67              Box::new(sheep), 0., 0);
68      }
69  }
70  fn generate_wolves(state: &mut WsgState,
71  schedule: &mut Schedule) {
72          [...]
73          let wolf = Wolf::new([...]);
74          state.wolves_grid
75          .set_object_location(wolf, &loc);
76          schedule.schedule_repeating(
77              Box::new(wolf), 0., 1);
78      }
79  }
```

Listing 2.7: WSG Simulation State.

At this point, we have defined all elements of the WSG model. Now, let's see how to run it and analyze its execution with the TUI.

Running the simulation. The WSG model includes several parameters influencing the system's evolution and stability, such as the number of sheep and wolves, the cost of each agent's step, the energy gained from food, and the grass growth rate. Listing 2.8 shows the `main.rs` file, which defines these parameters (Lines $1-7$). Here, the function `main()` instantiates the simulation State with the required parameters and runs the macro `simulate!` to launch the simulation.

```
1       pub const ENERGY_CONSUME: f64 = 1.0;
2       pub const FULL_GROWN: u16 = 20;
3       pub const GAIN_ENERGY_SHEEP: f64 = 4.0;
4       pub const GAIN_ENERGY_WOLF: f64 = 20.0;
5       pub const SHEEP_REPR: f64 = 0.2;
6       pub const WOLF_REPR: f64 = 0.1;
7       pub const MOMENTUM_PROBABILITY: f64 = 0.8;
8       fn main() {
9           let step = 200;
10          let dim: (i32, i32) = (50, 50);
11          let initial_animals: (u32, u32) =
12              ((200. * 0.6) as u32,
```

```
13              (200. * 0.4) as u32);
14          let state = WsgState::new(dim, initial_animals);
15          let _ = simulate!(state, step, 10);
16      }
```

Listing 2.8: main.rs file.

Analyzing the simulation. The krABMaga GUI terminal helps us investigate the stability of the ecosystem predator/prey by creating dynamic plots of the simulation status. These plots are managed within the `State` object and, hence, have access to all simulation data (see Listing 2.9). The structure of each plot is defined within the function `init()` via the macro `add_plot!()`, which specifies the plot's title and labels (Lines $3-14$). Then, the plot is populated using the macro `plot!()` called within the function `after_step()`, which adds a data point after each simulation step (Lines $20-49$). The code listed in Listing 2.9 results in two plots: the first shows the trend of the wolf and sheep population (see Figure 2.5a), while the second tracks newly born wolf and sheep agents, as well as the number of sheeps killed by wolves (see Figure 2.5b).

After having the WSG simulation up and running, we can use the Visualization component of krABMaga to visually monitor the behavior of the developed model. The trait `VisualizationState` manages the entire visualization model similar to the trait `State` in the sense that we need to initialize (function `on_init()`) and update (function `update()`) the visualization of the grass field and the agents. The function `get_agent_render()` fetches the data structures defined in the model (`grass_field`, `wolves_grid`, `sheep_grid`).

In this specific use case, wolf and sheep agents move in the space, while the grass is a static environmental element. To render the agents' movement, we let the `Wolf` and `Sheep` objects implement the trait `AgentRender`. Then, we defined how the engine must draw the agent at each step, specifying its position, orientation, and scale in the function `update()`. The only difference between sheep and wolf agents resides in their representing sprites, specified in the `sprite()` function. Eaten sheep disappear from the field. In the method `before_render()`, run before each simulation step, we inform the visualization to render the newly born agents. To graphically represent the growing grass, we implemented the trait `BatchRender` in the field `DenseNumberGrid2D`. In more detail, we used the method `get_pixel()` to specify the

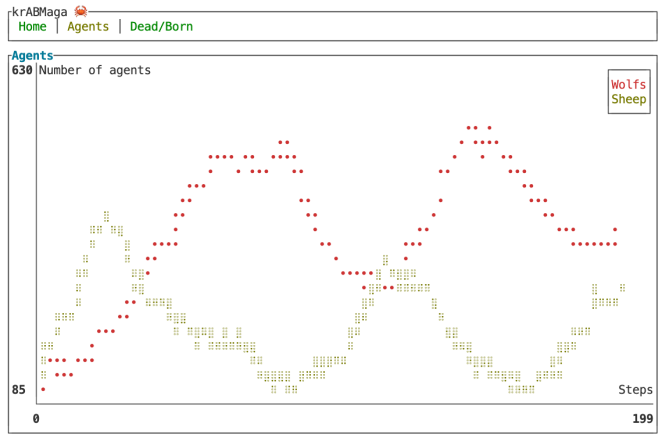
color of each cell based on the amount of grass contained (the greener the cell, the higher the grass available).

To actually use the visualization component, we need to define a `Visualization` object within the function `main()`, which allows us to set up the graphical properties of the visualization and run it.

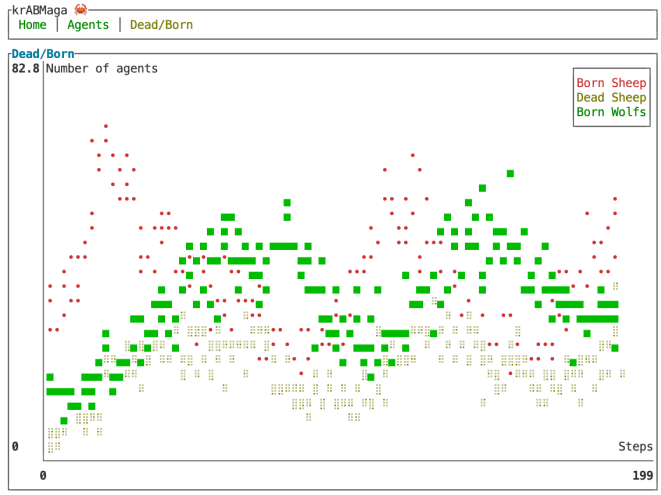
A snapshot of the visualization of the WSG model is depicted in Figure2.6. The complete code is available on the krABMaga repository[15].

```
fn init(&mut self, schedule: &mut Schedule) {
    [...]
    addplot!(
        String::from("Agents"),
        String::from("Steps"),
        String::from("Number_of_agents"),
        true
    );
    addplot!(
        String::from("Dead/Born"),
        String::from("Steps"),
        String::from("Number_of_agents"),
        true
    );
}
[...]
fn after_step(&mut self, schedule: &mut Schedule) {
    let agents = schedule.get_all_events();
    [...]
    plot!(
        String::from("Agents"),
        String::from("Wolfs"),
        schedule.step as f64,
        num_wolves as f64
    );
    plot!(
        String::from("Agents"),
        String::from("Sheep"),
        schedule.step as f64,
        num_sheep as f64
    );
    plot!(
        String::from("Dead/Born"),
        String::from("Dead_Sheep"),
        schedule.step as f64,
        self.killed_sheep.len() as f64
    );
    plot!(
        String::from("Dead/Born"),
        String::from("Born_Wolfs"),
        schedule.step as f64,
        self.new_wolves.len() as f64
```

15 WSG model repository - `https://github.com/krABMaga/examples/tree/main/wolfsheepgrass`



(a) **Wolf and Sheep Populations.**
Plot describing the trend of the wolf and sheep populations.



(b) **Births and Deaths.**
Plot depicting the trend of newly born wolves and sheep, as well as killed sheep.

Figure 2.5: Simulation results of the Wolf–Sheep model showing (a) population trends and (b) birth and death dynamics.

```
43      );
44      plot!(
45          String::from("Dead/Born"),
46          String::from("Born_Sheep"),
47          schedule.step as f64,
48          self.new_sheep.len() as f64
49      );
```

```
50 }
```

Listing 2.9: Setting the krABMaga TUI interface.

Figure 2.6: The GUI interface for the Wolf, Sheep & Grass simulation.

2.3.3 *Performance evaluation*

This section presents a two-fold performance evaluation of krABMaga: first, we investigated the library's efficiency in running different ABM simulations against the most adopted ABM tools, and then we evaluated the scalability potential of the model optimization module.

In this experiment, we are interested in showing the performance of krABMaga in running ABM models and comparing it with the most representative ABM tools, namely Agents.jl, MASON, Mesa, NetLogo and Repast.

Models. As a benchmark, we employed the following ABM, each with its peculiarities regarding data structures, agents' behaviors, and environment types.

- *Flockers*. Developed by Craig Reynolds, this is one of the most famous ABM that simulate a flock's flying behavior. In this model, the agents move within a continuous toroidal space according to a simple set of rules.
- *Schelling*. This is a simple segregation model based on a 2D grid in which agents decide whether to move into a new cell based on the status of their neighbors.

- *Wolf, Sheep and Grass*. This multi-agent model simulates the population dynamics of predators and prey coexisting in a shared environment.
- *ForestFire*. This stochastic spreading model is realized as a cellular automaton to reproduce the fire diffusion in a forest.

To perform a fair comparison, we benchmarked only the official released model for each platform; still, some differences could exist due to the variance in the frameworks, mainly because of the different programming languages involved. The model and benchmark implementations are available on the krABMaga repository[16].

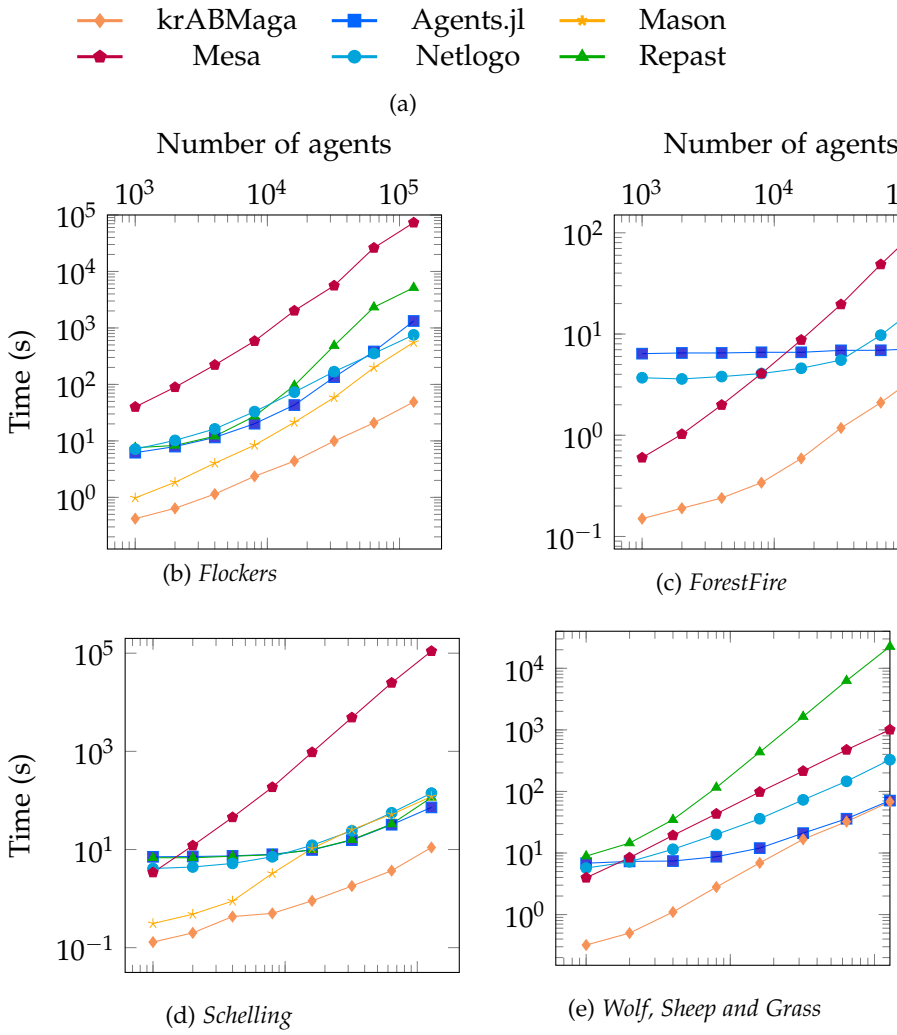
Benchmark configurations. All experiments have been performed on the same virtual machine running over a VMware Esxi hypervisor and equipped with: Ubuntu 21.10 x86_64 machine, 8 VCPU, 16GB RAM, and 256 storage (on SDD). The performance of each framework has been tested with different model configurations, starting with a field of size 100×100, 1000 agents, and 200 steps, while keeping an agent density of $\cong 10\%$, calculated as $\frac{\text{width} * \text{height}}{\text{number of agents}}$. We obtained the other configurations by doubling the number of agents and changing the field dimension to preserve the agent density:

Table 2.9: Agent count and corresponding field size configurations.

#	Agents	Field Size ($x \times y$)
1	1,000	100×100
2	2,000	141×141
3	4,000	200×200
4	8,000	282×282
5	16,000	400×400
6	32,000	565×565
7	64,000	800×800
8	128,000	1131×1131

Each experiment has been run 10 times.

16 https://github.com/krABMaga/ABM-Comparisons



(a)

(b) *Flockers*

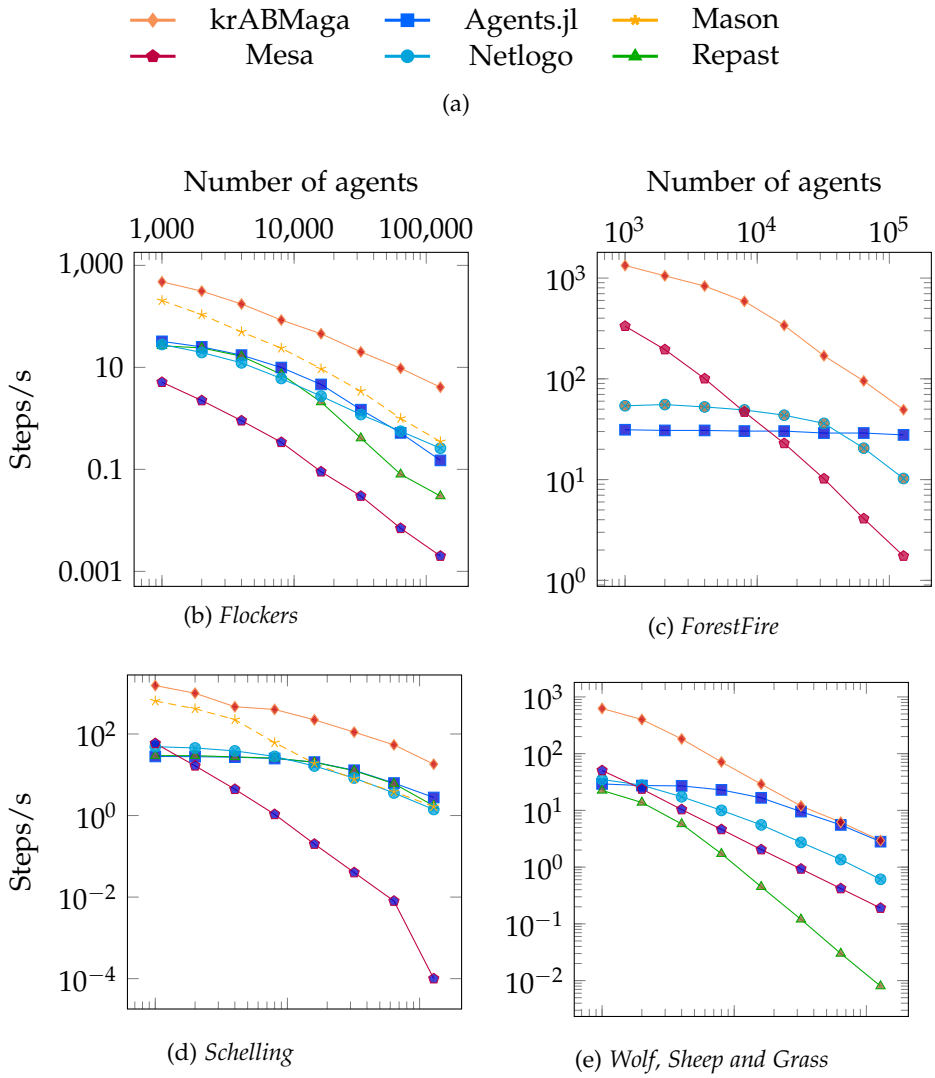
(c) *ForestFire*

(d) *Schelling*

(e) *Wolf, Sheep and Grass*

Figure 2.7: Running times of each framework on the four ABMs, varying the computational load by increasing the number of agents, while preserving the agent density.

Results. Figure 2.7 shows the results of our experiments, focusing on each framework's average running time (in seconds) when varying the model dimension while keeping the agent density fixed. Such values only keep into account the actual simulation time and exclude the initialization time required by each engine. Overall, krABMaga always performs better than the other platforms, requiring the lowest computational time regardless of the



(a)

(b) *Flockers*

(c) *ForestFire*

(d) *Schelling*

(e) *Wolf, Sheep and Grass*

Figure 2.8: Simulation steps per second required by each framework on the four ABMs, varying the computational load by increasing the number of agents while preserving the agent density.

computational load. In only a single scenario, Agents.jl reached the same performance as our platform (see Figure 2.1d) but consumed up to the 90% of the system's available memory. Figure 2.8 refers to the same benchmark, reporting the average number of simulation steps per second. In general, the heavy memory usage

of some platforms limited the maximum computational workload tested, as these platforms were unable to simulate larger systems. For that reason, we were unable to assess configurations with a higher computational complexity. It is important to stress that these results come from adequately using the different fields, data structures, and methods in each framework since every model exposes different peculiarities that must be appropriately addressed when implementing the corresponding ABM.

In this second experiment, we analyzed the scalability of krABMaga when calibrating an ABM using the model-optimization functionality in our simulation engine. The full code of this experiment is available on the official krABMaga example repository[17].

Model. As a benchmark model, we built an ABM for simulating an epidemic spreading in a population, where agents can get infected via their neighbors in a similar fashion of the work of [91]. Specifically, we implemented the Susceptible (S), Infectious (I), Recovered (R) compartmental model, which specifies how agents move from the state S to I (i.e., become infected with a probability β proportional to the number of infected neighbors), and from I to R (i.e., recover from the infection with a probability γ). The agent contact network follows the Barabási-Albert preferential attachment rule. Some details about the simulation follow.

The simulation State, in addition to the network structure, includes the definition of the parameters governing the dynamics of virus spread: *(i)* the probability that a susceptible node transitions to an infected state, and *(ii)* the probability that an infected node transitions to a resistant state. The simulation begins with the network containing only one infected node randomly placed within it and continues until either the maximum number of steps is reached, or there are no more infected nodes. Each step corresponds to a day, during which nodes may alter their status. Specifically, susceptible nodes examine the status of their neighbors, and if any neighbor is infected, there's a chance that the susceptible node becomes infected as well. Infected nodes, on the other hand, make an attempt to recover during each step, changing their status to resistant. In contrast, resistant nodes remain inactive since they are immune to infection and cannot infect others.

17 https://github.com/krABMaga/examples/tree/main/sir_ga_exploration

This sample scenario particularly fits the ABM simulations that krABMaga intends to support. Fitting an epidemic model with actual data and then performing experiments on top of the designed model usually requires *(i)* exploring the parameter space to calibrate the model's input parameters, *(ii)* verifying the model's correctness, *(iii)* validating the model's output, *(iv)* analyze the sensitivity of the model, and finally *(v)* run the intended experiments. Since each phase requires running the simulation multiple times, guaranteeing that each run is efficient and reliable is critical. For simplicity, we will only focus on the first listed phase in this use case.

Calibration data. As ground-truth data, we used the Italy's average number of daily new infections during the SARS-CoV-2 virus pandemic (moving average over seven days). In particular, we focused on a period of 45 days from December 2021 to January 2022, when the Omicron variant caused a new outbreak. This dataset is available on the official website of the Istituto Superiore di Sanità, the leading public health body in Italy[18]. Models of the like are widely described in the literature [144, 177].

Simulation setting. We simulated 10,000 agents for 51 days (three days more than the calibration time window to compute the average of new simulated infections). We normalized the infection data by the size of the Italian population (divided by 60M).

Optimization strategy. To find the input parameter configuration generating outputs that best fit real-world data, we used the evolutionary search approach offered by krABMaga. In more detail, we considered an initial population of 128 randomly generated individuals, 20 simulation repetitions for evaluating a single configuration (e.g., individual), and 2000 optimization loops (e.g., generations). Specifically, each individual comprised two real value genes, representing the infection and recovery probabilities $[\beta, \gamma]$. At each generation, the evolutionary strategy computed the new population by selecting the best 20% of individuals and generating the remaining 80% using a crossover operation (combining two random individuals from the previous generations). A mutation operator was then applied to change one of the two genes of the individuals randomly. The simulation

18 https://covid19.infn.it/iss/

output represented the average error between simulated and real data.

The overall calibration process resulted in about 5 million simulation executions. If all these simulations were carried out sequentially, it would have taken approximately 172 days (this estimate derives from summing the running time of each simulation execution). However, we completed the task in just 45 hours by leveraging distributed computing.

Results. Figure 2.9 shows the infection curves derived from simulated data based on the best configuration computed by the optimization process and the real infection curve. Table 2.10 focuses on some statistics comparing a sequential execution of the optimization process against its parallel/distributed version.

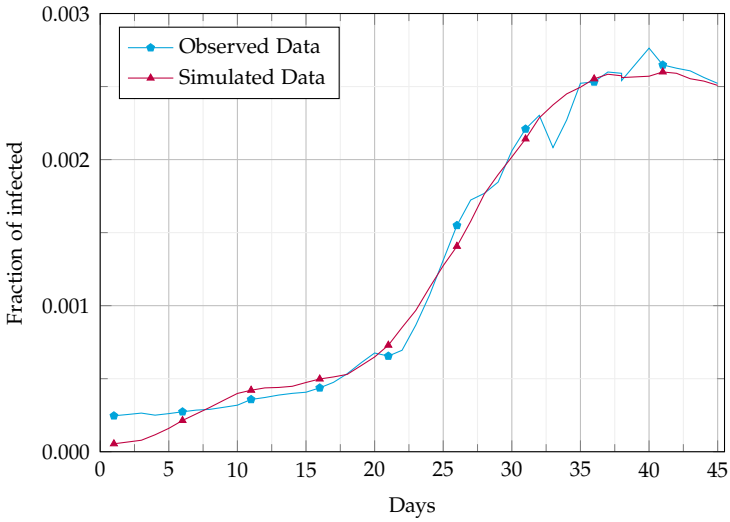


Figure 2.9: SIR calibration results.

We performed this comparison on a virtual cluster machine of 16 nodes running over Amazon AWS EC2, where each node used the *c4.2xlarge* instance type and was equipped with an Intel Xeon Processor with 8 virtual CPUs, 15 GB of memory, and a high-speed network. We analyzed the system's performance by running the optimization process for 1 hour while varying the number of VCPUs and incrementing the number of nodes. We collected the number of computed generations per hour (Gs/h), the time (seconds) per generation (s/G), and the speedup relative to sequential execution for each setting. As highlighted in the

Table 2.10: Evaluation of krABMaga's scalability on an Amazon EC2 virtual cluster machine.

Backend	#Instances	VCPUs	Gs/h	s/G	Speedup
Sequential	1	1	0	7459	1
Parallel	1	8	3.1	1149	6.5
Distributed	2	16	6.2	579	12.9
Distributed	4	32	12.1	297	25.1
Distributed	8	64	23.2	155	48
Distributed	16	128	44.3	81	92

table, the best performance is achieved in the last distributed setting using 16 nodes and 128 VCPU, which is 96 times faster than the sequential setting. It is worth noting that we report only the simulation speedup to stress krABMaga's scalability in such scenarios. Given the results of the previous experiment (which showed that krABMaga achieved the lowest simulation time when varying the workload), we did not replicate this scenario with the other simulation engines.

2.4 EXPANDING ON DISTRIBUTED ABM

Modeling real-world phenomena is incredibly challenging due to the intricate interactions among numerous interconnected elements. Understanding these systems is nearly impossible when they are viewed in isolation. Consequently, such systems are often referred to as complex systems, though a precise definition of complexity remains elusive [179]. These systems typically share features such as non-linearity, decentralized control, and feedback mechanisms. In recent years, Computational Science has leveraged data-intensive computing and analysis to study such issues. ABM offer a bottom-up approach for analyzing complex systems, allowing modelers to design the behaviors of individual agents (e.g., members of a population) and the environments in which they operate. The interactions among these agents in the simulated environment produce emergent properties and phenomena that the modeler aims to examine and understand. These three components (behavior, environment, and interactions) are

the building blocks of an ABM and have been proven to be very effective in modeling different scenarios across a vast corpora of fields [309].

ABM are also a recurring theme in High-Performance Computing (HPC), since these models are designed to mimic social interactions, the global economy, or natural phenomena. In addition, they can help predict potential outcomes involving numerous entities. However, as the number of agents grows, traditional ABM engines fail because the computational power required per execution becomes an unbearable limitation. The literature states that ABM can intersect with HPC in two distinct ways: the outer and inner loops [86]. The former usually describes optimization techniques such as model parameter sweeping [20]. The latter, which is also the focus of this paper, typically involves distributing a model across multiple computational nodes using de facto standards such as MPI (Message Passing Interface) [210] or OpenMP (OpenMP) [75].

This section presents a distributed version of krABMaga, an ABM engine written entirely in Rust. We have employed our experiences in ABM and the distributed computing field to enhance the capabilities of the krABMaga engine, assessing the possibilities and opportunities for deploying a highly optimized tool to an HPC cluster with minimal effort and achieving good results [20].

A key aspect of ABM computation is the communication between agents. Typically, each simulation agent needs to identify its neighbors to exchange information and perform its tasks. In sequential execution, this process is routine and has a moderate impact on performance. However, in distributed execution, where neighbors may be located on different machines, discovering and interacting with them can become a significant performance bottleneck. Moreover, when an agent moves between partitions or is removed from the simulation, the process must proceed seamlessly as if all agents were in a single field. Addressing these challenges is crucial when developing an efficient distributed system that can manage multiple partitions of the same field across remote machines.

To facilitate the distribution of the simulation, we began by modifying an existing field in our framework. Our first attempt takes as a reference the *Field2D* implementation in the krABMaga

repository[19], which serves as the standard field for an agent to move on a continuous 2D space.

Sequential structure. The *Field2D* uses a two-dimensional toroidal grid as a simulation field characterized by an origin point (x;y), a width, and a height. Each grid coordinate identifies a cell in which an agent can reside and interact with other agents.

Distributed structure. K-Dimensional Tree (K-D Tree) data structures are widely used in distributed computing, particularly for tasks like clustering and closest neighbors search when a scalable solution is required [74]. A K-D Tree is a tree structure in which each node has exactly two children and can be split until the desired number of nodes is reached. For each pair of children created, the parent node keeps references to them, allowing us to reach any leaf, starting from the root, in a short time [64]. We implemented a customized K-D Tree, where each child node stores references to all other nodes in the tree, called blocks. Each block represents a segment of the simulation field and includes an ID corresponding to the process rank it will be assigned to, its origin point (x, y), as well as its width and height, as shown in Figure 2.10. By maintaining references to every child node, each node gains access to comprehensive information about all nodes, facilitating efficient neighbor search and synchronization operations. Although this modification increases the physical space required, the resulting efficiency in communication justifies this trade-off. Moreover, this approach remains practical since the growth rate of machines — and thus partitions — does not scale as rapidly as the number of agents.

After the main process has computed each block, it sends a reference to each block to all other processes. These processes then receive and store the reference in local memory, allowing them to communicate using the associated ID when necessary.

When an agent changes position during the simulation and exits from a block's border, it must be sent to another block. Since every block knows the exact size and ID of all the others, when an agent moves outside its border, it can easily determine the processor responsible for it based on its position. To make this exchange possible, the object is saved in an array whenever an agent moves outside the border of its process field. At the end of every step, every process sends all the agents who have moved

19 https://github.com/krABMaga/krABMaga/blob/main/src/engine/fields/field_2d.rs

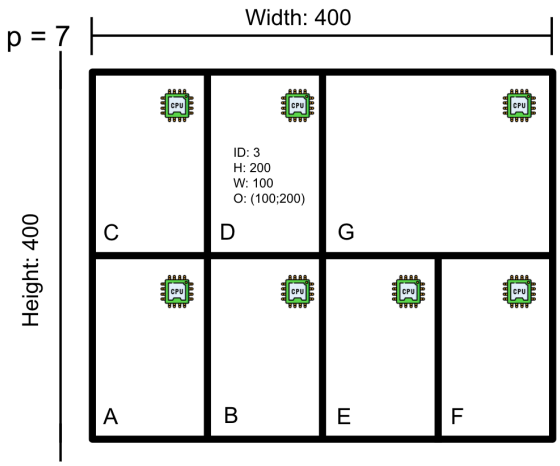


Figure 2.10: Partitioning of the field when there are 7 processors. Each block has a unique ID, the height and width of its perimeter, an origin point, and is assigned to a different processor (or machine).

off their partition to their respective neighbors and receives all agents sent by their neighbors. This phase is preceded by a message exchange phase, where each process sends the number of agents it will exchange with each neighbor and receives the value from all its neighbors. When this process completes, each process allocates the buffer with sufficient slots for the incoming agents. This communication phase is handled using MPI collective operations, such as scatter and gather, combined with non-blocking send and receive operations to efficiently exchange data across processors.

Additionally, in many simulations, an agent needs to be aware of nearby agents within a specific area of interest (AOI), usually defined by a fixed-size radius (see Figure 2.11). This can be particularly challenging in distributed simulations, as an agent's AOI may be split across multiple processes. To accomplish this task, it is essential to identify first the agents that could be neighbors of agents from other processes. This process is facilitated by Halo Regions, of which an example is illustrated in Figure 2.11. A Halo Region is a fixed-length zone located near the borders. When an agent moves into the field and enters a Halo region, a copy of the agent is placed in an array of agents, indexed by the Halo Region that contains it. At the end of every step, these agents

are sent to potential neighbors and received from each of them. This operation uses the same principles as the exchange between processors. Once all agents have been received, the process can calculate each agent's neighbors, including both local agents and received agents within the AOI.

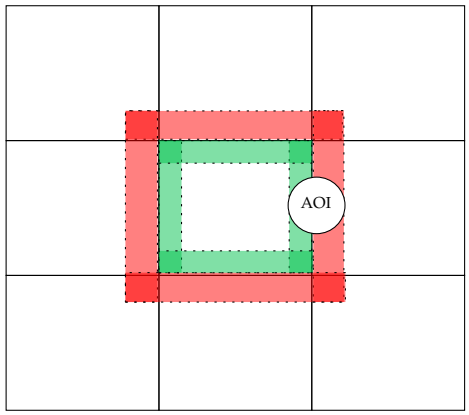


Figure 2.11: An example of a halo region. In this example, the central square represents the main actor. The red-highlighted areas are the Halo regions of its neighboring agents, while the green areas indicate the portions of the field shared with those neighbors. Agents located within these green regions are locally stored by the processor managing them. At the end of each step, processors sharing borders exchange information about the agents in the red halo regions and, if needed, transfer agents between processors.

To evaluate the proposed implementation, we have chosen Flockers as the main benchmark model [244]. Tests are based on known model parameters [24]. The number of steps per simulation is fixed at 200. The model is evaluated using the configurations for the number of agents and field dimensions listed in Table 2.11.

The code to reproduce the model is available on the GitHub repository[20]. We have built a cluster on Microsoft Azure[21] to make the benchmark as fair as possible, eliminating the noise from background activities on local machines. Each virtual ma-

20 https://github.com/krABMaga/examples/tree/main/flockers_mpi
21 https://azure.microsoft.com

Table 2.11: Size of the examples.

# Agents	Field size	Density
1Million	3162x3162	$\approx 10\%$
2Million	4472x4472	$\approx 10\%$
5Million	7071x7071	$\approx 10\%$
10Million	10000x10000	$\approx 10\%$

chine on the Azure cluster was created within a Proximity Placement Group[22] and has the following specifications:

- *Size*: standard_DS1_v2
- *Number vCPU*: 1
- *CPU family*: Intel Xeon Platinum 8370C (or similar[23]).
- *Memory*: 3.5 GiB
- *S.O.*: Ubuntu Server 22.04 LTS
- *Disk*: 8 GB Standard SSD

All numerical results obtained with an average execution time of 10 runs for each setup are displayed in Table 2.12. The trend of this model's curve is shown as the number of processors is varied in Figure 2.12. It is evident that when computation is the main task of the distributed system, every configuration performs efficiently, closely approaching the optimal curve. However, it is also apparent that the curves tend to slow down at specific thresholds and can sometimes show a speed-up exceeding 8, in the communication phase, which becomes a bottleneck when many halo regions are filled with agents and need to exchange information at each step. The simulation with 1M agents running on 8 processing nodes reveals a speed-up exceeding 8, which is an unexpected result that warrants further investigation. This anomaly could be due to near-perfect system load balancing or optimal memory alignment with the machine's cache. However,

22 https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/

23 https://learn.microsoft.com/it-it/azure/virtual-machines/dv2-dsv2-series

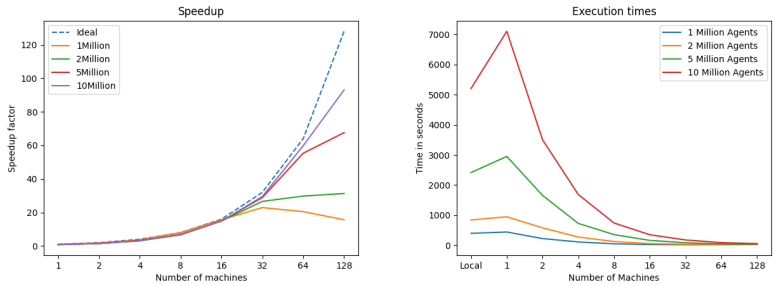


Figure 2.12: Speedup and execution times of the experiments.

Table 2.12: Numerical results of the experiments.

Virtual Machines (vCPU)									
Speedup	**Seq.**	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
1M agents	1,00	0,89	1,78	3,60	8,04	15,64	22,90	20,49	15,68
2M agents	1,00	0,89	1,45	3,06	6,68	14,88	26,59	29,78	31,29
5M agents	1,00	0,82	1,46	3,33	6,80	14,90	28,87	55,32	67,57
10M agents	1,00	0,73	1,49	3,09	7,04	14,86	29,51	59,76	93,05
Virtual Machines (vCPU)									
Exec. time (sec)	**Seq.**	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
1M agents	397,2	441,1	223,4	110,2	49,3	25,4	17,3	19,4	25,3
2M agents	841,4	946,9	578,8	274,4	125,8	56,5	31,6	28,2	26,8
5M agents	2417,1	2950,2	1656,4	726,2	355,3	162,2	83,7	43,6	35,7
10M agents	5206,9	7107,3	3501,8	1685,0	739,8	350,4	176,4	87,1	55,9

these explanations seem improbable, considering the benchmark was executed multiple times with randomized seed values. These results highlight the need for load balancing to optimize partition sizes, thereby reducing the number of agents that need to be exchanged at each step.

2.5 SCALING ABM VISUALIZATION

In this section, we present an ECS experiment within krABMaga. The ECS architecture is a software design pattern widely used in game development. Still, its flexibility and efficiency make it suitable for other computational domains like simulations, including ECS. ECS separates data (components) from logic (systems) and organizes them into entities, providing a highly modular, scal-

able framework for managing complex, dynamic systems. An example of this architecture is shown in Figure 2.13.

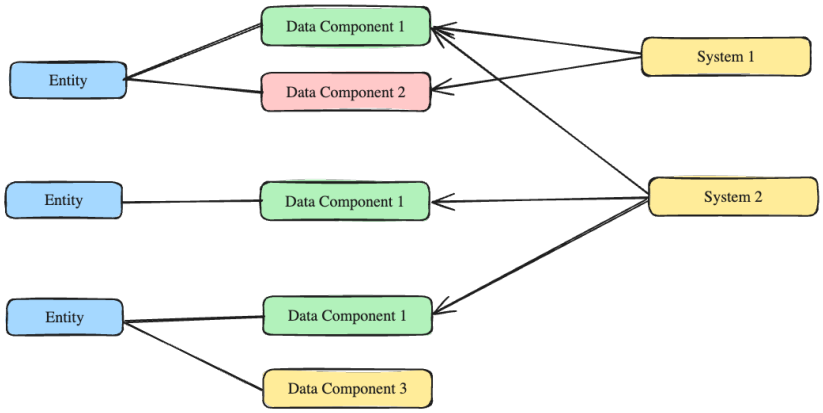


Figure 2.13: Entities are designed to be independent and point to the relevant data. The data associated with an entity is stored in components, while systems are responsible for processing and updating them.

In agent-based modeling, agents can be represented as entities, with attributes such as wealth or location defined as components. This analogy creates a natural link between ABM and the ECS model, merging the benefits of both worlds, such as performance and usability, without increasing complexity. Breaking agents into individual components in ECS enables better reusability and fine-tuned control over each simulation aspect. Systems, such as a movement system, may update agent positions based on velocity components. ECS systems can also process entities in batches, enabling high-performance execution by applying the same operations to multiple entities within a system.

In ABM, large scale refers not only to simulations with many agents but also to the challenge of managing the complexity of such simulations [231]. Parry identified several methods to address the challenges of implementing large-scale ABM. These techniques range from adjusting the level of agent complexity, investing in new hardware, restructuring the software, or rewriting the program using a parallel approach. Each method has pros and cons, but the parallel approach appears to be the most efficient. However, developing new parallel algorithms and accurately defining the communication layer can be challenging and time-consuming.

To the best of our knowledge, the ECS model has never been employed to implement an ABM engine. We have experimented with ECS to develop the visualization module, but it was not involved in managing the entire simulation. For this reason, we aim to fill the gap in the literature investigating the feasibility of applying such a model in the context of ECS. Therefore, in this paper, we demonstrate that the ECS architecture is feasible for agent-based modeling and examine its impact on the development cycle. We also show that the ECS architecture can improve performance and analyze the performance of the parallelized ECS model in agent-based simulations under high computational loads. To support these claims, we empirically evaluate the parallelization of the ECS model, focusing on scalability and performance through tests on two well-known agent-based models that pose high computational demands.

In most ABM toolkits, simulations usually follow a structured workflow divided into three phases: pre-step, step, and post-step. Each iteration corresponds to a discrete time step, with the step phase being essential and the others optional. The pre-step phase prepares the simulation by organizing agents, allocating resources, and scheduling entities. The step phase is the core of the simulation, executing the model's main logic and handling agent behaviors and interactions. In the post-step phase, the toolkit finalizes tasks by updating shared data, releasing unused resources, and possibly performing data logging or analysis. This phased structure ensures efficient computation and supports scalability for complex models with many agents.

Specifically in krABMaga, the simulation progresses in steps using double buffering, meaning that agents scheduled at step i cannot see any changes happening in the current step unless the modeler forces a read on the specific structure of the buffering. Instead, they only see the state of other agents, fields, and the simulation as they were at step $i - 1$. This mechanism's clear separation of read and write structures allows the engine to perform better on several tasks.

In sequential execution, as the paradigm suggests, each stage of the process must wait for the previous one to complete, even when it is unnecessary. Moreover, the simulation's logic, encapsulated in the step phase, can be executed independently for each agent. This naturally leads to a scenario where parallel execution is both possible and highly beneficial. However, the original ar-

chitecture of krABMaga, influenced by Rust's inherent features, makes parallel programming more difficult due to its strict concepts of ownership and borrowing [25]. These constraints make it challenging to implement parallel execution on the original architecture without significant restructuring.

By contrast, modeling an agent-based model using an ECS architecture offers a more suitable approach. The main building blocks of an ECS are described as follows:

- *Entities*: An entity is essentially a unique identifier that serves as a container for various components. Unlike objects in object-oriented programming, entities do not hold data or logic themselves; they are purely identifiers. This makes entities lightweight and flexible.

- *Components*: Components are the pieces of data that define the characteristics or state of an entity. Each component typically holds a single aspect of data, such as position, velocity, or health.

- *Systems*: Systems contain the logic that operates on entities by modifying their components. They are responsible for processing data component-by-component.

As mentioned earlier, ECS decouples data from behavior, facilitating the independent processing of agents and making the transition to parallel code much more straightforward. This decoupling enables better scalability and performance, especially in large-scale simulations, where parallel execution can significantly reduce computational time. The ECS design inherently aligns with parallelism by enabling simultaneous updates to entities without direct dependencies, making it an ideal solution for ABM simulations that require high performance.

The original version of KrABMaga leverages Bevy as its visualization module. Bevy is a straightforward, data-driven game engine that lets users leverage the ECS architecture in the Rust ecosystem. By default, Bevy supports highly parallel and cache-efficient programming. It offers various features and libraries for making queries, managing global and local resources, and implementing a lock-free parallel scheduler.

To assess our framework, we adapted two widely used examples from the ABM field: the Boids simulation [244], which mimics bird flocking behavior, and WolfSheepGrass [77], a predator-prey

model that balances the interactions among three different populations, following Lotka-Volterra equations [45]. These examples are valuable for evaluating the engine's performance with single and multiple agent types. To better describe our efforts and improvements, we will provide insights into the development of the WolfSheepGrass implementation.

First experiment. The initial implementation forcefully imposed parallelism on agent data structures using mutable queries for sheep and wolves. When a wolf detects and eats a sheep, it gains energy; the sheep must be removed, and the field updated to prevent other wolves from eating the same dead sheep. Parallel implementation proved challenging due to the overhead of mutually exclusive field updates.

This experiment showed that the focus should be on tracking sheep deaths and wolf energy gains per location. To overcome this, we implemented atomic counters tracking wolves per grid cell. The system processed sheep and grid in parallel, despawning appropriate sheep from each cell. Using wolf counts, sheep fields, and wolf locations, we calculated energy gains by computing the minimum between sheep and wolves per cell. This approach failed to improve performance because atomic operations introduced locks during execution, creating bottlenecks without resolving field update issues.

Second experiment. The constraint that agents only interact within the same location was intended to isolate potential conflicts. Wolves kill sheep and gain energy in the same grid cell. The challenge was modifying wolves' energy values during parallel execution, as Rust's mutable objects cannot be shared across threads. To address this, we used locks, ensuring only one thread could access an agent's data at a time. Instead of locking entire query objects, locks were applied directly to agents' data, enabling efficient parallel execution while satisfying Rust's memory-safety requirements.

Third experiment. The previous experiment did not fix the grid's weak performance. Placing a lock on the entire grid would cause race conditions among agents. However, each agent only creates race conditions within its specific grid bag. Since we modify only the bag corresponding to the agent's location rather than the entire grid, the solution is to place locks on individual bags. This

allows locking only the required bag, reducing thread contention and improving parallel efficiency.

Fourth experiment. Despite fixing some grid problems, agent management became a bottleneck. Spawning and despawning agents when they die or are born influenced engine performance. To optimize agent lifecycles, we replaced the spawning system with Bevy's parallel system, which creates a buffer for the *spawn_batch* function. This allows all agents to be instantiated in a single call using an iterator that provides initialization data, streamlining the process.

Final experiment. Previous attempts at the new implementation failed to simultaneously improve performance across all critical sections of the architecture. Each attempt addressed individual issues, but none provided a complete solution. The main bottlenecks were sequential field updates, agent spawning and despawning, and certain Bevy command executions. In the current implementation, which we call the cemetery system, we aim to resolve all these issues concurrently, resulting in a more comprehensive and efficient approach. The cemetery system is a memory management technique that optimizes memory reuse in simulations, especially when agents frequently die or need to respawn. In traditional approaches, when an agent dies or is removed from the simulation, the associated memory is typically deallocated and must be reallocated later when a new agent spawns. Since this allocation-deallocation cycle repeats each iteration, it introduces performance overhead, especially in large-scale simulations where many agents are created and destroyed. In contrast, the cemetery system avoids this overhead by maintaining a pool of freed but unallocated memory locations when agents die. Instead of removing the memory entirely, the system places dead agents' memory into a designated region called the cemetery. This region serves as a holding area, preserving space for later use. When new agents need to be created, the system reuses these empty cemetery spaces, assigning them to agents that need spawning. By reusing already-allocated memory, the cemetery system reduces the need for frequent memory allocations, which are expensive in both time and system resources. The cemetery system operates within a single iteration window. Agent memory need not persistance across the entire simulation, only during the current step. This approach enhances performance more granu-

larly, avoiding overhead associated with Bevy's internal functions, such as querying and reordering, which occur when managing entities.

To answer our research questions, we have tested Boids and WolfSheepGrass with the last architecture shown in the previous section (ECS with a cemetery system). The code for each attempt and benchmark is available as a Github repository [24]. As a baseline, we measured the execution times of these models using the original sequential krABMaga implementation. Starting from 32000 agents, we have doubled this number until 2048000. The field size was selected to maintain an agent density of approximately 10%. Moreover, each configuration is tested with different numbers of threads (1-20), with an average elapsed time across 5 runs. The machine is configured as follows:

- OS: Ubuntu 24.04.1 LTS x86_64
- CPU: Intel i9-14900KF @ 5.800GH
- Memory: 64 GB

Table 2.13 compares the performance of ECS-based models against their original implementations. The values in the tables reflect the execution times of the models in their sequential versions, with the ECS variant constrained to a single thread.

Table 2.13: Sequential krABMaga execution time vs the ECS implementation runned with 1 thread (time in seconds).

# agents	Field	WSG (ECS)	WSG (seq)	Boids (ECS)	Boids (seq)
32000	565x565	3,50	5,61	4,32	2,18
64000	800x800	7,64	13,73	9,57	4,69
128000	1131x1131	16,29	30,40	20,90	9,92
256000	1600x1600	33,49	61,52	47,12	22,80
512000	2262x2262	69,07	112,54	130,17	63,78
1024000	3200x3200	141,52	238,70	378,11	162,79
2048000	4525x4525	287,79	487,66	966,40	394,37

Interestingly, the multi-agent model (WSG) shows improved performance compared to its original counterpart, whereas the simpler Boids model exhibits slower execution times. This seemingly counterintuitive result can be attributed to the Boids model's

24 https://github.com/Tonaiono2/KrABMagaPersonalDev

internal structure, which does not fully leverage the advantages of the ECS framework. As a result, the model incurs additional overhead throughout computation, reducing efficiency. On the other hand, WSG appears to benefit more from ECS optimizations, even in its single-threaded form.

Table 2.14 presents the execution times of the WSG model, running with 2,048,000 agents, as the number of threads varies, while Table 2.15 describes the execution times of the Boids model. To provide deeper insights, the table includes a "step" column that highlights performance improvements during the most computationally intensive tasks.

Table 2.14: Relative speedup of WSG model on 2048000 agents (time in seconds).

# threads	Total time	Speedup	Step time	Speedup
1	287,79	1	0,52	1
2	161,09	1,78	0,28	1,84
4	98,72	2,91	0,15	3,42
8	64,55	4,45	0,08	5,97
16	56,43	5,09	0,07	6,85
20	54,54	5,27	0,07	7,02

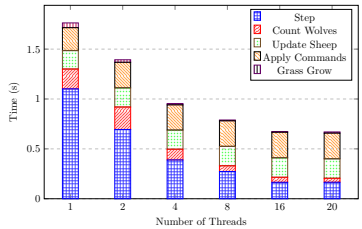
Table 2.15: Relative speedup of Boids model on 2048000 agents (time in seconds).

# threads	Total time	Speedup	Step time	Speedup
1	966,40	1	4,51	1
2	486,25	1,99	2,27	1,98
4	249,81	3,86	1,16	3,87
8	142,76	6,76	0,66	6,78
16	96,24	10,04	0,44	10,24
20	84,89	11,38	0,38	11,70

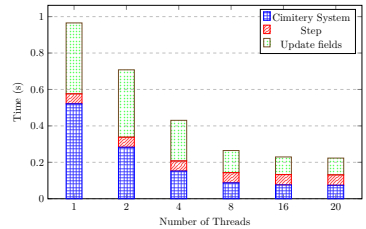
As discussed in the methodology section, the primary bottleneck in the simulation arises from updating the system's internal structure. Up to 8 threads, there is a noticeable and consistent increase in speedup, reflecting efficient parallelization. However,

beyond this point, the performance plateaus. This plateau is due to the communication and synchronization overhead between threads, which becomes more pronounced as the number of threads increases. At this stage, the benefits of additional parallelism are offset by the cost of managing inter-thread dependencies and coordination.

Figure 2.14a illustrates a snapshot of the initial implementation of the ECS methodology, as described in the previous section. The computation is divided into distinct phases, each occupying separate slots within a single iteration of the model's execution. Notably, some of these phases overlap, requiring execution in sequence and extending the time required to complete each iteration. This segmentation reduces the potential for parallelism and introduces inefficiencies, as some tasks must wait for others to complete before proceeding.



(a) This snapshot captures a single iteration of the first implementation of ECS architecture.



(b) This snapshot shows a single iteration from the final implementation of the ECS architecture.

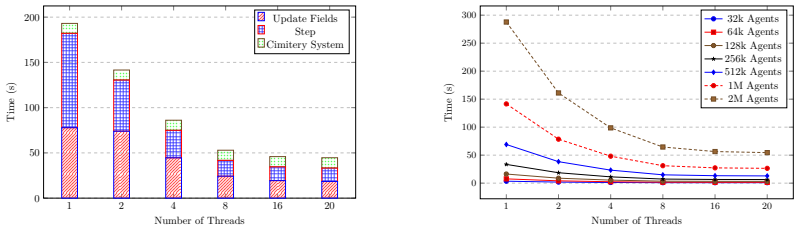
Figure 2.14: Performance of the WSG model along different implementations on 2,048,000 agents varying the number of threads.

Figure 2.14b shows the execution time of the WSG model in its final iteration, incorporating the cemetery system. In this implementation, entities are allocated at the start of the simulation. When an agent dies, instead of being deallocated, it leaves an empty space in the "cemetery" that can be reused for future agents in subsequent steps. This approach eliminates the overhead of dynamic memory deallocation and reallocation during runtime. As a result, the previously time-consuming "apply commands" phase, which handled entity management, is now optimized and no longer consumes significant processing time. This improvement allows the system to more effectively parallelize operations, enabling simultaneous execution of different

components of the simulation. The cemetery system thus contributes to a more efficient overall workflow, reducing bottlenecks and improving parallel performance.

Figure 2.15b shows the total execution time of the final implementation of the WSG model. The cemetery system is stable, reaching a constant value across different thread counts, indicating it no longer significantly influences overall performance. The computation time per step scales well with the number of threads, but performance plateaus after 8 threads. This suggests that the simulation benefits from parallelization up to a point, beyond which the overhead of thread synchronization and communication becomes a bottleneck.

While the update fields still occupy a portion of the total execution time, the introduction of parallelism allows various field management tasks to be handled concurrently. As a result, the field's overall impact on the simulation's total runtime is reduced.



(a) Stacked values of the components in last experiment.

(b) Total execution times.

Figure 2.15: Performance of the WSG model across different implementations on 2,048,000 agents varying the number of threads.

Additionally, Figure 2.15a highlights the performance of the WSG model across various simulation sizes and thread counts. Notably, the performance plateau discussed earlier is not affected by field size or the number of agents. Instead, it is a direct consequence of the ECS architecture's inherent limitations. This suggests that while the ECS framework offers certain advantages, its current implementation reaches a point at which additional parallelism no longer yields significant performance improvements. Given this observation, future research should focus on exploring alternative methods for managing the simulation's different phases or structures. By optimizing how the simulation's

tasks are organized and executed, it may be possible to overcome the current performance bottleneck and achieve further efficiency gains.

However, factors such as load balancing, synchronization costs, and the nature of agent interactions influence the level of parallel efficiency. In tightly coupled simulations, where agent interactions are frequent and complex, the overhead from synchronization offsets the gains from parallelization. This is highlighted by the simulation's performance plateau, as shown in the previous chapter.

The framework was designed as a versatile tool to support a wide range of applications. However, this broad compatibility may make specific components less efficient due to the need to support diverse use cases. To optimize performance and adequately evaluate the benefits of this approach, additional workloads should be tested, and code profiling should be conducted to identify issues such as caching and to evaluate the impact of tightly vs. loosely coupled agents.

It is crucial to recognize that redesigning an engine for parallel architecture presents significant challenges, and the associated development costs must be carefully weighed when determining the feasibility of this solution.

2.6 CONCLUSION

ABM embodies a powerful approach to unraveling the complexity governing real-world systems. Over the last decade, the need for more complex, computationally intensive models has given rise to numerous frameworks and tools for running ABM simulations. The development of robust ABM frameworks remains an evolving challenge as researchers continually seek to balance usability, scalability, and computational performance. Our exploration with krABMaga reaffirms that next-generation ABM tools must provide more than simply intuitive interfaces or raw computational speed—they must also deliver rigorous reliability and consistent scalability to support a broad spectrum of scientific purposes, from small explorations to computationally intensive scenarios.

krABMaga distinguishes itself by leveraging Safe Rust to ensure both stability and efficiency, enabling researchers to run long-running simulations with confidence in the integrity of their

results. The comprehensive architecture, as detailed in this chapter, provides a foundation for advanced model experimentation and high-performance optimization. At the same time, empirical benchmarks validate that krABMaga matches or surpasses established open-source alternatives in both speed and scalability.

Looking ahead, our focus will be two-pronged: expanding krABMaga's modeling expressiveness—such as enabling 3D environments, facilitating fine-grained parallel simulations, and supporting the complex integration of GIS data—and deepening the toolset for every phase of the ABM workflow. By streamlining processes such as calibration and ensuring transparent, reproducible results, we intend for krABMaga to become not just a technical platform but a comprehensive tool for researchers developing agent-based models across disciplines.

3

A TOP-DOWN PERSPECTIVE ON COMPLEX SYSTEM ANALYSIS

Hypergraphs provide a natural abstraction of a wide spectrum of systems characterized by group (or higher-order) relations among their components. These mathematical structures extend the concept of graphs by allowing a (hyper)edge to connect any number of nodes [61]. This ability to model complex, non-linear interactions among multiple elements makes hypergraphs an invaluable tool for analyzing systems in which their constituents interact in groups of varying sizes [18, 193]. In practice, this consideration translates into using hyperedges to model (possibly indecomposable) group interactions that cannot be adequately described by simple pairwise connections, as happens with conventional graphs [38].

Formally, a hypergraph is an ordered pair $\mathcal{H} = (V, E)$, where V is the set of vertices, and E is the set of hyperedges (or hyperlinks). Hypergraphs can be described by an incidence matrix $\mathbf{H} \in \{0,1\}^{|V|\times|E|}$, where each element $H_{v,e}$ indicates whether the vertex v belongs to the hyperedge e. Figure 3.1 shows an example of a hypergraph (right) and its clique expansion (left). Formally, the *clique expansion* of a hypergraph $\mathcal{H} = (V, E)$ is an undirected graph $\mathcal{G} = (V, \mathcal{E})$ where:

$$\mathcal{E} = \Big\{ \{u, v\} \mid u, v \in V, u \neq v \wedge \exists e \in E \text{ s.t. } \{u, v\} \subseteq e \Big\}.$$

A hypergraph can also be transformed into a graph via its line graph representation. Essentially, each hyperedge is transformed into a node, and an edge is added between two nodes if the corresponding hyperedges intersect in the original hypergraph. Formally, the *line graph* of a hypergraph $\mathcal{H} = (V, E)$ is the graph $L(\mathcal{H}) = (V', E')$ such that $V' \equiv E$, and $\{i, j\} \in E', i \neq j \iff e_i \cap e_j \neq \emptyset$.

The powerful expressiveness of hypergraphs has a few drawbacks: the complexity of these data structures and the need for appropriate tools and algorithms for their study. For this reason, hypergraphs have been little used in literature in favor of their

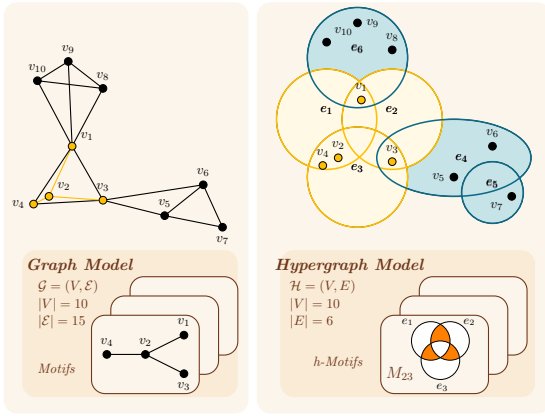


Figure 3.1: Graph *vs* hypergraph models.

graph counterpart. Recently, the trend of using hypergraphs to graph representations is drifting, thanks to an increasing number of systematic studies demonstrating how the transformation of a hypergraph to a classical graph either leads to an inevitable loss of information or creates a large number of extra nodes/edges that increase space and time requirements in downstream graph analytic tasks [216, 311].

3.1 ANALYSIS OF THE STATE-OF-THE-ART

All graph-related problems and corresponding challenges persist in the hypergraph context, often with amplified computational costs due to the presence of high-order interactions [18]. This complexity is particularly pronounced when investigating local structural patterns, known as motifs. In traditional graphs, network motifs consist of small, recurrently connected sets of nodes that offer common structural functionalities [211] (Figure 3.1, left) and serve as fundamental building blocks across the most disparate domains, from social science to biology to a plethora of other fields [233, 306, 313, 314]. However, extending the notion of motifs from graphs to hypergraphs is not straightforward as there can be an endless number of interaction patterns among a fixed number of nodes due to the flexibility in the size of hyperedges [180]. The concept of high-order motifs (h-motifs) has been recently introduced by Lee et al. [180] and describes the connectivity patterns of three or more connected hyperedges (Fig-

ure 3.1, right). As graph motifs, h-motifs have been shown to be critical for identifying unique local structural patterns across different domains [181, 193]. For instance, in the biological domain, Gaudelet et al. [126] leverage hypergraphs to represent protein complex networks and introduce hypergraphlets to capture local node wiring patterns. Their empirical analysis reveals a significant correlation between recurring hypergraphlet patterns in molecular hypernetworks and specific biological functions. In the chemical field, Andersen et al. [13] propose modeling pathways in chemical reaction networks using directed multi-hypergraphs, with vertices corresponding to molecules and hyperedges corresponding to reactions. They embed the concept of chemical transformation motifs in this context to model coherent subsets of chemical reactions induced by possible hyperpaths in the hypernetwork. Looking at the social domain, Arregui-García et al. [32] propose the concept of hyperegocentric temporal motifs to identify significant patterns in temporal offline social networs. In their work, the authors show that these temporal high-order motifs play a statistical role across all datasets analyzed, but exhibit greater variability and diversity across domains. Hence, the authors empirically demonstrate the relevance of higher-order structures in shaping social and network dynamics. Further exploring social interactions, Juul et al. [160] develop the concept of m-patterns in hypergraphs to formalize and analyze collaborator relationships and group dynamics. Their study reveals that certain m-patterns are significantly over- or underrepresented across various datasets, highlighting how prior collaborations influence the formation of new teams.

Existing literature on the topic has proposed exact and approximated algorithms for mining motifs in hypergraphs [180, 182–184, 192, 219], and has exploited these substructures to improve downstream analytical tasks, such as link prediction [180, 183, 208], and node classification [183]. As for graphs, several works address the task of (hyper)link prediction [78, 202], i.e., predicting the formation of connections between nodes. However, there is a noticeable gap in research dedicated explicitly to predicting the appearance of given motifs [47], which involves understanding whether specific complex sub-structures, composed of multiple interconnected nodes and (hyper)edges, may emerge in the data. Providing methods to advance our understanding of the formation and evolution of crucial local patterns can be beneficial in

fields such as chemistry and computational biology, where h-motif prediction can be used to identify probable missing clusters of interactions and, hence, limit the number of expensive experiments to find missing meaningful connections [38, 47, 78]. Online Social Networks (OSN) have become a central part of modern digital life, transforming how individuals communicate, share information, and build relationships. Over the past two decades, platforms such as Facebook or Twitter have reshaped social interaction by enabling unprecedented connectivity across geographic and cultural boundaries. As their influence expands, OSN have attracted extensive scholarly attention across fields such as communication studies, sociology, psychology, and computer science. This literature review examines key themes and findings in the study of online social networks, highlighting their benefits and challenges and identifying gaps warranting further investigation. More specifically, QA websites have become a valuable source of shared knowledge related to a wide variety of fields [258].

Stack Overflow has been extensively studied in the literature, including tasks such as predicting question tags, detecting duplicate questions, analyzing the use of code snippets, and examining user behavior and community evolution [5, 207]. In particular, from this perspective, some relevant works focus on characterizing the social evolution of user communities over a significant temporal period [49, 212] and on detecting patterns of interest change [123]. Specifically, Blanco et al. [49] explore the social evolution experienced by the Java developer community over 10 years, assessing the evolution of topics, user reputation, and cross-references of internal contents and external sources to improve the user experience by, for instance, reducing the number of questions with no answers. Moutidis et al. [212] examine user evolution from a broader perspective by considering all questions, answers, votes, and tags from Stack Overflow between 2008 and 2020 to evaluate trends in domain-related technologies and user persistence within the associated communities. Similarly, Fu et al. [123] tackle the problem of detecting changes in patterns of interest, but from a microscopic perspective, by quantifying how individuals shift their focus of interest over time based on question tags.

Reddit has become a golden pot for researchers thanks to the openness, richness, and quality of its data [206]. In the context of developer communities, this platform has been analyzed to

gain insights into how users contribute to global knowledge and learn from others regarding computer science-specific issues. For instance, Waller et al. [296] explore users' activity diversity, characterizing whether they are specialists in some topics or contribute to broader knowledge. Lu et al. [194] focus on communications of game developers about their game development experiences, processes, and practice. Both Aniche et al. [15] and Zang et al. [316] compare Reddit with HackerNews [137] via developer surveys. While the former focuses on understanding what motivates developers from both communities to contribute and what kind of content is shared, the latter targets identifying barriers to the adoption of Rust. More recently, Liu et al. [191] explored how Reddit has been utilized by online learners to learn programming-related topics.

To our knowledge, few works currently compare developer communities across similar structured social QA sites. An example is the work by Sengupta [260], which explores differences in discourse patterns on Stack Overflow and Reddit (r/AskProgramming), specifically focusing on the learning practices these collectives support and scaffold. Wu et al. [308] examine similarities and discrepancies of the Reddit, Stack Overflow, and Stack Exchange cybersecurity communities with the purpose of better understanding how and why users choose one platform over the others when their needs are identical. Another example is the work of Luo et al. [197], which analyzes practitioners' opinions about low-code development on Stack Overflow and Reddit.

3.2 DATA COLLECTION AND DISTRIBUTION

The progression of research hinges critically on the quality and accessibility of data. Yet, the landscape of data collection is fraught with challenges, notably the scarcity and openness of comprehensive datasets. While science is placing greater value on making information freely accessible and on working together, the reality remains that many valuable datasets are fragmented, proprietary, or hidden behind institutional barriers. This lack of freely and openly available data limits reproducibility and transparency. As we examine the centrality of data in contemporary research, this section highlights the persistent obstacles to data collection and distribution, specifically around hypergraphs, and empha-

sizes the essential role of open, freely accessible repositories in fostering a truly collaborative scientific ecosystem.

OSN analysis relies heavily on the quality and completeness of relational data, yet real-world networks are rarely observed in full. Incompleteness, caused by missing actors, unreported ties, sampling limitations, privacy restrictions, or platform-specific constraints, can substantially distort the observed network structure and, in turn, the conclusions drawn from it. Data availability is also uneven: some networks benefit from rich digital traces, while others depend on self-reports or fragmentary observations, introducing additional biases. For traditional graph-based representations, these limitations can often be addressed with established tools and relatively few additional assumptions. Graphs typically require only pairwise relations, and despite missing or noisy edges, analysts have long developed methods to clean or approximate the underlying structure. As a result, the ecosystem of graph data formats and analysis libraries is mature, standardized, and widely interoperable. However, hypergraphs, where relationships may involve more than two entities, demand richer, more expressive data and more careful handling when data are incomplete. In this context, the Hypergraph Interchange Format (HIF) plays a crucial role [84]. By collecting hypergraph data from diverse sources and making it openly available in a standardized, interoperable format, HIF helps reduce inconsistencies, improve comparability across datasets, and mitigate the effects of data gaps. This shared infrastructure supports more reliable analyses and fosters cumulative research by ensuring that complex relational structures are documented and accessible to all.

In contrast to graphs, the literature on hypergraphs is in its adolescence thanks to a recent rise of systematic studies demonstrating how the transformation of a hypergraph to a classical graph either leads to an unavoidable loss of information or creates a large number of extra nodes/edges that increase space and time requirements in downstream graph analytic tasks [18]. This recent attention toward hypergraphs is reflected in the absence of one or more established repositories that offer readily available benchmark hypergraphs. The lack of standardized benchmark datasets may pose a challenge for researchers aiming to assess and compare claims, hypotheses, and algorithms specifically designed for hypergraphs in different application scenarios. Addressing this gap could contribute to the growth of hypergraph

research and foster a more robust understanding of their applications and implications across diverse fields, while favoring open science principles [89].

To this end, we introduce *HypergraphRepository*, the first open-source, community-driven, and interactive hypergraph collection. Our project stands on two fundamental pillars. First, its primary objective is to establish a dedicated space, crafted by and for the community, that allows users to contribute by uploading their own datasets. This collaborative approach ensures that the repository evolves organically, reflecting the diverse needs and interests of the hypergraph research community. Second, the repository's interactive design aims to facilitate the exploration and comparison of a broad spectrum of datasets. HypergraphRepository is open-source and available at `hypergraphrepository.di.unisa.it`. We provide an open-source, community-driven data repository where users can contribute by sharing their own data and insight through a system of pull requests handled via git. The hypergraph database is available at `github.com/HypergraphRepository/datasets`.

Graph repositories. The most famous graph repositories are probably SNAP [187] and Network Repository [249]. The SNAP collection, managed by the Stanford Graph Learning Research Group, comprises more than 200 real-world network datasets from diverse domains and types. These datasets are provided in multiple formats, ensuring their compatibility with a broad spectrum of graph analysis tools. Launched in July 2009, the SNAP website primarily hosts datasets gathered to serve the specific research objectives of the research group. NetworkRepository is a web-based data repository for interactively exploring, visualizing, and comparing a large number of networks. This platform also integrates social and collaborative features, enabling users to engage in discussions, share observations, and exchange visualizations related to each network. Currently, the repository contains more than 5,000 networks sourced from 19 diverse categories (e.g., social and biological). These collections cover various network types, such as bipartite and time-series networks, and span domains like social sciences, physics, and bioinformatics. Both repositories include their own programmatic libraries or tools for graph analysis, with possible support for machine learning frameworks.

The KONECT Project [175, 176] is another platform where users can download network datasets and visualize their statis-

tics online. The project is run by Jérôme Kunegis, and the entire source code is available under a free software license. The repository also includes a network analysis toolbox for GNU Octave, a network extraction library, and code to generate all statistics and plots shown on the website. Currently, the KONECT project comprises 1,326 network datasets across 24 categories.

Two websites where it is possible to download large graph datasets are the website of the `Laboratory for Web Algorithmics` [50, 51] and Amazon's project `GraphChallenge` [8]. The former offers a diverse array of graph categories, accompanied by comprehensive statistics, including plots of degree distributions, the size of the giant component, average and median distances, and the harmonic diameter. The latter refers to the Graph Challenge data sets available to the community at no cost as part of the AWS Public Data Sets program.

The Open Graph Benchmark (OGB) [147] and the Illinois Graph Benchmark (IGB) [167] are two recent yet mature projects designed to advance graph machine learning, aiming to facilitate the development of novel models and comparisons with existing approaches. The OGB repository covers graphs of diverse scales from various domains, including biological networks, molecular graphs, academic networks, and knowledge graphs. This project fully automates dataset processing, providing graph objects that are fully compatible with PyTorch Geometric [113] and DGL [321], as well as standardized dataset splits and evaluators that enable easy, reliable comparisons of different models in a unified manner. The IGB project specifically focuses on training and evaluating graph neural network models. In particular, it provides highly labeled graphs, including both homogeneous and heterogeneous large-scale real-world citation graphs, with more than 40% of their nodes labeled. IGB is open-sourced and, like OGB, supports DGL and Pytorch Geometric frameworks.

Hypergraph repositories. As of our current knowledge, there is a lack of a comparable repository dedicated to hypergraphs. The limited online resources typically consist of personal web pages where authors provide downloadable datasets used in their own work. Although this practice contributes to the reproducibility of their research, it falls short of establishing a universal platform. Moreover, this approach does not guarantee increased dataset availability, as it relies on individual efforts. The most notable example is Professor Austin R. Benson's personal website [43].

Among other datasets, users can access 17 temporal hypergraphs, 11 node-labeled hypergraphs, 10 edge-labeled hypergraphs, and 2 hypergraphs labeled with a core-fringe structure. A short list of resources can also be found in the following pages [168, 189, 201, 229].

HypergraphRepository is the first open-source, community-driven web platform for interactively exploring, comparing, and downloading hypergraph data. The ultimate objective of our platform is to assist users throughout the workflow of hypergraph-related tasks by serving as a centralized hub that enables them to access various hypergraph types (e.g., temporal, directed, weighted interactions) from diverse application domains (e.g., social, transportation, biological networks). This approach is designed to streamline the initial stages of hypergraph processing, providing users with readily available hypergraphs tailored for different tasks, such as community detection, hyperedge prediction, or node classification/ranking.

The dual nature of being both open-source and community-driven opens up a two-fold avenue for user contribution. First, users can actively participate in dataset creation, modification of existing datasets, and take on roles as community reviewers. Second, users can contribute directly to the platform by discussing potential updates to the repository. This collaborative process empowers users to request and add new features to the website, including the integration of novel interactive plots. The entire workflow, from dataset management to platform enhancements, operates seamlessly through the GitHub platform, fostering a transparent and collaborative environment. Specifically, users can use GitHub's pull request system to propose modifications and improvements. This strategy not only promotes accessibility but also enhances the overall quality and integrity of the datasets. By implementing a stringent review mechanism as part of the submission process, the platform ensures that the available hypergraphs meet high standards set by the community, thereby enriching the data's exploitable for current research challenges. This aspect, often overlooked by similar platforms, sets us apart by strongly emphasizing data integrity and quality assurance approved by the relevant communities.

This approach not only promotes accessibility but also elevates the overall quality and integrity of the datasets. By implementing a stringent review mechanism as part of the submission process,

the platform ensures that submitted hypergraphs meet high standards set by the community. This commitment enhances the data's usability for addressing current research challenges.

Technically speaking, our platform is made up of two main software components: *(i)* a dataset manager and *(ii)* an interactive analytics manager. The dataset manager is responsible for handling all aspects of datasets, from their creation to possible publication on the site. This includes tasks such as dataset organization and maintenance, as well as ensuring seamless user accessibility. The interactive analytics manager is dedicated to computing hypergraph statistics, dataset search, and comparison. This component provides users with robust capabilities for exploring and deriving insights from the datasets through interactive and analytical features. Together, these components synergize to provide a comprehensive, user-friendly environment for managing, analyzing, and sharing hypergraph data on our platform.

In the following, we detail our platform, its underlying hypergraph model, and its features, with a particular emphasis on the dataset creation and uploading pipeline. Additionally, we delve into the currently accessible analytics functionalities. More technical details about how to contribute can be found in the repositories of the platform[1], the hypergraph database[2], or on the website FAQ page[3].

Under the hood, HypergraphRepository exploits the SimpleHypergraphs.jl Julia library to represent and analyze the hypergraph datasets [17, 26]. This library represents a hypergraph $H = (V, E)$ as an $n \times k$ matrix, where n is the number of vertices and k is the number of hyperedges. Vertices and hyperedges are uniquely identified by progressive integer ids, corresponding to rows $(1, \ldots, n)$ and columns $(1, \ldots, k)$, respectively. Each position (i, j) of the matrix denotes the weight of the vertex i within the hyperedge j. SimpleHypergraphs.jl also provides several constructors for defining meta-information types and enables attaching metadata values of arbitrary type to both vertices and hyperedges. In this manner, the library naturally supports a wide range of hypergraph types, including heterogeneous, weighted, attributed, and temporal hypergraphs.

1 github.com/HypergraphRepository/website
2 github.com/HypergraphRepository/datasets
3 hypergraphrepository.di.unisa.it/f-a-q

Currently, SimpleHypergraphs.jl offers two mechanisms for loading and saving a hypergraph to or from a stream. In our framework, we use plain-text storage (the HGF format). In this format, the first line consists of two integers n and k, representing the number of vertices and the number of hyperedges of H, respectively. The following k rows describe the structure of H; specifically, each line represents a hyperedge as a list of all vertex-weight pairs within that hyperedge. While this format's simplicity facilitates seamless interoperability among hypergraph software libraries, its drawback is the need to define supplementary information (such as vertex metadata or hyperedge weights) in separate files. Still, it is worth noting that the framework-agnostic nature of the HGF format means it is not bound to the development of any specific library, including SimpleHypergraphs.jl, in our case. For instance, this flexibility allows users to upload directed hypergraph datasets, even though SimpleHypergraphs.jl currently lacks support for these structures. We also support the use of HIF for each dataset, thereby guiding the community toward a standard format.

As previously discussed, our project's primary aim is to provide the research community with a comprehensive hypergraph collection that evolves with the community's interests. This approach enables users to actively contribute by uploading their own datasets. Figure 3.2 summarizes the dataset management pipeline.

Given that the dataset repository, storing all the hypergraphs accessible on the platform, is publicly accessible, any user can propose modifications. To facilitate this collaborative effort, we established a designated template to guide the upload of a new dataset to the repository. The overall pipeline comprises three main phases, described below.

Dataset creation phase. The first step of the pipeline is delegated to the end user who intends to upload a novel dataset. The same process applies if a user wants to modify existing hypergraph data. Upon forking the current dataset repository, the user has to create a new folder for each additional dataset they wish to contribute. This folder must include the hypergraph stored in the HGF format, a markdown-formatted file offering a concise dataset description along with any supplementary information the user deems pertinent, and a metadata file describing the hypergraph's

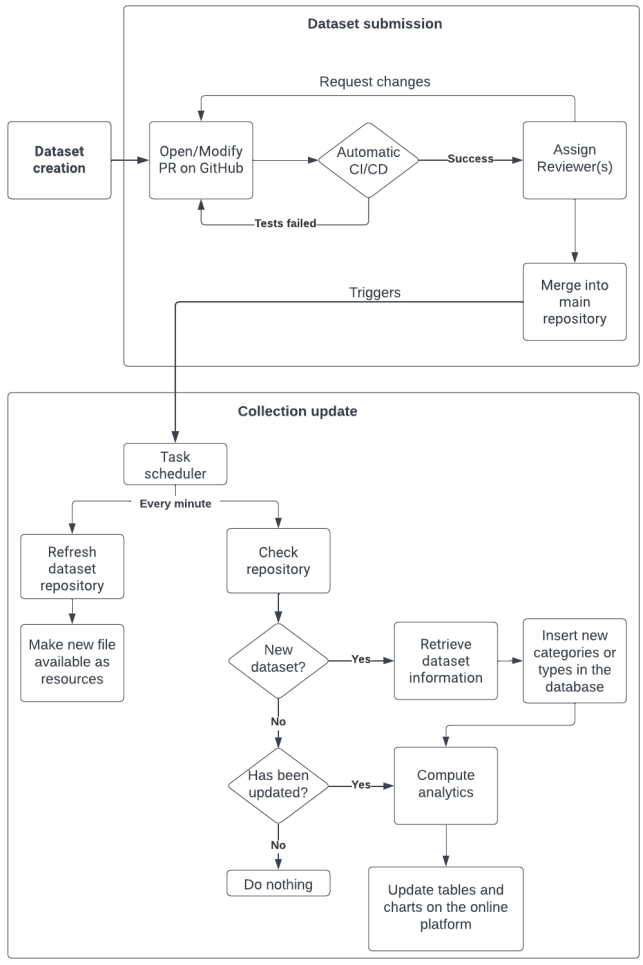


Figure 3.2: Workflow of the dataset management pipeline.

type (e.g., homogeneous, temporal) and its application domain (e.g., OSN, infrastructure network). These three files are essential for initiating a successful pull request (PR) on the main repository. Moreover, the user has the option to upload further files describing hypergraph features, such as node and vertex labels, timestamps, and hyperedge weights.

Dataset submission. Once the user is satisfied with the dataset and any supplementary data generated, they can submit all

components to the central dataset repository. Specifically, the user must initiate a PR via the GitHub system, adhering to the provided template that outlines a checklist to be fulfilled before officially requesting a merge. Figure 3.3 illustrates the template.

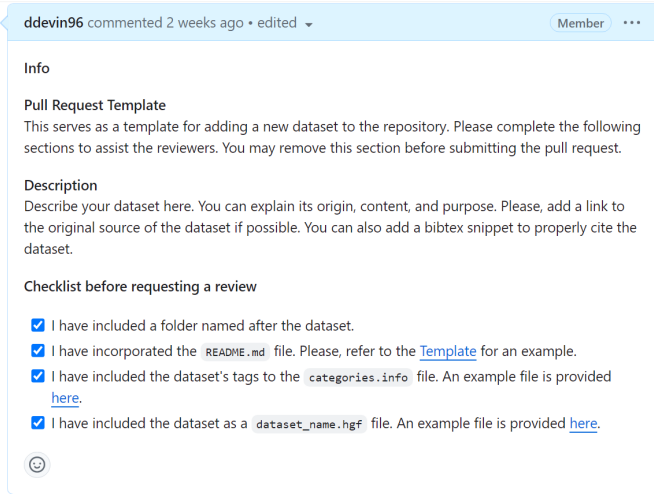


Figure 3.3: PR template for adding or modifying a dataset.

Upon opening the PR, the first round of Continuous Integration/Continuous Development (CI/CD) starts. This automated process allows for validating the PR integrity before assigning one or more reviewers. This mechanism is handled through GitHub actions, which trigger specific Python scripts. The first action verifies the presence of all requested files and ensures that their format aligns with expectations (e.g., the hypergraph data adheres to the HGF format). If any checks fail, the action will halt, and the bot will promptly report the missing elements (as a comment on the PR), delineating necessary revisions. If all these conditions are satisfied, two additional actions come into play: assigning the appropriate label(s) to the PR (e.g., creation, change, documentation) and subsequently updating the PR after each new commit. Simultaneously, the final action, which assigns a reviewer to the PR, is triggered. Each reviewer is chosen randomly from a list of volunteers, making this step a highly community-driven process open to collaboration from anyone interested. At this point, the

assigned reviewer(s) can verify whether all uploaded files are compliant with the guidelines and eventually merge the new/modified dataset into the main repository. Taking place on GitHub, it is worth noting that the overall review process is public and unblinded. Figure 3.4 shows the GitHub actions pipeline.

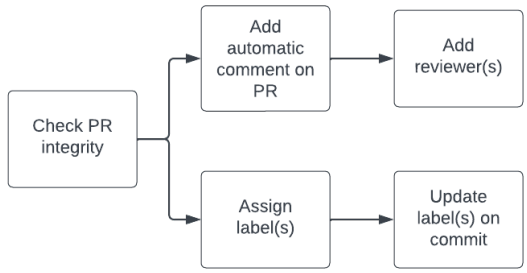


Figure 3.4: CI/CD pipeline for validating PR integrity during dataset addition or modification, along with the assignment of at least one reviewer.

Collection update. The final step of the pipeline is entirely automated and managed through CI/CD. This phase starts upon the successful merging of the PR into the main repository. Specifically, the server is configured to execute two primary scripts every minute.

- A Bash script triggers a pull operation on the dataset repository, capturing every change made. Then, it updates a folder associated with the public online collection, ensuring that all dataset files are promptly updated and made accessible for download.
- A Python script orchestrates all system calls, interacts with the GitHub APIs, and updates the database. In particular, this script makes an external call to a Julia script that computes all metrics, statistics, and plots for the newly added datasets. Leveraging the SimpleHypergraphs.jl library, the Julia script operates behind the scenes to enable the generation, manipulation, and analysis of hypergraphs at runtime. If an existing hypergraph has been modified, the script verifies which properties have changed and subsequently updates all

Table 3.1: Currently available statistics.

Symbols	Description
$\|V\|$	Number of nodes
$\|E\|$	Number of hyperedges
d_{max}	Maximum node degree
d_{avg}	Average node degree
d_{median}	Median node degree
e_{max}	Maximum hyperedge size
e_{avg}	Average hyperedge size
e_{median}	Median hyperedge size

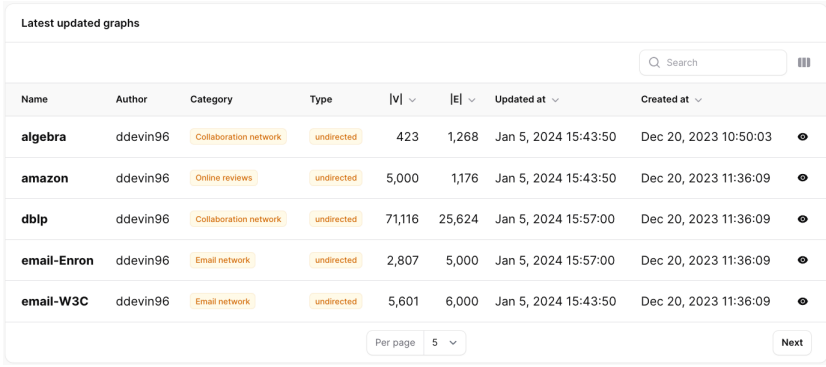
fields associated with those changes. To accomplish this, we utilize the GitHub API to retrieve the commit timestamps and verify which files have been updated. The coordination between these two scripts ensures that newly computed values are updated seamlessly on the platform's website.

Table 3.1 summarizes the statistics currently available on the website.

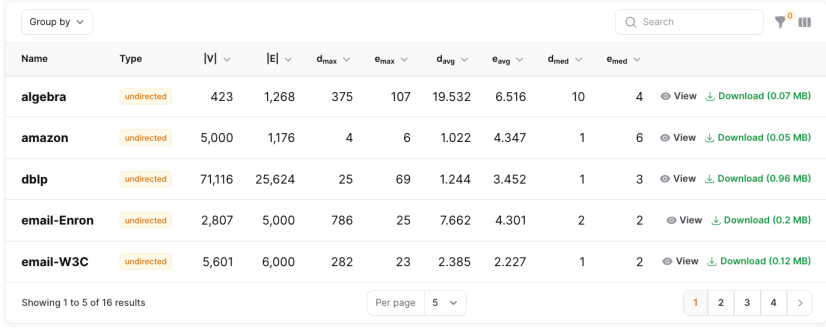
- The homepage provides users with an overview of the datasets accessible on the platform, including key information such as the number of available hypergraphs, the variety of hypergraph types and categories, and a summary table detailing the notation conventions in use. Additionally, it lists the latest updated hypergraphs. As shown in Figure 3.5a, users can access details about dataset names, authors, categories, types, and the number of nodes and hyperedges. The table also includes timestamps indicating when each dataset was initially uploaded and last modified. Finally, for a more in-depth exploration of a specific hypergraph, users can click the eye icon to open its page.

- The dataset page showcases all hypergraphs within the collection, as shown in Figure 3.5b. Beyond the information available on the homepage, this table provides additional

insights, such as the maximum, average, and median node degree, as well as the maximum, average, and median hyperedge size. Users can download a specific hypergraph of interest directly from this page, with the file size conveniently displayed for reference.



(a) Partial view of the HypergraphRepository's homepage.



(b) Partial view of the HypergraphRepository's dataset page.

Figure 3.5: The HypergraphRepository's website.

An essential feature of any online data repository is the provision of two core functionalities: the ability to search for specific datasets and to compare them. In the following, we elaborate on the implementation of these features within our platform.

Regardless of the page they visit, users can search for a specific dataset using the tables shown in Figures 3.5a and 3.5b. More precisely, users can search for a hypergraph by name, author, category, or type. Additionally, while navigating the dataset page, users can group datasets based on author, application

domain (i.e., category), or type. Further, users can apply filters to hypergraph types and categories and define minimum thresholds for the number of nodes or hyperedges. As illustrated in Figure 3.6, users can customize the settings of the main table to identify the most relevant datasets for their specific use cases. All numeric values are sortable in the table view, enabling users to navigate and explore datasets efficiently.

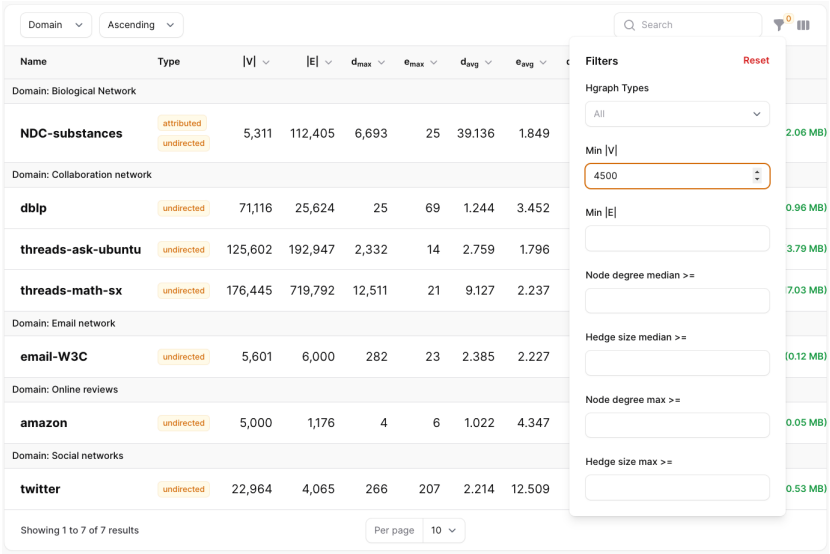


Figure 3.6: Example of *filtering* and *group by* operations.

The tables accessible on both the home and dataset pages facilitate seamless dataset comparison for users. Leveraging the table headers introduced earlier (see Figure 3.5), users can effortlessly compare datasets within the collection along the defined dimensions. Although trivial, this approach provides users with a bird's-eye view of the datasets' primary characteristics. A more in-depth comparison across datasets is the object of future work.

Users can access a detailed view of each dataset by clicking on the "view" button, as illustrated in Figure 3.7. Currently, the user can access two tabs that report information about the dataset.

- The first tab reports all statistics computed for the given hypergraph, details about the dataset creator, and the dates of both the upload and the most recent modification (retrieved through the GitHub API). This page also renders the README file, including all the information provided

at the time of submission. Users can also download the dataset directly from this page.

- The second tab contains interactive visualizations that allow users to explore various hypergraph properties. This functionality is achieved by exploiting Chart.js[4], a widely acclaimed and highly customizable JavaScript library for data plotting. The incorporation of this library is designed to facilitate the development of community-driven custom plugins and the integration of features into our platform. Currently, our platform supports the visualization of node degree and hyperedge size distribution, visualized via a histogram and a scatter plot on a log-log scale, as shown in Figure 3.8.

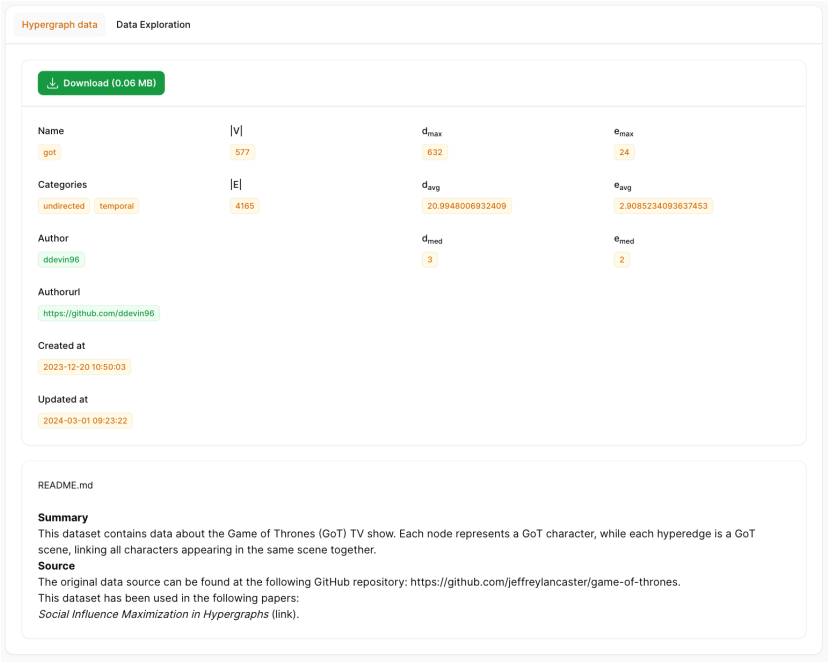
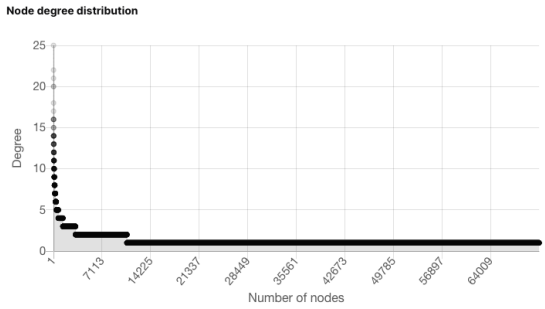
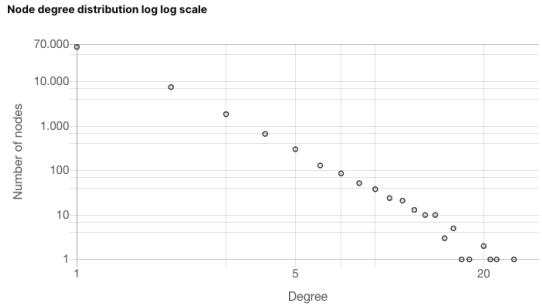


Figure 3.7: A snapshot of the web page reporting the details of a hypergraph.

4 https://www.chartjs.org/



(a)



(b)

Figure 3.8: Examples of node-related interactive charts.

3.3 MODELLING SOCIAL NETWORKS

Today, coding skills are among the most sought-after competencies worldwide, extending far beyond the domain of computer scientists. As a result, community-driven QA platforms have become essential resources for navigating programming challenges. For years, Stack Overflow has dominated as the primary venue for technical questions, while programming-related subreddits have recently emerged as popular alternatives for both troubleshooting and discussion. At the same time, OSN constitute a substantial portion of our daily interactions, providing rich conversational data that offer insights into human behavior, opinion formation, and the structural dynamics of online communities. These interactions are frequently group-based, making hypergraphs, rather than traditional graphs, particularly well suited to capture their complex, multi-actor nature.

This thesis investigates developers' behavior and community dynamics across the 20 most popular programming languages by examining 2 years of programming-related questions from Stack Overflow and Reddit. Through a longitudinal analysis of user activity and higher-order interaction patterns abstracted via hypergraphs, we uncover structural similarities and differences between these platforms. Our findings highlight crucial distinctions in how each QA environment is used and align with existing literature documenting Stack Overflow's steady decline in favor of more community-oriented platforms like Reddit.

Beyond the programming domain, this work also examines subreddit communities more broadly, demonstrating how structural patterns can reveal affinities, divergences, and latent behavioral signatures across diverse interest groups. By comparing communities such as gun-enthusiast subreddits and pro-life activist spaces, we show how hypergraph-based representations capture both the unique characteristics of each group and the analogies that emerge across ideologically charged or highly engaged communities. This structural perspective offers a deeper understanding of how online social groups form, evolve, and interact within the broader Reddit ecosystem.

3.3.1 *The shape of developer communities*

Programming enthusiasts love spending their time coding solutions to complex challenges or printing a trivial "Hello World" in the newest, most popular programming languages. In most cases, they rely on QA websites to acquire knowledge, solve problems, seek snippets of code for reuse, improve their own code, and discuss technical concepts [212]. The social mechanics of these platforms shape the formation of topic-specific communities and play a critical role in how information is shared within them. Still, these platforms' similar structures and functionalities can make it challenging to find the right place to seek help. We are interested in unraveling developers' behavior and community formation around the most widely used programming languages on QA platforms. Specifically, our first objective was to examine whether and, if so, how users' activity patterns differ across websites in terms of conversation length and duration, as well as platform-specific trending topics. Our second intent was to investigate the community structure of different platforms, characterize their

trending technologies, and evaluate their persistence or evanescence over time.

This study is a step toward answering these questions by collecting data on the 20 most widely used programming languages across two popular QA platforms, namely Stack Overflow and Reddit. The main contributions of our work can be summarized as follows:

- Stack Overflow and Reddit profoundly differ in how their users communicate in terms of the length and duration of conversations. These patterns naturally reflect the platforms' different intents in sharing knowledge.
- The most discussed languages in Stack Overflow do not correspond to those in Reddit. As in the previous case, this outcome may be due to the inherent nature of both QA platforms, as users prefer Reddit over Stack Overflow to discuss newer and trending languages.
- Both platforms exhibit the typical inverse pyramid of contribution [217], where most users contribute poorly to creating new content while only a few produce most of it.
- Overall, the analysis of the evolution of Stack Overflow communities suggests the falling out of the QA website in favor of other platforms. At the same time, it highlights Reddit's rise in the programming domain.

Test

Our work falls under the general research direction of understanding the dynamics of participation on QA platforms to encourage valuable user engagement and improve the value of crowd-sourced knowledge. Nowadays, these objectives are becoming even more critical since we are entering the era of AI-generated code thanks to tools like GitHub Copilot [130] or ChatGPT [224]. Despite their help, developers must still have some coding skills to fully exploit their potential; in this context, supporting communities will continue playing a crucial role as learning environments [191].

Reddit and Stack Overflow. QA online social platforms lay their foundations in the wisdom of crowds [276]: everyone can ask and answer questions, thus, contributing to increasing community knowledge. Currently, there is a myriad of QA websites, which

share some common features, such as the use of reputation/-points awarded for providing good questions and answers. In this work, we consider two of the most popular social QA platforms where users can ask programming language-related questions: Stack Overflow [225] and Reddit [241].

Stack Overflow is one of the most popular QA platforms specifically designed for developers. Users can submit a programming-related question, attach tags to it, and receive answers from software developer communities. According to its guidelines, a core characteristic of Stack Overflow is that users should avoid opinion-based questions that could generate discussions rather than answers. Reddit is one of the most popular OSN websites focused on news aggregation and discussion, mainly in the form of question and answer. The key element of this platform is its organization in subreddits, which represent online communities centered around a specific topic, like technology, politics, or sport. Although quite different in their intent, Stack Overflow and Reddit share some similarities: both *(i)* are community-driven, and *(ii)* live on the content generated by their users, *(iii)* use a scoring system to assess user reputation, and *(iv)* allow to filter their content by categories easily.

This section concisely describes the methodology followed, starting from the data collection and preparation process to the analysis of users' activity and interaction patterns. All code used to analyze the data and the results obtained are available on GitHub [16].

Data collection. In this study, we restrict our analysis to programming language-related questions. Specifically, we picked the top 20 most used programming languages in the Stack Overflow survey [267] to define the data filtering criteria. We queried Stack Overflow data from Kaggle [161] while we used the Pushshift API [237] to retrieve Reddit data. For both platforms, we collected: *(i)* the identifier of the comment, *(ii)* the identifier of the Stack Overflow question/Reddit submission, *(iii)* the id of the comment/question author, (iv) the creation date, *(v)* the date of last interaction with the question, and *(vi)* the topic of the question/comment (i.e., Stack Overflow tag or subreddit name). After the data collection process, we anonymized all ids. The resulting data set (available on Zenodo) spans over two years, starting from 2020/09/24 to 2022/09/24.

Data Analysis. To assess the evolution of the posting patterns and interactions of Stack Overflow and Reddit users over the two years, we split our data set into eight snapshots, each 3-months long. For each period, we evaluated both quantitative and structural information.

Mining users' activity patterns. As a first insight into each platform's usage, we evaluated: *(i)* the number of unique users, *(ii)* the number of questions/comments, *(iii)* the distribution of the length and *(iv)* time span of conversations. We then focused on examining the popularity of the 20 languages in terms of the number of questions/submissions related to the specific topic.

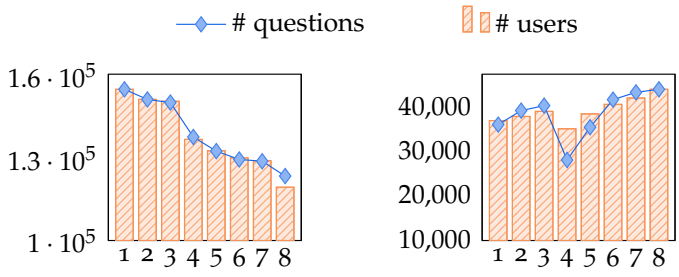
Mining users' interaction patterns. A set of users commenting and discussing the same question can be seen as part of the same many-to-many (or high-order) relation. Such type of relationship does not imply a direct pairwise interaction among each pair of users, but it only indicates they have commented on the same topic. In this work, we leverage hypergraphs to model the underlying question-answers interaction network, which allows us to highlight the existing relationships between users and questions rather than between users. The rationale behind this network creation approach is to link users that have interacted with the same question, even though they did not directly answer each other. In this way, we are able to cluster together groups of users with the same interests even if they have never interacted directly.

To address the above task, we built one hypergraph per trimester, whose vertices represent Stack Overflow and Reddit users, and hyperedges their indirect interactions when they answer the same question or submission. After labeling each hyperedge with the topic of the question, we ran the Label Propagation community detection algorithm [17] on each hypergraph snapshot and characterized each community based on the most prominent tag within it. For the bigger communities, we further investigated their persistence in terms of the users that constitute them over time.

Mining users' activity patterns. It may be argued that among the first indicators of a platform's liveliness, we can name the number of users actively engaging with others as well as the number of their activities on the platform itself. In the realm of QA websites, looking at how users connect with each other may guide toward a deeper comprehension of how users perceive and actually use the platform.

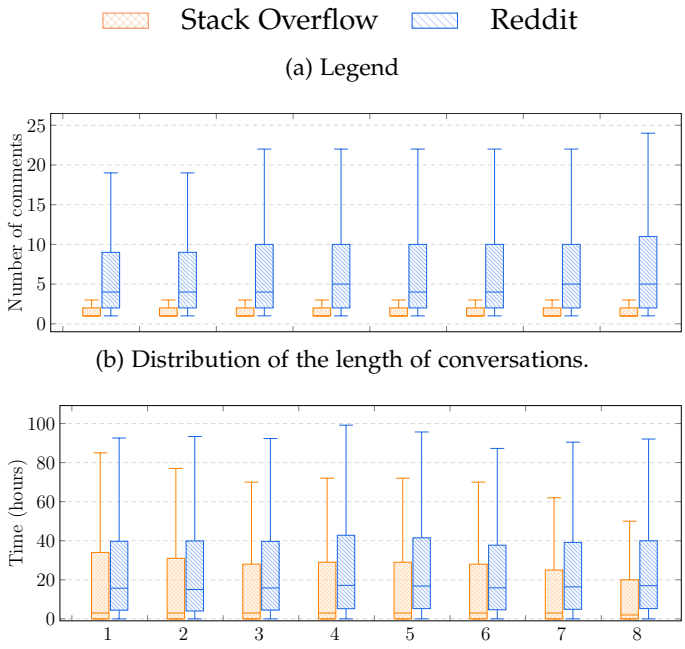
Usage of Stack Overflow and Reddit over time. Although Stack Overflow and Reddit represent two examples of QA online platforms, they profoundly differ in the way they are supposed to be used and in the intended user base. These differences are clearly evident from the data we collected in terms of *(i)* the number of users posting and/or answering questions about the selected programming languages, and *(ii)* the length (number of comments) and time-span of each conversation (i.e., *questions* for Stack Overflow, *submissions* for Reddit).

Figure 3.9 shows, for each trimester, the number of users that have asked or answered at least one question plotted against the overall number of questions. The first interesting outcome is the steeply decreasing trend in the number of users and questions/-comments on Stack Overflow (see Figure 2a). This phenomenon may be justified by the increasing knowledgeability of the developers, which could be more skilled in tuning existing answers to their problems [49], as well as by the ever-increasing database of questions which makes it easier for someone to find another user who had already experienced the same issue. Another reason for this negative trend could also be sought in Stack Overflow's increasing churn rate (i.e., users who leave the platform after an initial use [215]), possibly due to the constant evolution of the proportions of content quality and user types (in relation to the posting and answering behavior) over time [266]. Generally, the number of questions is comparable to the number of users, hence, suggesting a ratio of 1:1 between users and questions. In contrast, the Reddit data set exhibits a rather different perspective (see Figure 2b). In this case, the number of Reddit users (i.e., Redditors) tends to increase gently over the two years of observation, possibly indicating the presence of a persistent developer



(a) Stack Overflow (b) Reddit

Figure 3.9: Number of users and questions per trimester.



(a) Legend

(b) Distribution of the length of conversations.

(c) Life span of conversations.

Figure 3.10: Comparison of the number of answers and durability of conversations in Stack Overflow and Reddit.

community on the platform. Reddit's characteristic of being a domain-agnostic discussion website explains why the number of Redditors is one order of magnitude less than the number of Stack Overflow users in our data set. In the case of Reddit, we are only considering a very small subset of its user base (52 million daily active users on average [272]); in the case of Stack Overflow, we are naturally including a bigger slice of the overall population considering that the platform specifically addresses developers' questions. As before, the number of questions is proportional to the number of Redditors, even though questions slightly outnumber users. An exception is made for the fourth and fifth trimesters, where the situation is reversed.

Figure 3.10 reveals two very different patterns of interaction across the two platforms regarding the length (see Figure 3.10b) and the time span (see Figure 3.10c) of conversations. Stack Overflow conversations are characterized by few comments, with most questions receiving only one answer. More extended conversations average 20 comments. On the contrary, Reddit discussions

are relatively longer than those on Stack Overflow, with half of the conversations consisting of 4/5 comments. Reddit's underlying nature as a discussion forum is reflected in the long tail of the distribution, where 10% of the submissions receive 20/25 replies on average, with some outliers reaching a mean of 600 comments. Regarding the lifespan of conversations, it is not surprising that Stack Overflow questions tend to be answered in less than 2/3 hours. A possible motivation for this behavior may lie in the platform's gamified mechanism, which stimulates volunteers' contributions by maximizing the likelihood of winning gamification rewards for fast answers [195]. Still, many Stack Overflow conversations span over one day or more, with some discussions still active after more than one year. On Reddit, half of the conversations last 15-17 hours, while another consistent portion lasts more than one day. However, unlike the QA computer science platform, the most extensive Reddit discussions span no more than three months. The reason for this result may be the focus of further research.

Technology trends. Another interesting perspective under which Stack Overflow and Reddit profoundly differ is the popularity (in terms of the number of questions) of each programming language. Figure 3.11 depicts this information, considering general-purpose and domain-specific languages independently to make it easier to read the plots. The y-axis reports the ratio of questions about a given language within the category of interest to enable comparison between the two platforms. The first noticeable outcome from this analysis regards the prevalence of Python-related questions on Stack Overflow (∼80%). Together with C#-related queries (∼20%), these constitute almost the totality of all raised issues about general-purpose languages. In the case of Reddit, Python, Rust, and Clojure-related questions share the stage with around 30/40% of questions each. Generally, no other technology sensibly prevails over the others, even though they follow different trends on both platforms. We can observe a similar situation when looking at the proportion of queries about domain-specific languages. JavaScript-related questions dominate the scene on Stack Overflow (∼40%), followed by SQL (∼20%), HTML (∼18%), and Go-related (∼12%) issues. In Reddit, there is no extreme gap across all technologies, but, in this case, the most frequent language is Go (∼25%), still followed by SQL (∼19%) and JavaScript (∼15%). As before, questions about

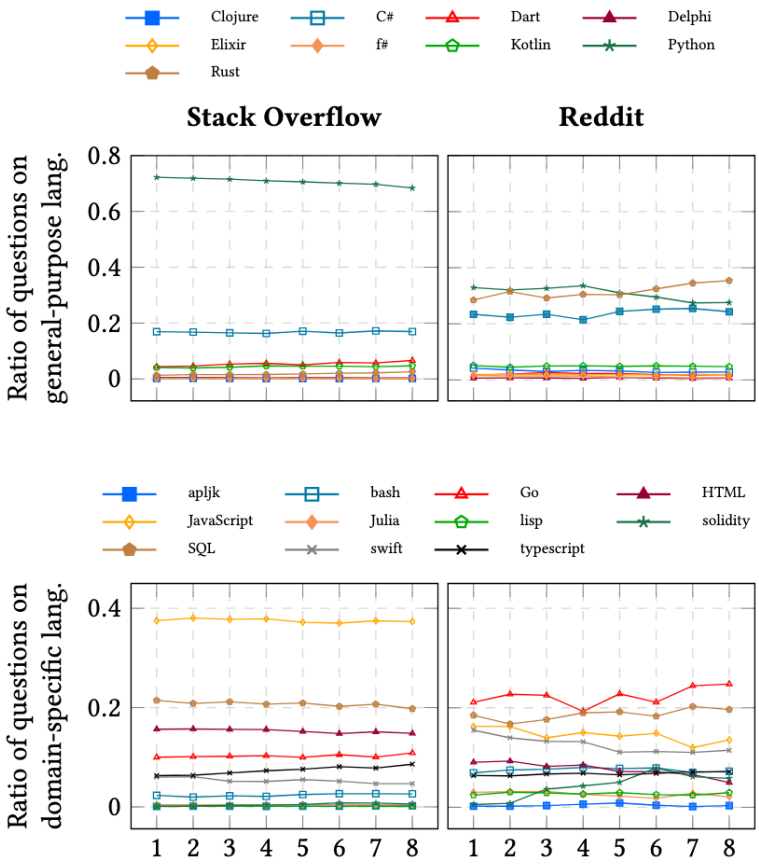


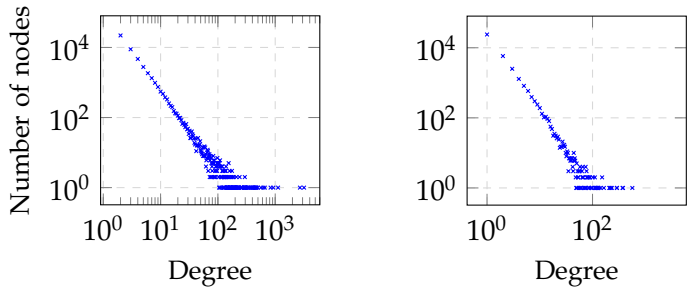
Figure 3.11: Proportion of the number of each language-related questions in Stack Overflow and Reddit.

the remaining languages follow different patterns on the two QA websites. Overall, these results suggest that Stack Overflow is a referring platform for asking questions related to the most commonly used programming languages, such as Python and JavaScript. On the contrary, Reddit appears to be preferred to discuss recently rising programming languages and their applications in the software development landscape (e.g., Rust). Once again, this trend may be due to the inherent nature of the Reddit website, where users can either discuss or ask for help on a technical topic.

Mining users' interaction patterns. Another distinctive aspect of how a QA platform is perceived by its users is how they spon-

taneously organize into informal communities. Understanding these dynamics would shed light not only on which technologies are more exploited for specific tasks but also on the platform's durability over time and content quality.

Users-Questions interaction hypergraph. We built one hypergraph per trimester for each platform representing the many-to-many interaction between users commenting on the same questions. Figure 3.9 indirectly reports the size of each hypergraph, indicating the number of users (i.e., vertices) and questions (i.e., hyperedges) throughout the observation period. Figure 3.12 shows the vertex degree distribution of the Stack Overflow and Reddit hypergraphs built upon the first trimester of data. Recalling that evaluating the degree of a vertex in a so-built hypergraph translates to counting the number of discussions a user has contributed, it is not surprising that both QA websites exhibit a similar contribution pattern. This long-tail distribution is typical of online social platforms, where most of the users contribute poorly to the creation of new content, while only a few of them account for most of it, according to the online participation inequality pyramid [19, 215, 217]. In both platforms, we have that, on average, around the 65% of users have contributed to a single question/submission (69% in Stack Overflow, 64% Reddit), the 15% to two discussions (14% in Stack Overflow, 16% Reddit), the 6% to three questions (6% in Stack Overflow, 7% Reddit), and less than 12% to four or more discussions (11% in Stack Overflow, 13% Reddit). We found the same contribution pattern repeating over all trimesters in both platforms with no statistical difference.



(a) Stack Overflow. (b) Reddit.

Figure 3.12: Vertex degree distributions of Stack Overflow and Reddit on a log–log scale during the first trimester.

Number of communities across all trimesters								
SO	59179	58564	56181	52382	50838	49936	48979	47417
Reddit	5140	5890	5478	4703	4894	5414	5851	5311
% of users within the top 5% biggest communities								
SO	26,70%	26,05%	27,40%	26,29%	25,90%	25,61%	25,96%	26,44%
Reddit	52,07%	43,28%	50,37%	50,98%	52,79%	52,74%	47,53%	53,78%
Trim.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Table 3.2: Community and user distributions.

Community detection and characterization. After building the user-question interaction hypergraphs, we evaluated their community structure via the label propagation algorithm. The upper part of Table 3.2 lists the number of communities found in each trimester for Stack Overflow (SO) and Reddit. The first remarkable outcome consists of the considerable number of communities discovered in both QA platforms, which decreases up to 20% for Stack Overflow while oscillating around the same value for Reddit. Further, the number of Reddit communities is always less than Stack Overflow communities by one order of magnitude throughout all trimesters. Although interesting, this result is not surprising and is consistent with previous literature. In a nutshell, the Reddit discussion model represents a fertile field for the growth of more supportive communities [15, 127, 191] whose interaction network appears to be less assortative and less clustered and, hence, less fragmented into small groups [141, 206]. On the contrary, Stack Overflow slowly changed over the last decade from a generalized discussion, with a small number of more extensive and diverse communities, to a more specialized mode with a larger number of smaller communities, each with a particular focus [212]. Having a deeper look at the size of the communities identified, we found that most of them were composed of very tiny groups of users and that, on average, only 5% of Stack Overflow and Reddit communities had a size bigger than 3 and 10, respectively. To filter out the noise such small groups could convey, we only focused on the top 5% largest communities on both platforms. The bottom part of Table 3.2 reports the percentage of users included in the communities considered: while we retained half of the original Reddit user base, we left with only 25% of the starting user base in the case of Stack Overflow.

To characterize the most dominant topic within each community, we considered the most common tag among the hyperedges belonging to it. We then labeled each user with the same topic of the community they belong. Table 3.3 reports the aggregated percentage of users associated with a given programming language community in Stack Overflow and Reddit. Table 3.5 lists the size of the communities for all programming languages. These values can be interpreted as a complement to the trends of language-related questions shown in Figure 3.11. In this case, we consider the label associated with each user as a proxy to count the number of users who have contributed to each topic. As expected, the biggest communities are represented by the most trending programming languages, such as Python and Javascript for Stack Overflow and Python, Rust, and Go for Reddit. Interestingly, we can observe how this pattern is not always true for Reddit. For instance, Clojure is the third most trending general-purpose language in our data set; however, less than 1% of the Reddit users analyzed belong to the Clojure community (a similar case happens for StackOverflow and C#). A completely reversed situation occurs for C#, where the C# community contains an average of 13% of the Reddit users. This result may suggest the existence of a few committed users who heavily discuss a given language

Trimester		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
C#	SO	2,96	3,32	2,95	2,89	3,52	2,66	4,24	3,48
	Reddit	14,52	15,77	11,33	12,50	14,12	14,34	15,55	13,62
Clojure	SO	0,00	0,02	0,04	0,00	0,06	0,00	0,00	0,04
	Reddit	1,10	0,79	0,63	1,02	0,33	0,68	0,61	0,65
Go	SO	0,31	0,36	0,20	0,44	0,43	0,34	0,55	0,27
	Reddit	7,74	7,50	6,88	9,08	9,67	8,33	12,30	7,02
Javascript	SO	25,29	26,72	24,02	27,64	28,84	27,81	24,22	25,29
	Reddit	7,68	9,42	9,95	7,84	7,93	8,47	7,14	6,26
Python	SO	61,13	58,36	63,51	58,76	56,51	57,24	58,78	59,22
	Reddit	33,71	36,21	28,37	32,34	32,84	25,70	31,45	30,22
Rust	SO	0,20	0,30	0,28	0,23	0,20	0,39	0,29	0,40
	Reddit	23,94	18,02	28,71	26,64	26,85	28,73	20,44	30,55
SQL	SO	4,01	3,88	3,86	4,01	4,93	4,56	5,86	5,51
	Reddit	1,66	2,29	2,16	2,14	1,65	4,12	4,61	3,04

Table 3.3: Percentage of users per programming language within the top 5% biggest communities.

		Percentiles			
Trimester	**Q&A Site**	**0.25**	**0.5**	**0.75**	**0.95**
1	Stack Overflow	1	1	2	3
	Reddit	1	2	5	11
2	Stack Overflow	1	1	2	3
	Reddit	1	2	5	12
3	Stack Overflow	1	1	2	3
	Reddit	1	2	5	12
4	Stack Overflow	1	1	2	3
	Reddit	1	2	5	12
5	Stack Overflow	1	1	2	3
	Reddit	1	2	5	12
6	Stack Overflow	1	1	2	3
	Reddit	1	2	5	11
7	Stack Overflow	1	1	2	3
	Reddit	1	2	5	13
8	Stack Overflow	1	1	2	3
	Reddit	1	2	5	13

Table 3.4: Distribution of the size of Stack Overflow and Reddit communities over each trimester.

(e.g., the Clojure community in Reddit) as well as the lack of solid interaction concerning a specific topic (e.g., C#-related questions in Reddit). Nevertheless, further analyses are needed to explain this behavior.

Community evolution. To verify how the composition of the communities changed over time, we examined the user behavior in terms of whether they stay active in the same community (i.e., continue to ask or answer questions related to their current community), migrate to another community (i.e., begin to consistently ask/answer questions in another community), go inactive (i.e., stop asking/answering questions anywhere on the platform of interest even though they can still lurk on the platform [19, 212]), or (re-)join the community (i.e., newly registered users or users that become active again after having been inactive the previous trimester(s)). Figure 3.13 shows the percentage of these four classes of users evaluated pairwise over consecutive trimesters. The values shown are averaged over all communities.

What clearly stands out from this analysis is the significant number of users that become inactive from one trimester to the

following (∼82% for Stack Overflow, ∼70% for Reddit) as well as the high number of new users (∼80% for Stack Overflow, ∼74% for Reddit). Further, it is also noticeable how the percent-

		Trimester							
Prog. lang.	**Q&A Site**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
APL	Stack Overflow	0,00	0,03	0,08	0,00	0,00	0,00	0,06	0,02
	Reddit	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Bash	Stack Overflow	0,23	0,44	0,30	0,46	0,51	0,71	0,73	0,74
	Reddit	1,25	1,06	2,11	1,38	0,47	1,88	1,60	1,05
C#	Stack Overflow	2,96	3,32	2,95	2,89	3,52	2,66	4,24	3,48
	Reddit	14,52	15,77	11,33	12,50	14,12	14,34	15,55	13,62
Clojure	Stack Overflow	0,00	0,02	0,04	0,00	0,06	0,00	0,00	0,04
	Reddit	1,10	0,79	0,63	1,02	0,33	0,68	0,61	0,65
Dart	Stack Overflow	0,77	1,06	1,05	0,57	0,77	1,21	1,16	1,16
	Reddit	0,15	0,49	0,57	0,14	0,13	0,32	0,27	0,32
Delphi	Stack Overflow	0,14	0,06	0,09	0,11	0,02	0,11	0,02	0,02
	Reddit	0,00	0,00	0,00	0,00	0,00	0,00	0,12	0,00
Elixir	Stack Overflow	0,06	0,01	0,02	0,00	0,01	0,02	0,00	0,02
	Reddit	0,67	0,57	0,67	0,65	0,18	0,21	0,88	0,47
f#	Stack Overflow	0,05	0,03	0,05	0,04	0,02	0,04	0,01	0,00
	Reddit	0,12	0,00	0,00	0,00	0,00	0,00	0,00	0,19
Go	Stack Overflow	0,31	0,36	0,20	0,44	0,43	0,34	0,55	0,27
	Reddit	7,74	7,50	6,88	9,08	9,67	8,33	12,30	7,02
HTML	Stack Overflow	1,55	1,72	1,36	1,55	1,57	1,59	1,65	1,05
	Reddit	0,11	0,00	0,45	0,38	0,29	0,32	0,55	0,00
Javascript	Stack Overflow	25,29	26,72	24,02	27,64	28,84	27,81	24,22	25,29
	Reddit	7,68	9,42	9,95	7,84	7,93	8,47	7,14	6,26
Julia	Stack Overflow	0,06	0,29	0,16	0,15	0,12	0,04	0,04	0,09
	Reddit	0,40	0,92	2,47	0,44	0,35	0,42	0,40	0,29
Kotlin	Stack Overflow	0,70	0,99	0,45	1,11	0,80	0,83	0,82	0,90
	Reddit	1,71	1,53	1,17	0,42	0,93	1,77	1,13	1,06
Lisp	Stack Overflow	0,07	0,07	0,00	0,04	0,00	0,02	0,08	0,00
	Reddit	1,41	0,67	1,03	1,58	1,50	0,71	0,51	0,82
Python	Stack Overflow	61,13	58,36	63,51	58,76	56,51	57,24	58,78	59,22
	Reddit	33,71	36,21	28,37	32,34	32,84	25,70	31,45	30,22
Rust	Stack Overflow	0,20	0,30	0,28	0,23	0,20	0,39	0,29	0,40
	Reddit	23,94	18,02	28,71	26,64	26,85	28,73	20,44	30,55
Solidity	Stack Overflow	0,00	0,02	0,03	0,16	0,15	0,15	0,10	0,00
	Reddit	0,00	0,00	0,11	0,19	0,14	0,99	0,00	0,59
SQL	Stack Overflow	4,01	3,88	3,86	4,01	4,93	4,56	5,86	5,51
	Reddit	1,66	2,29	2,16	2,14	1,65	4,12	4,61	3,04
Swift	Stack Overflow	1,92	1,71	1,13	0,92	1,01	1,58	0,78	1,04
	Reddit	2,44	3,62	2,28	1,78	1,78	1,81	1,24	1,87
Typescript	Stack Overflow	0,55	0,62	0,42	0,93	0,54	0,70	0,62	0,75
	Reddit	1,40	1,15	1,12	1,47	0,85	1,21	1,19	1,98

Table 3.5: Percentage of users per programming language within the top 5% biggest communities per trimester.

age of users remaining in the same community or migrating from one community to another is extremely low in the Stack Overflow platform (∼4% and ∼3%, respectively). At the same time, we can observe slightly higher values for Reddit (∼5% and ∼25%, respectively). Overall, these results suggest that most Stack Overflow users are interested in receiving feedback for solving their problems rather than contributing and building a community around specific topics. However, it is worth noting that this outcome is strictly related to the division in trimesters of the observation period and that this phenomenon may change if we observe a user's activity over more extended periods, for instance, from one year to another [212]. In the case of Reddit, we can observe a more significant portion of users (around 30% considering both users remaining in the same community and migrating to another) that consistently contribute to the platform. This behavior may reflect a greater sense of bonding among community members in addition to task-specific discussions [260]. Still, the migration pattern across communities must be further analyzed by considering each community independently.

3.3.2 *Deeper analysis on communities*

OSNs have revolutionized our lives, greatly impacting how we interact with other people. Not only our lives, but these platforms have reshaped the web, changing protocols and utilities to support a completely different consumer pervaded by the online world [223, 228]. The rapid advancement of technology has transformed our interactions from face-to-face to screen-based communication. However, this modern form of dialogue may not

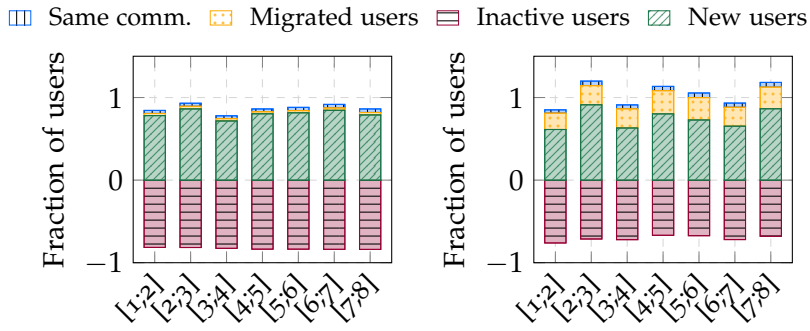


Figure 3.13: Community evolution.

carry the same significance. Messages exchanged on screens are prone to various interpretations, making it challenging to discern the intended tone, whether it is a joke, sarcasm, or a straightforward remark. As a consequence, analyzing these conversations is challenging and requires the selection of appropriate tools to represent and examine such data.

OSNs are usually composed of interactions between multiple people (e.g., friends within the same group) who discuss a specific topic. In literature, graphs are the most used representation of these interactions. For example, an OSN can be modeled as a graph $G = (V, E)$, where V denotes a set of users and E represents the connections between two users. This link can easily define a real-world friendship or, for instance, the typical follow-follow relation [173]. However, representing these interactions with pairwise links between nodes could be a limitation when modeling many-to-many relationships [61]. In many real-world systems, relations are non-dyadic and often involve more than two nodes. Social networks are a clear example of this phenomenon, as social groups are based on relationships among multiple users who interact simultaneously (e.g., a group of users commenting on the same post), leading to the emergence of high-order behaviors that are poorly encoded in the topology of a traditional graph [37]. In recent years, alternatives like hypergraphs have gained popularity for representing these higher-order interactions [18]. Hypergraphs generalize the concept of graphs, allowing a hyperedge to connect an arbitrary number of nodes [44]. As motifs for graphs, h-motifs have been proven to be critical in identifying unique local structural patterns in different domains [180, 193]. Hence, enhancing our understanding of their formation and evolution can be used to identify meaningful connections to, for instance, stop misinformation spreading [105]. Figure 3.14 shows on the left a simple example of this mathematical structure and on the right an example of h-motif.

We explore the conversational patterns of online discussions on Reddit, a web platform that hosts various communities engaged in diverse topics [206]. Specifically, we model and examine the structural characteristics of the networks inferred from these conversations using hypergraphs and leverage the notion of h-motifs to describe the conversational patterns involving different

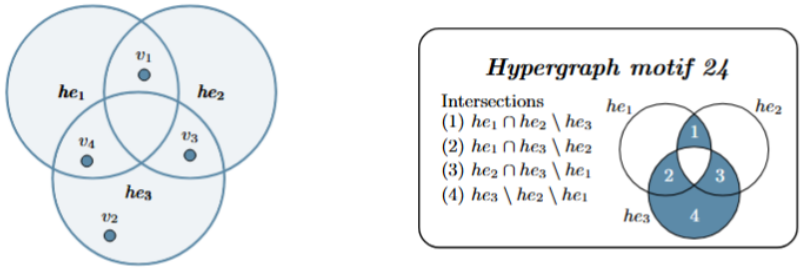


Figure 3.14: Example of a hypergraph with three hyperedges and 4 nodes (on the left). From this configuration, it is possible to extract an instance of h-motif (on the right), given by the intersections: (1) $he_1 \cap he_3 \setminus he_2$; (2) $he_1 \cap he_2 \setminus he_3$; (3) $he_2 \cap he_3 \setminus he_1$; (4) $he_3 \cap he_2 \setminus he_1$.

topics. In particular, we aim to answer the following research question:

> Within an OSN such as Reddit, where each subreddit functions as a distinct community, are there comparable patterns of interaction between these subreddits?

Results show that the conversational networks related to each Reddit community have their own structural characteristics, whose comparison within the same domain yielded unexpected results. In contrast to existing literature, our analysis revealed that conversations related to the same topic do not exhibit similar local interaction patterns. This suggests that high-order dynamics, such as groups of users interacting in the same conversation thread, cannot be overlooked when studying complex online human behavior.

The study of OSNs. OSNs are probably the most visited and used websites on the Internet. Platforms such as Facebook, YouTube, and Reddit have facilitated how people create and maintain relationships, share information, and form communities. The study of OSNs has emerged as a dynamic and interdisciplinary field of research, involving experts from computer science [308], sociology [152] and many others [150]. The fast and easy access to these platforms has revolutionized the dissemination of information and coordination of activities. OSNs influence public opinion formation and can contribute to idea polarization. One example is the echo chamber effect, where individuals are exposed to

homogeneous points of view, reinforcing existing beliefs and leading to increased polarization within society [82]. Analyzing these platforms has helped research to model and understand the behavior of people, diving into the shape of societal dynamics in the online world [76, 289].

Hypergraphs. Despite the prevalence of graphs, interactions in real-world systems often involve relationships between groups of entities. While graphs are ideal for representing pairwise relationships between them, the literature suggests that hypergraphs are better suited for capturing these more complex group interactions. Hypergraphs are, hence, the natural representation of a broad range of systems where group (or high-order) relationships exist among their interacting parts. Hypergraphs are a generalization of graphs where a (hyper)edge can group an arbitrary number of nodes. A hypergraph is formally defined as an ordered pair $H = (V, E)$, where V is the node set, and E is the hyperedge set [61]. Such structures can easily abstract social systems where individuals interact in groups of any size [193]; for instance, in the case of a co-authorship collaboration network, a hyperedge may represent an article and link together all authors (nodes) having collaborated on it [18]. Similar situations, characterized by high-order interactions, also exist in biology, ecology, and neuroscience [37]. In the social context, hypergraphs have been employed to model various scenarios, including the relationship between mobility and social connections [312], the formation of links between users using hypergraph neural networks [132, 208], the growth of OSNs over time [298], and the impact of social influence on OSNs [29, 322].

The concept of high-order motifs (h-motifs) has been recently introduced by Lee et al. [180] and describes the connectivity patterns of three or more connected hyperedges. As motifs for graphs, h-motifs have been proven to be critical in identifying unique local structural patterns in different domains [180, 193]. Each h-motif has a different shape, specified by the non-empty intersections between the hyperedges: (1) $he_1 \setminus he_2 \setminus he_3$ (2) $he_2 \setminus he_1 \setminus he_3$ (3) $he_3 \setminus he_1 \setminus he_2$ (4) $he_1 \cap he_2 \setminus he_3$ (5) $he_1 \cap he_3 \setminus he_2$ (6) $he_2 \cap he_3 \setminus he_1$ (7) $he_1 \cap he_2 \cap he_3$. An example of an h-motif is shown in Figure 3.14. It is important to note that an arbitrary number of hyperedges can be used to describe an h-motif, but the possible combinations increase drastically (e.g., 1,853 and 18,656,322 h-motifs for four and five connected hyperedges).

In this section, we detail the experiments conducted to address the research question outlined in the Introduction. All code, images, and tools used in these experiments are available in a public GitHub repository[5].

Dataset. The dataset built for our analyses includes 22 different subreddits, spanning from January 4th, 2019, to July 31st, 2020, counting 1.173.347 messages and 5271 users. For each subreddit, we retrieved all conversations, storing information about authors and submissions. It is worth noting that we anonymized all sensitive information.

Modeling conversational networks with hypergraphs. In this work, we model each conversation with a Text Attributed Hypergraph (TAH). Formally, a TAH is defined as a tuple $H = (V, E, T)$, where each node $v \in V$ is associated with at least one text $t \in T$, which contains all the messages in a Reddit conversation [103].

Structural analysis. In this study, we focused on analyzing the local interaction patterns of each conversational network by following the approach outlined by Lee et al. [180]. Specifically, we compared the local structural patterns of the various conversational hypergraphs by examining their Characteristic Profiles (CPs). CPs characterize a hypergraph by quantifying the significance of its h-motifs, which is achieved by normalizing and concatenating the significance of all h-motif instances. Figure 3.15 illustrates the CPs of the subreddits we analyzed. As illustrated in

5 https://github.com/ddevin96/DevCommunities

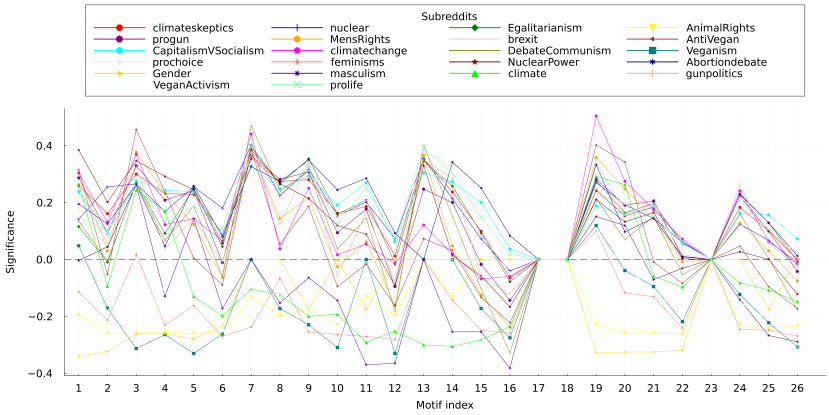


Figure 3.15: CPs of the subreddits analyzed in this work.

the plot, conversations in the subreddits can vary significantly in terms of local interaction patterns, suggesting that different network structures can emerge under similar conditions. To explore the common characteristics underlying similar conversational hypernetworks, we employed two grouping methods. The first method involves clustering the CP vectors, while the second measures cosine similarity between the CPs, providing a more nuanced categorization. Although both methods yield similar representations, they highlight different matches among subreddits. Figure 3.16 and Figure 3.17 illustrate the outcomes of both methods through a matrix of similarity for CPs.

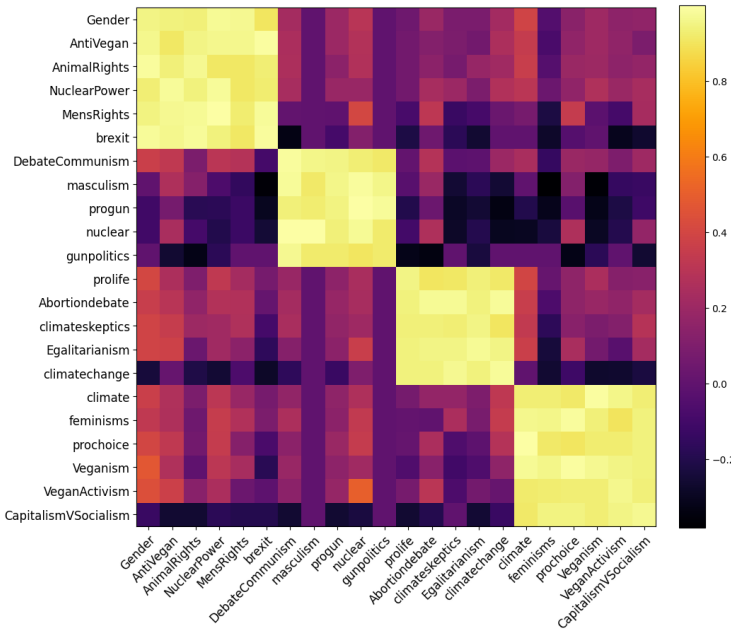


Figure 3.16: Clustering-based similarity.

To better understand the underlying motivations behind these two different categorizations, we selected three samples from each similarity matrix. Specifically, we chose the following triples of subreddits:

- *climateskeptics, progun, prolife*;
- *climateskeptics, nuclear, egalitarianism*;
- *climateskeptics, climate and climate change*.

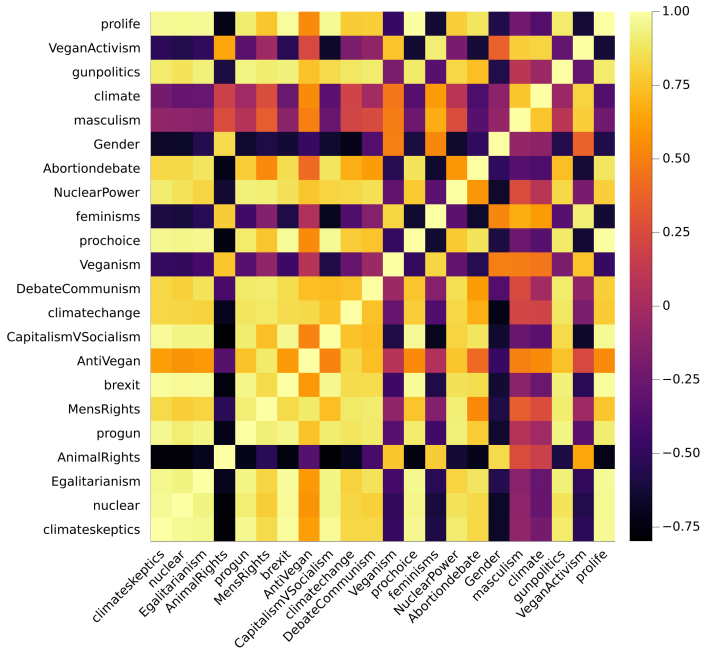


Figure 3.17: Cosine-based similarity.

Each group includes a conversation hypergraph of varying sizes and shapes, but all groups contain the *climateskeptics* subreddit for comparison.

Figure 3.18 illustrates the curve trends for the CPs of the three distinct groups. The first two groups, which are clustered together in the matrix, show similar curve patterns. In contrast, subreddits from the third group, despite being from the same real-world domain (i.e., climate), display a different trend. These observations imply that local patterns can highlight diverse phenomena based on the composition of the groups.

First, we can observe that the size of the hypergraph does not influence the number of h-motifs. All the hypergraphs generated exhibit different shapes, and neither cluster differentiates based on a quantitative measure. Second, contrary to the assertions in [180], real-world hypergraphs within the same domain do not follow the same curve trend. This discrepancy is primarily due to two reasons. Assuming these hypergraphs belong to the conversations or OSN domain, Figure 3.15 clearly shows that multiple behaviors follow different curve trends. Therefore, this

finding highlights that domain alone is not a sufficient feature to categorize different communities into the same group. Our results further suggest that the observed similarities are more closely related to the dynamic behavior of the Reddit platform rather than the specific topics considered. Conversely, if we consider these hypergraphs based on the domains suggested by their names (e.g., the progun subreddit not hosting discussions about vegan food), we find clear instances of hypergraphs from different semantic domains sharing similar patterns. This similarity is not observed in hypergraphs from the same domains.

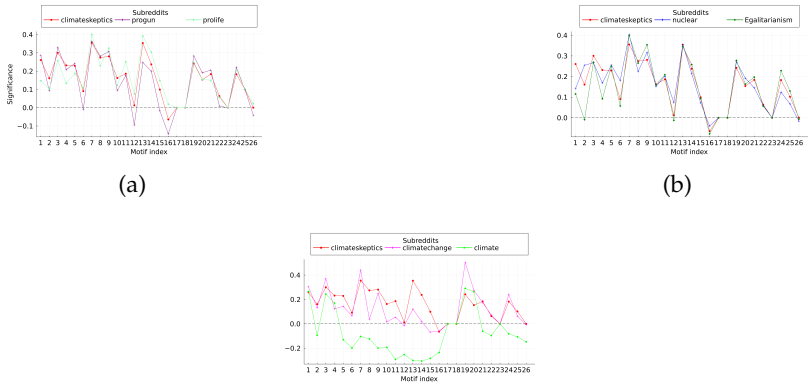


Figure 3.18: a) Three subreddits that have similar CPs using clustering. b) Three subreddits that have similar CPs using cosine similarity. c) Three subreddits with similar domains.

In summary, while domain similarity offers a starting point for studying similar communities, users' dynamic behavior and interactions play a more crucial role in defining their structural analogies.

Some limitations of this study relate to the data and their analysis. First, the temporal span and the selection of messages may limit the representation of how diverse users impact the observed communities. It is also important to acknowledge that the study considers a temporal slice of one and a half years in its entirety. Future research should explore both longer and shorter temporal slices to better understand the evolution of these communities.

3.4 EXPLOITING LLMS TO UNDERSTAND SEMANTICS

Understanding the structural and linguistic properties of conversational data in social media is crucial for extracting meaningful insights to understand opinion dynamics, (mis-)information spreading, and the evolution of harmful behavior. Current state-of-the-art mathematical frameworks, such as hypergraphs and linguistic tools, such as large language models (LLMs), offer robust methodologies for modeling high-order group interactions and unprecedented capabilities for dealing with natural language-related tasks. In this study, we propose an innovative approach that blends these worlds by abstracting conversational networks via hypergraphs and analyzing their dynamics through LLMs. Our aim is to enhance the stance detection task by incorporating the high-order interactions naturally embedded within a conversation, thereby enriching the contextual understanding of LLMs regarding the intricate human dynamics underlying social media data.

Social interactions through online media are increasingly pervasive [294], offering opportunities to study a wide range of social human behavior, including the mechanisms behind collective action, the spread of (mis)information, and the evolution of harmful behaviors. The widespread adoption of Large Language Models (LLMs) like ChatGPT or Gemini are changing the way in which people produce and consume content in online social media. LLMs are influencing how people create, share, and consume content from textual descriptions to images to videos [96]. Moreover, from an academic perspective, a growing body of work demonstrates how LLMs outperform state-of-the-art models in various tasks [142, 290, 300]. Traditionally, online interactions between social media users have been successfully studied through graphs abstracting the underlying relations with nodes and edges connecting pairs of interacting components. Yet, many online actions are characterized by group interactions that cannot be described simply in terms of dyads (e.g., all users commenting on the same discussion thread). Hypergraphs are the perfect candidates to model group-wise interactions, as these structures are a generalization of graphs where a (hyper)edge allows the connection of an arbitrary number of nodes.

In this line of work, we focus on analyzing social media conversational data and, specifically, on the task of stance detection.

In this preliminary study, we introduce an innovative approach to address this task, which combines the use of hypergraphs for abstracting conversational networks with the analytical power of LLMs to examine their dynamics. We opted to use LLMs based on compelling evidence from the literature demonstrating their rising dominance over graph neural networks as the predominant tool for graph analysis [148, 149, 190]. Additionally, LLMs have proven their efficacy in stance detection tasks, demonstrating versatility across datasets with varying prompting schemes and even surpassing supervised models in performance while demanding fewer resources [93]. While previous studies have explored the integration of LLMs and graphs for stance detection [67], they have overlooked group-wise interactions, which are recognized as pivotal in understanding many real-world phenomena [38]. Building upon these foundations, our objective is to leverage LLMs to address the stance detection task within online social media conversations represented as hypergraphs. We envision that this modeling approach has the potential to not only advance the current state-of-the-art but also deepen our comprehension of social media dynamics.

Problem definition. In more formal terms, our objective is to address the following task: given input represented as a conversational hypergraph and a target pair, our goal is to classify each comment within the conversation based on the author's stance toward the target. In our case, the stance is categorized into one of three labels: favor, against, or neutral.

Based on previous literature [39], we model a conversational hypernetwork as a Text-Attributed Hypergraph (TAH). A TAH is formally defined as a tuple $H = (V, E, T)$, where V is the node set representing the users of a conversation, E is the hyperedge set denoting the group-wise relationship of the users in the conversation and T is the comment set. Specifically, each node $v \in V$ is associated with a textual comment $t \in T$.

In our work, we aim to address the following research questions. First, we investigate whether model performance can benefit from integrating high-order interactions that capture nuanced local context in stance detection tasks. Second, given the scarcity of labeled conversational datasets, we explore the feasibility of using LLMs to generate ground truth data.

Methodology. To address the RQs outlined above, we devised the pipeline shown in Figure 3.19, and opted to focus on the

specific topic of climate change. Our initial step involved selecting a dataset of conversational threads. We began by choosing the SPINOS dataset [255], which features human annotations of Reddit comments across six distinct topics, including climate change. To enrich this dataset, we collected all messages from the three climate-related subreddits (`climate`, `climatechange`, `climateskeptics`) spanning from January 4th, 2019, to July 31st, 2020. After constructing the dataset ($|\text{messages}| = 230122$, $|\text{authors}| = |V| = 21496$, $|\text{submissions}| = |E| = 18771$), the next step involved assigning a stance to each unlabeled message. For this task, we employed Gemma 1.1 7B[6], using the same prompt as the human annotators who compiled the SPINOS dataset, by obtaining an accuracy of 68%. The second step involved building the TAH associated with the so-built dataset and feeding it to *HyperStance*, our proposed stance detection model. This model will then evaluate the stance attributed to a user by simultaneously learning the semantics of the textual description and the hypergraph structure, which encapsulates the high-order local context of a conversation. In the final stage of validation, we will assess our model by comparing it with the current state-of-the-art approaches, which either leverage the conversational structure or solely focus on individual pieces of text. The model, experiments, and data are freely available on GitHub[7].

Challenges. Our primary challenge lies in the absence of ground truth data against which we can assess the developed model. Current labeled datasets include only isolated text fragments, yet we require access to complete conversation threads to capture the high-order dynamics of online platforms. Leveraging pre-trained LLMs to construct ground truths for this purpose could bridge this gap, albeit potentially introducing bias. Moreover, handling (potentially large-scale) hypergraphs introduces an additional layer of computational complexity, which significantly challenges the scalability and efficiency of the developed pipeline.

This paper presented a novel approach to leverage LLMs to address the stance detection task within online social media conversations modeled as hypergraphs. Currently, we are working on enhancing the training dataset and implementing HyperStance.

6 https://ai.google.dev/gemma
7 https://github.com/ddevin96/HyperStance

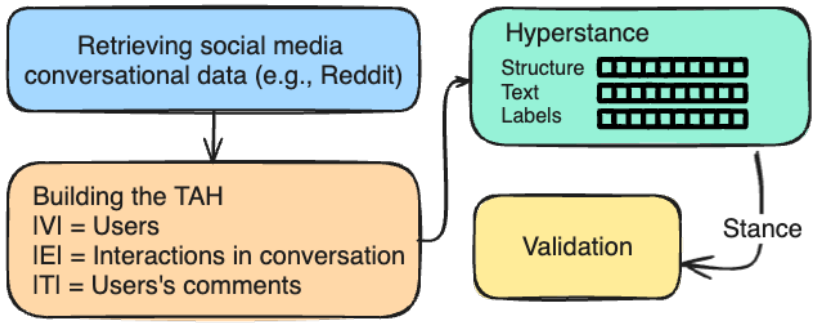


Figure 3.19: Overview of the pipeline.

3.5 MACHINE LEARNING TECHNIQUES TO BETTER UNDERSTAND NETWORK

Hypergraphs have emerged as a powerful tool for representing high-order connections in real-world complex systems. Similar to graphs, local structural patterns in hypergraphs, known as high-order motifs (h-motifs), play a crucial role in network dynamics and serve as fundamental building blocks across various domains. For this reason, predicting h-motifs can be highly beneficial in different fields. In this paper, we aim to advance our understanding of such complex high-order dynamics by introducing and formalizing the problem of h-motifs prediction. To address this task, we propose a novel solution that leverages both high-order and pairwise information by combining hypergraph and graph convolutions to capture hyperedges correlation within h-motifs, along with an innovative negative sampling approach designed to generate close-to-positive negative samples. To evaluate the effectiveness of our approach, we defined several baselines inspired by existing literature on hyperedge prediction methods. Our extensive experimental assessments demonstrate that our approach consistently outperforms all the considered baselines, showcasing its superior performance and robustness in predicting h-motifs.

h-Motif definition. The concept of h-motif was introduced by Lee et al. in [180]. In their work, the authors proposed 26 h-motifs describing all possible connectivity patterns among three connected hyperedges. They analyzed various real-world networks and compared them based on the normalized significance of each h-motif's occurrences, empirically demonstrating that

real-world hypergraphs from the same domain tend to exhibit similar h-motif occurrence patterns. Lee and Shin [182] extended this definition to include the temporal dimension, introducing 96 temporal hypergraph motifs to describe connectivity patterns among three connected temporal hyperedges. Another definition of h-motif is provided by Lotito et al. [193], who generalized the seminal notion and analysis of network motifs proposed by Milo et al. [211] to hypergraphs. In this context, h-motifs are defined as statistically over-expressed connected subgraphs comprising a given number of nodes, which can be interconnected by high-order interactions of arbitrary order. In our work, we leverage the definition from Lee et al. [180].

h-Motif analysis and prediction. Existing literature dealing with h-motifs has proved how these substructures identify unique local interaction patterns in different domains [180, 193], proposed exact and approximated algorithms for mining motifs in hypergraphs [180, 182–184, 192, 219], and exploited these substructures to improve downstream analytical tasks, such as link prediction [180, 183, 208].

Hyperedge prediction. Hyperedge prediction is the most similar task to h-motif prediction. The h-motif prediction task can be approached by using the representations of hyperedges learned from the hyperedge prediction task. However, predicting h-motifs in this manner may be suboptimal as this approach treats the hyperedge representations that form the h-motif independently, thus overlooking potential correlations and interactions among the hyperedges. In this work, we used the similarity-based method HPRA [174], and two representation learning-based approaches, namely VilLain [181] and N2V-HGCN. For a comprehensive review of existing hyperedge prediction methods, we refer the reader to [18, 78].

In this work, we aim to begin filling this gap by introducing and formalizing the problem of h-motifs prediction. Specifically, to tackle this task, we propose a novel neural networks-based approach that exploits both high-order and pairwise information by combining hypergraph and graph convolutions to capture hyperedges correlation within h-motifs. A key component of our solution is the generation of non-trivial negative samples, designed to generate close-to-positive negative samples. To assess the effectiveness of our approach, we defined several baselines inspired by existing hyperedge prediction techniques. Our

comprehensive experimental evaluations demonstrate that our method consistently outperforms all the baselines considered, highlighting its superior performance and robustness in h-motif prediction. In this work, we address the emerging challenge of h-motif prediction in hypergraphs by proposing and formalizing the task within a robust methodological framework. First, we clearly define the h-motif prediction problem, establishing formal foundations for further exploration. To strengthen the learning process, we introduce an innovative negative sampling strategy that generates complex negative examples by systematically replacing nodes in existing h-motifs using specific scoring functions. Furthermore, we present several new baseline methods for h-motif prediction, structured on the assumption of hyperedge independence, enabling fair and diverse benchmarks. Our primary contribution is a novel approach that learns latent representations of h-motifs while explicitly capturing correlations between hyperedges, thereby more accurately reflecting the complex relationships inherent in real-world networks. The effectiveness of our method is empirically validated through an extensive suite of experiments, consistently demonstrating superior performance and robustness compared to the proposed baselines. This work not only advances the field of hypergraph learning but also provides practical tools and datasets for the broader research community.

Reproducibility. The datasets and the implementation and experimental code are available at the following link: `https://github.com/hypernetwork-research-group/motif_representation_learning`.

Hypergraph motifs (or h-motifs) describe connectivity patterns between hyperedges. In this work, we focus on h-motifs involving three hyperedges as defined by Lee et al. in [180]. Specifically, each h-motif $m = (e_i, e_j, e_k)$, with $e_i, e_j, e_k \in E$, is defined by the emptiness of the following seven sets: [1] $e_i \setminus (e_j \cup e_k)$; [2] $e_j \setminus (e_k \cup e_i)$; [3] $e_k \setminus (e_i \cup e_j)$; [4] $(e_i \cap e_j) \setminus e_k$; [5] $(e_j \cap e_k) \setminus e_i$; [6] $(e_k \cap e_i) \setminus e_j$; [7] $e_i \cap e_j \cap e_k$. Trivially, if the h-motif m exists in a hypergraph then $e_i, e_j, e_k \in E$. Figure 3.1 illustrates two motif types: on the left, a star sub-network representing a traditional graph motif; on the right, an h-motif characterized by non-empty pairwise intersections. There exist 26 distinct types of h-motifs without duplicated hyperedges, which we identify using the MoCHy algorithm [180].

Hypergraphs embedding methods are techniques able to learn a low-dimensional representation of a hypergraph or parts of it, such as nodes, hyperedges, or substructures. Formally, for a given hypergraph $\mathcal{H}(V, E)$, a hypergraph embedding is a mapping function $\Psi : V \to \mathbb{R}^{|V| \times d}$, where $d \ll |V|$, such that Ψ defines the latent representation (a.k.a. embedding) of each node $v \in V$, which captures certain network topological information in E [18].

Hyperedge prediction aims to identify the most likely existent hyperlinks missing from an observed set E. Chen et al. [78] comprehensively review hyperlink prediction methods. As highlighted in their work, most methods aim to learn a scoring function $\phi : E \to \mathbb{R}$ such that a higher score indicates a higher likelihood of e being a missing hyperlink. When treating hyperedge prediction as a binary classification task, the score is binarized using a threshold ϵ. Thus, given a hyperedge e, e is predicted to be in E if $\phi(e) \geq \epsilon$.

The h-motif prediction problem.

Let $M_t \subseteq (2^V \setminus \{\emptyset\})^3$ be the set of *all possible h-motifs* of type t that can be formed by any three hyperedges involving nodes in V. Let $M_t^+ \subseteq \mathbb{M}$ be the subset containing all instances of type t *observed* in $\mathcal{H}$, and $\mathbb{M} \equiv \bigcup_{t=1}^{26} M_t^+$ be the set of all h-motifs in $\mathcal{H} = (V, E)$. Finally, let $\overline{\mathcal{H}} = (V, \overline{E})$ be a sub-hypergraph of $\mathcal{H}$, with $\overline{E} \subset E$, where the set of observed h-motif $\overline{M}_t^+$ in $\overline{\mathcal{H}}$ is strictly contained in M_t^+, i.e., some h-motifs of type t are missing in $\overline{\mathcal{H}}$.

> The **h-motif prediction problem** consists in predicting whether a given h-motif $m \in M_t \setminus \overline{M}_t^+$ will appear in $M_t^+ \setminus \overline{M}_t^+$ or not.

In this work, we frame this problem as a classification task where our goal is to learn a scoring function $\psi_{M_t} : M_t \to [0, 1]$, predicting the likelihood that a h-motif m of type t occurs in M_t^+. To get the final classification, i.e., predicting if m will be in M_t^+, we threshold the scoring function ψ_{M_t} in such a way that we predict $m \in M_t^+$ if $\psi_{M_t}(m) \geq \epsilon$, where $\epsilon \in [0, 1]$.

The h-motif prediction problem can be approached by either considering that each hyperedge exists independently from others or by accounting for correlations underlying their formation (see Figure 3.20). In the former case, the problem can be addressed as a hyperedge prediction task, where the probability

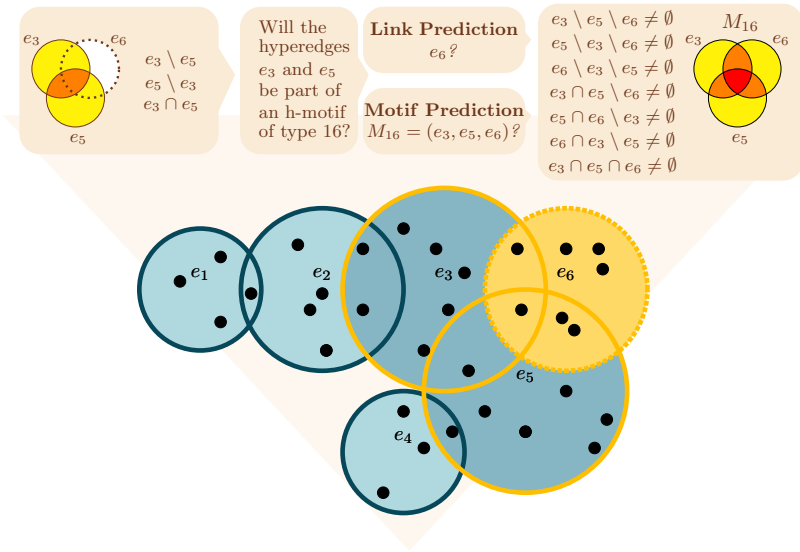


Figure 3.20: A high-level description of the h-motif prediction problem.

score ψ_{M_t} is equal to the product of the independent probabilities of each hyperedge's presence. However, this assumption is quite strong, as it suggests that the formation of hyperedges occurs independently, without any mutual influence. This scenario may be unrealistic for real-world systems, such as OSNs, where social patterns significantly affect node relationships [124]. The latter case does not assume any independence among hyperedges forming a motif. In this setting, evaluating the likelihood of a motif appearing becomes more complex.

Solving the h-motif prediction task under the assumption that hyperedges forming an h-motif will independently appear allows us to model the probability of an h-motif forming as the product of the probabilities of each individual hyperedge appearing. Formally, the h-motif scoring function for $m = (e_i, e_j, e_k)$, $e_i, e_j, e_k \in E$, can be expressed as the product of the scoring functions evaluated on each hyperedge, i.e.,

$$\psi_{M_t}(m) = \phi(e_i) \cdot \phi(e_j) \cdot \phi(e_k).$$

A straightforward consequence of this assumption is that we can leverage a wide range of hyperedge prediction methods available in the literature [78] to evaluate the probability that a single hyperedge will form. In our work, we considered two categories

of methods for estimating the score function $\phi(\cdot)$: *similarity*-based and *representation learning*-based methods.

Similarity-based methods *exploit the likeness between nodes in hyperedges.*

CN Common Neighbors [319] is a method initially used for pairwise link prediction on graphs, which can be easily extended to hyperedge prediction. Formally, let $\mathcal{N}(v) = \{u \in V \mid \{u, v\} \in \mathcal{E}\}$ be the neighborhood of a node v. The CN index between two nodes $v, u \in V$ can be computed as:

$$CN(v, u) = |\mathcal{N}(v) \cap \mathcal{N}(u)|,$$

which measures the number of common neighbors of v and u. CN can be applied to hyperedge prediction by aggregating (e.g., via the mean, max, min, product) the pairwise CN between all the nodes within a hyperedge. For instance, the CN index for a hyperedge $e \in E$ using the *mean* aggregation function can be computed as follows:

$$CN(e) = \frac{2}{|e|(|e|-1)} \sum_{v,u \in e} CN(v, u).$$

We similarly extend all the following similarity-based methods to compute hyperedge scores.

AA Adamic Adar [4] is an index originally proposed for link prediction in social networks. Similar to CN, it is based on the number of common neighbors between two nodes. Specifically, this index is evaluated as the sum of the inverse logarithmic frequency between common neighbors, i.e.,

$$AA(v, u) = \sum_{z \in \mathcal{N}(v) \cap \mathcal{N}(u)} \frac{1}{log|\mathcal{N}(z)|}$$

JC Jaccard Coefficient [151] is a statistical index used to compare sets of elements. This index is defined as

$$JC(v, u) = \frac{|\mathcal{N}(v) \cap \mathcal{N}(u)|}{|\mathcal{N}(v) \cup \mathcal{N}(u)|}$$

HPRA Hyperedge Prediction using Resource Allocation [174] is a method directly applicable on hypergraphs. HPRA computes a hypergraph resource allocation (HRA) index between nodes combining structural information. The HRA index between two nodes is defined as

$$HRA(v,u) = SD(v,u) + \sum_{s \in \mathcal{N}(u) \cap \mathcal{N}(v)} \frac{SD(u,s) \cdot SD(s,v)}{|\mathcal{N}(s)|},$$

where $SD(v,u) = \sum_{v,u \in e, v \neq u} \frac{1}{|e|-1}$ is the connection score.

Representation learning-based methods *are end-to-end techniques that learn a latent representation of the nodes/hyperedges while estimating the likelihood of a hyperlink's existence through the minimization of a loss function.*

N2V node2vec [131] is a very popular method based on the random-walk technique used to learn node embeddings. When applied to hyperlink prediction, *N2V* operates on the clique expansion of a hypergraph. This method computes scores by measuring the average similarity between nodes within the same hyperedge. Then, for each hyperedge $e \in E$, node embeddings are aggregated and transformed through a sigmoid function to produce the final hyperedge scores.

VL VilLain [181] is a self-supervised representation learning approach that operates directly on hypergraphs. Specifically, VL learns a sparse probability distribution of virtual labels propagated through the network. It produces node representations and uses a maximum pooling function to construct hyperedge embeddings. These embeddings are then processed through a single-layer perceptron to compute the final hyperedge scores.

N2V HGCN N2V with HyperGraph Convolutional Network [78] applies a convolutional operator directly on a hypergraph to learn latent node representations $\mathbf{Z}'$ from an initial representation $\mathbf{Z}$ which is computed using N2V. The convolutional layer is defined as

$$\mathbf{Z}' = \mathbf{L}\mathbf{Z}\boldsymbol{\theta}_H,$$

where $\mathbf{L}$ is the normalized Laplacian matrix of the hypergraph, and $\boldsymbol{\theta}_H$ are the learnable parameters of the layer. The resulting node representations $\mathbf{Z}'$ are combined through an aggregation function to generate hyperedge embeddings, which are then processed by a shallow multi-layer perceptron (MLP) for final classification.

To ensure comparable scores across different motifs, we normalize the scores deriving from similarity-based link prediction methods that produce values outside the range $[0, 1]$. Following Chen et al. [78], we normalize such score vector $\phi(\cdot)$ by computing $\phi(\cdot)/c$, where $c \geq \lceil |\phi(\cdot)|_\infty \rceil$.

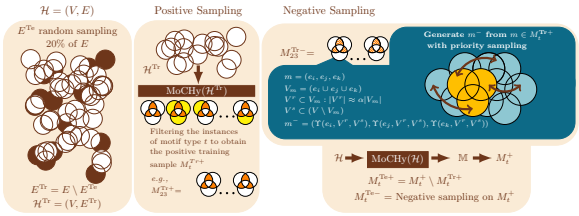


Figure 3.21: Overview of the positive and negative sampling strategy along with the training-test data splitting procedure.

In this section, we present our proposed solution that leverages both high-order and pairwise information to capture hyperedges correlation within h-motifs.

Defining a good negative sample is crucial for ensuring a model's effectiveness. Unlike the hyperedge prediction setting, building a negative h-motif means generating a collection of three hyperedges rather than a single one. As discussed in the following, generating such negative examples is a challenging task.

To build the training and test sets, we first split the hyperedge set E of the input hypergraph $\mathcal{H}$ into two subsets: (i) the test hyperedge set $E^{\text{Te}+}$, formed by randomly sampling hyperedges from E, and (ii) the training hyperedge set $E^{\text{Tr}+} = E \setminus E^{\text{Te}+}$. The training hypergraph can, hence, be defined as $\mathcal{H}^{\text{Tr}} = (V, E^{\text{Tr}+})$.

Given $\mathcal{H}^{\text{Tr}}$, we compute the set of all h-motif instances $\mathbb{M}$ using the *MoCHy* algorithm [180]. Since our prediction task focuses on

a specific type of h-motif t, we select the subset $M_t^{\text{Tr}+} \subseteq M_t^+ \subseteq \mathbb{M}$ as the set of positive motif samples of type t.

The negative sampling procedure requires a more sophisticated approach since it impacts the difficulty of the h-motifs prediction problem itself [232]. A straightforward approach to obtain negative h-motifs would be to randomly generate a negative set of hyperedges $\widehat{E}$, where each negative hyperedge is computed by randomly sampling nodes in V. Then, the negative h-motif sample set would be $M_t^{\text{Tr}-} = M_t^+ \setminus M_t^{\text{Tr}+}$, where M_t^+ is the set of h-motifs of type t computed on $\widehat{\mathcal{H}} = (V, E^{\text{Tr}+} \cup \widehat{E})$. However, this naïve approach would fail to ensure both the proportion of negative samples relative to positive ones as well as the "plausibility" of such negative examples. Our objective is, hence, to generate negative motifs that are non-trivial to classify; otherwise, we would risk underfitting issues. Therefore, we propose a novel negative sampling strategy that addresses these problems.

Figure 3.21 provides a schematic representation of our generation approach, denoted as $\text{NS}_m(\cdot)$. The methodology involves generating negative samples derived from positive instances by substituting a fraction α of nodes with different nodes from the network. This approach aligns with the methodology employed by Besta et al. [47] in predicting motifs in graphs. However, our technique introduces an improved node replacement selection mechanism specifically designed for hypergraphs, enabling us to obtain more plausible (i.e., close-to-positive) negative samples. Specifically, the nodes to be replaced are selected uniformly at random, while the replacement nodes are chosen according to a node-substitution score. These scores are computed based on the node's distance from the h-motif and the node degree, thereby prioritizing proximal nodes with higher degrees of connectivity. We define the following distances:

Distances between $v, u \in V$, $e \in E$, and $m \in M_t$

$\mathbf{v} \rightarrow \mathbf{u}$ $d_V(v,u) =$ number of hyperedges in the shortest path between v and u;

$\mathbf{v} \rightarrow \mathbf{e}$ $d_E(v,e) = min_{u \in e} d_V(u,v)$ (3.1)

$\mathbf{v} \rightarrow \mathbf{m}$ $d_M(v,m) = min_{e \in m} d_E(v,e)$.

Given an instance of an h-motif $m = (e_i, e_j, e_k)$, we denote with $V_m \equiv e_i \cup e_j \cup e_k$ the set of all nodes in the motif. To obtain the set of nodes to be replaced, we select uniformly at random a subset of nodes $V_r \subseteq V_m$ such that $|V_r| = \lfloor \alpha |V_m| \rfloor$, where $\alpha \in [0, 1)$ defines the proportion of nodes to be replaced. Then, we sample the substitution set $V_s \subseteq (V \setminus V_m)$ according to one of the following three mechanisms:

Uniformly at random

$$p_m(v) = \frac{1}{(|V \setminus V_m|)}$$

Probability-based

$$p_m(v) = \frac{1}{2}\left(\frac{\deg(v)}{\sum_{u \in V \setminus V_m} \deg(u)} + \frac{d_M(v,m)^{-1}}{\sum_{u \in V \setminus V_m} d_M(u,m)^{-1}}\right)$$

Rank-based

We deterministically select the $\lfloor \alpha |V_m| \rfloor$ nodes with the highest $p_m(v)$ according to the probability-based method.

Finally, a random matching function $\pi : V_r \rightarrow V_s$ is established, and we generate a new negative motif instance using the substitution procedure $\Upsilon(e, \pi)$. Specifically, Υ takes a hyperedge e and replaces each node $v \in V_r \cap e$ with the corresponding match $\pi(v) \in V_s$, thus forming the negative hyperedge e'. Therefore, given a positive motif $m = (e_i, e_j, e_k)$, its negative counterpart is defined as

$$m^- = (\Upsilon(e_i, \pi), \Upsilon(e_j, \pi), \Upsilon(e_k, \pi)).$$

For each motif $m \in M_t^+$, we generate $\beta \in \mathbb{N}^+$ negative samples, obtaining the set $M_t^- = M_t^{\mathrm{Tr}-} \cup M_t^{\mathrm{Te}-}$.

In essence, we build the following sets:

	Positive	Negative
Tr	$M_t^{\text{Tr}+} \subseteq M_t^+ \subseteq M_t$	$M_t^{\text{Tr}-} = \text{NS}_m(M_t^{\text{Tr}+})$
Te	$M_t^{\text{Te}+} = M_t^+ \setminus M_t^{\text{Tr}+}$	$M_t^{\text{Te}-} = \text{NS}_m(M_t^+)$ [8]

To generate the testing hypergraph $\mathcal{H}^{\text{Te}}$, we define the set of negative hyperedges $E^{\text{Te}-}$ as:

$$E^{\text{Te}-} = \bigcup_{\substack{e \in m \\ m \in M_t^-}} e,$$

such that $\mathcal{H}^{\text{Te}} = (V, E \cup E^{\text{Te}-})$.

Time complexity. The $v \to m$ distance can be computed using the *Floyd-Warshall Algorithm*, resulting in a time complexity of $O(|V|^3 + \beta|M_t^+||V|\log|V|)$. Appendix 3.5 provides further details.

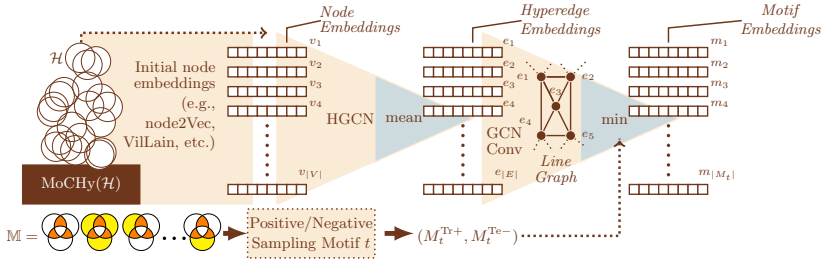


Figure 3.22: The overall architecture of the proposed HM learning framework. The input hypergraph $\mathcal{H}$ and the initial node embeddings are refined and combined using hypergraph convolution to generate hyperedge embeddings. These representations are further refined using graph convolution over the line graph derived from $\mathcal{H}$. Finally, the representations are aggregated to construct the h-motif embeddings.

Figure 3.22 provides an overview of our proposed **H**ypergraph **M**otifs representation learning architecture, which consists of three principal computational phases, sequentially pipelined: i) the *initial node embedding* step, which establishes a base representation for the nodes; ii) the *hyperedge representation learning* module, which refines the node embeddings and computes the

8 We use M_t to ensure that the generated negative samples are not present in $M_t^{\text{Tr}+}$.

hyperedge representations through a pooling operation; and iii) the *h-motifs representation learning* module, which leverages the line graph of the input hypergraph to generate h-motif embeddings.

Initial node embeddings. The first step of our pipeline is computing the *initial node representations* $\mathbf{X} \in \mathbb{R}^{|V| \times d}$, where $\mathbf{x}_i$ is the representation of the node $v_i \in V$ (for some arbitrary ordering of the nodes). These initial node embeddings can be learned using any hyper(graph) representation learning technique such as node2vec [131] (*N2V*) or ViLlain [181] (*VL*), just to name a few [18].

Hyperedge representation learning. The second part of the model is a Hypergraph Convolutional Neural Network [310] (*HGCN*), that takes in input the initial representation of the nodes $\mathbf{X}$ and the adjacency matrix of the hypergraph $\mathcal{H}$. The convolutional layers of HGCN improve the node representations by leveraging the message-passing framework over the hypergraph structure. Then, the hyperedge representations are computed using a mean pooling operation. Formally, a hypergraph convolutional layer is defined as

$$\mathbf{X}^{l+1} = \mathbf{L}\mathbf{X}^{l}\boldsymbol{\theta}_H^{l}, \quad l \in [0, L-1], \tag{3.2}$$

where L is the number of convolutional layers, $\mathbf{X}^0 = \mathbf{X}$, while $\mathbf{L}$ and $\boldsymbol{\theta}_H$ are the same as in *N2V HGCN* computed on $\mathcal{H}$.

The last HGCN layer is then followed by an MLP, which, in our implementation, consists of a single linear layer with LeakyReLU activation, i.e., the refined node representations are defined as

$$\widehat{\mathbf{X}} = \text{LeakyReLU}(\boldsymbol{\theta}_w \mathbf{X}^L + \boldsymbol{\theta}_b),$$

where $\boldsymbol{\theta}_w$ and $\boldsymbol{\theta}_b$ are learnable weights and biases, respectively. Finally, the node representations $\widehat{\mathbf{X}}$ are combined via a mean pooling operation to obtain the hyperedge representations, i.e.,

$$Z \equiv \left\{ \frac{1}{|e|} \sum_{v_i \in e} \hat{\mathbf{x}}_i \right\}_{e \in E},$$

which can be arranged in a matrix $\mathbf{Z} \in \mathbb{R}^{|E| \times d}$, for some arbitrary ordering of the hyperedges.

h-Motifs representation learning. The last module of our model is responsible for learning the h-motif representations. To achieve this goal, we rely on the line graph transformation of the input hypergraph $\mathcal{H}$. This transformation represents each hyperedge as a node and creates edges between overlapping hyperedges, enabling us to model their interactions explicitly. This approach seems particularly suitable for learning h-motifs, where hyperedges may exhibit strong interdependencies rather than occurring independently. We process this transformed graph using a Graph Convolutional Network (GCN) [169] to learn refined hyperedge representations that capture these structural correlations. These improved representations are then pooled to get the h-motif embeddings.

Formally, the hyperedge representations $\mathbf{Z}$ are enhanced via L' GCN layers as in the following

$$\mathbf{Z}^{l+1} = \widehat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{A}_{\mathcal{L}}\widehat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}^{l}\boldsymbol{\theta}_{G}^{l}, \quad l \in [0, L'-1],$$

where $\mathbf{A}_{\mathcal{L}} = \mathbf{H}^{\top}\mathbf{H}$ is the adjacency matrix of the line graph of $\mathcal{H}$, $\widehat{\mathbf{D}}$ is the nodes degree diagonal matrix, and $\boldsymbol{\theta}_{G}^{l}$ are the learnable weights of the GCN layers. Similarly to the previous module, the last graph convolutional layer is followed by an MLP. In this case, we also use a single LeakyReLU-activated linear layer, i.e.,

$$\widehat{\mathbf{Z}} = \text{LeakyReLU}(\boldsymbol{\theta}'_{w}\mathbf{Z}^{L'} + \boldsymbol{\theta}'_{b}),$$

where $\boldsymbol{\theta}'_{w}$ and $\boldsymbol{\theta}'_{b}$ are the learnable parameters of the linear layer. Finally, we build the h-motif representations $\mathbf{Y}$ by aggregating the refined hyperedge representations using the *min*-pooling operator:

$$\mathbf{Y} \in \mathbb{R}^{|M_t| \times d} \text{ s.t. } \quad \mathbf{y}_{ij} = \min_{e_k \in m_i \in M_t} \hat{z}_{kj}$$

Here, the pooling operation happens between the hyperedges, potentially forming the target motif. Although our specific task concerns h-motif with exactly three hyperedges involved, our architecture can also handle h-motifs with more than three hyperedges.

h-Motif prediction head. To get the final prediction scores for the h-motifs, i.e., the likelihood that a h-motif will appear in the hypergraph, we employ a standard MLP head with the sigmoid

as an activation function. This type of prediction head has also been used during the ablation study for each sub-module of our model. In our experiments, we used a single-layer perceptron as the head. We adopted the Binary Cross-Entropy as a loss function, and the training happens in an end-to-end fashion[9].

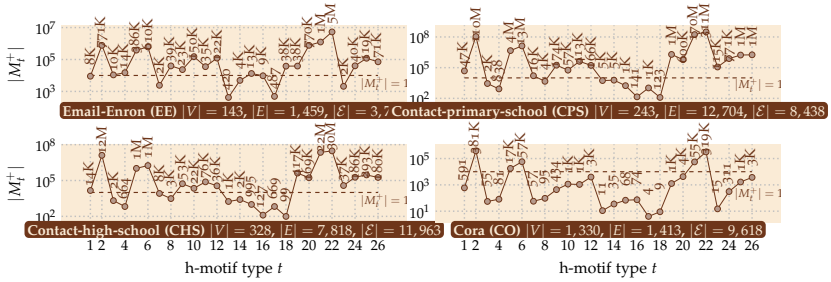


Figure 3.23: Datasets statistics—Number of nodes $|V|$, hyperedges $|E|$, clique expansion edges $|\mathcal{E}|$, and h-motifs (y-log scale).

In this section, we present the experimental results for the h-motifs prediction task, discussing the performance of our method HM using both $N2V$ and VL as initial embeddings. To the best of our knowledge, no other methods exist in the literature for learning h-motif embeddings or solving h-motif prediction tasks.
Experimental configuration. In our experiments, we employed the following training protocol for each dataset and model. We allocated 80% of the data for the training set and the remaining 20% for the test set. Networks were divided over hyperedges and we report the average results over three iterations. Due to computational constraints, we limited the h-motif positive sampling algorithm to a maximum of $|M_t^+| \leq 10K$ h-motifs, selected randomly. It is worth noting that, as shown in Figure 3.23, not all datasets provided the specified number of instances for all h-motif types. In such cases, the pipeline used all available instances. This aspect should be carefully considered when interpreting the results. To evaluate the models, we used the `AUROC`, a commonly used metric for binary classification problems and is commonly applied to the (hyper)edge prediction task [198]. Our

9 When used with the VL initial embedding, this step is performed first, and then the rest of the model is trained end-to-end.

experiments are reproducible using the code available in a public repository on GitHub[10].

Datasets. We ran our experiments on the following four real-world networks: (i) *Email-Enron* (EE) [170], representing email exchanges among employees of the Enron company; (ii) *Contact Primary School* (CPS) [273], modeling interactions between students and teachers; (iii) *Contact High School* (CHS) [203], modeling interactions between students; and (iv) *Cora* (CO) [205], a citation network. Specifically, hyperedges in these datasets are (i) employees involved in the same mail, (ii and iii) individuals interacting in groups, and (iv) papers cited by the same work. Figure 3.23 presents the statistical characteristics of these datasets, as well as their h-motif distribution. As a preprocessing step, we only filtered out hyperedges of size one.

In total, we run 4 *(datasets)* $\times$ 26 *(h-motif types t)* $\times$ 3 *(experiments)* $=$ 312 on a virtual cluster machine of 26 instances, each equipped with 8 vCPU, 16 GB of RAM, and 150 GB of storage.

In this experimental suite, we set $\beta = 1$ (# of negative samples generated from a positive sample), $\alpha = 0.5$ (proportion of nodes to be replaced in a positive sample), and $\eta = 0.01$ (learning rate) for all datasets except for *CO*, where $\eta = 0.0001$. In the following, we report the results using the rank-based node-substitution mechanism to build the negative h-motifs. Table 3.6 presents the AUROC scores across all datasets. To enhance clarity, we used a color-coded scheme: a color gradient marks the highest (green) and lowest (red) performances, while the best score per h-motif type is underlined. The last row of the table displays each model's score averaged over all h-motif types. For each h-motif type, we report the number of available positive instances $|M_t^+|$. We indicate this value with a ▲ symbol when the count exceeds $10K$, with ▼ otherwise.

First, we can observe that our method (either *HM N2V* or *HM VL*) achieves the best overall average performance across all datasets. Notably, both variants obtain comparable results on the *CPS* and *CHS* datasets, while *HM VL* outperforms other techniques on the *EE* dataset and *HM N2V* on the *CO* dataset. For the *CHS* dataset, we can also note that, on average, our method does not surpass *N2V HGCN* (a sub-module of *HM N2V*). This

10 https://github.com/hypernetwork-research-group/motif_representation_learning

suggests that, for this dataset, adding the final GCN module does not yield a significant improvement. However, it is worth emphasizing that for *HM VL*, the average result is significantly impacted by a single poor result, specifically for the h-motif type 22. For all other h-motif types, *HM VL* consistently achieves the best performance, albeit with a small margin.

Table 3.6: Performance comparison of prediction methods across all **26** h-motif types.

EE $\eta = 0.01$

t	$\lvert M_t\rvert \approx$	CC	AA	JC	HPRA	NaV	NaV HGCN	VL	HM NaV	HM VL
1	▼8K	0.45 ± 0.01	0.60 ± 0.01	0.70 ± 0.01	0.68 ± 0.02	0.74 ± 0.05	0.79 ± 0.01	0.88 ± 0.01	0.97 ± 0.01	0.96 ± 0.00
2	▲771K	0.38 ± 0.00	0.53 ± 0.01	0.51 ± 0.01	0.64 ± 0.04	0.82 ± 0.02	0.84 ± 0.01	0.89 ± 0.04	0.85 ± 0.08	0.95 ± 0.00
3	▲10K	0.44 ± 0.04	0.61 ± 0.04	0.73 ± 0.07	0.71 ± 0.03	0.77 ± 0.03	0.84 ± 0.00	0.90 ± 0.02	0.96 ± 0.03	0.98 ± 0.01
4	▲14K	0.53 ± 0.02	0.67 ± 0.00	0.77 ± 0.01	0.71 ± 0.01	0.73 ± 0.02	0.77 ± 0.01	0.95 ± 0.01	0.90 ± 0.07	0.97 ± 0.01
5	▲386K	0.44 ± 0.01	0.59 ± 0.01	0.63 ± 0.01	0.66 ± 0.01	0.80 ± 0.01	0.82 ± 0.00	0.93 ± 0.01	0.94 ± 0.04	0.97 ± 0.01
6	▲610K	0.47 ± 0.00	0.64 ± 0.00	0.67 ± 0.00	0.71 ± 0.03	0.80 ± 0.01	0.82 ± 0.01	0.95 ± 0.01	0.94 ± 0.04	0.97 ± 0.00
7	▼2K	0.33 ± 0.02	0.50 ± 0.03	0.67 ± 0.06	0.64 ± 0.06	0.87 ± 0.04	0.96 ± 0.01	0.92 ± 0.01	0.98 ± 0.01	0.98 ± 0.01
8	▲39K	0.48 ± 0.01	0.66 ± 0.02	0.72 ± 0.01	0.72 ± 0.04	0.76 ± 0.01	0.77 ± 0.03	0.92 ± 0.07	0.86 ± 0.10	0.96 ± 0.01
9	▲23K	0.39 ± 0.03	0.55 ± 0.01	0.70 ± 0.06	0.65 ± 0.02	0.89 ± 0.01	0.91 ± 0.00	0.90 ± 0.01	0.95 ± 0.04	0.98 ± 0.00
10	▲150K	0.54 ± 0.01	0.68 ± 0.02	0.75 ± 0.01	0.74 ± 0.01	0.74 ± 0.02	0.76 ± 0.03	0.97 ± 0.01	0.87 ± 0.08	0.95 ± 0.04
11	▲75K	0.45 ± 0.01	0.68 ± 0.04	0.76 ± 0.02	0.77 ± 0.03	0.81 ± 0.02	0.84 ± 0.02	0.88 ± 0.01	0.99 ± 0.01	0.98 ± 0.00
12	▲122K	0.58 ± 0.01	0.73 ± 0.02	0.81 ± 0.01	0.82 ± 0.02	0.75 ± 0.02	0.79 ± 0.02	0.92 ± 0.01	0.95 ± 0.04	0.96 ± 0.02
13	▼42O	0.31 ± 0.01	0.52 ± 0.04	0.71 ± 0.04	0.68 ± 0.05	0.83 ± 0.03	0.96 ± 0.01	0.91 ± 0.05	0.98 ± 0.01	0.97 ± 0.00
14	▲4K	0.39 ± 0.01	0.58 ± 0.02	0.75 ± 0.01	0.73 ± 0.05	0.85 ± 0.02	0.88 ± 0.00	0.86 ± 0.02	0.96 ± 0.03	0.98 ± 0.00
15	▲13K	0.56 ± 0.01	0.71 ± 0.01	0.81 ± 0.01	0.79 ± 0.03	0.72 ± 0.05	0.78 ± 0.01	0.93 ± 0.02	0.98 ± 0.02	0.98 ± 0.01
16	▼9K	0.70 ± 0.02	0.76 ± 0.01	0.88 ± 0.02	0.81 ± 0.01	0.68 ± 0.12	0.83 ± 0.02	0.87 ± 0.03	0.96 ± 0.03	0.97 ± 0.01
17	▼48Z	0.35 ± 0.00	0.53 ± 0.03	0.67 ± 0.02	0.59 ± 0.04	0.88 ± 0.03	0.97 ± 0.01	0.97 ± 0.00	0.98 ± 0.01	0.98 ± 0.01
18	▲8K	0.66 ± 0.04	0.80 ± 0.02	0.80 ± 0.04	0.68 ± 0.04	0.69 ± 0.05	0.72 ± 0.04	0.94 ± 0.03	0.87 ± 0.09	0.92 ± 0.04
19	▲38K	0.36 ± 0.02	0.51 ± 0.02	0.62 ± 0.06	0.57 ± 0.01	0.90 ± 0.02	0.93 ± 0.01	0.86 ± 0.03	0.95 ± 0.03	0.97 ± 0.01
20	▲770K	0.59 ± 0.02	0.71 ± 0.02	0.74 ± 0.03	0.70 ± 0.01	0.74 ± 0.02	0.76 ± 0.02	0.97 ± 0.00	0.96 ± 0.01	0.95 ± 0.02
21	▲1M	0.38 ± 0.01	0.54 ± 0.01	0.54 ± 0.04	0.58 ± 0.01	0.86 ± 0.00	0.87 ± 0.01	0.88 ± 0.04	0.96 ± 0.02	0.97 ± 0.00
22	▲5M	0.52 ± 0.03	0.67 ± 0.02	0.68 ± 0.04	0.71 ± 0.01	0.79 ± 0.01	0.81 ± 0.01	0.95 ± 0.01	0.95 ± 0.02	0.96 ± 0.00
23	▼2K	0.33 ± 0.01	0.48 ± 0.00	0.53 ± 0.02	0.58 ± 0.09	0.86 ± 0.04	0.96 ± 0.01	0.91 ± 0.04	0.97 ± 0.01	0.97 ± 0.01
24	▲40K	0.44 ± 0.02	0.59 ± 0.03	0.64 ± 0.04	0.62 ± 0.01	0.83 ± 0.00	0.86 ± 0.02	0.94 ± 0.03	0.93 ± 0.06	0.96 ± 0.01
25	▲119K	0.52 ± 0.00	0.68 ± 0.00	0.72 ± 0.01	0.73 ± 0.01	0.79 ± 0.02	0.81 ± 0.02	0.96 ± 0.00	0.93 ± 0.07	0.96 ± 0.02
26	▲71K	0.62 ± 0.01	0.75 ± 0.01	0.84 ± 0.02	0.80 ± 0.02	0.75 ± 0.01	0.81 ± 0.02	0.92 ± 0.01	0.94 ± 0.06	0.98 ± 0.01
	avg	0.47	0.62	0.71	0.69	0.79	0.84	0.92	0.94	0.97

CPS $\eta = 0.01$

$\lvert M_t\rvert \approx$	CC	AA	JC	HPRA	NaV	NaV HGCN	VL	HM NaV	HM VL
▲47K	0.40 ± 0.01	0.73 ± 0.01	0.70 ± 0.03	0.60 ± 0.03	0.91 ± 0.01	0.96 ± 0.01	0.94 ± 0.03	0.97 ± 0.01	0.97 ± 0.00
▲110M	0.29 ± 0.01	0.62 ± 0.02	0.45 ± 0.01	0.61 ± 0.02	0.95 ± 0.01	0.97 ± 0.00	0.95 ± 0.04	0.97 ± 0.01	0.98 ± 0.01
▼2K	0.37 ± 0.04	0.72 ± 0.03	0.66 ± 0.05	0.67 ± 0.04	0.84 ± 0.05	0.96 ± 0.01	0.95 ± 0.03	0.98 ± 0.01	0.97 ± 0.02
▼83K	0.38 ± 0.03	0.76 ± 0.03	0.66 ± 0.03	0.64 ± 0.01	0.79 ± 0.06	0.93 ± 0.01	0.96 ± 0.02	0.97 ± 0.01	0.96 ± 0.03
▲4M	0.32 ± 0.00	0.70 ± 0.02	0.55 ± 0.01	0.60 ± 0.01	0.95 ± 0.01	0.98 ± 0.01	0.95 ± 0.03	0.97 ± 0.03	0.98 ± 0.01
▲13M	0.35 ± 0.02	0.69 ± 0.02	0.61 ± 0.03	0.72 ± 0.04	0.85 ± 0.02	0.97 ± 0.01	0.95 ± 0.03	0.97 ± 0.00	0.97 ± 0.00
▲16K	0.36 ± 0.01	0.78 ± 0.01	0.64 ± 0.03	0.58 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.92 ± 0.03	0.96 ± 0.03	0.95 ± 0.03
▼4K	0.37 ± 0.01	0.75 ± 0.03	0.65 ± 0.02	0.57 ± 0.02	0.81 ± 0.02	0.95 ± 0.01	0.96 ± 0.03	0.96 ± 0.01	0.96 ± 0.02
▲174K	0.36 ± 0.01	0.76 ± 0.02	0.65 ± 0.01	0.64 ± 0.01	0.96 ± 0.01	0.97 ± 0.01	0.94 ± 0.01	0.98 ± 0.01	0.98 ± 0.01
▲57K	0.37 ± 0.02	0.72 ± 0.03	0.68 ± 0.01	0.66 ± 0.05	0.90 ± 0.01	0.96 ± 0.00	0.95 ± 0.03	0.97 ± 0.02	0.97 ± 0.02
▲413K	0.36 ± 0.01	0.71 ± 0.03	0.70 ± 0.02	0.75 ± 0.04	0.94 ± 0.01	0.98 ± 0.01	0.95 ± 0.03	0.98 ± 0.02	0.98 ± 0.02
▲166K	0.39 ± 0.01	0.71 ± 0.03	0.73 ± 0.01	0.74 ± 0.03	0.87 ± 0.03	0.97 ± 0.00	0.97 ± 0.02	0.98 ± 0.01	0.99 ± 0.01
▼5K	0.37 ± 0.02	0.77 ± 0.01	0.71 ± 0.01	0.78 ± 0.01	0.87 ± 0.06	0.98 ± 0.01	0.96 ± 0.01	0.97 ± 0.01	0.98 ± 0.01
▼5K	0.37 ± 0.02	0.77 ± 0.01	0.71 ± 0.01	0.78 ± 0.01	0.87 ± 0.06	0.98 ± 0.01	0.96 ± 0.01	0.97 ± 0.01	0.98 ± 0.01
▼1K	0.37 ± 0.03	0.70 ± 0.05	0.72 ± 0.03	0.79 ± 0.04	0.88 ± 0.03	0.98 ± 0.01	0.97 ± 0.01	0.98 ± 0.00	0.98 ± 0.01
▼141	0.36 ± 0.03	0.70 ± 0.05	0.69 ± 0.04	0.77 ± 0.03	0.89 ± 0.04	0.90 ± 0.04	0.97 ± 0.01	0.95 ± 0.03	0.95 ± 0.04
▼1K	0.34 ± 0.01	0.74 ± 0.01	0.63 ± 0.02	0.52 ± 0.02	0.86 ± 0.02	0.97 ± 0.01	0.96 ± 0.01	0.99 ± 0.00	0.99 ± 0.01
▼12O	0.46 ± 0.04	0.76 ± 0.13	0.81 ± 0.04	0.69 ± 0.04	0.76 ± 0.08	0.72 ± 0.09	0.95 ± 0.03	0.86 ± 0.07	0.86 ± 0.02
▲1M	0.32 ± 0.00	0.69 ± 0.01	0.55 ± 0.00	0.60 ± 0.01	0.96 ± 0.01	0.97 ± 0.02	0.93 ± 0.03	0.97 ± 0.02	0.97 ± 0.03
▲600K	0.34 ± 0.01	0.70 ± 0.03	0.59 ± 0.01	0.62 ± 0.01	0.93 ± 0.02	0.97 ± 0.01	0.98 ± 0.02	0.99 ± 0.01	0.99 ± 0.01
▲170M	0.27 ± 0.01	0.61 ± 0.01	0.43 ± 0.01	0.58 ± 0.04	0.96 ± 0.00	0.98 ± 0.00	0.95 ± 0.02	0.97 ± 0.02	0.97 ± 0.01
▲311M	0.28 ± 0.01	0.64 ± 0.03	0.50 ± 0.02	0.63 ± 0.02	0.94 ± 0.01	0.97 ± 0.00	0.96 ± 0.02	0.98 ± 0.01	0.99 ± 0.01
▲155K	0.29 ± 0.01	0.68 ± 0.01	0.49 ± 0.02	0.56 ± 0.03	0.97 ± 0.01	0.98 ± 0.01	0.94 ± 0.03	0.97 ± 0.01	0.97 ± 0.01
▲771K	0.31 ± 0.01	0.68 ± 0.01	0.54 ± 0.02	0.60 ± 0.04	0.96 ± 0.01	0.97 ± 0.01	0.97 ± 0.02	0.98 ± 0.01	0.98 ± 0.01
▲1M	0.34 ± 0.01	0.70 ± 0.02	0.63 ± 0.00	0.63 ± 0.03	0.95 ± 0.02	0.97 ± 0.01	0.96 ± 0.02	0.97 ± 0.01	0.97 ± 0.01
▲1M	0.37 ± 0.02	0.77 ± 0.01	0.71 ± 0.01	0.78 ± 0.01	0.87 ± 0.06	0.98 ± 0.01	0.96 ± 0.01	0.97 ± 0.01	0.98 ± 0.01
	0.35	0.71	0.63	0.66	0.90	0.96	0.95	0.97	0.97

CHS $\eta = 0.01$

t	$\lvert M_t\rvert \approx$	CC	AA	JC	HPRA	NaV	NaV HGCN	VL	HM NaV	HM VL
1	▲14K	0.69 ± 0.02	0.80 ± 0.02	0.90 ± 0.01	0.68 ± 0.02	0.79 ± 0.04	0.97 ± 0.01	0.94 ± 0.00	0.98 ± 0.01	0.98 ± 0.01
2	▲381K	0.66 ± 0.01	0.71 ± 0.01	0.80 ± 0.01	0.73 ± 0.02	0.89 ± 0.05	0.97 ± 0.00	0.93 ± 0.01	0.98 ± 0.01	0.99 ± 0.00
3	▼2K	0.67 ± 0.02	0.79 ± 0.01	0.87 ± 0.01	0.71 ± 0.01	0.80 ± 0.07	0.98 ± 0.00	0.97 ± 0.01	0.98 ± 0.01	0.99 ± 0.00
4	▼66M	0.77 ± 0.03	0.84 ± 0.02	0.91 ± 0.01	0.78 ± 0.01	0.76 ± 0.01	0.97 ± 0.01	0.95 ± 0.01	0.99 ± 0.00	0.99 ± 0.00
5	▲1M	0.68 ± 0.01	0.78 ± 0.01	0.85 ± 0.01	0.70 ± 0.01	0.82 ± 0.01	0.98 ± 0.00	0.92 ± 0.01	0.99 ± 0.00	0.99 ± 0.00
6	▲1M	0.73 ± 0.01	0.80 ± 0.01	0.87 ± 0.00	0.80 ± 0.02	0.85 ± 0.01	0.97 ± 0.00	0.93 ± 0.00	0.95 ± 0.05	0.99 ± 0.00
7	▼8K	0.66 ± 0.01	0.79 ± 0.01	0.87 ± 0.01	0.64 ± 0.03	0.86 ± 0.03	0.99 ± 0.00	0.93 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
8	▼2K	0.68 ± 0.02	0.81 ± 0.01	0.87 ± 0.01	0.65 ± 0.01	0.86 ± 0.04	0.98 ± 0.01	0.97 ± 0.01	0.99 ± 0.00	0.99 ± 0.01
9	▲53K	0.71 ± 0.01	0.89 ± 0.00	0.77 ± 0.03	0.81 ± 0.01	0.98 ± 0.00	0.94 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.00
10	▲22K	0.76 ± 0.02	0.85 ± 0.01	0.90 ± 0.00	0.77 ± 0.03	0.83 ± 0.02	0.98 ± 0.00	0.95 ± 0.01	0.99 ± 0.00	0.99 ± 0.00
11	▲76K	0.79 ± 0.01	0.86 ± 0.01	0.93 ± 0.00	0.85 ± 0.03	0.83 ± 0.03	0.98 ± 0.00	0.95 ± 0.01	0.99 ± 0.00	0.98 ± 0.01
12	▲8K	0.82 ± 0.02	0.87 ± 0.01	0.93 ± 0.01	0.90 ± 0.02	0.87 ± 0.04	0.98 ± 0.00	0.97 ± 0.00	1.00 ± 0.00	0.99 ± 0.00
13	▼1K	0.70 ± 0.01	0.81 ± 0.01	0.91 ± 0.01	0.82 ± 0.01	0.87 ± 0.09	0.98 ± 0.00	0.97 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
14	▼2K	0.77 ± 0.03	0.85 ± 0.02	0.93 ± 0.01	0.85 ± 0.01	0.85 ± 0.04	0.99 ± 0.00	0.96 ± 0.01	0.99 ± 0.00	0.99 ± 0.00
15	▼39S	0.85 ± 0.00	0.89 ± 0.00	0.95 ± 0.00	0.92 ± 0.00	0.85 ± 0.07	0.98 ± 0.00	0.96 ± 0.02	0.99 ± 0.00	0.99 ± 0.00
16	▼2T	0.86 ± 0.02	0.87 ± 0.00	0.96 ± 0.01	0.88 ± 0.01	0.78 ± 0.03	0.96 ± 0.02	0.98 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
17	▼66Y	0.67 ± 0.01	0.77 ± 0.01	0.88 ± 0.01	0.69 ± 0.07	0.89 ± 0.03	0.98 ± 0.01	0.96 ± 0.03	0.99 ± 0.01	0.99 ± 0.01
18	▼9Y	0.59 ± 0.03	0.73 ± 0.01	0.84 ± 0.06	0.64 ± 0.03	0.82 ± 0.10	0.93 ± 0.04	0.94 ± 0.07	0.96 ± 0.04	0.97 ± 0.04
19	▲417K	0.66 ± 0.01	0.76 ± 0.01	0.83 ± 0.01	0.67 ± 0.03	0.90 ± 0.02	0.98 ± 0.00	0.93 ± 0.02	0.99 ± 0.00	0.99 ± 0.00
20	▲69K	0.71 ± 0.01	0.79 ± 0.00	0.85 ± 0.00	0.75 ± 0.01	0.86 ± 0.03	0.98 ± 0.00	0.97 ± 0.01	0.99 ± 0.00	0.99 ± 0.00
21	▲22M	0.64 ± 0.00	0.68 ± 0.00	0.79 ± 0.00	0.69 ± 0.02	0.92 ± 0.04	0.98 ± 0.00	0.91 ± 0.03	0.95 ± 0.05	0.98 ± 0.00
22	▲30M	0.71 ± 0.00	0.75 ± 0.01	0.84 ± 0.00	0.75 ± 0.03	0.88 ± 0.03	0.97 ± 0.00	0.92 ± 0.02	0.99 ± 0.00	0.84 ± 0.20
23	▲7K	0.67 ± 0.00	0.66 ± 0.00	0.87 ± 0.00	0.61 ± 0.02	0.97 ± 0.01	0.99 ± 0.00	0.93 ± 0.02	0.98 ± 0.00	0.98 ± 0.00
24	▲186K	0.73 ± 0.01	0.74 ± 0.01	0.88 ± 0.01	0.76 ± 0.02	0.88 ± 0.02	0.98 ± 0.00	0.96 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
25	▲293K	0.79 ± 0.01	0.83 ± 0.00	0.92 ± 0.01	0.79 ± 0.02	0.83 ± 0.01	0.97 ± 0.00	0.96 ± 0.01	0.99 ± 0.00	0.97 ± 0.02
26	▲190K	0.85 ± 0.01	0.89 ± 0.01	0.95 ± 0.00	0.91 ± 0.02	0.90 ± 0.05	0.97 ± 0.00	0.96 ± 0.01	0.99 ± 0.00	0.99 ± 0.00
	avg	0.73	0.80	0.89	0.76	0.85	0.98	0.95	0.98	0.98

CO $\eta = 0.0001$

$\lvert M_t\rvert \approx$	CC	AA	JC	HPRA	NaV	NaV HGCN	VL	HM NaV	HM VL
▼31K	0.66 ± 0.02	0.70 ± 0.02	0.87 ± 0.02	0.16 ± 0.04	0.71 ± 0.10	0.80 ± 0.05	0.79 ± 0.05	0.93 ± 0.04	0.90 ± 0.07
▲71K	0.33 ± 0.03	0.58 ± 0.03	0.51 ± 0.01	0.45 ± 0.01	0.95 ± 0.01	0.96 ± 0.01	0.70 ± 0.06	0.96 ± 0.03	0.98 ± 0.01
▼5S	0.68 ± 0.06	0.76 ± 0.06	0.95 ± 0.03	0.07 ± 0.04	0.79 ± 0.02	0.82 ± 0.03	0.96 ± 0.05	0.94 ± 0.03	0.93 ± 0.01
▼8Z	0.73 ± 0.01	0.74 ± 0.08	0.95 ± 0.00	0.09 ± 0.02	0.71 ± 0.01	0.71 ± 0.02	0.93 ± 0.02	0.95 ± 0.02	0.93 ± 0.01
▲27K	0.49 ± 0.01	0.64 ± 0.02	0.68 ± 0.01	0.36 ± 0.01	0.86 ± 0.03	0.90 ± 0.02	0.80 ± 0.05	0.97 ± 0.03	0.98 ± 0.01
▲57K	0.46 ± 0.01	0.60 ± 0.00	0.71 ± 0.00	0.26 ± 0.00	0.90 ± 0.03	0.95 ± 0.01	0.83 ± 0.05	0.97 ± 0.01	0.99 ± 0.01
▼57	0.70 ± 0.06	0.85 ± 0.04	0.90 ± 0.00	0.21 ± 0.01	0.88 ± 0.01	0.94 ± 0.02	0.96 ± 0.02	0.92 ± 0.02	0.90 ± 0.02
▼9S	0.65 ± 0.07	0.77 ± 0.06	0.84 ± 0.08	0.21 ± 0.03	0.85 ± 0.03	0.84 ± 0.06	0.96 ± 0.02	0.90 ± 0.02	0.90 ± 0.01
▼31M	0.58 ± 0.03	0.68 ± 0.04	0.86 ± 0.03	0.21 ± 0.01	0.77 ± 0.02	0.89 ± 0.00	0.81 ± 0.01	0.93 ± 0.02	0.92 ± 0.01
▼1K	0.63 ± 0.03	0.71 ± 0.04	0.87 ± 0.03	0.13 ± 0.04	0.79 ± 0.02	0.88 ± 0.02	0.80 ± 0.03	0.97 ± 0.01	0.96 ± 0.02
▼1K	0.61 ± 0.01	0.67 ± 0.03	0.87 ± 0.01	0.12 ± 0.01	0.77 ± 0.04	0.92 ± 0.01	0.64 ± 0.03	0.96 ± 0.00	0.95 ± 0.00
▼3K	0.65 ± 0.04	0.65 ± 0.07	0.89 ± 0.02	0.09 ± 0.01	0.89 ± 0.06	0.95 ± 0.00	0.69 ± 0.07	0.99 ± 0.00	0.98 ± 0.00
▼1I	0.78 ± 0.08	0.78 ± 0.07	0.99 ± 0.01	0.05 ± 0.04	0.73 ± 0.05	0.84 ± 0.09	0.87 ± 0.06	0.95 ± 0.01	0.80 ± 0.08
▼3S	0.69 ± 0.04	0.83 ± 0.13	0.97 ± 0.02	0.02 ± 0.01	0.75 ± 0.02	0.69 ± 0.05	0.84 ± 0.07	0.95 ± 0.02	0.93 ± 0.01
▼6I	0.62 ± 0.01	0.75 ± 0.04	0.94 ± 0.01	0.06 ± 0.01	0.86 ± 0.03	0.72 ± 0.09	0.91 ± 0.07	0.95 ± 0.02	0.94 ± 0.02
▼7K	0.76 ± 0.06	0.75 ± 0.13	0.94 ± 0.05	0.05 ± 0.02	0.79 ± 0.13	0.89 ± 0.04	0.66 ± 0.03	0.98 ± 0.01	0.95 ± 0.03
▼4	0.21 ± 0.16	0.50 ± 0.20	0.71 ± 0.21	0.17 ± 0.24	0.75 ± 0.20	0.92 ± 0.12	0.83 ± 0.24	0.58 ± 0.42	0.75 ± 0.02
▼9	0.86 ± 0.10	0.90 ± 0.14	0.96 ± 0.05	0.22 ± 0.09	0.90 ± 0.11	0.85 ± 0.14	0.83 ± 0.14	0.86 ± 0.10	0.62 ± 0.20
▼1K	0.46 ± 0.02	0.57 ± 0.01	0.72 ± 0.01	0.20 ± 0.02	0.86 ± 0.03	0.91 ± 0.03	0.78 ± 0.07	0.95 ± 0.01	0.95 ± 0.00
▼1K	0.47 ± 0.02	0.61 ± 0.02	0.72 ± 0.02	0.16 ± 0.01	0.89 ± 0.02	0.93 ± 0.01	0.69 ± 0.08	0.96 ± 0.02	0.95 ± 0.00
▲55K	0.47 ± 0.02	0.61 ± 0.02	0.72 ± 0.02	0.16 ± 0.01	0.89 ± 0.02	0.93 ± 0.01	0.69 ± 0.08	0.96 ± 0.05	0.95 ± 0.01
▲319K	0.39 ± 0.01	0.56 ± 0.01	0.63 ± 0.01	0.16 ± 0.01	0.88 ± 0.01	0.93 ± 0.01	0.89 ± 0.02	0.96 ± 0.01	0.96 ± 0.01
▼1S	0.75 ± 0.13	0.72 ± 0.13	0.93 ± 0.05	0.21 ± 0.05	0.91 ± 0.04	0.87 ± 0.04	0.91 ± 0.02	0.96 ± 0.05	0.88 ± 0.05
▼31I	0.51 ± 0.04	0.63 ± 0.05	0.77 ± 0.02	0.19 ± 0.02	0.83 ± 0.04	0.85 ± 0.02	0.88 ± 0.03	0.92 ± 0.00	0.92 ± 0.01
▼1K	0.51 ± 0.00	0.61 ± 0.03	0.79 ± 0.02	0.12 ± 0.01	0.82 ± 0.01	0.88 ± 0.02	0.77 ± 0.02	0.96 ± 0.00	0.94 ± 0.01
▼3K	0.53 ± 0.01	0.59 ± 0.03	0.85 ± 0.02	0.06 ± 0.01	0.89 ± 0.04	0.96 ± 0.00	0.56 ± 0.05	0.99 ± 0.00	0.98 ± 0.00
	0.58	0.68	0.83	0.16	0.83	0.87	0.81	0.94	0.92

Among the similarity-based baselines, *JC* is the best performing one, except for the *CPS* dataset, where both *AA* and *HPRA* achieve higher AUROC scores. However, as discussed in the following, *JC* also proves to be a very strong baseline when combined with other node-substitution mechanisms. In Appendix 3.5, we present the accuracy and F1 scores achieved on the *EE* dataset, while in Appendix 3.5, we discuss experiments varying the negative sampling node-substitution mechanisms. These experimental suites revealed performance variations between *similarity* and *representation learning*-based methods. In particular, with the *random* and *probability-based* node-substitution mechanisms, the similarity method *JC* performs exceptionally well, achieving an average AUROC of 0.92 across all h-motif types. However, with the *rank-*

based mechanism, our method obtains the highest average AUROC of 0.97 across all types. Although not directly comparable, when considering the best results across all mechanisms, our approach significantly surpasses the performance of *JC*. Detailed analyses of these findings are provided in Appendix 3.5. Still, it is crucial to recognize a critical limitation of $\perp\!\!\!\perp_m$ solutions, like *JC*, which assume hyperedge independence within h-motifs. This assumption fails to capture the inherent dependencies between hyperedges in real-world scenarios, making these approaches unsuitable for many applications.

Finally, regarding the different number of positive instances across h-motif types, our model demonstrates strong performance, consistently achieving scores between 0.92 and 1 when considering the best values from *HM VL* and *HM N2V*. However, in some cases, when the number of positive h-motif instances is significantly low, performance decreases. This behavior is expected, as it is well-known that deep neural networks require an adequate number of training instances to be trained effectively.

In this section, we empirically show that all three main submodules of *HM* play a key role in the method's success. To achieve this, we evaluated the model using only the initial node representations (i.e., *N2V* and *VL*), the model without the final GCN (namely, *N2V HGCN*), and the complete model (*HM N2V/VL*). To compute the prediction for the submodules, we added pooling layers to obtain hyperedges and/or motif representations, which are then used to compute the final score using the same MLP head. The results of all these models are shown in Table 3.6 and 3.7, specifically columns *N2V*, *VL*, *N2V HGCN*, *HMN2V*, and *HMVL*. The results clearly show that, with very few exceptions, the inclusion of each submodule consistently improves performance. We also conducted an ablation study where we removed it while preserving all other components of the pipeline, including the aggregation method. We replicated the experiment from Table 3.6 for motif type 2 on the *EE* dataset, using *VL* as the initial node feature. The results showed a significant performance drop, with an average AUC of 0.73 ($\pm$0.0087) and values ranging from 0.72 to 0.74, compared to the full pipeline's AUC of 0.95. This substantial difference demonstrates the crucial role of the HGCN component in our architecture's performance.

Additional details on results.

Table 3.7: Performance comparison of h-motif prediction across different motif types using ACC and F1 scores on the Email-Enron dataset.

ACC

t	$\lvert M_t \rvert \sim$	CC	AA	JC	HPRA	N2V	N2V HGCN	VL	**HM N2V**	**HM VL**
1	▼8K	0.50 ± 0.03	0.70 ± 0.01	0.65 ± 0.01	0.64 ± 0.01	0.56 ± 0.05	0.66 ± 0.11	0.79 ± 0.03	0.91 ± 0.02	0.90 ± 0.00
2	▲771K	0.46 ± 0.02	0.64 ± 0.01	0.53 ± 0.00	0.61 ± 0.03	0.61 ± 0.10	0.60 ± 0.07	0.72 ± 0.13	0.67 ± 0.16	0.86 ± 0.02
3	▲10K	0.48 ± 0.03	0.71 ± 0.01	0.66 ± 0.06	0.65 ± 0.02	0.62 ± 0.09	0.78 ± 0.00	0.85 ± 0.01	0.88 ± 0.05	0.92 ± 0.02
4	▲14k	0.52 ± 0.01	0.73 ± 0.01	0.70 ± 0.01	0.64 ± 0.02	0.55 ± 0.02	0.71 ± 0.00	0.89 ± 0.02	0.84 ± 0.09	0.91 ± 0.02
5	▲386K	0.50 ± 0.01	0.68 ± 0.01	0.60 ± 0.01	0.62 ± 0.01	0.66 ± 0.06	0.71 ± 0.01	0.88 ± 0.01	0.86 ± 0.09	0.90 ± 0.01
6	▲610K	0.49 ± 0.01	0.72 ± 0.00	0.64 ± 0.00	0.66 ± 0.03	0.71 ± 0.03	0.68 ± 0.07	0.87 ± 0.03	0.87 ± 0.07	0.91 ± 0.01
7	▼2K	0.52 ± 0.01	0.70 ± 0.02	0.62 ± 0.03	0.60 ± 0.05	0.52 ± 0.00	0.91 ± 0.01	0.83 ± 0.02	0.92 ± 0.01	0.91 ± 0.03
8	▲39K	0.54 ± 0.01	0.75 ± 0.02	0.68 ± 0.01	0.67 ± 0.01	0.68 ± 0.02	0.62 ± 0.09	0.88 ± 0.07	0.81 ± 0.08	0.89 ± 0.06
9	▲23K	0.50 ± 0.02	0.45 ± 0.01	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.73 ± 0.04	0.79 ± 0.01	0.85 ± 0.08	0.95 ± 0.01
10	▲150K	0.54 ± 0.01	0.75 ± 0.02	0.68 ± 0.01	0.67 ± 0.01	0.68 ± 0.02	0.62 ± 0.09	0.88 ± 0.07	0.81 ± 0.08	0.89 ± 0.06
11	▲35K	0.46 ± 0.02	0.76 ± 0.02	0.72 ± 0.02	0.69 ± 0.02	0.72 ± 0.02	0.76 ± 0.01	0.85 ± 0.02	0.94 ± 0.02	0.94 ± 0.01
12	▲122K	0.58 ± 0.01	0.77 ± 0.01	0.74 ± 0.01	0.72 ± 0.01	0.70 ± 0.02	0.61 ± 0.08	0.90 ± 0.01	0.87 ± 0.07	0.89 ± 0.03
13	▼420	0.47 ± 0.05	0.70 ± 0.03	0.65 ± 0.06	0.63 ± 0.03	0.58 ± 0.07	0.89 ± 0.04	0.74 ± 0.14	0.93 ± 0.01	0.93 ± 0.01
14	▲4K	0.46 ± 0.04	0.70 ± 0.02	0.68 ± 0.01	0.66 ± 0.03	0.50 ± 0.00	0.77 ± 0.06	0.78 ± 0.08	0.89 ± 0.09	0.93 ± 0.00
15	▲13K	0.56 ± 0.01	0.75 ± 0.02	0.73 ± 0.01	0.70 ± 0.03	0.65 ± 0.07	0.67 ± 0.08	0.90 ± 0.01	0.93 ± 0.04	0.94 ± 0.02
16	▼9K	0.66 ± 0.01	0.77 ± 0.02	0.81 ± 0.02	0.72 ± 0.01	0.56 ± 0.06	0.72 ± 0.03	0.82 ± 0.04	0.91 ± 0.04	0.92 ± 0.03
17	▼487	0.52 ± 0.01	0.72 ± 0.02	0.62 ± 0.01	0.57 ± 0.04	0.53 ± 0.04	0.92 ± 0.01	0.89 ± 0.01	0.94 ± 0.01	0.92 ± 0.01
18	▲38K	0.54 ± 0.03	0.67 ± 0.03	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.56 ± 0.01	0.78 ± 0.10	0.76 ± 0.15	0.84 ± 0.07
19	▲38K	0.49 ± 0.01	0.43 ± 0.01	0.49 ± 0.00	0.49 ± 0.00	0.49 ± 0.00	0.68 ± 0.04	0.76 ± 0.01	0.86 ± 0.07	0.91 ± 0.02
20	▲770K	0.58 ± 0.01	0.74 ± 0.02	0.69 ± 0.02	0.65 ± 0.01	0.62 ± 0.05	0.62 ± 0.09	0.90 ± 0.01	0.88 ± 0.01	0.89 ± 0.02
21	▲1M	0.47 ± 0.04	0.65 ± 0.01	0.54 ± 0.02	0.57 ± 0.01	0.73 ± 0.04	0.72 ± 0.06	0.63 ± 0.13	0.89 ± 0.04	0.91 ± 0.00
22	▲5M	0.52 ± 0.03	0.72 ± 0.01	0.64 ± 0.03	0.66 ± 0.00	0.69 ± 0.03	0.65 ± 0.10	0.88 ± 0.01	0.88 ± 0.04	0.88 ± 0.01
23	▲2K	0.53 ± 0.01	0.64 ± 0.01	0.57 ± 0.00	0.58 ± 0.03	0.60 ± 0.11	0.90 ± 0.02	0.84 ± 0.03	0.86 ± 0.03	0.89 ± 0.03
24	▲40K	0.52 ± 0.01	0.70 ± 0.02	0.62 ± 0.03	0.59 ± 0.01	0.65 ± 0.06	0.77 ± 0.02	0.65 ± 0.09	0.81 ± 0.15	0.90 ± 0.00
25	▲119K	0.54 ± 0.01	0.74 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.65 ± 0.10	0.70 ± 0.07	0.85 ± 0.05	0.80 ± 0.19	0.90 ± 0.03
26	▲71K	0.61 ± 0.02	0.78 ± 0.01	0.78 ± 0.03	0.70 ± 0.02	0.61 ± 0.08	0.67 ± 0.01	0.88 ± 0.02	0.80 ± 0.21	0.92 ± 0.01
	avg	0.52	0.70	0.64	0.63	0.61	0.72	0.82	0.86	🏆 **0.91**

F1

1	▼8K	0.58 ± 0.12	0.77 ± 0.00	0.68 ± 0.00	0.68 ± 0.01	0.42 ± 0.16	0.70 ± 0.04	0.80 ± 0.03	0.91 ± 0.02	0.90 ± 0.00
2	▲771K	0.55 ± 0.12	0.70 ± 0.01	0.60 ± 0.02	0.65 ± 0.03	0.72 ± 0.04	0.71 ± 0.03	0.79 ± 0.07	0.76 ± 0.10	0.86 ± 0.04
3	▲10K	0.63 ± 0.05	0.78 ± 0.01	0.67 ± 0.06	0.67 ± 0.02	0.66 ± 0.04	0.75 ± 0.02	0.86 ± 0.00	0.88 ± 0.05	0.92 ± 0.02
4	▲14K	0.60 ± 0.04	0.79 ± 0.01	0.74 ± 0.01	0.66 ± 0.01	0.56 ± 0.16	0.70 ± 0.03	0.89 ± 0.02	0.83 ± 0.10	0.91 ± 0.02
5	▲386K	0.65 ± 0.01	0.74 ± 0.01	0.65 ± 0.03	0.66 ± 0.01	0.72 ± 0.02	0.66 ± 0.05	0.88 ± 0.01	0.87 ± 0.07	0.90 ± 0.01
6	▲610K	0.55 ± 0.04	0.77 ± 0.00	0.67 ± 0.01	0.69 ± 0.02	0.72 ± 0.01	0.68 ± 0.03	0.87 ± 0.04	0.88 ± 0.05	0.91 ± 0.00
7	▼2K	0.68 ± 0.00	0.77 ± 0.01	0.65 ± 0.01	0.64 ± 0.07	0.68 ± 0.00	0.91 ± 0.01	0.84 ± 0.02	0.92 ± 0.01	0.90 ± 0.03
8	▲39K	0.59 ± 0.03	0.80 ± 0.01	0.70 ± 0.00	0.70 ± 0.01	0.65 ± 0.04	0.66 ± 0.06	0.86 ± 0.10	0.79 ± 0.09	0.89 ± 0.05
9	▲23K	0.64 ± 0.02	0.36 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.64 ± 0.07	0.77 ± 0.01	0.87 ± 0.07	0.95 ± 0.01
10	▲150K	0.59 ± 0.03	0.80 ± 0.01	0.70 ± 0.00	0.70 ± 0.01	0.65 ± 0.04	0.66 ± 0.06	0.86 ± 0.10	0.79 ± 0.09	0.89 ± 0.05
11	▲35K	0.49 ± 0.06	0.80 ± 0.01	0.74 ± 0.02	0.74 ± 0.02	0.73 ± 0.06	0.74 ± 0.02	0.86 ± 0.01	0.94 ± 0.02	0.94 ± 0.01
12	▲122K	0.61 ± 0.01	0.81 ± 0.00	0.76 ± 0.02	0.75 ± 0.01	0.69 ± 0.02	0.69 ± 0.01	0.91 ± 0.01	0.87 ± 0.07	0.88 ± 0.03
13	▼420	0.49 ± 0.25	0.77 ± 0.02	0.65 ± 0.06	0.64 ± 0.06	0.34 ± 0.23	0.89 ± 0.06	0.64 ± 0.22	0.93 ± 0.01	0.92 ± 0.01
14	▲4K	0.51 ± 0.16	0.76 ± 0.02	0.71 ± 0.01	0.70 ± 0.03	0.02 ± 0.02	0.79 ± 0.03	0.75 ± 0.11	0.87 ± 0.12	0.93 ± 0.00
15	▲13K	0.60 ± 0.02	0.79 ± 0.02	0.74 ± 0.02	0.74 ± 0.02	0.60 ± 0.13	0.70 ± 0.04	0.91 ± 0.01	0.94 ± 0.03	0.94 ± 0.02
16	▼9K	0.70 ± 0.02	0.80 ± 0.02	0.82 ± 0.03	0.75 ± 0.01	0.48 ± 0.27	0.72 ± 0.08	0.83 ± 0.04	0.91 ± 0.05	0.91 ± 0.04
17	▼487	0.67 ± 0.01	0.78 ± 0.01	0.64 ± 0.03	0.56 ± 0.03	0.33 ± 0.28	0.92 ± 0.00	0.89 ± 0.02	0.94 ± 0.01	0.92 ± 0.01
18	▲38K	0.67 ± 0.02	0.64 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.02 ± 0.02	0.28 ± 0.06	0.71 ± 0.18	0.81 ± 0.10	0.85 ± 0.06
19	▲38K	0.64 ± 0.01	0.36 ± 0.02	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.55 ± 0.08	0.78 ± 0.02	0.88 ± 0.05	0.91 ± 0.02
20	▲770K	0.60 ± 0.01	0.77 ± 0.02	0.70 ± 0.03	0.67 ± 0.02	0.68 ± 0.04	0.70 ± 0.03	0.91 ± 0.01	0.87 ± 0.01	0.89 ± 0.02
21	▲1M	0.50 ± 0.20	0.70 ± 0.01	0.59 ± 0.02	0.63 ± 0.02	0.75 ± 0.01	0.71 ± 0.07	0.73 ± 0.07	0.89 ± 0.03	0.91 ± 0.00
22	▲5M	0.41 ± 0.15	0.75 ± 0.01	0.63 ± 0.01	0.69 ± 0.01	0.72 ± 0.01	0.68 ± 0.06	0.87 ± 0.00	0.89 ± 0.03	0.89 ± 0.01
23	▼2K	0.69 ± 0.00	0.72 ± 0.01	0.67 ± 0.01	0.62 ± 0.02	0.73 ± 0.06	0.90 ± 0.02	0.83 ± 0.04	0.86 ± 0.02	0.88 ± 0.03
24	▲40K	0.67 ± 0.00	0.75 ± 0.01	0.67 ± 0.01	0.63 ± 0.01	0.71 ± 0.01	0.76 ± 0.03	0.72 ± 0.02	0.85 ± 0.10	0.91 ± 0.01
25	▲119K	0.57 ± 0.01	0.78 ± 0.01	0.69 ± 0.00	0.70 ± 0.01	0.68 ± 0.02	0.74 ± 0.03	0.84 ± 0.07	0.85 ± 0.12	0.90 ± 0.03
26	▲71K	0.64 ± 0.02	0.81 ± 0.01	0.79 ± 0.03	0.75 ± 0.01	0.62 ± 0.08	0.73 ± 0.01	0.88 ± 0.01	0.85 ± 0.13	0.92 ± 0.01
	avg	0.60	0.73	0.61	0.60	0.53	0.71	0.83	0.87	🏆 **0.91**

Table 3.7 reports the accuracy (ACC) and F1 scores for the *EE* dataset. We employed 5-fold cross-validation to ensure robust performance assessment and minimize data partitioning bias.

For classification thresholds, we applied the sensitivity specificity cutoff method [117] on each fold's validation set and used their average for test set predictions. Consistently with our findings discussed in Section 3.5, our method systematically outperforms all $\perp\!\!\!\perp_m$ baselines, including VL, with an average improvement of 9% in both metrics across all h-motif types.

Further analysis on negative sampling .

Details on the negative sampling algorithm. Algorithm in Figure 3.24 details the negative sampling strategy presented above. The algorithm takes in input a hypergraph $\mathcal{H}$, a set M_t^+ of h-motifs of type t, the proportion of nodes α to be replaced in order to generate negative h-motifs, and the number of negative h-motifs β to be generated for each positive h-motif. The output of the algorithm is the set M_t^- of negative h-motifs, such that $|M_t^-| = \beta|M_t^+|$.

Algorithm 1 Negative sampling algorithm

Input: $\mathcal{H} = (V, E)$, M_t^+, α, β
Output: M_t^- a set of $\beta|M_t^+|$ negative samples.

Compute node-hyperedge distances $d_E(v, e)$, $\forall v \in V, e \in E$
$M_t^- \leftarrow \varnothing$
for $m \in M_t^+$ **do**
  Compute node-motif distances $d_M(v, m)$, $\forall v \in V \setminus V_m$
  Compute $p_m(v)$, $\forall v \in V \setminus V_m$
  $V_s \leftarrow$ the set of $\lfloor \alpha|V_m| \rfloor$ nodes from $V \setminus V_m$ sampled according to the node-substitution strategy (see Section 5.1)
  for $\ell = 1$ to β **do**
    $V_\ell^r \leftarrow$ any random subset of $\lfloor \alpha|V_m| \rfloor$ nodes from V_m
    Let $\pi_\ell : V_\ell^r \rightarrow V_s$ be a matching between V_ℓ^r and V_s
    $m_\ell^- \leftarrow (\Upsilon(e_i, \pi_\ell), \Upsilon(e_j, \pi_\ell), \Upsilon(e_k, \pi_\ell))$
    $M_t^- \leftarrow M_t^- \cup m_\ell^-$
  end for
end for
return M_t^-

Figure 3.24: Negative sampling algorithm.

Time complexity. The algorithm consists of two main phases: preprocessing and negative h-motif generation.

In the preprocessing phase, we first compute all-pair-shortest paths in the clique expansion of hypergraph $\mathcal{H}$ using the Floyd-

Warshall algorithm, requiring $O(|V|^3)$ time. We then calculate the distances between hyperedges and nodes using 3.1, which takes an additional $O(|V|^2)$ time.

For each positive h-motif $m \in M_t^+$, the generation phase performs these operations: *(i)* computes distances between m and each node in $V \setminus V_m$ in $O(|V|)$ time; *(ii)* calculates priority values for nodes in $V \setminus V_m$ in $O(|V|)$ time; and *(iii)* sorts nodes to determine replacing nodes in $O(|V| \log |V|)$ time. The algorithm then generates β negative h-motifs through an iterative process: *(i)* selects $|V_s|$ nodes from V_m to form set V_r; *(ii)* creates a random matching π between V_r and V_s; and *(iii)* generates and stores the negative h-motif m^-. Each iteration requires $O(|V_m|)$ time.

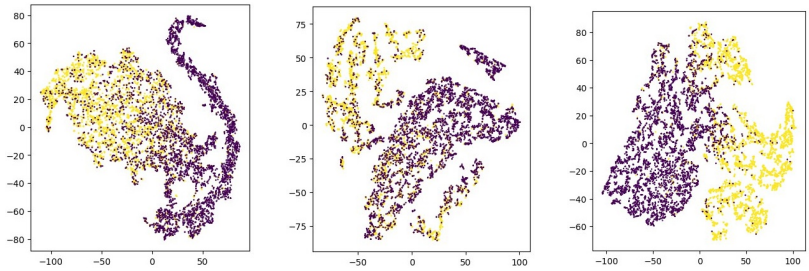
Combining all components, the total time complexity is

$$O(|V|^3 + \beta |M_t^+||V| \log |V|).$$

The impact of node-substitution mechanisms. Table 3.8 demonstrates how different node-substitution mechanisms in our negative sampling strategy affect the task difficulty, as measured by AUROC scores. We evaluate three approaches: our proposed

Table 3.8: AUROC scores when varying the negative sampling node-substitution mechanisms on the Email-Enron dataset.

	JC			VL			HM-VL		
t	random	probability	ranked	random	probability	ranked	random	probabilty	ranked
1	0.92 ± 0.00	0.92 ± 0.01	0.70 ± 0.01	0.87 ± 0.02	0.91 ± 0.04	0.88 ± 0.01	0.90 ± 0.02	0.88 ± 0.02	0.96 ± 0.00
2	0.88 ± 0.01	0.83 ± 0.00	0.51 ± 0.01	0.78 ± 0.10	0.87 ± 0.03	0.89 ± 0.04	0.87 ± 0.00	0.79 ± 0.01	0.95 ± 0.00
3	0.94 ± 0.01	0.92 ± 0.00	0.73 ± 0.07	0.87 ± 0.03	0.93 ± 0.02	0.90 ± 0.02	0.94 ± 0.01	0.89 ± 0.05	0.98 ± 0.01
4	0.94 ± 0.01	0.94 ± 0.01	0.77 ± 0.01	0.68 ± 0.11	0.96 ± 0.00	0.95 ± 0.01	0.91 ± 0.01	0.87 ± 0.03	0.97 ± 0.01
5	0.9± 0.01	o.88± 0.00	o.63± 0.01	0.81 ± 0.05	0.91 ± 0.03	0.93 ± 0.01	0.88 ± 0.02	0.81 ± 0.02	0.97 ± 0.01
6	0.93 ± 0.01	0.92 ± 0.01	0.67 ± 0.00	0.92 ± 0.02	0.94 ± 0.01	0.95 ± 0.01	0.91 ± 0.02	0.87 ± 0.01	0.97 ± 0.00
7	0.93 ± 0.00	0.92 ± 0.01	0.67 ± 0.06	0.84 ± 0.07	0.78 ± 0.01	0.92 ± 0.01	0.86 ± 0.01	0.83 ± 0.01	0.98 ± 0.01
8	0.94 ± 0.01	0.91 ± 0.01	0.72 ± 0.01	0.89 ± 0.01	0.93 ± 0.01	0.92 ± 0.07	0.89 ± 0.02	0.82 ± 0.02	0.96 ± 0.01
9	0.93 ± 0.00	0.91 ± 0.00	0.70 ± 0.06	0.91 ± 0.02	0.9± 0.03	o.90± 0.01	0.90 ± 0.01	0.83 ± 0.03	0.98 ± 0.00
10	0.95 ±0.01	0.94 ±0.01	0.75 ±0.01	0.90± 0.03	o.96± 0.01	0.97± 0.01	0.93 ± 0.01	0.90 ± 0.01	0.95 ± 0.04
11	0.95 ±0.01	0.95 ±0.00	0.76± 0.02	0.89 ± 0.04	0.82 ± 0.02	0.88 ± 0.01	0.94 ± 0.01	0.91 ± 0.01	0.98 ± 0.00
12	0.97 ±0.00	0.96 ±0.00	0.81 ±0.01	0.89 ± 0.05	0.85 ± 0.02	0.92 ± 0.01	0.96 ± 0.01	0.88 ± 0.02	0.96 ± 0.02
13	0.96 ±0.00	0.95 ±0.01	0.71 ±0.04	0.97 ± 0.01	0.95 ± 0.04	0.91 ± 0.05	0.88 ± 0.02	0.87 ± 0.04	0.97 ± 0.00
14	0.96 ±0.01	0.94 ±0.00	0.75 ±0.01	0.85 ± 0.05	0.83 ± 0.01	0.86 ± 0.02	0.94 ± 0.01	0.90 ± 0.01	0.98 ± 0.00
15	0.97 ±0.00	0.97 ±0.00	0.81 ±0.01	0.94 ± 0.01	0.85 ± 0.03	0.93 ± 0.02	0.95 ± 0.00	0.90 ± 0.03	0.98 ± 0.01
16	0.99 ±0.00	0.98 ±0.00	0.88 ±0.02	0.84 ± 0.03	0.79 ± 0.03	0.87 ± 0.03	0.96 ± 0.01	0.91 ± 0.03	0.97 ± 0.01
17	0.95 ±0.00	0.94 ±0.02	0.67 ±0.02	0.98 ± 0.01	0.95 ± 0.01	0.97 ± 0.00	0.85 ± 0.02	0.75 ± 0.02	0.98 ± 0.01
18	0.93 ±0.01	0.91 ±0.01	0.80 ±0.04	0.86 ± 0.05	0.82 ± 0.06	0.94 ± 0.03	0.91 ± 0.01	0.82 ± 0.04	0.92 ± 0.04
19	0.9 ± 0.01	0.87 ± 0.01	0.62 ± 0.06	0.83 ± 0.09	0.91 ± 0.02	0.86 ± 0.03	0.87 ± 0.02	0.77 ± 0.06	0.97 ± 0.04
20	0.92 ±0.01	0.92 ±0.00	0.74 ±0.04	0.87 ± 0.02	0.89 ± 0.04	0.97 ± 0.00	0.90 ± 0.04	0.82 ± 0.04	0.95 ± 0.02
21	0.88 ±0.01	0.84 ±0.01	0.54 ±0.04	0.8± 0.05	0.87± 0.02	0.88± 0.04	0.85 ± 0.01	0.79 ± 0.02	0.97 ± 0.00
22	0.92 ±0.01	0.89 ±0.00	0.68 ±0.04	0.88 ± 0.02	0.93 ± 0.01	0.95 ± 0.01	0.90 ± 0.01	0.83 ± 0.01	0.96 ± 0.00
23	0.89 ±0.02	0.87 ±0.02	0.53 ±0.02	0.93 ± 0.01	0.9± 0.01	0.91± 0.01	0.77 ± 0.02	0.72 ± 0.03	0.97 ± 0.01
24	0.9 ± 0.01	0.88 ± 0.02	0.64 ± 0.04	0.77 ± 0.12	0.93 ± 0.02	0.94 ± 0.03	0.87 ± 0.01	0.73 ± 0.06	0.96 ± 0.01
25	0.93 ±0.00	0.92 ±0.00	0.72 ±0.01	0.84 ± 0.03	0.92 ± 0.03	0.96 ± 0.00	0.93 ± 0.01	0.88 ± 0.03	0.96 ± 0.02
26	0.97 ±0.01	0.97 ±0.01	0.84 ±0.02	0.92 ± 0.03	0.82 ± 0.02	0.92 ± 0.01	0.95 ± 0.00	0.89 ± 0.00	0.98 ± 0.00
avg	**0.93**	0.92	0.70	0.87	0.89	**0.92**	0.90	0.84	🏆 **0.97**



(a) Random (b) Probability (c) Ranked

Figure 3.25: t-SNE visualization of h-motif embeddings learned by *HM VL* for type **t=2**, comparing different negative sampling strategies. Positive h-motifs are shown in violet and negative h-motifs in yellow.

method (*HM VL*), the standalone *VL* method, and the best-performing similarity-based approach, *JC*. The comparison uses the *EE* dataset for h-motif prediction, maintaining the experimental configuration detailed above.

The results show that *JC* achieves the best performance when using the *Random* and *Probability* node-substitution strategies. While this might appear counterintuitive, the relatively lower performance of *VL* and *HM VL* can be attributed to these sampling strategies being suboptimal for learning-based approaches. To validate this interpretation, we analyzed the distribution of h-motif embeddings produced by *HM VL* across all three substitution strategies, as visualized in Figure 3.25 using t-SNE for motif type $t = 2$. With the ranking-based substitution strategy, *HM VL* learns an embedding function that clearly separates positive and negative motifs. In contrast, the other strategies result in overlapping distributions, making the classification task more challenging. Furthermore, these alternative node-substitution strategies generate negative h-motifs that are less similar to positive ones, inherently favoring similarity-based approaches like *JC*.

Analysis of the confusion matrices (see Figure 3.26) for *JC* and *HM VL* reveals distinct behavioral patterns across different sampling strategies. With the probability-based strategy, *HM VL* achieves high precision but suffers from poor recall, suggesting the model learns an overly restrictive decision boundary that increases false negatives. In contrast, under the rank-based

strategy—which generates negative h-motifs structurally similar to positive ones—our method maintains balanced precision and recall with strong overall performance. This aligns with the embedding visualization in Figure 3.25, demonstrating effective h-motif representation learning. The rank-based strategy particularly challenges *JC*, as simple similarity metrics prove insufficient for distinguishing between positive samples and these carefully constructed negative examples.

Impact of positive training set size. We empirically determined the number of positive training instances ($|M_t| = 10K$) by evaluating all approaches on the *EE* dataset for h-motif type $t = 2$. To find this value, we conducted 10 experimental runs using

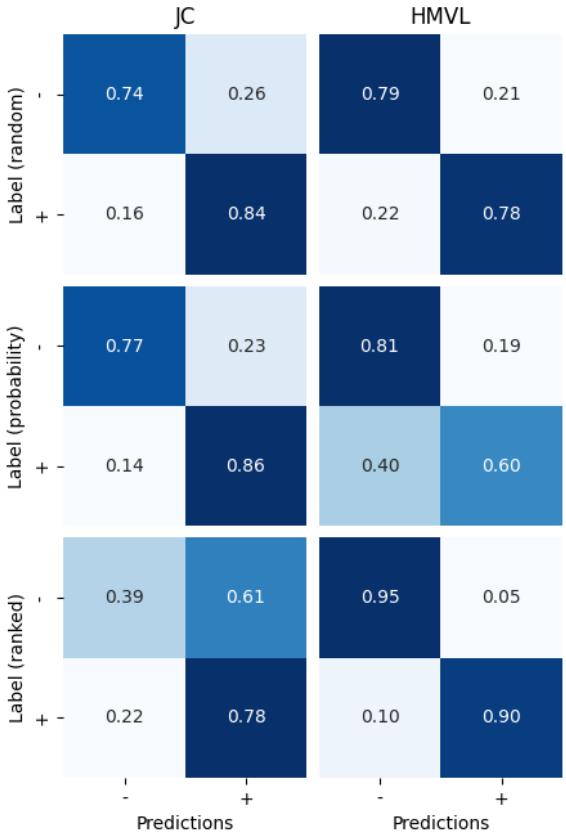


Figure 3.26: Confusion matrices comparing the classification performance of **JC** and **HMVL** methods for h-motif type **t=2** on the Email-Enron dataset.

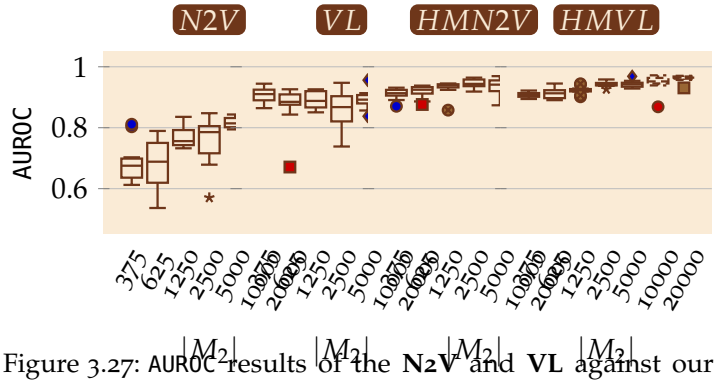


Figure 3.27: AUROC results of the **N2V** and **VL** against our methods **HMN2V** and **HMVL** to predict h-motifs instances on the *Email-Enron* hypergraph by varying the number of training motif instances for a specific type **t=2** averaged results over **10** experiments.

positive instances ranging from 375 to 20K. Overall, our analysis revealed that model performance stabilizes after 5000 instances. While the VL method showed decreased performance beyond 10K instances, other methods maintained consistent results for $|M_2| \geq 5000$. These findings led us to select 10K as the standard number of instances for our experiments. Two important considerations should be noted: first, not all h-motif types across all datasets contain this many positive instances; second, we excluded similarity-based methods from this analysis since their training process is independent of the number of positive instances. Figure 3.27 compares the AUROC results between our HM method (using both $N2V$ and VL initial embeddings) and the baseline methods $N2V$ and VL when varying the number of positive training instances.

3.6 CONCLUSION

This chapter presents an in-depth exploration of complex systems, with a specific emphasis on social network analysis using hypergraph-based methodologies. By leveraging advanced sampling and computational strategies, such as local sampling within ego networks, approximation sampling for specialized motifs, and parallel or distributed computing, the work addresses the scalability challenges of analyzing large and intricate datasets. The proposed pipeline is not only oriented toward three-sided

h-motifs, as established in recent literature, but also demonstrates flexibility to accommodate motifs of arbitrary size, albeit at increased computational cost. Our research provides the first significant solution to the h-motif prediction problem, and initial results suggest that conversation patterns within platforms such as Reddit transcend topical boundaries, with structural dynamics playing a pivotal role. The comparative analysis between Stack Overflow and Reddit further uncovers fundamental differences in user behaviors, community evolution, and platform engagement, shaped mainly by each platform's design and social mechanisms. Recognizing limitations in current data collection and processing, the thesis also identifies opportunities for future enhancement—ranging from enriched dataset descriptors to programmatic access and scalable visualization features. Looking forward, we aim to refine our embedding learning architectures, expand evaluation to tasks such as node classification and community detection, and implement generative adversarial frameworks to produce synthetic, motif-aware networks. Collectively, these efforts position our methodology as a scalable, generalizable tool for understanding and simulating the higher-order dynamics inherent in modern complex systems and social platforms.

4

CONCLUSION

This dissertation has explored complex social systems through both bottom-up and top-down perspectives, integrating methodological innovation, data infrastructure design, and empirical analysis. Across the chapters, a consistent theme emerges: understanding collective behavior in online environments requires complementary approaches that connect granular simulation with large-scale observational data.

Chapter 1 established the conceptual foundations of the dissertation, motivating the use of agent-based models, hypergraphs, and computational analytics as mutually reinforcing tools for studying complex systems. It articulated the research questions that guided the work and underscored the need for frameworks that capture interactions beyond simple pairwise relations.

Chapter 2 examined agent-based modeling as a bottom-up methodological paradigm. Through a sequence of complementary works, it demonstrated how ABM tools can be evaluated, improved, and scaled. The contributions span from practical assessments of existing simulation environments to rigorous approaches for ensuring reliability to the development of a high-performance Rust-based distributed engine capable of running large-scale simulations. The chapter highlights the methodological maturity of ABMs while acknowledging the persistent challenges of validation, benchmarking, and computational efficiency.

Chapter 3 provided a top-down analytical lens on online communities. It leveraged large-scale datasets, hypergraph analytics, and language models to study behavioral patterns, information exchange, and ideological discourse. The studies demonstrated how structural and semantic signals can be combined to extract high-level behavioral signatures, whether by examining knowledge-sharing practices among developers or by identifying stance patterns and community structures in politically charged environments. The chapter also introduced a significant infrastructural contribution: a community-driven repository for hypergraph datasets. By structuring, documenting, and standardizing data from diverse domains, the repository fills a critical gap in

the ecosystem of complex systems research. Beyond its technical architecture, this chapter emphasized the role of open science and collaborative data management in enabling reproducibility and cross-disciplinary insights.

Together, these chapters demonstrate the value of integrating data infrastructure, scalable modeling, and empirical analytics to advance the study of complex social systems.

4.1 DIRECTIONS

Although the dissertation provides a broad set of methodological and empirical contributions, it also opens numerous avenues for future work.

Future developments in data and infrastructure may include automated data ingestion and validation pipelines, such as machine-assisted documentation, schema validation, and metadata enrichment, which will enhance usability and ensure the long-term sustainability of computational research.

Within agent-based modeling and simulation, the implementation of formalized benchmarking suites would allow researchers to systematically compare performance, reliability, and reproducibility across diverse simulation environments. Hybrid modeling approaches, such as combining ABMs with machine learning components (e.g., LLM-based agents or surrogate models), promise to expand the representational scope of simulations while maintaining interpretability. Additionally, advances in distributed simulation—particularly further development of Rust-based engines that emphasize fault tolerance, adaptive load balancing, and real-time data integration could push the computational boundaries of ABM research. In the domain of computational social science, future research may explore a deeper integration of hypergraph-based structural models with language models to enable richer analyses of discourse, influence, and polarization in online discussions. Cross-platform behavioral studies that extend analyses of developer communities and ideological groups to accommodate multiple social platforms (such as GitHub, Twitter, and Discord) may reveal how differing platforms shape interaction patterns. Moreover, longitudinal studies tracking the evolution of behaviors, stances, and community structures over extended periods can provide insights into stability, fragmentation, and transformation within digital ecosystems.

Opportunities also exist for bridging bottom-up and top-down approaches: empirical findings from top-down analyses can be used to calibrate agent behaviors in ABMs based on real-world patterns observed in online communities. Such models could enable hypothesis testing, creating a feedback loop between simulation and data-driven analysis. Ultimately, deeper integration of hypergraph data, ABM simulations, and large-scale analytics may yield unified methodological pipelines capable of spanning the micro, meso, and macro levels of complex social system analysis.

Part I

APPENDIX

A

APPENDIX

This appendix brings together two categories of supplementary work. The first consists of manuscripts currently in the peer review process, which might have been incorporated into the main chapters of the dissertation due to their thematic relevance. The second category includes published papers that, despite contributing to my broader scholarly development, do not directly align with the core research questions of the dissertation. They are included here to provide a comprehensive view of the academic output produced during the course of this Ph.D. project.

A.1 OTHER PAPERS

A.1.1 *Hyperlink Prediction on Hypergraphs of Text*

Hypergraphs have recently emerged as powerful tools for modeling high-order relationships in complex real-world data. Nevertheless, existing hyperlink prediction methods primarily emphasize the structural connectivity encoded by these structures, often over- looking the rich semantic information associated with nodes and relations, which can often be expressed in the form of text. In this work, we propose a novel framework for hyperlink pre- diction on Hypergraphs of Text (HoTs), where both nodes and hyperedges are enriched with textual attributes. Our model jointly leverages these semantic and structural signals by combining hy- pergraph convolutional operators with cross-attention mechanisms that iteratively refine node and hyperedge representations. Experi- mental results demonstrate that integrating semantic information from nodes and hyperedges with structural properties consistently improves performance over baselines relying solely on topology, hence highlighting the effectiveness of contextual representations for hyperlink prediction and opening new directions for semantic- aware hypergraph learning.

A.1.2 *HyperBench: A Python library for Hyperlink prediction based on Pytorch and Pytorch Geometric*

This paper presents *HyperBench*, a Python library designed to support the evaluation of Hyperlink Prediction models. These models operate on hypergraph structures, which introduce particular challenges related to the complex nature of the relationships between nodes. In particular, hypergraphs often exhibit a strong imbalance between positive and negative sample classes, making it difficult to obtain accurate evaluations and consistent comparisons between different models. The Hypernegative library aims to address these critical issues by providing tools for data management, negative example generation, and integration with different learning strategies, simplifying the entire experimentation process.

A.1.3 *The Metaverse through the Eyes of University Students.*

The Metaverse is perceived as a promising approach that can possibly transform HCI guaranteeing high engagement and promoting participation and cooperation even at a distance. Its adoption is increasing in the last few years, with different areas and domains starting to use it and exploring if the Metaverse can be beneficial for them. This article explores university students' opinions regarding contexts and applications that can take advantage of the Metaverse and which benefits can be achieved. According to the results, students are aligned with the literature, considering the Metaverse useful in many application contexts thanks to its immersiveness and engaging environment promoting high-quality collaboration. However, their optimistic point of view should be mitigated by making them aware of the challenges the Metaverse poses in terms of (economic) sustainability and security concerns.

A.1.4 *CrossWarp: A Framework for the Integration of Virtual Worlds and Augmented Reality*

In recent years, the concept of the Reality-Virtuality Continuum (RVC) has taken a central role in research on immersive technologies, delineating the spectrum between physical and virtual

reality. However, while Augmented Reality (AR) and Virtual Reality (VR) have reached a technological maturity, existing solutions tend to be constrained to specific devices or use cases, limiting the possibilities for collaboration. In this context, the Cross-Reality (CR) paradigm aims to overcome such barriers, offering seamless integration between desktop devices, mobile AR, and immersive VR headsets. This paper introduces CrossWarp, a framework designed to enable real-time collaboration experiences between users with different devices within a common virtual space. The framework adds support for a smooth transition of digital objects between different realities, preserving state and context, facilitates interoperability between heterogeneous platforms, enabling hybrid interactions between desktop and mobile AR, and provides flexibility and extensibility, adapting to different application domains.

BIBLIOGRAPHY

[1] Sameera Abar, Georgios K. Theodoropoulos, Pierre Lemarinier, and Gregory M.P. O'Hare. "Agent Based Modelling and Simulation tools: A review of the state-of-art software." In: *Computer Science Review* 24 (2017), pp. 13–33. ISSN: 1574-0137.

[2] Russ Abbott and Jung Lim. *PyLogo*. https://pylogo.sourceforge.net/. Last accessed: 31st October 2022.

[3] Russ Abbott. and Jung Lim. "PyLogo: A Python Reimplementation of (Much of) NetLogo." In: *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - SIMULTECH*, INSTICC. SciTePress, 2021, pp. 199–206. DOI: 10.5220/0010466401990206.

[4] L. A. Adamic and E. Adar. "Friends and neighbors on the Web." In: *Social Networks* 25.3 (2003), pp. 211–230. ISSN: 0378-8733. DOI: 10.1016/S0378-8733(03)00009-1.

[5] A. Ahmad, C. Feng, S. Ge, and A. Yousif. "A survey on mining stack overflow: question and answering (Q&A) community." In: *Data Technologies and Applications* 52.2 (2018), pp. 190–247. ISSN: 25149288. DOI: 10.1108/DTA-07-2017-0054.

[6] Robert John Allan et al. *Survey of agent based modelling and simulation tools*. Science & Technology Facilities Council New York, 2010.

[7] Bernardo Alves Furtado. "PolicySpace2: Modeling Markets and Endogenous Public Policies." In: *Journal of Artificial Societies and Social Simulation* 25.1 (2022), p. 8. ISSN: 1460-7425. DOI: 10.18564/jasss.4742.

[8] MIT Amazon. *GraphChallenge - Data sets*. https://graphchallenge.mit.edu/data-sets. Accessed on 4th January 2024. 2017.

[9] Amethys. https://github.com/amethyst/legion.

[10] Amethyst. https://github.com/amethyst/specs.

[11] Li An et al. "Challenges, tasks, and opportunities in modeling agent-based complex systems." In: *Ecological Modelling* 457 (2021), p. 109685. ISSN: 0304-3800.

[12] Philipp Andelfinger and Wentong Cai. "Advanced Tutorial: Parallel and Distributed Methods for Scalable Discrete Simulation." In: *2022 Winter Simulation Conference (WSC)*. 2022, pp. 268–282. DOI: 10.1109/WSC57314.2022.10015291.

[13] J. L. Andersen, C. Flamm, D. Merkle, and P. F. Stadler. "Chemical Transformation Motifs—Modelling Pathways as Integer Hyperflows." In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 16 (2019), pp. 510–523. ISSN: 1557-9964. DOI: 10.1109/TCBB.2017.2781724.

[14] P. W. Anderson. "More Is Different." In: *Science* 177 (1972).

[15] M. Aniche, C. Treude, I. Steinmacher, I. Wiese, G. Pinto, M.-A. Storey, and M.A. Gerosa. "How Modern News Aggregators Help Development Communities Shape and Share Knowledge." In: vol. 2018-January. IEEE Computer Society, 2018, pp. 499–510. DOI: 10.1145/3180155.3180180.

[16] A. Antelmi, G. Cordasco, D. De Vinco, and C. Spagnuolo. *Github Repository*. https://github.com/ddevin96/DevCommunities. [Last accessed: March 2023]. 2023.

[17] A. Antelmi, G. Cordasco, B. Kamiński, P. Prałat, V. Scarano, C. Spagnuolo, and P. Szufel. "Analyzing, Exploring, and Visualizing Complex Networks via Hypergraphs using SimpleHypergraphs.jl." In: *Internet Mathematics* (2020). DOI: 10.24166/im.01.2020.

[18] A. Antelmi, G. Cordasco, M. Polato, V. Scarano, C. Spagnuolo, and D. Yang. "A Survey on Hypergraph Representation Learning." In: *ACM Computing Surveys* 56.1 (2023). ISSN: 0360-0300. DOI: 10.1145/3605776.

[19] A. Antelmi, D. Malandrino, and V. Scarano. "Characterizing the behavioral evolution of Twitter users and the truth behind the 90-9-1 rule." In: Association for Computing Machinery, Inc, 2019, pp. 1035–1038. ISBN: 9781450366755. DOI: 10.1145/3308560.3316705.

[20] Alessia Antelmi, Pasquale Caramante, Gennaro Cordasco, Giuseppe D'Ambrosio, Daniele De Vinco, Francesco Foglia, Luca Postiglione, and Carmine Spagnuolo. "Reliable and Efficient Agent-Based Modeling and Simulation." In: *Journal of Artificial Societies and Social Simulation* 27.2 (2024). DOI: 10.18564/jasss.5300.

[21] Alessia Antelmi, Gennaro Cordasco, Matteo D'Auria, Daniele De Vinco, Alberto Negro, and Carmine Spagnuolo. "On Evaluating Rust as a Programming Language for the Future of Massive Agent-Based Simulations." In: *Methods and Applications for Modeling and Simulation of Complex Systems*. Singapore: Springer Singapore, 2019, pp. 15–28.

[22] Alessia Antelmi, Gennaro Cordasco, Daniele De Vinco, Valerio Di Pasquale, Mirko Polato, and Carmine Spagnuolo. "Hypergraph Motif Representation Learning." In: *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*. 2025, pp. 13–24.

[23] Alessia Antelmi, Gennaro Cordasco, Daniele De Vinco, and Carmine Spagnuolo. "The Age of Snippet Programming: Toward Understanding Developer Communities in Stack Overflow and Reddit." In: *Companion Proceedings of the ACM Web Conference 2023*. WWW '23 Companion. New York, NY, USA: Association for Computing Machinery, 2023. ISBN: 9781450394192. DOI: 10.1145/3543873.3587673.

[24] Alessia Antelmi, Gennaro Cordasco, Giuseppe D'Ambrosio, Daniele De Vinco, and Carmine Spagnuolo. "Experimenting with Agent-Based Model Simulation Tools." In: *Applied Sciences* 13.1 (2023). ISSN: 2076-3417.

[25] Alessia Antelmi, Gennaro Cordasco, Matteo D'Auria, Daniele De Vinco, Alberto Negro, and Carmine Spagnuolo. "On evaluating rust as a programming language for the future of massive agent-based simulations." In: *Methods and Applications for Modeling and Simulation of Complex Systems: 19th Asia Simulation Conference, AsiaSim 2019, Singapore, October 30–November 1, 2019, Proceedings 19*. Springer Singapore. 2019, pp. 15–28.

[26] Alessia Antelmi, Gennaro Cordasco, Bogumił Kamiński, Paweł Prałat, Vittorio Scarano, Carmine Spagnuolo, and

Przemyslaw Szufel. "SimpleHypergraphs.jl—novel software framework for modelling and analysis of hypergraphs." In: *Algorithms and Models for the Web Graph: 16th International Workshop, WAW 2019, Brisbane, QLD, Australia, July 6–7, 2019, Proceedings 16*. Springer. 2019, pp. 115–129.

[27] Alessia Antelmi, Gennaro Cordasco, Mirko Polato, Vittorio Scarano, Carmine Spagnuolo, and Dingqi Yang. "A survey on hypergraph representation learning." In: *ACM Computing Surveys* 56.1 (2023), pp. 1–38. DOI: 10.1145/3605776.

[28] Alessia Antelmi, Gennaro Cordasco, Carmine Spagnuolo, and Vittorio Scarano. "A Design-Methodology for Epidemic Dynamics via Time-Varying Hypergraphs." In: *19th International Conference on Autonomous Agents and Multi-Agent Systems*. AAMAS '20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, 61–69. ISBN: 9781450375184. DOI: 10.5555/3398761.3398774.

[29] Alessia Antelmi, Gennaro Cordasco, Carmine Spagnuolo, and Przemysław Szufel. "Social influence maximization in hypergraphs." In: *Entropy* 23.7 (2021), p. 796.

[30] Alessia Antelmi, Daniele De Vinco, and Carmine Spagnuolo. "HypergraphRepository: A Community-Driven and Interactive Hypernetwork Data Collection." In: *International Workshop on Algorithms and Models for the Web-Graph*. Springer. 2024, pp. 159–173. DOI: 10.1007/978-3-031-59205-8_11.

[31] Alessia Antelmi, Daniele De Vinco, and Carmine Spagnuolo. "HypergraphRepository: a community-driven and interactive hypernetwork data collection." In: *International Workshop on Algorithms and Models for the Web-Graph*. Springer. 2024, pp. 159–173.

[32] B. Arregui-García, A. Longa, Q. F. Lotito, S. Meloni, and G. Cencetti. "Patterns in Temporal Networks with Higher-Order Egocentric Structures." In: *Entropy* 26.3 (2024). DOI: 10.3390/e26030256.

[33] Robert Axelrod. "Advancing the Art of Simulation in the Social Sciences." In: *Simulating Social Phenomena*. Ed. by Rosaria Conte, Rainer Hegselmann, and Pietro Terna.

Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 21–40.

[34] Robert Axtell, Robert Axelrod, Joshua M. Epstein, and Michael D. Cohen. "Aligning simulation models: A case study and results." In: *Computational & Mathematical Organization Theory* 1.2 (1996), pp. 123–141. DOI: 10.1007/BF01299065.

[35] Steven C Bankes. "Agent-based modeling: A revolution?" In: *Proceedings of the National Academy of Sciences* 99.suppl_3 (2002), pp. 7199–7200. DOI: 10.1073/pnas.072081299.

[36] Brendan Bartley. "Mobility impacts, reactions and opinions : traffic demand management options in Europe : the MIRO Project." In: *Traffic engineering and control* 36 (1995), pp. 596–602.

[37] Federico Battiston, Enrico Amico, Alain Barrat, Ginestra Bianconi, Guilherme Ferraz de Arruda, Benedetta Franceschiello, Iacopo Iacopini, Sonia Kéfi, Vito Latora, Yamir Moreno, et al. "The physics of higher-order interactions in complex systems." In: *Nature Physics* 17.10 (2021), pp. 1093–1098.

[38] Federico Battiston, Giulia Cencetti, Iacopo Iacopini, Vito Latora, Maxime Lucas, Alice Patania, Jean-Gabriel Young, and Giovanni Petri. "Networks beyond pairwise interactions: Structure and dynamics." In: *Physics Reports* 874 (2020), pp. 1–92. DOI: 10.1016/j.physrep.2020.05.004.

[39] A. Bazaga, P. Liò, and G. Micklem. *HyperBERT: Mixing Hypergraph-Aware Layers with Language Models for Node Classification on Text-Attributed Hypergraphs*. 2024. arXiv: 2402.07309 [cs.LG].

[40] Fabio Bellifemine, Giovanni Caire, Giovanni Rimassa, Agostino Poggi, Federico Bergenti, Tiziana Trucco, Danilo Gotta, Elisabetta Cortese, Filippo Quarta, and Giosuè Vitaglione. *JADE CI/CD*. https://jade-project.gitlab.io/docs/add-on/JADE_TestSuite.pdf. Last accessed: 9st December 2022.

[41] Fabio Bellifemine, Giovanni Caire, Giovanni Rimassa, Agostino Poggi, Federico Bergenti, Tiziana Trucco, Danilo Gotta, Elisabetta Cortese, Filippo Quarta, and Giosuè Vitaglione. *JADE site | Java Agent Development Framework*.

https://jade.tilab.com/. Last accessed: 31st October 2022.

[42] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. "Developing multi-agent systems with a FIPA-compliant agent framework." In: *Software: Practice and Experience* 31.2 (2001), pp. 103–128. DOI: https://doi.org/10.1002/1097-024X(200102)31:2<103::AID-SPE358>3.0.CO;2-O.

[43] Austin R. Benson. *Data!* https://www.cs.cornell.edu/ arb/data/. Accessed on 30th December 2023. 2021.

[44] Claude Berge. *Hypergraphs: combinatorics of finite sets*. Vol. 45. Elsevier, 1984.

[45] Alan A Berryman. "The orgins and evolution of predator-prey theory." In: *Ecology* 73.5 (1992), pp. 1530–1535.

[46] Federico Bert, Michael North, Santiago Rovere, Eric Tatara, Charles Macal, and Guillermo Podestá. "Simulating agricultural land rental markets by combining agent-based models with traditional economics concepts: The case of the Argentine Pampas." In: *Environmental Modelling & Software* 71 (2015), pp. 97–110. ISSN: 1364-8152. DOI: 10.1016/j.envsoft.2015.05.005.

[47] M. Besta et al. "Motif Prediction with Graph Neural Networks." In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '22. New York, NY, USA: Association for Computing Machinery, 2022, 35–45. ISBN: 9781450393850. DOI: 10.1145/3534678.3539343.

[48] Bevy. https://github.com/bevyengine.

[49] G. Blanco, R. Pérez-López, F. Fdez-Riverola, and A.M.G. Lourenço. "Understanding the social evolution of the Java community in Stack Overflow: A 10-year study of developer interactions." In: *Future Generation Computer Systems* 105 (2020), pp. 446–454. ISSN: 0167739X. DOI: 10.1016/j.future.2019.12.021.

[50] P. Boldi, M. Rosa, M. Santini, and S. Vigna. "Layered Label Propagation: A Multiresolution Coordinate-Free Ordering for Compressing Social Networks." In: *Proceedings of the 20th International Conference on World Wide Web*. WWW '11. New York, NY, USA: Association for Comput-

ing Machinery, 2011, 587–596. ISBN: 9781450306324. DOI: 10.1145/1963405.1963488.

[51] P. Boldi and S. Vigna. "The Webgraph Framework I: Compression Techniques." In: *Proceedings of the 13th International Conference on World Wide Web*. WWW '04. New York, NY, USA: Association for Computing Machinery, 2004, 595–602. ISBN: 158113844X. DOI: 10.1145/988672.988752.

[52] Pierre Bommel, Nicolas Becu, Christophe Le Page, and François Bousquet. *CORMAS CI/CD*. https://github.com/cormas/cormas/blob/master/.github/workflows/test.yml. Last accessed: 9st December 2022.

[53] Pierre Bommel, Nicolas Becu, Christophe Le Page, and François Bousquet. *CORMAS: COmmon-pool Resources and Multi-Agent Simulations*. http://cormas.cirad.fr/indexeng.html. Last accessed: 31st October 2022.

[54] Pierre Bommel, Nicolas Becu, Christophe Le Page, and François Bousquet. "Cormas: An Agent-Based Simulation Platform for Coupling Human Decisions with Computerized Dynamics." In: *Simulation and Gaming in the Network Society*. Springer Singapore, 2016, pp. 387–410. DOI: 10.1007/978-981-10-0575-6_27.

[55] Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah-Seghrouchni, Jorge J. Gómez-Sanz, João Leite, Gregory M. P. O'Hare, Alexander Pokahr, and Alessandro Ricci. "A Survey of Programming Languages and Platforms for Multi-Agent Systems." In: *Informatica (Slovenia)* 30.1 (2006), pp. 33–44.

[56] Francisco Borges, Albert Gutierrez-Milla, Emilio Luque, and Remo Suppi. "Care HPS: A high performance simulation tool for parallel and distributed agent-based modeling." In: *Future Generation Computer Systems* 68 (2017), pp. 59–73. ISSN: 0167-739X. DOI: 10.1016/j.future.2016.08.015.

[57] Andrei Borshchev, Yuri Karpov, and Vladimir Kharitonov. "Distributed Simulation of Hybrid Systems with AnyLogic and HLA." In: *Future Generation Computer System* 18.6 (2002), 829–839. DOI: 10.1016/S0167-739X(02)00055-9.

[58] T. RJ Bossomaier and D. G. Green. *Complex systems*. Cambridge university press, 2000.

[59] Wang Boyu and Jackie Kazil. *Mesa-Geo: GIS Extension for Mesa Agent-Based Modeling*. https://github.com/projectmesa/mesa-geo. Last accessed: 31st October 2022.

[60] Corey Brady, Jeremy, Robert Grider, and Aaron Brandes. *NetLogo View2.5D Extension*. https://github.com/NetLogo/View2.5D. Last accessed: 31st October 2022.

[61] Alain Bretto. "Hypergraph theory." In: *An introduction. Mathematical Engineering. Cham: Springer* 1 (2013). DOI: 10.1007/978-3-319-00080-0.

[62] Daniel G. Brown, Rick Riolo, Derek T. Robinson, Michael North, and William Rand. "Spatial process and data models: Toward integration of agent-based models and GIS." In: *Journal of Geographical Systems* 7.1 (2005), pp. 25–47. ISSN: 1435-5949. DOI: 10.1007/s10109-005-0148-5.

[63] Daniel Brown, Scott Page, Rick Riolo, Moira Zellner, and William Rand. "Path dependence and the validation of agent-based spatial models of land use." In: *International Journal of Geographical Information Science* 19.2 (2005), pp. 153–174. DOI: 10.1080/13658810410001713399.

[64] Russell A. Brown. "Building a Balanced k-d Tree in $O(kn \log n)$ Time." In: *Journal of Computer Graphics Techniques (JCGT)* 4.1 (2015), pp. 50–68. ISSN: 2331-7418.

[65] William Bugden and Ayman Alahmar. "Rust: The programming language for safety and performance." In: *arXiv preprint arXiv:2206.05503* (2022). DOI: 10.48550/arXiv.2206.05503.

[66] Andrey Bychkov and Vsevolod Nikolskiy. "Rust Language for Supercomputing Applications." In: *Communications in Computer and Information Science* 1510 CCIS (2021), pp. 391–403. DOI: 10.1007/978-3-030-92864-3_30.

[67] R. Cao, X. Luo, Y. Xi, and Y. Qiao. "Stance detection for online public opinion awareness: An overview." In: *International Journal of Intelligent Systems* 37.12 (2022), pp. 11944–11965.

[68] Marcos Cardinot, Colm O'Riordan, Josephine Griffith, and Matjaž Perc. *Evoplex - CI/CD*. https://github.com/evoplex/evoplex/blob/master/.travis.yml. Last accessed: 9st December 2022.

[69] Marcos Cardinot, Colm O'Riordan, Josephine Griffith, and Matjaž Perc. *Evoplex - agent-based modeling on networks*. https://evoplex.org/. Last accessed: 31st October 2022.

[70] Marcos Cardinot, Colm O'Riordan, Josephine Griffith, and Matjaž Perc. "Evoplex: A platform for agent-based modeling on networks." In: *SoftwareX* 9 (2019), pp. 199–204. DOI: 10.1016/j.softx.2019.02.009.

[71] K.M. Carley, D.B. Fridsma, E. Casman, A. Yahja, N. Altman, Li-Chiou Chen, B. Kaminsky, and D. Nave. "BioWar: scalable agent-based model of bioattacks." In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 36.2 (2006), pp. 252–265. DOI: 10.1109/TSMCA.2005.851291.

[72] Ernesto Carrella. "No Free Lunch when Estimating Simulation Parameters." In: *Journal of Artificial Societies and Social Simulation* 24.2 (2021), p. 7. ISSN: 1460-7425. DOI: 10.18564/jasss.4572.

[73] Christian Castle and Andrew Crooks. "Principles and Concepts of Agent-Based Modelling for Developing Geospatial Simulations." In: *Castle, C.J.E. and Crooks, A.T. (2006) Principles and concepts of agent-based modelling for developing geospatial simulations. Working paper. CASA Working Papers (110). Centre for Advanced Spatial Analysis (UCL), London, UK.* 110 (2006).

[74] Aritra Chakravorty, William S Cleveland, and Patrick J Wolfe. "Scalable *k*-d trees for distributed data." In: *arXiv preprint arXiv:2201.08288* (2022).

[75] Rohit Chandra. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[76] Eshwar Chandrasekharan, Shagun Jhaver, Amy Bruckman, and Eric Gilbert. "Quarantined! Examining the effects of a community-wide moderation intervention on Reddit." In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 29.4 (2022), pp. 1–26.

[77] Joydev Chattopadhyay and Ovide Arino. "A predator-prey model with disease in the prey." In: *Nonlinear analysis* 36 (1999), pp. 747–766.

[78] C. Chen and Y.-Y. Liu. "A Survey on Hyperlink Prediction." In: *IEEE Transactions on Neural Networks and Learning Systems* 35.11 (2024), pp. 15034–15050. DOI: 10.1109/TNNLS.2023.3286280.

[79] Shu-Heng Chen, Chia-Ling Chang, and Ye-Rong Du. "Agent-based economic models and econometrics." In: *The Knowledge Engineering Review* 27.2 (2012), pp. 187–219. DOI: 10.1017/S0269888912000136.

[80] Z. Chen, H. Mao, H. Li, W. Jin, H. Wen, X. Wei, S. Wang, D. Yin, W. Fan, H. Liu, et al. "Exploring the potential of large language models (llms) in learning on graphs." In: *ACM SIGKDD Explorations Newsletter* 25.2 (2024), pp. 42–61. DOI: 10.1145/3655103.3655110.

[81] Timothy Chuang and Munehiro Fukuda. "A Parallel Multi-agent Spatial Simulation Environment for Cluster Systems." In: *2013 IEEE 16th International Conference on Computational Science and Engineering*. 2013, pp. 143–150. DOI: 10.1109/CSE.2013.32.

[82] Matteo Cinelli, Gianmarco De Francisci Morales, Alessandro Galeazzi, Walter Quattrociocchi, and Michele Starnini. "The echo chamber effect on social media." In: *Proceedings of the National Academy of Sciences* 118.9 (2021), e2023301118.

[83] Simon Coakley, Marian Gheorghe, Mike Holcombe, Shawn Chin, David Worth, and Chris Greenough. "Exploitation of High Performance Computing in the FLAME Agent-Based Simulation Framework." In: *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. 2012, pp. 538–545. DOI: 10.1109/HPCC.2012.79.

[84] MartĂn Coll, Cliff A Joslyn, Nicholas W Landry, Quintino Francesco Lotito, Audun Myers, Joshua Pickard, Brenda Praggastis, PrzemysĹ Szufel, et al. "HIF: The hypergraph interchange format for higher-order networks." In: *arXiv preprint arXiv:2507.11520* (2025).

[85] N. Collier and M. North. "Parallel agent-based simulation with Repast for High Performance Computing." In: *Simulation* 89.10 (2013), pp. 1215–1235. DOI: 10.1177/0037549712462620.

[86] Nicholson T. Collier, Jonathan Ozik, and Eric R. Tatara. "Experiences in Developing a Distributed Agent-based Modeling Toolkit with Python." In: *2020 IEEE/ACM 9th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*. 2020, pp. 1–12.

[87] Nick Collier, John T. Murphy, Jonathan Ozik, Sara Rimer, Daniel Sheeler, and Eric Tatara. *Repast HPC*. `https://repast.github.io/repast_hpc.html`. Last accessed: 31st October 2022.

[88] Nick Collier, John T. Murphy, Jonathan Ozik, Sara Rimer, Daniel Sheeler, and Eric Tatara. *Repast Suite*. `https://repast.github.io/`. Last accessed: 31st October 2022.

[89] European Commission. *Open Science*. https://research-and-innovation.ec.europa.eu/strategy/strategy-2020-2024/our-digital-future/open-science_en. Accessed on 29th December 2023. 2020.

[90] Gennaro Cordasco, Vittorio Scarano, and Carmine Spagnuolo. "Distributed mason: A scalable distributed multi-agent simulation environment." In: *Simulation Modelling Practice and Theory* 89 (2018), pp. 15–34. DOI: `10.1016/j.simpat.2018.09.002`.

[91] Andrew T. Crooks and Atesmachew B. Hailegiorgis. "An agent-based modeling approach applied to the spread of cholera." In: *Environmental Modelling & Software* 62 (2014), pp. 164–177. ISSN: 1364-8152. DOI: `10.1016/j.envsoft.2014.08.027`.

[92] Andrew T Crooks and Alison J Heppenstall. "Introduction to agent-based modelling." In: *Agent-based models of geographical systems*. Springer, 2011, pp. 85–105. DOI: `10.1007/978-90-481-8927-4_5`.

[93] I. J Cruickshank and L. H. X. Ng. "Use of large language models for stance classification." In: *arXiv preprint arXiv:2309.13734* (2023). DOI: `10.48550/arXiv.2309.13734`.

[94] Erik Cuevas. "An agent-based model to evaluate the COVID-19 transmission risks in facilities." In: *Computers in biology and medicine* 121 (2020), p. 103827. DOI: `10.1016/j.compbiomed.2020.103827`.

[95] Giuseppe D'Ambrosio and Daniele De Vinco. "The Metaverse through the Eyes of University Students." In: *S3C@ CHItaly*. 2023, pp. 73–82.

[96] L. Da Silva, J. Samhi, and F. Khomh. "ChatGPT vs LLaMA: Impact, Reliability, and Challenges in Stack Overflow Discussions." In: *arXiv preprint arXiv:2402.08801* (2024).

[97] George Datseris, Ali R. Vahdati, and Timothy C. DuBois. *Agents.jl CI/CD*. `https://github.com/JuliaDynamics/Agents.jl/blob/main/.github/workflows/ci.yml`. Last accessed: 9st December 2022.

[98] George Datseris, Ali R. Vahdati, and Timothy C. DuBois. *Agents.jl*. `https://juliadynamics.github.io/Agents.jl/`. Last accessed: 31st October 2022.

[99] George Datseris, Ali R. Vahdati, and Timothy C. DuBois. *Distributed Computing - The Julia Language*. `https://docs.julialang.org/en/v1/stdlib/Distributed/`. Last accessed: 31st October 2022.

[100] George Datseris, Ali R. Vahdati, and Timothy C. DuBois. "Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity." In: *SIMULATION* 0.0 (2022), p. 003754972110688. DOI: `10.1177/00375497211068820`.

[101] Daniele De Vinco. "Beyond the Surface: Navigating Complex Systems via ABMs and Hypergraphs." In: *International Conference on Advances in Social Networks Analysis and Mining*. Springer. 2024, pp. 148–163.

[102] Daniele De Vinco. "Fifty Shapes of Reddit: Do Prolife Activists Have the Same Interaction Patterns of Gun Fanatics?" In: *International Conference on Advances in Social Networks Analysis and Mining*. Springer. 2024, pp. 245–253.

[103] Daniele De Vinco, Alessia Antelmi, Carmine Spagnuolo, and Luca Maria Aiello. "Deciphering Conversational Networks: Stance Detection via Hypergraphs and LLMs." In: *Companion Publication of the 16th ACM Web Science Conference*. Websci Companion '24. New York, NY, USA: Association for Computing Machinery, 2024. DOI: `10.1145/3630744.3658418`.

[104] Daniele De Vinco, Andrea Tranquillo, Alessia Antelmi, Carmine Spagnuolo, Vittorio Scarano, et al. "High-Performance Computation on a Rust-based distributed ABM engine." In: *CEUR WORKSHOP PROCEEDINGS*. Vol. 3785. CEUR-WS. 2024, pp. 52–60.

[105] Michela Del Vicario, Alessandro Bessi, Fabiana Zollo, Fabio Petroni, Antonio Scala, Guido Caldarelli, H Eugene Stanley, and Walter Quattrociocchi. "The spreading of misinformation online." In: *Proceedings of the national academy of Sciences* 113.3 (2016), pp. 554–559.

[106] Elizabeth Donkin, Peter Dennis, Andrey Ustalakov, John Warren, and Amanda Clare. "Replicating complex agent based models, a formidable task." In: *Environmental Modelling & Software* 92 (2017), pp. 142–151. DOI: 10.1016/j.envsoft.2017.01.020.

[107] Unreal Engine. https://www.unrealengine.com/marketplace/en-US/product/apparatus.

[108] Joshua M. Epstein, D. Michael Goedecke, Feng Yu, Robert J. Morris, Diane K. Wagener, and Georgiy V. Bobashev. "Controlling Pandemic Flu: The Value of International Air Travel Restrictions." In: *PLoS ONE* 2.5 (2007). Ed. by Maurizio Del Poeta. DOI: 10.1371/journal.pone.0000401.

[109] Ernesto Estrada. "What is a Complex System, After All?" In: *Foundations of Science* (2023). ISSN: 1572-8471. DOI: 10.1007/s10699-023-09917-w.

[110] Roland Ewald and Adelinde M. Uhrmacher. "SESSL: A Domain-Specific Language for Simulation Experiments." In: *ACM Trans. Model. Comput. Simul.* 24.2 (2014). ISSN: 1049-3301. DOI: 10.1145/2567895.

[111] J. Doyne Farmer and Duncan Foley. "The economy needs agent-based modelling." In: *Nature* 460.7256 (2009), pp. 685–686. ISSN: 1476-4687. DOI: 10.1038/460685a.

[112] J. Doyne Farmer, Cameron Hepburn, Penny Mealy, and Alexander Teytelboym. "A Third Wave in the Economics of Climate Change." In: *Environmental and Resource Economics* 62.2 (2015), pp. 329–357. ISSN: 1573-1502. DOI: 10.1007/s10640-015-9965-2.

[113] M. Fey and J.E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric." In: *CoRR* abs/1903.02428 (2019). workshop paper at ICLR 2019.

[114] M.T. Fischer, A. Frings, D.A. Keim, and D. Seebacher. "Towards a Survey on Static and Dynamic Hypergraph Visualizations." In: *2021 IEEE Visualization Conference (VIS)*. 2021, pp. 81–85. DOI: 10.1109/VIS49827.2021.9623305.

[115] Luciano Floridi and Massimo Chiriatti. "GPT-3: Its nature, scope, limits, and consequences." In: *Minds and Machines* 30 (2020), pp. 681–694. DOI: 10.1007/s11023-020-09548-1.

[116] Leon Florin. *ActressMas*. https://github.com/florinleon/ActressMas. Last accessed: 31st October 2022.

[117] R. Fluss, D. Faraggi, and B. Reiser. "Estimation of the Youden Index and its Associated Cutoff Point." In: *Biometrical Journal* 47.4 (2005), pp. 458–472. DOI: 10.1002/bimj.200410135.

[118] Joël Foramitti. *AgentPy - Agent-based modeling in Python*. https://agentpy.readthedocs.io/en/latest/. Last accessed: 31st October 2022.

[119] Joël Foramitti. *AgentPy CI/CD*. https://github.com/JoelForamitti/agentpy/blob/master/.github/workflows/test.yml. Last accessed: 9st December 2022.

[120] Joël Foramitti. "AgentPy: A package for agent-based modeling in Python." In: *Journal of Open Source Software* 6.62 (2021), p. 3065. DOI: 10.21105/joss.03065.

[121] Scott Fortmann-Roe. *Insight Maker*. https://insightmaker.com/. Last accessed: 31st October 2022.

[122] Scott Fortmann-Roe. "Insight Maker: A general-purpose tool for web-based modeling & simulation." In: *Simulation Modelling Practice and Theory* 47 (2014), pp. 28–45. ISSN: 1569-190X. DOI: 10.1016/j.simpat.2014.03.013.

[123] C. Fu, X. Yue, B. Shen, S. Yu, and Y. Min. "Patterns of interest change in stack overflow." In: *Scientific Reports* 12.1 (2022). ISSN: 20452322. DOI: 10.1038/s41598-022-15724-3.

[124] L. Gallo, L. Lacasa, V. Latora, and F. Battiston. "Higher-order correlations reveal complex memory in temporal hypergraphs." In: *Nature Communications* 15.1 (2024), p. 4754. ISSN: 2041-1723. DOI: 10.1038/s41467-024-48578-6.

[125] Iván García-Magariño, Andrés S. Lombas, Inmaculada Plaza, and Carlos Medrano. "ABS-SOCI: An Agent-Based Simulator of Student Sociograms." In: *Applied Sciences* 7.11 (2017). ISSN: 2076-3417. DOI: 10.3390/app7111126.

[126] T. Gaudelet, N. Malod-Dognin, and N. Pržulj. "Higher-order molecular organization as a source of biological function." In: *Bioinformatics* 34.17 (2018), pp. i944–i953. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty570.

[127] R.P. Gauthier, M.J. Costello, and J.R. Wallace. ""I Will Not Drink With You Today": A Topic-Guided Thematic Analysis of Addiction Recovery on Reddit." In: Association for Computing Machinery, 2022. ISBN: 9781450391573. DOI: 10.1145/3491102.3502076.

[128] George Mason University's ECLab Evolutionary Computation Laboratory. *ECJ A Java-based Evolutionary Computation Research System*. https://cs.gmu.edu/~eclab/projects/ecj/. Last accessed: 31st October 2022.

[129] Nigel Gilbert and Steven Bankes. "Platforms and methods for agent-based modeling." In: *Proceedings of the National Academy of Sciences* 99.suppl_3 (2002), pp. 7197–7198. DOI: 10.1073/pnas.072079499.

[130] Github. *Github Copilot*. https://github.com/features/copilot. [Last accessed: March 2023]. 2022.

[131] A. Grover and J. Leskovec. "node2vec: Scalable Feature Learning for Networks." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD'16. New York, NY, USA: Association for Computing Machinery, 2016, 855–864. ISBN: 9781450342322. DOI: 10.1145/2939672.2939754.

[132] Yuanshen Guan, Xiangguo Sun, and Yongjiao Sun. "Sparse relation prediction based on hypergraph neural networks in online social networks." In: *World Wide Web* 26.1 (2023), pp. 7–31.

[133] Ritu Gupta and Gaurav Kansal. "A Survey on Comparative Study of Mobile Agent Platforms." In: *International Journal of Engineering Science and Technology* 3 (2011).

[134] Olivier Gutknecht and Jacques Ferber. *MaDKit CI/CD*. `https://github.com/fmichel/MaDKit/blob/master/.travis.yml`. Last accessed: 9st December 2022.

[135] Olivier Gutknecht and Jacques Ferber. *MaDKit*. `https://www.madkit.net/madkit/`. Last accessed: 31st October 2022.

[136] Olivier Gutknecht and Jacques Ferber. "The MadKit Agent Platform Architecture." In: *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*. Ed. by Tom Wagner and Omer F. Rana. Springer Berlin Heidelberg, 2001, pp. 48–55.

[137] HackerNews. `https://news.ycombinator.com/`. [Last accessed: March 2023]. 2007.

[138] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. "A survey on large language models: Applications, challenges, limitations, and practical usage." In: *Authorea Preprints* (2023). DOI: `10.36227/techrxiv.23589741.v1`.

[139] Ho Bich Hai, Lucie Contamin, Marc Choisy, and Arthur Brugière. *gamar: An R interface to the GAMA platform*. `https://github.com/r-and-gama/gamar`. Last accessed: 31st October 2022.

[140] Atesmachew Hailegiorgis, Andrew Crooks, and Claudio Cioffi-Revilla. "An Agent-Based Model of Rural Households' Adaptation to Climate Change." In: *Journal of Artificial Societies and Social Simulation* 21.4 (2018), p. 4. ISSN: 1460-7425. DOI: `10.18564/jasss.3812`.

[141] W.L. Hamilton, J. Zhang, C. Danescu-Niculescu-Mizil, D. Jurafsky, and J. Leskovec. "Loyalty in online communities." In: AAAI Press, 2017, pp. 540–543. ISBN: 9781577357889.

[142] X. He, X. Bresson, T. Laurent, A. Perold, Y. LeCun, and B. Hooi. "Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning." In: *The Twelfth International Conference on Learning Representations*. 2023.

[143] Brian Heath, Raymond Hill, and Frank Ciarallo. "A Survey of Agent-Based Modeling Practices (January 1998 to July 2008)." In: *Journal of Artificial Societies and Social Simulation* 12.4 (2009), p. 9. ISSN: 1460-7425.

[144] Nicolas Hoertel, Martin Blachier, Carlos Blanco, Mark Olfson, Marc Massetti, Marina Sánchez Rico, Frédéric Limosin, and Henri Leleu. "A stochastic agent-based model of the SARS-CoV-2 epidemic in France." In: *Nature Medicine* 26.9 (2020), pp. 1417–1421. ISSN: 1546-170X. DOI: 10.1038/s41591-020-1001-6.

[145] Mike Holcombe, Simon Coakley, and Rod Smallwood. "A general framework for agent-based modelling of complex systems." In: 2006.

[146] Chung-Chi Hsieh and Yi-Che Hsieh. "Reliability and cost optimization in distributed computing systems." In: *Computers & Operations Research* 30.8 (2003), pp. 1103–1119.

[147] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. "Open Graph Benchmark: Datasets for Machine Learning on Graphs." In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 22118–22133.

[148] Y. Hu, Z. Zhang, and L. Zhao. "Beyond Text: A Deep Dive into Large Language Models' Ability on Understanding Graph Data." In: *arXiv preprint arXiv:2310.04944* (2023).

[149] J. Huang, X. Zhang, Q. Mei, and J. Ma. "Can llms effectively leverage graph structural information: when and why." In: *arXiv preprint arXiv:2309.16595* (2023).

[150] Md Rafiqul Islam, Shaowu Liu, Xianzhi Wang, and Guandong Xu. "Deep learning for misinformation detection on online social networks: a survey and new perspectives." In: *Social Network Analysis and Mining* 10.1 (2020), p. 82.

[151] P. Jaccard. "Étude comparative de la distribution florale dans une portion des Alpes et des Jura." In: *Bull Soc Vaudoise Sci Nat* 37 (1901), pp. 547–579.

[152] Ankit Kumar Jain, Somya Ranjan Sahoo, and Jyoti Kaubiyal. "Online social networks security and privacy: comprehensive review and analysis." In: *Complex & Intelligent Systems* 7.5 (2021), pp. 2157–2177.

[153] Marc Jaxa-Rozen and Jan H. Kwakkel. *pyNetLogo*. https://pynetlogo.readthedocs.io/en/latest/. Last accessed: 31st October 2022.

[154] Marc Jaxa-Rozen and Jan H. Kwakkel. "PyNetLogo: Linking NetLogo with Python." In: *Journal of Artificial Societies and Social Simulation* 21.2 (2018), p. 4. DOI: 10.18564/jasss.3668.

[155] Longbin Jiang and Chunxiao Zhao. "The Netlogo-Based Dynamic Model for the Teaching." In: *2009 Ninth International Conference on Hybrid Intelligent Systems*. Vol. 2. 2009, pp. 49–53. DOI: 10.1109/HIS.2009.121.

[156] Donald R. Jones, Matthias Schonlau, and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions." In: *Journal of Global Optimization* 13.4 (1998), 455 – 492. DOI: 10.1023/A:1008306431147.

[157] R. Jung, J.-H. Jourdan, R. Krebbers, and D. Dreyer. "RustBelt: Securing the foundations of the rust programming language." In: *Proceedings of the ACM on Programming Languages* 2.POPL (2018). DOI: 10.1145/3158154.

[158] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. "RustBelt: Securing the foundations of the Rust programming language." In: *Proceedings of the ACM on Programming Languages* 2.POPL (2017), pp. 1–34. DOI: 10.1145/3158154.

[159] Edward Junprung. "Exploring the intersection of large language models and agent-based modeling via prompt engineering." In: *arXiv preprint arXiv:2308.07411* (2023). DOI: 10.48550/arXiv.2308.07411.

[160] J. L. Juul, A. R. Benson, and J. Kleinberg. "Hypergraph patterns and collaboration structure." In: *Frontiers in Physics* 11 (2024). ISSN: 2296-424X. DOI: 10.3389/fphy.2023.1301994.

[161] Kaggle. *Kaggle dataset for Stackoverflow platform*. https://www.kaggle.com/datasets/stackoverflow/stackoverflow?datasetId=2268. [Last accessed: March 2023]. 2022.

[162] John Kallaugher, Michael Kapralov, and Eric Price. "The sketching complexity of graph and hypergraph counting." In: *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2018, pp. 556–567. DOI: 10.1109/FOCS.2018.00059..

[163] Takumi Kato and Ryota Kamoshida. "Multi-Agent Simulation Environment for Logistics Warehouse Design Based on Self-Contained Agents." In: *Applied Sciences* 10.21 (2020). ISSN: 2076-3417. DOI: 10.3390/app10217552.

[164] Jackie Kazil, David Masad, and Andrew Crooks. *Mesa: Agent-based modeling in Python 3+*. https://mesa.readthedocs.io/en/latest/. Last accessed: 31st October 2022.

[165] Jackie Kazil, David Masad, and Andrew Crooks. *Mesa CI/CD*. https://github.com/projectmesa/mesa/blob/main/.github/workflows/build_lint.yml. Last accessed: 9st December 2022.

[166] Jackie Kazil, David Masad, and Andrew Crooks. "Utilizing Python for Agent-Based Modeling: The Mesa Framework." In: *Social, Cultural, and Behavioral Modeling*. Springer International Publishing, 2020, pp. 308–317. DOI: 10.1007/978-3-030-61255-9_30.

[167] A. Khatua, V. Sharma Mailthody, B. Taleka, T. Ma, X. Song, and W. Hwu. "IGB: Addressing The Gaps In Labeling, Features, Heterogeneity, and Size of Public Graph Datasets for Deep Learning Research." In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '23. New York, NY, USA: Association for Computing Machinery, 2023, 4284–4295. ISBN: 9798400701030. DOI: 10.1145/3580305.3599843.

[168] S. Kim, D. Lee, Y. Kim, J. Park, T. Hwang, and K. Shin. "Datasets, tasks, and training methods for large-scale hypergraph learning." In: *Data Mining and Knowledge Discovery* 37.6 (2023), pp. 2216–2254. ISSN: 1573-756X. DOI: 10.1007/s10618-023-00952-6.

[169] T. N. Kipf and M. Welling. "Semi-supervised classification with graph convolutional networks." In: International Conference on Learning Representations, ICLR, 2017. DOI: https://doi.org/10.48550/arXiv.1609.02907.

[170] B. Klimt and Y. Yang. "Introducing the Enron Corpus." In: *CEAS 2004 - First Conference on Email and Anti-Spam, July 30-31, 2004, Mountain View, California, USA*. 2004.

[171] Daniel Kornhauser, Uri Wilensky, and William Rand. "Design Guidelines for Agent Based Model Visualization." In: *Journal of Artificial Societies and Social Simulation* 12.2 (2009), p. 1. ISSN: 1460-7425.

[172] Kalliopi Kravari and Nick Bassiliades. "A Survey of Agent Platforms." In: *Journal of Artificial Societies and Social Simulation* 18.1 (2015), p. 11. DOI: 10.18564/jasss.2661.

[173] Sanjay Kumar, Abhishek Mallik, Anavi Khetarpal, and Bhawani Sankar Panda. "Influence maximization in social networks using graph embedding and graph neural network." In: *Information Sciences* 607 (2022), pp. 1617–1636.

[174] T. Kumar, K. Darwin, S. Parthasarathy, and B. Ravindran. "HPRA: Hyperedge Prediction using Resource Allocation." In: *Proceedings of the 12th ACM Conference on Web Science*. WebSci '20. New York, NY, USA: Association for Computing Machinery, 2020, 135–143. ISBN: 9781450379892. DOI: 10.1145/3394231.3397903.

[175] J. Kunegis. "KONECT: The Koblenz Network Collection." In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13 Companion. New York, NY, USA: Association for Computing Machinery, 2013, 1343–1350. ISBN: 9781450320382. DOI: 10.1145/2487788.2488173.

[176] J. Kunegis. *The KONECT Project*. http://konect.cc/. Accessed on 30th December 2023. 2017.

[177] Clara L. and Fei Liu. "Effect of control measure on the development of new COVID-19 cases through SIR model simulation." In: *medRxiv* (2020). DOI: 10.1101/2020.10.27.20220590.

[178] STFC Rutherford Appleton Laboratory. *FLAME*. http://flame.ac.uk/. Last accessed: 31st October 2022.

[179] James Ladyman, James Lambert, and Karoline Wiesner. "What is a complex system?" In: *European Journal for Philosophy of Science* 3 (2013), pp. 33–67.

[180] G. Lee, J. Ko, and K. Shin. "Hypergraph motifs: concepts, algorithms, and discoveries." In: *Proceedings of the VLDB Endowment* 13.12 (2020), 2256–2269. ISSN: 2150-8097. DOI: 10.14778/3407790.3407823.

[181] G. Lee, S. Y. Lee, and K. Shin. "VilLain: Self-Supervised Learning on Homogeneous Hypergraphs without Features via Virtual Label Propagation." In: *Proceedings of the ACM on Web Conference 2024*. WWW '24. New York, NY, USA: Association for Computing Machinery, 2024, 594–605. ISBN: 9798400701719. DOI: 10.1145/3589334.3645454.

[182] G. Lee and K. Shin. "THyMe+: Temporal Hypergraph Motifs and Fast Algorithms for Exact Counting." In: *2021 IEEE International Conference on Data Mining (ICDM)*. 2021, pp. 310–319. DOI: 10.1109/ICDM51629.2021.00042.

[183] G. Lee, S. Yoon, J. Ko, H. Kim, and K. Shin. "Hypergraph motifs and their extensions beyond binary." In: *The VLDB Journal* 33.3 (2024), pp. 625–665. ISSN: 0949-877X. DOI: 10.1007/s00778-023-00827-8.

[184] Geon Lee and Kijung Shin. "Temporal hypergraph motifs." In: *Knowledge and Information Systems* 65.4 (2023), pp. 1549–1586. DOI: 10.1007/s10115-023-01837-2.

[185] SingLing Lee and Hann-Jang Ho. "Algorithms and complexity for weighted hypergraph embedding in a cycle." In: *First International Symposium on Cyber Worlds, 2002. Proceedings.* IEEE. 2002, pp. 70–75.

[186] Florin Leon. "ActressMAS, a .NET Multi-Agent Framework Inspired by the Actor Model." In: *Mathematics* 10.3 (2022). DOI: 10.3390/math10030382.

[187] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. 2014.

[188] Leudz. https://github.com/leudz/shipyard.

[189] Pan Li. *Datasets*. https://sites.google.com/view/panli-purdue/datasets. Accessed on 30th December 2023. 2020.

[190] Y. Li, Z. Li, P. Wang, J. Li, X. Sun, H. Cheng, and J.X. Yu. "A survey of graph meets large language model: Progress and future directions." In: *arXiv preprint arXiv:2311.12399* (2023).

[191] Y. Liu and M. Anwar. "Learning Programming in Social Media: An NLP-powered Reddit Study." In: Institute of Electrical and Electronics Engineers Inc., 2022, pp. 55–58. ISBN: 9781665471848. DOI: 10.1109/TransAI54797.2022.00015.

[192] Q. F. Lotito, F. Musciotto, F. Battiston, and A. Montresor. "Exact and sampling methods for mining higher-order motifs in large hypergraphs." In: *Computing* 106.2 (2024), pp. 475–494. ISSN: 1436-5057. DOI: 10.1007/s00607-023-01230-5.

[193] Q. F. Lotito, F. Musciotto, A. Montresor, and F. Battiston. "Higher-order motif analysis in hypergraphs." In: *Communications Physics* 5.1 (2022), p. 79. ISSN: 2399-3650. DOI: 10.1038/s42005-022-00858-7.

[194] C. Lu, J. Peltonen, and T. Nummenmaa. "Game postmortems vs. developer Reddit AMAs: Computational analysis of developer communication." In: Association for Computing Machinery, 2019. ISBN: 9781450372176. DOI: 10.1145/3337722.3337727.

[195] Y. Lu, X. Mao, M. Zhou, Y. Zhang, T. Wang, and Z. Li. "Haste Makes Waste: An Empirical Study of Fast Answers in Stack Overflow." In: Institute of Electrical and Electronics Engineers Inc., 2020, pp. 23–34. ISBN: 9781728156194. DOI: 10.1109/ICSME46990.2020.00013.

[196] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. "Mason: A multiagent simulation environment." In: *Simulation* 81.7 (2005), pp. 517–527.

[197] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan. "Characteristics and challenges of low-code development: The practitioners perspective." In: IEEE Computer Society, 2021. ISBN: 9781450386654. DOI: 10.1145/3475716.3475782.

[198] L. Lü and T. Zhou. "Link prediction in complex networks: A survey." In: *Physica A: Statistical Mechanics and its Applications* 390.6 (2011), pp. 1150–1170. ISSN: 0378-4371. DOI: 10.1016/j.physa.2010.11.027.

[199] C. M. Macal. "Everything you need to know about agent-based modelling and simulation." In: *Journal of Simulation* 10.2 (2016), pp. 144–156. DOI: 10.1057/jos.2016.7.

[200] C. M. Macal and M. J. North. "Tutorial on agent-based modelling and simulation." In: *Journal of Simulation* 4.3 (2010), pp. 151–162. DOI: 10.1057/jos.2010.3.

[201] Peter Macgregor. *Datasets*. https://pmacg.io/datasets.html. Accessed on 30th December 2023. 2021.

[202] V. Martínez, F. Berzal, and J.-C. Cubero. "A Survey of Link Prediction in Complex Networks." In: *ACM Comput. Surv.* 49.4 (2016). ISSN: 0360-0300. DOI: 10.1145/3012704.

[203] R. Mastrandrea, J. Fournet, and A. Barrat. "Contact Patterns in a High School: A Comparison between Data Collected Using Wearable Sensors, Contact Diaries and Friendship Surveys." In: *PLOS ONE* 10.9 (2015), pp. 1–26. DOI: 10.1371/journal.pone.0136497.

[204] Nicholas D. Matsakis and Felix S. Klock. "The rust language." In: HILT '14 (2014), 103–104. DOI: 10.1145/2663171.2663188.

[205] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. "Automating the Construction of Internet Portals with Machine Learning." In: *Information Retrieval* 3.2 (2000), pp. 127–163. ISSN: 1573-7659. DOI: 10.1023/A:1009953814988.

[206] Alexey N Medvedev, Renaud Lambiotte, and Jean-Charles Delvenne. "The anatomy of Reddit: An overview of academic research." In: *Dynamics on and of Complex Networks III: Machine Learning and Statistical Physics Approaches 10* (2019), pp. 183–204.

[207] S. Meldrum, S.A. Licorish, and B.T.R. Savarimuthu. "Crowdsourced knowledge on stack overflow: A systematic mapping study." In: vol. Part F128635. Association for Computing Machinery, 2017, pp. 180–185. ISBN: 9781450348041. DOI: 10.1145/3084226.3084267.

[208] C. Meng and H. Motevalli. "Link prediction in social networks using hyper-motif representation on hypergraph." In: *Multimedia Systems* 30.3 (2024), p. 123. ISSN: 1432-1882. DOI: 10.1007/s00530-024-01324-w.

[209] Mesa Community. *Mesa 3D graphics library*. https://github.com/Mesa3D/mesa. Last accessed: 31st October 2022.

[210] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*. 2023. URL: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf.

[211] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. "Network Motifs: Simple Building Blocks of Complex Networks." In: *Science* 298.5594 (2002), pp. 824–827. DOI: 10.1126/science.298.5594.824.

[212] I. Moutidis and H.T.P. Williams. "Community evolution on stack overflow." In: *PLoS ONE* 16.6 June 2021 (2021). ISSN: 19326203. DOI: 10.1371/journal.pone.0253010.

[213] Fukuda Munehiro, Christian Freksa, Eric Salathe, Kim Wooyoung, and Kiyoki Yasushi. *MASS: A Parallelizing Library for Multi-Agent Spatial Simulation*. https://depts.washington.edu/dslab/MASS/. Last accessed: 31st October 2022.

[214] Fukuda Munehiro, Christian Freksa, Eric Salathe, Kim Wooyoung, and Kiyoki Yasushi. *MASS CI/CD*. https://bitbucket.org/mass_library_developers/mass_java_core/src/master/bitbucket-pipelines.yml. Last accessed: 9st December 2022.

[215] A. Nadiri and F.W. Takes. "A Large-scale Temporal Analysis of User Lifespan Durability on the Reddit Social Media Platform." In: Association for Computing Machinery, Inc, 2022, pp. 677–685. ISBN: 9781450391306. DOI: 10.1145/3487553.3524699.

[216] Leonie Neuhäuser, Andrew Mellor, and Renaud Lambiotte. "Multibody interactions and nonlinear consensus dynamics on networked systems." In: *Physical Review E* 101.3 (2020), p. 032310. DOI: 10.1103/PhysRevE.101.032310.

[217] J. Nielsen. *The 90-9-1 Rule for Participation Inequality in Social Media and Online Communities*. https://www.nngroup.com/articles/participation-inequality/. Last accessed: March 2023. 2006.

[218] Cynthia Nikolai and Gregory Madey. "Tools of the Trade: A Survey of Various Agent Based Modeling Platforms." In: *Journal of Artificial Societies and Social Simulation* 12.2 (2009), pp. 1–2.

[219] D. Nóbrega and P. Ribeiro. "Computing Motifs in Hypergraphs." In: *Complex Networks XV*. Cham: Springer Nature Switzerland, 2024, pp. 55–70. ISBN: 978-3-031-57515-0. DOI: 10.1007/978-3-031-57515-0_5.

[220] Michael J. North, Nicholson T. Collier, Jonathan Ozik, Eric R. Tatara, Charles M. Macal, Mark Bragen, and Pam Sydelko. "Complex adaptive systems modeling with Repast Simphony." In: *Complex Adaptive Systems Modeling* (2013). DOI: 10.1186/2194-3206-1-3.

[221] Jalil Nourisa. *CppyABM*. https://pypi.org/project/cppyabm/. Last accessed: 31st October 2022.

[222] Jalil Nourisa, Berit Zeller-Plumhoff, and Regine Willumeit-Römer. "CppyABM: An open-source agent-based modeling library to integrate C++ and Python." In: *Software: Practice and Experience* 52.6 (2022), pp. 1337–1351. DOI: 10.1002/spe.3067.

[223] Karynna Okabe-Miyamoto, Lisa C Walsh, Daniel J Ozer, and Sonja Lyubomirsky. "Measuring the experience of social connection within specific social interactions: The Connection During Conversations Scale (CDCS)." In: *Plos one* 19.1 (2024), e0286408.

[224] OpenAI. *ChatGPT website*. https://chat.openai.com/. [Last accessed: March 2023]. 2023.

[225] Stack Overflow. https://stackoverflow.com/. [Last accessed: March 2023]. 2008.

[226] Jonathan Ozik, Nicholson Collier, Justin Wozniak, Carmine Spagnuolo, and Gary An. *EMEWS: Extreme-scale Model Exploration with Swift*. https://emews.github.io/. Last accessed: 31st October 2022.

[227] Constantin-Valentin Pal, Florin Leon, Marcin Paprzycki, and Maria Ganzha. "A Review of Platforms for the Development of Agent Systems." In: *CoRR* abs/2007.08961 (2020).

[228] George Pallis, Demetrios Zeinalipour-Yazti, and Marios D Dikaiakos. "Online social networks: status and trends." In: *New directions in web data management 1* (2011), pp. 213–234.

[229] Marios Papachristou. *Datasets*. https://papachristoumarios.github.io/dat Accessed on 30th December 2023. 2020.

[230] Hazel R Parry. "Agent-based modeling, large-scale simulations." In: *Complex Social and Behavioral Systems: Game Theory and Agent-Based Models* (2020), pp. 913–926.

[231] Hazel R Parry and Mike Bithell. "Large scale agent-based modelling: A review and guidelines for model scaling." In: *Agent-based models of geographical systems* (2011), pp. 271–308. DOI: 10.1007/978-90-481-8927-4_14.

[232] P. Patil, G. Sharma, and M. N. Murty. "Negative Sampling for Hyperlink Prediction in Networks." In: *Advances in Knowledge Discovery and Data Mining*. Cham: Springer International Publishing, 2020, pp. 607–619. ISBN: 978-3-030-47436-2. DOI: 10.1007/978-3-030-47436-2_46.

[233] S. Patra and A. Mohapatra. "Review of tools and algorithms for network motif discovery in biological networks." In: *IET Systems Biology* 14.4 (2020), pp. 171–189. DOI: 10.1049/iet-syb.2020.0004.

[234] David J. Pearce. "A Lightweight Formalism for Reference Lifetimes and Borrowing in Rust." In: *ACM Transactions on Programming Languages and Systems* 43.1 (2021). ISSN: 0164-0925. DOI: 10.1145/3443420.

[235] L. Felipe Perrone, Christopher S. Main, and Bryan C. Ward. "SAFE: Simulation automation framework for experiments." In: *Proceedings of the 2012 Winter Simulation Conference (WSC)*. 2012, pp. 1–12. DOI: 10.1109/WSC.2012.6465286.

[236] Donovan Platt. "A comparison of economic agent-based model calibration methods." In: *Journal of Economic Dynamics and Control* 113 (2020), p. 103859. DOI: 10.1016/j.jedc.2020.103859.

[237] Pushshift. *API to retrieve data from Reddit dump*. https://api.pushshift.io. [Last accessed: March 2023]. 2023.

[238] Steven F. Railsback, Daniel Ayllón, Uta Berger, Volker Grimm, Steven Lytinen, Colin Sheppard, and Jan Thiele. "Improving Execution Speed of Models Implemented in NetLogo." In: *Journal of Artificial Societies and Social Simulation* 20.1 (2017), p. 3. ISSN: 1460-7425. DOI: 10.18564/jasss.3282.

[239] Steven F. Railsback, Steven L. Lytinen, and Stephen K. Jackson. "Agent-based Simulation Platforms: Review and Development Recommendations." In: *SIMULATION* 82.9 (2006), pp. 609–623. DOI: 10.1177/0037549706073695.

[240] Ralith. https://github.com/Ralith/hecs.

[241] Reddit. https://reddit.com/. [Last accessed: March 2023]. 2005.

[242] Carl Orge Retzlaff, Laura Burbach, Lilian Kojan, Patrick Halbach, Johannes Nakayama, Martina Ziefle, and André Calero Valdez. "Fear, Behaviour, and the COVID-19 Pandemic: A City-Scale Agent-Based Model Using Socio-Demographic and Spatial Map Data." In: *Journal of Artificial Societies and Social Simulation* 25.1 (2022), p. 3. ISSN: 1460-7425. DOI: 10.18564/jasss.4723.

[243] Romain Reuillon, Mathieu Leclaire, and Sebastien Rey-Coyrehourcq. "OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models." In: *Future Generation Computer Systems* 29.8 (2013). Including Special sections: Advanced Cloud Monitoring Systems & The fourth IEEE International Conference on e-Science 2011 — e-Science Applications and Tools & Cluster, Grid, and Cloud Computing, pp. 1981–1990. ISSN: 0167-739X. DOI: 10.1016/j.future.2013.05.003.

[244] Craig W. Reynolds. "Flocks, Herds, and Schools: A Distributed Behavioral Model." In: *SIGGRAPH Computer Graphics (ACM)* 21.4 (1987), 25–34. ISSN: 0097-8930. DOI: 10.1145/37402.37406.

[245] Matteo G. Richiardi and Ross E. Richardson. *JAS-mine*. http://jas-mine.net/. Last accessed: 31st October 2022.

[246] Matteo G. Richiardi and Ross E. Richardson. "JAS-mine: A new platform for microsimulation and agent-based modelling." In: *IJM* 10.1 (2017), pp. 106–134. DOI: 10.34196/ijm.00151.

[247] Paul Richmond. *FLAME GPU CI/CD*. https://github.com/FLAMEGPU/FLAMEGPU2/blob/master/.github/workflows/Draft-Release.yml. Last accessed: 9st December 2022.

[248] Paul Richmond. *FLAME GPU*. https://flamegpu.com/. Last accessed: 31st October 2022.

[249] R.A. Rossi and N.K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization." In: *AAAI*. 2015. URL: https://networkrepository.com.

[250] Alban Rousset, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe. "A survey on parallel and distributed multi-agent systems for high performance computing simulations." In: *Computer Science Review* 22 (2016), pp. 27–46. DOI: 10.1016/j.cosrev.2016.08.001.

[251] Xavier Rubio-Campillo. *Pandora*. http://xrubio.github.io/pandora/. Last accessed: 31st October 2022.

[252] Xavier Rubio-Campillo. "Pandora: A Versatile Agent-Based Modelling Platform for Social Simulation." In: *Proceedings of SIMUL 2014, The Sixth International Conference on Advances in System Simulation*. 2014, pp. 29–34. DOI: 10.13140/2.1.5149.4086.

[253] Eric Russell and James Hovet. *NetLogo Gis Extension*. https://ccl.northwestern.edu/netlogo/docs/gis.html. Last accessed: 31st October 2022.

[254] Rust Doc. *Rust Doc Lang - Meet Safe and Unsafe*. https://doc.rust-lang.org/nomicon/meet-safe-and-unsafe.html. 2023.

[255] Flora Sakketou, Allison Lahnala, Liane Vogel, and Lucie Flek. "Investigating User Radicalization: A Novel Dataset for Identifying Fine-Grained Temporal Shifts in Opinion." In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. 2022, pp. 3798–3808. DOI: 10.48550/arXiv.2204.10190.

[256] Jan Salecker, Marco Sciaini, Katrin M. Meyer, and Kerstin Wiegand. "The nlrx r package: A next-generation framework for reproducible NetLogo model analyses." In: *Methods in Ecology and Evolution* 10.11 (2019), pp. 1854–1863. DOI: 10.1111/2041-210X.13286.

[257] SanderMertens. https://github.com/SanderMertens/flecs.

[258] A. Saxena and H. Reddy. "Users roles identification on online crowdsourced Q&A platforms and encyclopedias: a survey." In: *Journal of Computational Social Science* 5.1 (2022), pp. 285–317. ISSN: 24322717. DOI: 10.1007/s42001-021-00125-9.

[259] Luke Sean, Gabriel Catalin Balan, Keith Sullivan, and Liviu Panait. *MASON Multiagent Simulation Toolkit*. https://cs.gmu.edu/~eclab/projects/mason/. Last accessed: 31st October 2022.

[260] S. Sengupta. "'Learning to code in a virtual world': A preliminary comparative analysis of discourse and learning in two online programming communities." In: Association for Computing Machinery, 2020, pp. 389–394. ISBN: 9781450380591. DOI: 10.1145/3406865.3418319.

[261] Adrian Serrano-Hernandez, Javier Faulin, Patrick Hirsch, and Christian Fikar. "Agent-based simulation for horizontal cooperation in logistics and transportation: From the individual to the grand coalition." In: *Simulation Modelling Practice and Theory* 85 (2018), pp. 47–59. ISSN: 1569-190X. DOI: 10.1016/j.simpat.2018.04.002.

[262] P. O. Siebers, C. M. Macal, J. Garnett, D. Buxton, and M. Pidd. "Discrete-event simulation is dead, long live agent-based simulation!" In: *Journal of Simulation* 4.3 (2010), pp. 204–210. DOI: 10.1057/jos.2010.14.

[263] A. F. Siegenfeld and Y. Bar-Yam. "An Introduction to Complex Systems Science and Its Applications." In: *Complexity* 2020 (2020), p. 6105872. ISSN: 1076-2787. DOI: 10.1155/2020/6105872.

[264] Carmine Spagnuolo, Giuseppe D'Ambrosio, Daniele De Vinco, Luca Postiglione, Francesco Foglia, and Pasquale Caramante. *krABMaga CI/CD*. https://github.com/

krABMaga/krABMaga/blob/main/.github/workflows/rust-ci.yml. Last accessed: 9st December 2022.

[265] Carmine Spagnuolo, Giuseppe D'Ambrosio, Daniele De Vinco, Luca Postiglione, Francesco Foglia, and Pasquale Caramante. *krABMaga*. https://krabmaga.github.io/. Last accessed: 31st October 2022.

[266] I. Srba and M. Bielikova. "Why is Stack Overflow Failing? Preserving Sustainability in Community Question Answering." In: *IEEE Software* 33.4 (2016), pp. 80–89. DOI: 10.1109/MS.2016.34.

[267] Stackoverflow. *annual survey of Stackoverflow platform*. https://survey.stackoverflow.co/2022. [Last accessed: March 2023]. 2022.

[268] Charles Staelin. *NetLogo stats Extension*. https://github.com/cstaelin/Stats-Extension. Last accessed: 31st October 2022.

[269] Russell K. Standish and Richard Leow. *EcoLab CI/CD*. https://github.com/highperformancecoder/ecolab/blob/master/.github/workflows/main.yml. Last accessed: 9st December 2022.

[270] Russell K. Standish and Richard Leow. *EcoLab*. https://ecolab.sourceforge.net/. Last accessed: 31st October 2022.

[271] Russell K. Standish and Richard Leow. "EcoLab: Agent Based Modeling for C++ programmers." In: *arXiv* (2004). DOI: 10.48550/ARXIV.CS/0401026.

[272] StartupBonsai. *23+ Reddit Statistics For 2023: Users, Revenue, And Growth*. https://startupbonsai.com/reddit-statistics. [Last accessed: March 2023]. 2023.

[273] J. Stehlé et al. "High-Resolution Measurements of Face-to-Face Contact Patterns in a Primary School." In: *PLOS ONE* 6.8 (2011), pp. 1–13. DOI: 10.1371/journal.pone.0023176.

[274] Forrest Stonedahl and Uri Wilensky. "Finding Forms of Flocking: Evolutionary Search in ABM Parameter-Spaces." In: *Multi-Agent-Based Simulation XI*. Springer Berlin Heidelberg, 2011, pp. 61–75. ISBN: 978-3-642-18345-4. DOI: 10.1007/978-3-642-18345-4_5.

[275] Keith Sullivan, Mark Coletti, and Sean Luke. *GeoMason: Geospatial Support for MASON*. https://cs.gmu.edu/~eclab/projects/mason/extensions/geomason/. 2010.

[276] J. Surowiecki. "The Wisdom of Crowds." In: 37 (3 2016), pp. 351–354.

[277] Vinoth Suryanarayanan, Georgios Theodoropoulos, and Michael Lees. "PDES-MAS: Distributed Simulation of Multi-agent Systems." In: *Procedia Computer Science* 18 (2013). 2013 International Conference on Computational Science, pp. 671–681. DOI: 10.1016/j.procs.2013.05.231.

[278] Patrick Taillandier, Benoit Gaudou, Arnaud Grignard, Quang-Nghi Huynh, Nicolas Marilleau, Philippe Caillou, Damien Philippon, and Alexis Drogoul. *GAMA Platform*. https://gama-platform.org/. Last accessed: 31st October 2022.

[279] Patrick Taillandier, Benoit Gaudou, Arnaud Grignard, Quang-Nghi Huynh, Nicolas Marilleau, Philippe Caillou, Damien Philippon, and Alexis Drogoul. "Building, composing and experimenting complex spatial models with the GAMA platform." In: *GeoInformatica* 23.2 (2019), pp. 299–322. DOI: 10.1007/s10707-018-00339-6.

[280] W. Tang and D.A. Bennett. "The Explicit Representation of Context in Agent-Based Models of Complex Adaptive Spatial Systems." In: *Annals of the Association of American Geographers* 100.5 (2010), pp. 1128–1155.

[281] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. "Gemma: Open models based on gemini research and technology." In: *arXiv preprint arXiv:2403.08295* (2024). DOI: 10.48550/arXiv.2403.08295.

[282] Georgios Theodoropoulos, Robert Minson, R Ewald, M Lees, AM Uhrmacher, and D Weyns. "Simulation Engines for Multi-Agent Systems." In: *Multi-Agent Systems: Simulation and Applications*. Taylor & Francis, 2009. Chap. 3. ISBN: 9781420070231.

[283] Jan C Thiele. *RNetLogo*. http://rnetlogo.r-forge.r-project.org/. Last accessed: 31st October 2022.

[284] Jan C Thiele. "R Marries NetLogo: Introduction to the RNetLogo Package." In: *Journal of Statistical Software* 58.2 (2014), 1–41. DOI: 10.18637/jss.v058.i02.

[285] Seth Tisue and Uri Wilensky. *BehaviorSpace*. https://ccl.northwestern.edu/netlogo/docs/behaviorspace.html. Last accessed: 31st October 2022.

[286] Seth Tisue and Uri Wilensky. "Netlogo: A simple environment for modeling complexity." In: *International conference on complex systems*. Vol. 21. Citeseer. 2004, pp. 16–21.

[287] Robert Tobias and Carole Hofmann. "Evaluation of free Java-libraries for social-scientific agent based simulation." In: *Journal of Artificial Societies and Social Simulation* 7 (2004).

[288] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. "Llama: Open and efficient foundation language models." In: *arXiv preprint arXiv:2302.13971* (2023). DOI: 10.48550/arXiv.2302.13971.

[289] Amaury Trujillo and Stefano Cresci. "Make reddit great again: assessing community effects of moderation interventions on r/the_donald." In: *Proceedings of the ACM on Human-computer Interaction* 6.CSCW2 (2022), pp. 1–28.

[290] M. Tsimpoukelli, J. L. Menick, S. Cabi, S.M. Eslami, O. Vinyals, and F. Hill. "Multimodal few-shot learning with frozen language models." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 200–212.

[291] Unity. https://unity.com/dots.

[292] Wilensky Uri, Arthur Hjorth, Connor Bain, Nicolas Payette, Bryan Head, and Jason Bertsche. *NetLogo CI/CD*. https://github.com/NetLogo/NetLogo/blob/hexy/.github/workflows/main.yml. Last accessed: 9st December 2022.

[293] Wilensky Uri, Arthur Hjorth, Connor Bain, Nicolas Payette, Bryan Head, and Jason Bertsche. *NetLogo*. https://ccl.northwestern.edu/netlogo. Last accessed: 31st October 2022.

[294] P. Verduyn, N. Gugushvili, K. Massar, K. Täht, and E. Kross. "Social comparison on social networking sites." In: *Current opinion in psychology* 36 (2020), pp. 32–37.

[295] Muhammad Waleed, Tai-Won Um, Tariq Kamal, Aftab Khan, and Zaka Ullah Zahid. "SIM-D: An Agent-Based Simulator for Modeling Contagion in Population." In: *Applied Sciences* 10.21 (2020). ISSN: 2076-3417. DOI: 10.3390/app10217745.

[296] I. Waller and A. Anderson. "Generalists and specialists: Using community embeddings to quantify activity diversity in online platforms." In: Association for Computing Machinery, Inc, 2019, pp. 1954–1964. ISBN: 9781450366748. DOI: 10.1145/3308558.3313729.

[297] Carmine Haoliang Wang et al. *Distributed MASON*. https://cs.gmu.edu/~eclab/projects/mason/extensions/distributed/. Last accessed: 31st October 2022.

[298] Zhiping Wang, Haofei Yin, and Xin Jiang. "Exploring the dynamic growth mechanism of social networks using evolutionary hypergraph." In: *Physica A: Statistical Mechanics and its Applications* 544 (2020), p. 122545.

[299] Duncan J Watts and Steven H Strogatz. "Collective dynamics of 'small-world'networks." In: *nature* 393.6684 (1998), pp. 440–442.

[300] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. "Emergent abilities of large language models." In: *arXiv preprint arXiv:2206.07682* (2022).

[301] David R. White. "Software review: the ECJ toolkit." In: *Genetic Programming and Evolvable Machines* 13.1 (2012), pp. 65–67. DOI: 10.1007/s10710-011-9148-z.

[302] Dennis Wiebusch and Marc Erich Latoschik. "Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems." In: *2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. IEEE. 2015, pp. 25–32.

[303] Uri Wilensky. "NetLogo 3.1. 3." In: (2006).

[304] Uri Wilensky and Kenneth Reisman. "ConnectedScience: Learning Biology through Constructing and Testing Computational Theories – an Embodied Modeling Approach." In: *InterJournal of Complex Systems*. 1998, pp. 1–12.

[305] Uri Wilensky and Kenneth Reisman. "Thinking Like a Wolf, a Sheep, or a Firefly: Learning Biology Through Constructing and Testing Computational Theories—An Embodied Modeling Approach." In: *Cognition and Instruction* 24.2 (2006), pp. 171–209. DOI: 10.1207/s1532690xci2402_1.

[306] E. Wong, B. Baur, S. Quader, and C.-H. Huang. "Biological network motif detection: principles and practice." In: *Briefings in Bioinformatics* 13.2 (2011), pp. 202–215. ISSN: 1467-5463. DOI: 10.1093/bib/bbr033.

[307] Junchao Wu, Shu Yang, Runzhe Zhan, Yulin Yuan, Derek F Wong, and Lidia S Chao. "A survey on llm-gernerated text detection: Necessity, methods, and future directions." In: *arXiv preprint arXiv:2310.14724* (2023). DOI: 10.48550/arXiv.2310.14724.

[308] M. Wu, R. Aranovich, and V. Filkov. "Evolution and differentiation of the cybersecurity communities in three social question and answer sites: A mixed-methods analysis." In: *PLoS ONE* 16.12 December (2021). ISSN: 19326203. DOI: 10.1371/journal.pone.0261954.

[309] Liwei Xue, Guo-Ping Liu, and Wenshan Hu. "All-in-One Framework for Design, Simulation, and Practical Implementation of Distributed Multiagent Control Systems." In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2024), pp. 1–14.

[310] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar. "HyperGCN: A New Method For Training Graph Convolutional Networks on Hypergraphs." In: *Advances in Neural Information Processing Systems*. Vol. 32. NeurIPS. Curran Associates, Inc., 2019.

[311] D. Yang, B. Qu, J. Yang, and P. Cudre-Mauroux. "Revisiting User Mobility and Social Relationships in LBSNs: A Hypergraph Embedding Approach." In: *The World Wide Web Conference*. WWW '19. Association for Computing

Machinery, 2019, 2147–2157. ISBN: 9781450366748. DOI: 10.1145/3308558.3313635.

[312] Dingqi Yang, Bingqing Qu, Jie Yang, and Philippe Cudré-Mauroux. "Lbsn2vec++: Heterogeneous hypergraph embedding for location-based social networks." In: *IEEE Transactions on Knowledge and Data Engineering* 34.4 (2020), pp. 1843–1855.

[313] S. Yu, Y. Feng, D. Zhang, H. D. Bedru, B. Xu, and F. Xia. "Motif discovery in networks: A survey." In: *Computer Science Review* 37 (2020), p. 100267. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2020.100267.

[314] S. Yu, J. Xu, C. Zhang, F. Xia, Z. Almakhadmeh, and A. Tolba. "Motifs in Big Networks: Methods and Applications." In: *IEEE Access* 7 (2019), pp. 183322–183338. DOI: 10.1109/ACCESS.2019.2960044.

[315] Tae-Sub Yun, Dongjun Kim, Il-Chul Moon, and Jang Won Bae. "Agent-Based Model for Urban Administration: A Case Study of Bridge Construction and its Traffic Dispersion Effect." In: *Journal of Artificial Societies and Social Simulation* 25.4 (2022), p. 5. ISSN: 1460-7425.

[316] A. Zeng and W. Crichton. "Identifying barriers to adoption for rust through online discourse." In: vol. 67. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2019. ISBN: 9783959770910. DOI: 10.4230/OASIcs.PLATEAU.2018.5.

[317] Haoxiang Zhang, Shaowei Wang, Heng Li, Tse-Hsun Chen, and Ahmed E. Hassan. "A Study of C/C++ Code Weaknesses on Stack Overflow." In: *IEEE Transactions on Software Engineering* 48.7 (2022), pp. 2359–2375. DOI: 10.1109/TSE.2021.3058985.

[318] Jiaxin Zhang and Derek T. Robinson. "Replication of an agent-based model using the Replication Standard." In: *Environmental Modelling & Software* 139 (2021), p. 105016. ISSN: 1364-8152. DOI: 10.1016/j.envsoft.2021.105016.

[319] M. Zhang, Z. Cui, S. Jiang, and Y. Chen. "Beyond link prediction: Predicting hyperlinks in adjacency space." In: AAAI press, 2018, 4430 – 4437. DOI: 10.1609/aaai.v32i1.11780.

[320] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. "A survey of large language models." In: *arXiv preprint arXiv:2303.18223* (2023). DOI: 10.48550/arXiv.2303.18223.

[321] D. Zheng, M. Wang, Q. Gan, X. Song, Z. Zhang, and G. Karypis. "Scalable Graph Neural Networks with Deep Graph Library." In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. New York, NY, USA: Association for Computing Machinery, 2021, 1141–1142. ISBN: 9781450382977. DOI: 10.1145/3437963.3441663.

[322] Jianming Zhu, Junlei Zhu, Smita Ghosh, Weili Wu, and Jing Yuan. "Social influence maximization in hypergraph in social networks." In: *IEEE Transactions on Network Science and Engineering* 6.4 (2018), pp. 801–811. DOI: 10.1109/TNSE.2018.2873759.

[323] Chengxiang Zhuge, Chunfu Shao, Shuling Wang, and Ying Hu. "An agent- and GIS-based virtual city creator: A case study of Beijing, China." In: *Journal of Transport and Land Use* 11.1 (2018). DOI: 10.5198/jtlu.2018.1270.

[324] Kashif Zia, Katayoun Farrahi, Andreas Riener, and Alois Ferscha. "An agent-based parallel geo-simulation of urban mobility during city-scale evacuation." In: *SIMULATION* 89.10 (2013), pp. 1184–1214. DOI: 10.1177/0037549713485468.

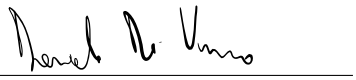
[325] *rngs - Random Number Generator extension for NetLogo*. https://github.com/AFMac/rngs. Last accessed: 31st October 2022.

[326] skypjack. https://github.com/skypjack/entt.

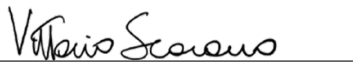
DECLARATION ON GENERATIVE AI

During the preparation of this work, the author(s) used Generative AI for grammar and spelling check through Grammarly. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

Fisciano, November 2025



Candidate Name



Supervisor Name