

CROWD: A FRAMEWORK FOR AGENT-BASED SOCIAL NETWORK SIMULATION AND ITS APPLICATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Ann Nedime Neşe Rende
June 2025

Crowd: A Framework for Agent-Based Social Network Simulation and
Its Applications

By Ann Nedime Neşe Rende

June 2025

We certify that we have read this thesis and that in our opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.

Özgür Ulusoy(Advisor)

İbrahim Körpeoğlu

İsmail Sengör Altıngövde

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

Copyright Information

Personal use of the material in full or in part in this thesis is permitted.

A. N. N. Rende, T. Yilmaz and O. Ulusoy, “Crowd: A Social Network Simulation Framework,” IEEE Transactions on Computational Social Systems, IEEE, 2025.
doi: 10.1109/TCSS.2025.3565377.

URL: <https://ieeexplore.ieee.org/document/11023536>

©IEEE 2025

ABSTRACT

CROWD: A FRAMEWORK FOR AGENT-BASED SOCIAL NETWORK SIMULATION AND ITS APPLICATIONS

Ann Nedime Neşe Rende

M.S. in Computer Engineering

Advisor: Özgür Ulusoy

June 2025

To observe how individual behaviors shape the actions of a broader community, agent-based modeling and simulation (ABMS) has been widely adopted by researchers in social sciences, economics, and epidemiology. While simulations can be executed on general-purpose ABMS frameworks, these tools are not specifically designed for social networks and, therefore, provide limited features, increasing the effort required for complex simulations. In this thesis, we first introduce Crowd, a social network simulator that adopts the agent-based modeling methodology to model real-world phenomena within a network environment. Designed to facilitate easy and quick modeling, Crowd supports simulation setup through YAML configuration and enables further customization through user-defined methods. Other features of Crowd include no-code simulations for diffusion tasks, interactive visualizations, data aggregation, and chart drawing facilities. Designed in Python, Crowd also supports generative agents that rely on LLMs for agents' decisions and connects easily with Python's libraries for data analysis and machine learning. Secondly, we present three case studies to illustrate the application of the framework, including generative agents in epidemics, influence maximization, and networked trust games. Finally, we design and implement a simulation that combines generative agents with social networks to analyze the effects of agents' personalities and social connections in the context of building energy modeling.

Keywords: Social network simulation, agent-based modeling, generative agents.

ÖZET

CROWD: AJAN TABANLI SOSYAL AĞ SİMÜLASYONUNA YÖNELİK BİR ÇERÇEVE VE UYGULAMALARI

Ann Nedime Neşe Rende

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Danışmanı: Özgür Ulusoy

Haziran 2025

Bireysel davranışların topluluk düzeyindeki etkilerini incelemek amacıyla ajan tabanlı modelleme ve simülasyon (ABMS), sosyal bilimler, ekonomi ve epidemiyoloji alanlarındaki araştırmacılar tarafından yaygın olarak kullanılmaktadır. Bu alanlardaki simülasyonlar genel amaçlı ajan tabanlı modelleme çerçeveleri üzerinde yürütülebilse de, bu araçlar sosyal ağlara özel olarak tasarlanmadığı için sınırlı özellikler sunmakta ve karmaşık simülasyonların geliştirilmesi için gereken çabayı arttırmaktadır. Bu tezde ilk olarak, gerçek dünya olgularını bir ağ yapısı içinde modellemek için ajan tabanlı modelleme metodolojisini benimseyen sosyal ağ simülatörü Crowd tanıtılmaktadır. Kolay ve hızlı modelleme imkânı sunmak üzere tasarlanan Crowd, YAML tabanlı konfigürasyonlarla simülasyon kurulumu sağlamak ve kullanıcı tarafından tanımlanan metotlarla özelleştirmelere olanak tanımaktadır. Crowd'un diğer özellikleri arasında, yayılım (diffusion) görevleri için kod yazmadan simülasyon çalıştırabilme, etkileşimli görselleştirmeler, veri toplama ve grafik çizim olanakları yer almaktadır. Python diliyle geliştirilen bu simülatör, ajan kararlarında büyük dil modellerini kullanan üretken ajanları da desteklemekte ve Python'un veri analizi ve makine öğrenmesi kütüphaneleriyle kolayca entegre olabilmektedir. Ek olarak, simülasyon çerçevesinin uygulamalarını göstermek amaçlı üç örnek vaka çalışmasına yer verilmiştir: üretken ajanlarla epidemik modelleme, etki maksimizasyonu ve ağ tabanlı güven oyunları. Son olarak, üretken ajanlar ile sosyal ağların birleştirildiği ve bina enerji modellemesi bağlamında ajanların kişilik özellikleri ve sosyal bağlantılarının etkilerinin analiz edildiği bir simülasyon tasarlanmış ve geliştirilmiştir.

Anahtar sözcükler: Sosyal ağ simülasyonu, ajan tabanlı modelleme, üretken ajanlar.

Acknowledgement

First and foremost, I would like to thank my advisor, Prof. Dr. Özgür Ulusoy, for his guidance and support during my graduate studies. His encouragement during difficult times helped me complete this thesis.

I would like to express my thanks to Dr. Tolga Yılmaz for his support throughout all stages of the framework's development, which has significantly helped with this research.

I am also thankful to the jury members, Prof. Dr. İbrahim Körpeoğlu and Prof. Dr. İsmail Sengör Altıngövde, for their time and valuable suggestions on my thesis.

I would also like to thank my friends and colleagues at our office, EA405, for their help and support throughout my graduate studies.

Last but not least, I would like to thank my mom, dad, and sisters for their support and encouragement at every stage of my life. Their belief in me has always motivated me to keep moving forward.

Contents

1	Introduction	1
2	Related Work	6
2.1	General Purpose ABMS Tools and Frameworks	8
2.1.1	Logo-style Simulators	8
2.1.2	Python-based Simulators	9
2.1.3	Larger Scale Simulators	11
2.2	Domain-specific ABMS Tools and Frameworks	13
2.2.1	Social Network Simulators	13
2.2.2	Epidemiological Simulators	16
3	Proposed Framework: Crowd	17
3.1	Architecture	17
3.1.1	Network Architecture	19
3.1.2	Data Collection, Batch Run and Model Exploration	20

3.1.3	Visualization	22
3.2	Simulation Development Approach	23
3.2.1	Step 1: Create or Load Project	24
3.2.2	Step 2: Modify Configuration	24
3.2.3	Step 3: Define Custom Methods and Run Simulation	28
3.2.4	Step 4: Analysis	29
4	Case Studies	31
4.1	Scenario 1: Generative Agents in Social Simulations	31
4.1.1	Scenario 2: Influence Maximization	37
4.1.2	Scenario 3: Trust Game on Networks	41
5	Personality-Driven Social Dynamics with LLM Agents in Shared Residential Settings	45
5.1	Methodology	46
5.1.1	Creating Families	46
5.1.2	Extracting Weather Data	48
5.1.3	Simulation Scenario	48
5.1.4	Simulation Methodology With Crowd	49
5.1.5	Agent Decisions With LLMs	51
5.2	Results	55

5.2.1	Network-level Analysis	56
5.2.2	Node-level Analysis	59
5.3	Discussion	63
6	Discussion	65
6.1	Modeling Other Social Network Problems With Crowd	65
6.2	Comparison With a General-Purpose Framework for the Three Case Studies	66
6.2.1	Modeling Stage	66
6.2.2	Execution Times	67
6.2.3	Visualization Stage	69
7	Conclusion	70

List of Figures

3.1	Architecture of the Crowd framework.	18
3.2	Simulation development and analysis steps of the Crowd framework.	23
3.3	(a) Simulation settings in the Project tab of Crowd. (b) Network initialization settings in Crowd GUI.	25
3.4	Rule addition in the Crowd GUI for the SIR example.	27
3.5	Network tab of Crowd's GUI.	29
3.6	Results tab of Crowd's GUI.	30
4.1	Execution order of a full iteration in the generative agents experi- ment.	34
4.2	Agents staying at home in the generative agents experiment. . . .	35
4.3	Agents on the fourth day of infection in the generative agents ex- periment.	35
4.4	Network visualization of the generative agents experiment: $n=100$ and $R_0 = 3$ (blue: Susceptible, orange: Infected, green: Recovered).	36

4.5	Network visualization of the IM experiment, $n = 4039$, $k = 100$, method = PageRank (blue: Active Spreader, green: Inactive, orange: Active). (a) Initial setting. (b) After 20 iterations.	39
4.6	IM: Node counts (seed node selection with PageRank)	39
4.7	IM: Comparison of total activation over time (seed node selection with PageRank, Degree, Betweenness, and Eigenvector centralities).	40
4.8	N-player trust game: Averaged number of investors (I), trustworthy trustees (T), untrustworthy trustees (U).	43
4.9	N-player trust game: Averaged global net wealth. Over 50 simulations for different temptation-to-defect ratios r_{UT} ($R_T = 6$).	43
4.10	Network visualization of the trust game experiment: $n = 1000$ and $r_{UT} = 0.4$ (blue: Investors (I), green: Trustworthy trustees (T), yellow: Untrustworthy trustees (U)). (a) Initial setting. (b) After 5000 iterations.	44
5.1	Network visualization of created families where nodes are colored (a) according to the roles (blue: Family Representative, orange: Family Member), (b) according to their family IDs.	46
5.2	The methods passed to Crowd framework and their execution order	50
5.3	Inputs provided in the query to LLM and outputs obtained in the first phase of the simulation	51
5.4	Phase 1 sample prompt	52
5.5	Phase 1 sample output	53
5.6	Inputs provided in the query to LLM and outputs obtained in the second phase of the simulation	53

5.7	Phase 2 sample prompt	54
5.8	Phase 2 Output	55
5.9	Charts generated with Crowd. (a) Average weight of edges between friends over time. (b) Number of strong friendships (weight > 3 where weight in range $[1,5]$). (c) Average happiness level over time. (d) Degree outside. (e) The temperature set in the building. (f) Cost over time.	56
5.10	Example output showing the agents choosing the higher limit of their preference category considering their needs and other people's comfort.	62

List of Tables

2.1	Comparison of other simulation tools with Crowd	7
5.1	Node parameters related to personality and associated adjectives .	47
5.2	OLS slope coefficients (with standard errors) over 15 days	57
5.3	OLS Regression results to analyze the influence of network-level variables on average happiness level	58
5.4	Correlated random effects model to estimate happiness level on 30-day node level data	60
5.5	Correlated random effects models to estimate degree choice on 30-day node-level data	61
6.1	Execution times of the generative agents case study in Mesa and Crowd frameworks.	68
6.2	Execution times of the influence maximization case study in Mesa and Crowd frameworks on Facebook, GitHub, and Twitch social network datasets.	68
6.3	Execution times of the networked trust game case study in Mesa and Crowd frameworks	69

Chapter 1

Introduction

Agent-based modeling and simulation (ABMS) serves as a valuable tool employed across diverse research fields, including analysis of adoption of electronic vehicles [1], mobile applications [2], marketing strategies [3], disease transmission between animals [4], virus spread and intervention strategies for the contagious diseases [5,6], dissemination of information and trends [7], trust management [8], transportation planning [9], and emergency evacuations [10].

ABMS employs a bottom-up approach to model real-world systems [11], allowing researchers to observe how the behaviors of individual entities (agents) influence the system through interactions with other agents and the environment [12]. By inspecting the agents' actions at the micro level, researchers can explain or predict the emergence of patterns at the system level, such as the diffusion of trends.

Agents of a simulation include a set of properties and actions defined by the modeler. Depending on the state of the agent and the environment, agents perform these specified actions in each time step of the simulation. Additionally, environments can also have properties and, in some simulation tools, their own actions (e.g., patches in NetLogo [13]). Changes in the environment's properties are directly reflected in the selection of agent actions.

ABMS has become an essential methodology in computational social sciences, particularly in studying social networks. In agent-based social network simulations, hereafter referred to simply as social network simulations, the environment is represented as a network. Examining the interaction between the agents in this structured environment provides insight into issues such as information diffusion, trust dynamics, and disease spread, which can be leveraged to prevent negative outcomes or achieve positive ones. In a network simulation, an agent typically represents a person, but may also represent an animal, an organization, or any other individual entity. A link between two nodes represents the existence of a relationship between two agents, and there may exist different kinds of relationships (e.g. family, coworker, or possibility of coordination between two organizations). Nodes interact with their neighbors, and the actions and state of these neighbors play a crucial role. For instance, while modeling the spread of information in simulations, the information is carried to an agent from its neighbors.

ABMS tools can be classified according to the platform’s intended purpose (general-purpose or domain-specific) or the motivation behind their development (prioritizing ease of use or efficiency) [12]. While there are various approaches to define a framework, all tools are expected to facilitate the creation of agents and environments (e.g., grid, continuous space, and network), define interactions among agents as well as between agents and the environment, and execute the simulation itself. Visualization of the model and the simulation data, tools for analysis, and model exploration (or parameter sweeps) [12] are also essential for understanding the model and addressing the research questions for a study.

Various existing ABMS tools also facilitate the simulation of social networks, with NetLogo [13], MASON [14], Repast [15], and Mesa [16] being the most widely used examples. While each framework meets the basic requirements to conduct a network simulation, they differ by the extensiveness of the features in each requirement. The general-purpose ABMS tools are not designed with a focus on social networks. Hence, the extent of features and the effort required to model a complex networked system increases.

An example of a commonly studied process, particularly in the fields of economics, epidemiology, and social sciences is diffusion. Despite the differences between these fields, the simulations have a common structure, while the rules, agent properties, and actions vary. By creating a framework that only requires the modeler to provide the varying components while the framework manages the rest, the time needed to implement a simulation is significantly reduced. For this purpose, the Network Diffusion Library (NDLib) [17] provides pre-built models for epidemiology and opinion analysis, enabling researchers to use them directly by only providing the network as a parameter or by implementing custom simulation through the library’s compartment structure.

In this thesis, we first introduce *Crowd*^{1,2}, a social network simulation framework that aims to simplify and fasten the process of developing agent-based models and simulations on networks. Key features of Crowd include interactive network visualizations, a configuration file approach for setting up simulations, which also allows no-code simulations for diffusion tasks and facilities for merging data and drawing charts for further analysis. While prioritizing ease of use and promoting less code writing, Crowd also allows the incorporation of custom models for more diverse use cases, due to its flexible and extensible architecture. Users of Crowd can take advantage of the Python environment by running their codes on Jupyter notebooks and accessing extensive Python libraries for tasks such as data analysis and machine learning. Although Crowd provides additional features for some use cases, such as the integration of Network Diffusion Library (NDLib)’s [17] compartment structure into configuration settings, it is designed to be a general-purpose tool that can help researchers across various fields.

Simulations in Crowd are driven by the settings defined in the configuration files, which can be modified either through the graphical user interface (GUI) or directly on the YAML file. This approach significantly reduces the amount of code needed to set up a simulation. With this approach, no-code simulations can be executed while also providing users with an interface to include any additional code in Python for data collection or the addition of use-case-specific methods.

¹<https://github.com/bilkent-sna/crowd>

²<https://github.com/bilkent-sna/crowd-ui>

After the completion of a simulation, users can explore the results either through the GUI or by loading the data files generated during the simulation. Within the GUI, users can observe the changes in the network state at different time steps through the interactive visualization, inspect node and edge properties, and modify the graph’s appearance. Moreover, Crowd allows users to merge data from multiple simulations and create charts by selecting data and chart settings from the provided options.

In the second part of the thesis, we present three case studies to show Crowd’s capabilities. We demonstrate the implementation of generative agents through an example of COVID-19 spread, and use NDLib’s compartment structures for simple diffusion simulations. Next, we implement an Independent Cascade model [50] for information diffusion and compare different seed node selection methods with Crowd’s facilities. We also present a networked trust game between investors and investees, emphasizing the use of user-defined methods.

In the final part of the thesis, we employ generative agents with varying personalities and heating preferences to simulate energy usage within a building. In this simulation, agents come together each day to decide a temperature for the central heating system. Each agent also sets a personal happiness level depending on the final decision on the previous day, and may choose to adjust the closeness levels with friends. Through a statistical analysis, we report on the impact of the average friendship weights on the network-level happiness and the effect of personality traits and temperature preferences on individual temperature choices and node-level happiness.

Our contributions can be summarized as follows:

- We present Crowd, an open-source framework that simplifies the setup and execution of agent-based network simulations through a configuration-based approach.
- Crowd supports customization with its extensible structure and allows the execution of user-defined methods written in Python providing access to

extensive data analysis and machine learning libraries.

- Crowd provides a user-friendly interface that facilitates simulation setup, interactive network visualization, chart generation, and data aggregation, enabling users to model and analyze various research problems.
- We demonstrate Crowd’s capabilities and ease of use through three case studies: the integration of generative agents into virus transmission modeling, influence maximization, and a networked trust game.
- We model energy-related decision-making in shared residential settings by integrating generative agents into social network simulations.

This thesis is structured as follows: Chapter 2 provides brief information about the previously developed tools for social network simulation and how Crowd compares to them. In Chapter 3, we explain the architecture of Crowd and the simulation development approach with this framework, detailing the design, execution, and analysis steps for both the tool and library options. In Chapter 4, we introduce three case studies to illustrate the use of Crowd, which include the use of generative models for agents’ decisions, influence maximization, and a trust game involving investors, trusted investees, and untrusted investees. In Chapter 5, we present our building energy simulation, where we explain the methodology, results of our simulations, and regression analyses. In Chapter 6, we further discuss how to represent common social network parameters with Crowd’s features and provide a detailed comparison of Crowd with a general-purpose framework to emphasize its contributions. Finally, we conclude with a summary of the framework’s key aspects and its applications, and a discussion of the future work in Chapter 7.

Chapter 2

Related Work

While creating a simulation tool, the researchers' first decision is to determine the scope of the framework. Although the general-purpose ABMS tools can be used for a wide range of simulations, the time and effort required to implement a system that can accurately model the real world for complex tasks significantly increases. As a consequence, many researchers develop domain-specific tools tailored to the common fields and topics where ABMS is applied. In this chapter, we first present the general-purpose ABMS tools, then move on to domain-specific tools in the fields of social networks and epidemiology, and explain where Crowd is positioned in this mapping.

To better illustrate this, we present a comparison in Table 2.1. This comparison is inspired by the classifications made by Antelmi et al. [12] and Abar et al. [18]. We evaluate the simulators based on several criteria: the programming language used for model development, the scope of ABMS, the available simulation environments, the type of graphical user interface (GUI), visualization capabilities, and functionalities for data collection, batch running, and model exploration. Additionally, we list the effort required for model development and the scalability of each tool, as an extension of Abar et al. [18]'s classification.

Table 2.1: Comparison of other simulation tools with Crowd

Category	Tool/ Property	PL for Model Dev.	ABMS Scope	Sim. Env.	GUI	Vis.	Data Coll.	Batch Run	Model Exp.	Model Dev. Effort	Model's Scalability Level
General- Purpose	NetLogo	NetLogo, Python*, R*	General purpose	Grid, network, continuous, GIS*	✓ ^{1,3}	2D, 3D*	✓	✓*	✓*	Simple ^b	Medium- large ^b
	Mason	Java	General purpose	Grid, network, continuous, GIS*	✓ ¹	2D, 3D	✓	✓ ^a	✓*	Complex ^b	Medium- large ^b
	Repast Simphony	Java, Groovy, ReLogo	General purpose	Grid, network, continuous, GIS	✓ ¹	2D, 3D	✓	✓	✓	Complex ^b	Large ^b
	Agents.jl	Julia	General purpose	Grid, network, continuous, GIS	✗	2D, 3D	✓	✓ ^a	✓	Moderate	Medium- large
	Mesa	Python	General purpose	Grid, network, continuous, GIS*	✓ ²	2D, 3D*	✓	✓	✓	Moderate	Small- medium ^b
Network- Focused	Covasim	Python	Covid-19	Network	✓ ³	2D	✓	✓	✓	Simple	Large
	HashKat	YAML, C++	Twitter- like networks	Network	✗	DAT & GEXF files	✓	✓	✗	Simple	Large
	Soil	Python	Social networks	Network	✓ ²	2D	✓	✓	✓	Moderate	Small- medium
	Crowd	Python, YAML	Social networks	Network	✓ ⁴	2D	✓	✓	✓	Simple	Small- medium

Features marked with an asterisk (*) are available through extensions. *a* and *b* refer to data from surveys [12] and [18], respectively.

1, 2, 3, 4 denote the GUI type: (1) desktop application, (2) local web application, (3) hosted web application, and (4) desktop application built with web frameworks.

2.1 General Purpose ABMS Tools and Frameworks

Even within the general-purpose tools, there exists a categorization with respect to the purpose of the tool. While some tools prioritize ease of use and simplicity, others may focus on extensibility, performance, and scalability. In this section, we present the general-purpose tools in three categories, in increasing order of difficulty for model development. We first present tools that adopt the Logo approach, continue with Python-based frameworks emphasizing ease of use and modularity, and finally, the performance-oriented tools for larger-scale simulations, including the high-performance computing (HPC) focused frameworks for simulations with millions of agents.

2.1.1 Logo-style Simulators

NetLogo [13] is an agent-based simulator widely used for education and research purposes. It introduces its own programming language, based on Logo, which aims to simplify the coding process for users with a non-programmer background. It includes different types of agents: turtles (agents that can move in the environment, e.g. sheep), patches (static agents, e.g. grass), links (which can be used to connect turtles for network-like environments), and the observer (used to write commands in the command center from an overseer point of view). The NetLogo language can be used in the tool to implement the user’s own models or to modify a sample model chosen from NetLogo’s extensive library, which is directly reachable from the desktop app itself.

Besides the code editor, NetLogo includes an interface tab, which is used to visualize the model, play with the parameters with the sliders that can be added on demand, as well as line, bar, and scatter plots that show the changes that happen in the simulation. Another tool, called BehaviorSpace, is integrated into NetLogo to run multiple simulations with different parameters. Moreover, NetLogo has a

web-based interface that allows simple simulations.

With the recent improvement of the large language models (LLMs), the researchers of NetLogo have introduced another study, ChatLogo [19], which aims to help new programmers with both starting up their model and fixing the bugs. Instead of directly giving an implementation of the user’s problem, it goes over the steps with the users, with the purpose of teaching.

Besides NetLogo, the ReLogo plugin within Repast Symphony [15] allows accessing the library’s functionality within a simpler programming language for a quicker and simpler start to simulation development. ReLogo’s advantage over other Logo-style programming facilities is that it can be mixed with Java and Groovy at any time, therefore providing access to extensive Java libraries.

Another Logo-style tool is StarLogo [20], which is the predecessor of NetLogo and serves more of an educational purpose.

2.1.2 Python-based Simulators

While Logo-style simulators allow a lower entry barrier to agent-based simulations, this approach also requires the users to learn numerous terms for modeling structure. It is not as practical as just using a general-purpose language, such as Python, which is widely used in many disciplines and includes various data analysis libraries in its ecosystem.

Mesa [16], the first Python ABMS framework, provides a modular architecture to support a variety of scenarios. It includes Model and Agent classes, schedulers that choose which agent to activate, and extensible environment definitions. As a consequence of the modular approach, it is considerably easier to define new spaces than it was with previous alternatives.

Furthermore, Mesa provides data collectors that will save the user-requested model-level and agent-level parameters. To utilize this feature, users provide

the model and agent-level variables and the function to collect them to the data collector. The collected values are saved in a dictionary, which can be converted to a Pandas DataFrame for further analysis or saved in CSV format.

In addition, Mesa’s batch runners enable multiple runs of simulations with the same and varying parameters for model exploration. Visualization in Mesa is conducted with a web-based UI and is created by users by importing the various visualization elements provided for the different types of environments in the library. Within this UI, users can include ”start,” ”stop,” and ”step” controls similar to those in NetLogo, and the addition of more controls can be achieved by additional coding.

Another Python framework is AgentPy [21], which emphasizes its ability to provide interactive computing within a single programming environment, Jupyter notebooks. This simulation package also includes batch running and parameter exploration, as well as a visualization similar to NetLogo, and parallel computing features. The syntax utilized in AgentPy further simplifies modeling and parameter exploration, requiring fewer lines of code compared to Mesa. Data collection is handled by the *record* method, which only requires the name of the parameter that will be used for analysis. Chart drawing can be done by utilizing external Python libraries such as seaborn.

NetLogo also allows the integration of Python code in its source code through its Python extension. This enables NetLogo to take advantage of data analysis and machine learning libraries specific to Python. Python code is passed as a string to the call function, and the results returned from Python can be utilized for visualization of the model and generation of charts. While this approach eliminates a limitation of NetLogo, regarding access to popular Python libraries, it is impractical compared to using a Python-based simulator like Mesa or AgentPy. This development environment does not provide any extensive support for Python in terms of code completion, debugging, and other features included in IDEs like VSCode, or interactive code execution that comes with Jupyter notebooks.

2.1.3 Larger Scale Simulators

As the number of agents in a simulation increases to hundreds of thousands or millions, execution times significantly increase. To address this issue, various tools provide performance optimizations with multithreading, distributed computing, or tensor-based approaches.

MASON [14] is a Java library designed to fasten the simulations for up to a million agents while excluding any platform dependencies so that the simulations can execute on various machines. It has a core package of basic simulation needs and provides features for more domain-specific features in external packages. One of its key features is that it allows checkpointing of the model to the file system so that the simulation can be stopped and run from that checkpoint at any time. This feature also allows switching machines.

MASON follows a three-layer approach, which is also adopted later by future simulators: utility, model, and visualization layers. 2D and 3D visualizers are provided in the graphical user interface (GUI), as well as object inspectors to watch and modify model parameters. Some notable extensions of MASON include GeoMASON [22] for GIS features, social network libraries, and D-MASON [23] for distributed execution of models with various machines.

Repast is a "family of agent-based modeling software" with multiple programming language support with Repast Symphony (Java) [15], Repast HPC (C++) [24] and Repast4Py (Python) [25]. Repast Symphony emphasizes a plug-in architecture for modularity, allowing the replacement of modules without affecting the other parts of the framework. UI tools, model specification, simulation execution, GIS, data saver, batch runner, charts, visualization, and distributed execution sets are all given as separate plugins. As stated in Section 2.1.1, Repast Symphony also includes a ReLogo plugin that simplifies the modeling stage.

To trigger a function depending on the model state, a watcher activation approach is introduced, similar to listeners in Java. Simulation information, such as agent properties, can be logged at runtime, and this data can be aggregated

with operations such as summation. Some of the features that are included in its visualization components include the creation of geometric or custom shapes, importing images to style the scene, and displaying GIS data. Similar to MASON, model checkpointing is supported (delimited file and XML formats).

Repast HPC is the C++ version of Repast Symphony, providing parallel distributed execution of simulations designed to be run on high-performance computers. Despite also being a distributed ABMS tool, Repast4Py aims to lower the entry barrier for modelers to try distributed simulators by simplifying the process. To increase performance, this framework utilizes Python C-API, NumPy, Numba, and PyTorch.

In Repast4Py, a PyTorch tensor is used to keep a shared value grid. Utilizing tensors is also adopted by another study that emphasizes scalability and efficiency, DeepABM [26]. It does not use an object-oriented structure that is adopted by other frameworks previously mentioned. In DeepABM, each agent and their states are stored in tensors and the interactions between them are modeled using message-passing operations in graph neural networks (GNNs). This approach provides increased scalability and GPU execution, consequently reducing the execution time for large-scale simulations. Moreover, DeepABM facilitates further optimized model exploration capabilities.

Recent additions to performance-oriented simulators include Agents.jl [27] and krABMaga [28]. Agents.jl is a tool that takes advantage of the Julia environment in terms of its performance (parallelization) and ease of defining complex mathematical expressions, such as differential equations, which might be the key point of some models. The creators of Agents.jl emphasize simplicity, ease of use, and fewer lines of code in addition to adding new features to the agent-based modeling space, such as N-dimensional space support and agent sampling based on certain properties. This framework does not include a GUI as a design consequence to allow the execution of full features on various settings, such as console, Jupyter notebooks, and remote servers, or as a part of a larger system.

krABMaga is a Rust framework that adopts MASON’s architecture design with

some modifications, while taking advantage of Rust’s efficiency and reliability. While the core simulation facilities, model explorations in parallel, distributed and cloud architectures, and some visualization features are available, others, such as checkpointing and GIS data support, are under development.

2.2 Domain-specific ABMS Tools and Frameworks

Previous simulators have all adopted an agent-based approach with various types of environments. While this allows them to be more general, it does not further provide functionalities commonly used for network simulations in social or natural sciences. In this section, we describe the needs of a modeler in the fields of social network and epidemiology research, how the previously developed simulators addressed them, and a comparison with Crowd.

2.2.1 Social Network Simulators

Social network platforms such as Facebook, X (Twitter), and Reddit enable the quick dissemination of information without the location barrier. Consequently, these platforms become a place to spread misinformation or trends for business purposes, which has been of interest to many researchers. Modeling these environments as graphs—with nodes representing individuals and edges representing their relationships—allows researchers to study the dynamics of information flow. The inspection of the effects of certain people on the general public aligns closely with the methodology of agent-based modeling simulations, where the influence of the individual entities on the system can be analyzed.

Online social network (OSN) simulators such as Krowdix [29] and HashKat [30] are specifically designed for this purpose. To model the user’s behaviors in the

social network platforms, Krowdix includes predefined agent actions such as creating a forum, posting content, and sending friendship requests and messages. With a duration-based scheduling approach, these agent actions may take more than a single step to get completed. This can enable different methodologies for analysis. In addition, Krowdix provides alternative instance state storage, which allows researchers to explore different scenarios without changing the configuration and rerunning the simulation.

Hashkat (#k@) is a C++ simulation tool that allows a detailed analysis of Twitter-like large-scale networks with its built-in functionalities. These functionalities include generating networks with agent properties such as the number of tweets and retweets, region, language, and ideology. Relations between the users, information flow, and trending topics can be modeled to test new features of OSNs or to develop business plans. HashKat requires users to create a configuration file to define the simulation properties, such as time limitation, follow models, and output settings (e.g., visualization, agent stats, tweet analysis). An online configuration generator is provided to simplify the process and support modelers who are not familiar with the YAML file structure. Despite providing extensive support for realistic social network simulations, it requires more effort to extend the simulator’s capabilities compared to modular simulators. The configuration aspect is similar to our approach; however, our tool is designed to be more general-purpose and extensible.

The Open Source Social Network Simulator proposed by Ye et al. [31] is a toolkit that focuses on spatiotemporal aspects of information diffusion and is implemented in Python. Similar to Crowd, this tool provides simple network generation and algorithms to select seed nodes for information propagation and an easy-to-navigate user interface. Two diffusion models, independent cascade (IC) and linear threshold (LT), are integrated into this tool, along with commonly used social network analysis (SNA) metrics and community detection algorithms. For spatial analysis, the tool provides the option to add a map to the visualization of the network, which benefits the analysis of information sources and propagation across regions. Moreover, composite networks can be generated by selecting the

connecting nodes with a specified rule, such as centrality metrics. The text analysis tool processes input files to return information about the usage of selected keywords or plot graphs of word frequency or shifts over time. Contour maps can also be generated to compare simulations with real-world data.

While Crowd also simplifies the model implementation by allowing no-code diffusion simulations, it still requires the user to set the rules. To calculate SNA metrics and run community detection algorithms, modelers can call the NetworkX functions from the Method Lab of Crowd. Even though this approach requires more effort, it also gives the user the flexibility to define other diffusion models easier, while also not limiting the methods to be used for metric calculations and community detection algorithms. These additions can be made in the Open Source Social Network Simulator as well, with further changes to the source code.

Soil [32] is an extensible Python-based simulator, designed to work with Mesa in the later versions to allow its users to take advantage of features provided by both libraries. Besides fundamental agent-based simulation facilities, Soil defines network, network agent, and event classes that can be extended by modelers for social network tasks. The tool provides functions to access NetworkX’s graph generators to initialize the networks with the selected parameters, as well as a web-based visualization of the network, settings, and charts. Soil utilizes Python to describe the simulation, SQLite database to store the results, and YAML files for simulation and visualization configurations.

With its Python environment, configuration file, network generation, and basic visualization approaches, Soil is the most similar study to Crowd. However, these frameworks differ in the simulation definition and execution methodologies, visualization capabilities, and ease of use. Crowd integrates compartment structure into its configuration style, a default data collection system for graphs and user-implemented methods, and a Tauri-based cross-platform GUI for project navigation, simulation definition, and execution, all through the user interface. All of these features lower the entry barrier for agent-based network simulations while requiring less time for coding at each step.

2.2.2 Epidemiological Simulators

Social networks are also utilized in epidemiological studies, such as the analysis of the spread of diseases like COVID-19, where modeling the dissemination of the virus, understanding the impact of different intervention strategies, and estimating resource requirements are typically the main objectives. The intervention strategies typically employed include vaccination, social distancing, closing schools and workplaces by certain percentages, and running regular tests to detect infected people. To significantly reduce the time required to model the effects of these factors in a simulation environment, tools such as Covasim [33] and Flu and Coronavirus Simulator (FACS) [34] have been developed.

In Covasim, a realistic simulation can be executed by only giving the data for the specific regions, as default parameters are set according to the values directly taken by recent COVID-19 studies. Besides the default intervention strategies mentioned previously, custom strategies can also be added. To represent a society, a multi-layered network structure is adopted, consisting of households, schools, workplaces, and community networks. While having contact with members of their household, an individual can also interact with agents from other networks they belong to. Furthermore, Covasim allows the use of SynthPops [35], an open-source contact network generator. Other key features of Covasim include performance optimizations that allow large simulations to be done in under a minute, parameter sweeps by simply giving the options as arguments, and a web application that can be used by entering parameters into default models.

FACS, also implemented in Python, focuses on local analyses with the geospatial data of buildings in a region. It can be utilized for local decision-making about determining the areas where spread occurs the most, hospital loads, and prevention methods such as lockdowns. The disease transmission is modeled with various parameters such as daily activities and mobility. Besides local machines, simulations can also be executed in remote supercomputers, increasing the scalability and efficiency of the tool. In addition to the features that model people's activities in a city, FACS provides visualizations with charts and maps.

Chapter 3

Proposed Framework: Crowd

In this study, we build upon the initial concept and methodology proposed for Crowd in [36]. We extend the available network types and visualization, further simplify simulation definitions, and introduce features such as a GUI, batch runner, model exploration, and no-code diffusion simulations.

In this chapter, we first explain the framework’s architecture, including the network environment types, and data collection and analysis facilities. We then describe the steps to design and run a simulation with an example of a Susceptible-Infected-Recovered (SIR) model [37].

3.1 Architecture

Crowd is built as a Python library consisting of modules for essential features for social network simulations. These extensible modules manage the network structure, execute simulations, save the collected data, and visualize the results. In addition, we developed a desktop application leveraging JavaScript’s visualization libraries, which interacts with the library through the API module. This application provides an extra layer of simplicity in the setup, execution, and visualization stages of simulations. The overall architecture of the framework,

showing the integration of the modules with the file system and the desktop app, is given in Fig. 3.1.

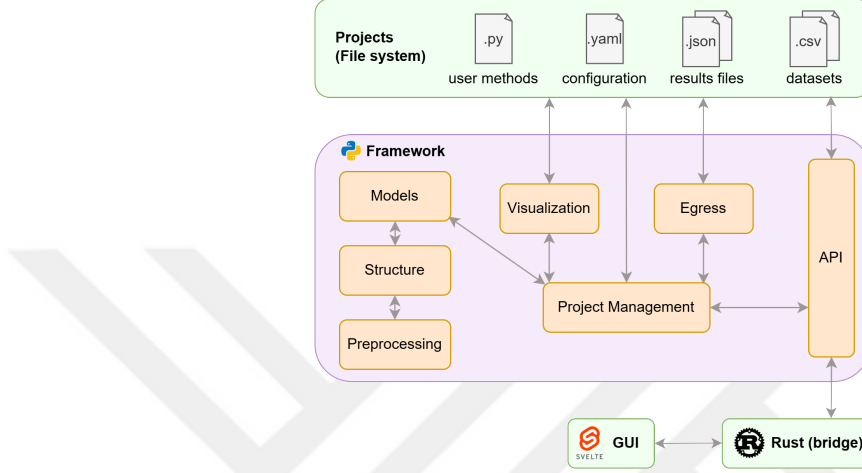


Figure 3.1: Architecture of the Crowd framework.

Through this desktop app, users can view their projects, navigate to previous simulations within a project, configure the settings with selectors, and execute a simulation. Complex simulation logic and additional data collection can be implemented by writing Python functions within the built-in editor and selecting their execution period in the simulation life cycle. After a simulation is executed, users can analyze the results with network visualizations, inspect node and edge data, merge results from multiple simulations, and generate charts in a simple and user-friendly fashion.

To describe network settings, node types, node and edge parameters, Crowd adopts a YAML-based configuration approach and eliminates the need for researchers to code for this step. For diffusion simulations, compartments and rules defining the simulation logic are also placed in the YAML files. This approach enables no-code simulations for simpler scenarios and reduces the coding effort required for more advanced models. In scenarios where this configuration based setup does not provide all the functionalities needed to initialize a specific simulation, in the library version of Crowd, modelers can add the required code before calling the library function that runs the simulation. For instance, in this part of the code, LLMs to be utilized in each step of the simulation can be initialized prior to the execution of iterations.

To increase ease of installation and make the tool more approachable for non-technical users, we utilize Tauri to provide a cross-platform application. The Rust layer acts as a bridge, relaying function calls from the GUI (Svelte) to the backend (Python). The functions that can be called with this structure are placed in another module called Crowd API, and these functions facilitate the execution of core simulation tasks and management of file operations, such as uploading datasets and accessing project data and results stored as JSON and YAML files in the file system.

3.1.1 Network Architecture

In the other ABMS frameworks, such as Mesa and MASON, two classes are expected to be implemented by the user, Model and Agent. Crowd adopts a different approach. In the current version, the only environment that the simulation can reside on is a network. This network environment consists of nodes and edges, where nodes are the agents and edges define the relations between the agents. Agents' attributes are stored as node parameters in the network, and agent-level methods can be defined separately to pass to the library method that runs the simulation.

Crowd provides two base classes, `Network` and `CustomSimNetwork`, as well as two extensions of them: `DiffusionNetwork` and `EdgeSimNetwork`. The internal representation of networks in our framework utilizes NetworkX [38], which is a library that provides data structures to represent graphs, network generators, and additional social network analysis algorithms. To extend the available network generators integrated into Crowd, we also use the `igraph-python` library [39].

The base `Network` class of Crowd is designed to provide a general framework for all possible user-defined scenarios. The *network-as-code* approach [36] is adopted for the design of this network type, where the network, agents, their interactions, as well as the details of the analysis and visualization, are defined using configuration files and code. It is useful for researchers who want to implement all low-level details of the simulation. We extend this network type for a use case

of edge-based simulation where we add links to the network in every iteration of the simulation, depending on the update method provided by the user for the selection. This class, which we call **EdgeSimNetwork**, illustrates the extensible nature of Crowd and how data savers and visualizers can be utilized in custom networks created by the modelers.

The third network type called **CustomSimNetwork** provides notably more features compared to the previously explained types. **CustomSimNetwork** manages the addition of network, node, and edge parameters, and stores and executes user-defined methods at the selected periods. *Before-iteration* methods allow setting or resetting some parameters before the main logic of the iteration. For instance, if the agents have to make a decision on staying at home or going out based on the number of COVID-19 cases in the town and updating their location parameter, this process should be done before the beginning of the daily interactions (case study 1). Between *before-iteration* and *after-iteration* methods, functions labeled as *every-iteration-agent* are executed, where we call the given methods for each agent in random order. *After-iteration* methods can be used to collect iteration-specific data, such as the number of people who decided to stay home that day. In addition, *after-simulation* methods can be utilized to handle the remaining operations for the simulation logic, allow the analysis of the final state of the simulation model through extra calculations, and save additional information for later inspection.

DiffusionNetwork network type, which extends **CustomSimNetwork**, enables simplified diffusion simulations by integrating NDLib’s compartment structure. With NDLib managing the simulation logic, the user is only responsible for writing the data collection functions.

3.1.2 Data Collection, Batch Run and Model Exploration

Simulation data collection is essential for visualization and temporal analysis. While all network types in Crowd support exporting the current state of the NetworkX graph, the level of automated data collection varies by network type.

When running a model based on the `CustomSimNetwork`, Crowd automatically logs the counts of each node type and their changes over time, along with outputs from any user-defined functions. If a user-defined function returns a value, it is saved to a JSON file named after that function. This data can later be analyzed through chart generation on the GUI or by loading it into a Python structure such as a dictionary or a pandas DataFrame.

Both the tool and library versions of Crowd allow users to import and use any installed libraries existing in the Python ecosystem. This includes NumPy, pandas, scikit-graph, and machine learning frameworks and libraries, such as PyTorch, TensorFlow, Keras, or Scikit-learn. This highlights another advantage of choosing Python-based simulators, compared to Java-based alternatives. Moreover, the modelers can utilize NetworkX’s comprehensive features for social network analysis, such as the algorithms for community detection, as well as network and node level metrics.

An important parameter to set is the snapshot period, which denotes the frequency at which the state of the simulation will be written to files. Setting the snapshot period to 1 results in saving the graph and other data in each iteration, which introduces an overhead for larger graphs or long simulations, consequently extending the simulation time. On the other hand, not saving the data results in data loss for analysis. Hence, it is crucial to find a balance in this trade-off.

As simulations may include some random processes, it is a desired requirement for the ABMS tools to have batch run options [12]. With batch runners, the simulations are executed multiple times with the same parameters. To aggregate the results, the modelers generally take the average of the collected data. For this purpose, we take the batch size as a parameter to a simulation. Currently, the multiple runs in Crowd run one after another, which leaves parallelization or optimization tasks as future work.

Model exploration, or parameter sweep, provides batch run options with varying parameter values. In Crowd, the list of values to explore can be given in the

related section of the configuration file. Crowd generates combinations of simulation settings where only one value is tested at a time, and executes all these configurations. The resulting data can then be used to identify optimal parameter values or inspect the impact of an independent variable on the dependent variables.

To aggregate the results, we provide two methods: merge within the parent simulation or merge with another simulation. The first option allows aggregating the results of the batches by a user-selected method, such as getting the mean or sum of the results of the user-implemented data collection methods. In the second option, data across different simulations can be merged, with the restriction of having the same file name and content format. The merge operations are conducted after the simulations conclude, either through the library or the app.

3.1.3 Visualization

The library version includes a basic visualizer, which saves the network states as PNG and GIF, while further types of visualizations can be obtained through the tool. First of all, the network information recorded in every snapshot period is shown as an interactive visualization utilizing the D3.js library. It can also be downloaded in various formats, such as PNG, SVG, JPEG, and GIF. Users can choose which iteration will be shown on the screen with buttons or a slider.

In addition, the Results page allows users to draw custom line, bar, scatter, and area charts with the data they choose. The source of this data is the data collection functions called in each iteration, as well as the default node type count files provided by Crowd. We pass the selected data to the ApexCharts Svelte library to create the charts and display the resulting interactive chart, which can also be downloaded in various formats. The graphs and charts provided by Crowd reduce the time needed to convert the results into visual form for researchers to put in research papers or reports.

3.2 Simulation Development Approach

In this section, we detail the steps to design, run, and analyze a simulation with both the library and tool versions of Crowd, while also providing a summary in Figure 3.2. We chose the widely used Susceptible-Infected-Recovered (SIR) model for a basic diffusion simulation that can demonstrate the distinct features of Crowd. The mathematical formulation and details of the SIR model are provided in [37].

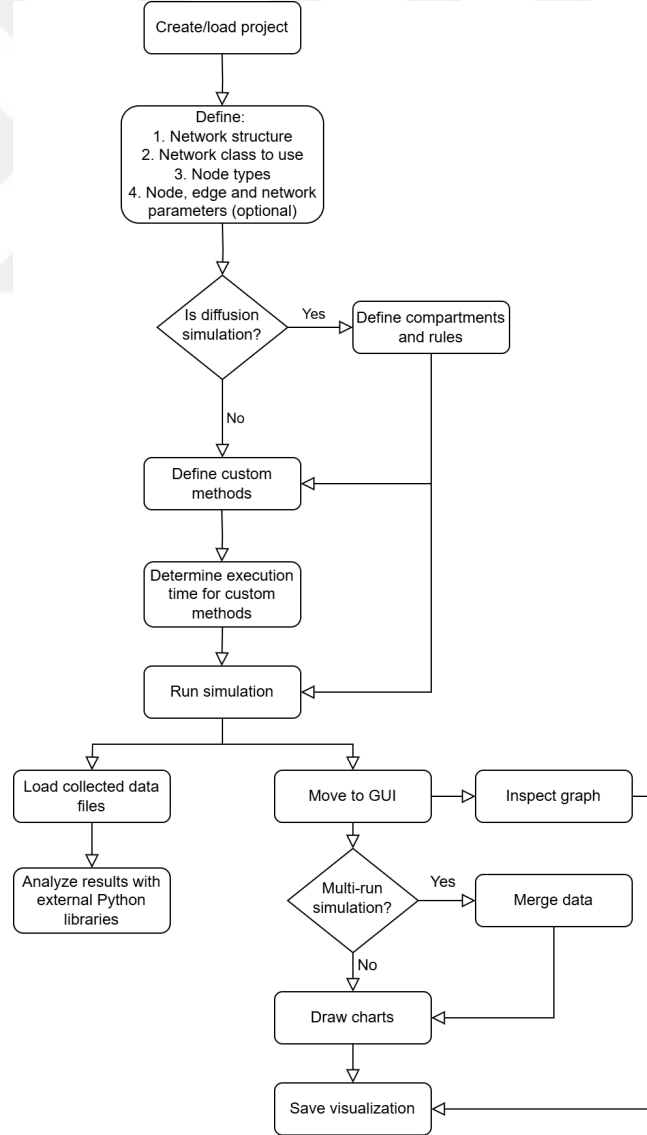


Figure 3.2: Simulation development and analysis steps of the Crowd framework.

3.2.1 Step 1: Create or Load Project

In Crowd, simulation settings, datasets and results are encapsulated in a Project structure. To begin a simulation process, a project should be either created or loaded using the library's functions as given in Listing 3.1 or through the GUI. Projects in Crowd are categorized into two as node-based or edge-based simulations. This classification is used for the network and visualization type selections.

Listing 3.1: Creating a project in Crowd

```
1 from crowd.project_management.project import Project
2
3 # 1. Project creation
4 project_name = "simplifiediffusion"
5 creation_date = "19/10/2024"
6 info = "Diffusion of a virus on a random network"
7
8 my_project = Project()
9 my_project.create_project(project_name, creation_date, info, "node")
```

3.2.2 Step 2: Modify Configuration

Defining all simulation settings with YAML configurations allows simple simulation setups with Crowd. The structure component describes how the network will be initialized, which can be from a file or randomly generated with various available methods. In Listing 3.2, we select our network to be a random regular network with 100 nodes, having a constant degree of 4.

Listing 3.2: Defining network structure in configuration file

```
1 name: SIR-example
2 structure:
3   random:
4     count: 100
5     degree: 4
6     type: random-regular
```

Alternatively, modelers can also use Crowd’s GUI for simulation configuration with buttons and dropdowns, and the selections are automatically converted into YAML. Fig. 3.3a shows the list of all configuration sections available in the Project tab, while Fig.3.3b illustrates the network source definition, which is the equivalent of Listing 3.2.

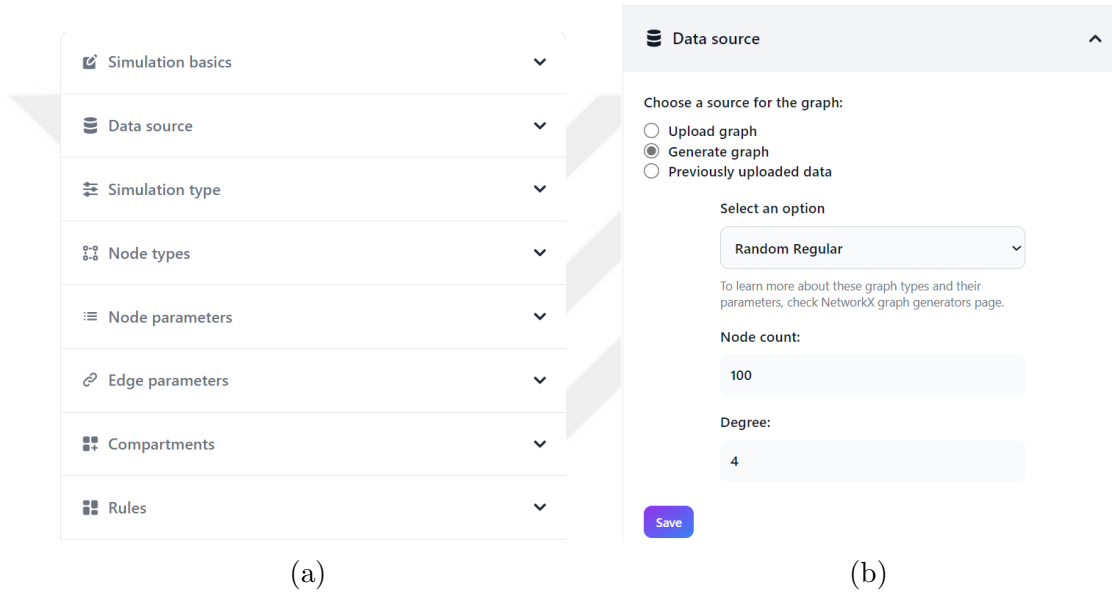


Figure 3.3: (a) Simulation settings in the Project tab of Crowd. (b) Network initialization settings in Crowd GUI.

Network initialization properties are defined in the *definitions* component. Compartments and rules can be included in this section when executing diffusion simulations. Selection between available network models is made through the inclusion of keywords, such as “pd-model” which denotes using a predefined model. In Listing 3.3, the predefined model for this example is chosen as `DiffusionNetwork` by the “diffusion” keyword provided in the next line. Following the network model selection, node types and their initialization methods are listed. Crowd provides four options for the initialization, which are *random with weight*, *random with count*, chosen with a metric, or from the file. Available metric options for this purpose include centrality metrics commonly used for social network analysis, such as degree centrality and Page Rank.

The agent and model parameters in Crowd are stored in the network objects,

and are renamed to node, edge and network parameters for the context of network-based simulations. Initialization of these parameters in the configuration file is optional, and can be achieved by giving a numerical range or categorical options for random selection.

Listing 3.3: Definitions for SIR example with Diffusion Network

```

1 definitions:
2   pd-model:
3     name: diffusion
4     nodetypes:
5       Susceptible:
6         random-with-weight:
7           initial-weight: 0.9
8       Infected:
9         random-with-weight:
10          initial-weight: 0.1
11       Recovered:
12         random-with-weight:
13           initial-weight: 0
14     node-parameters:
15       numerical:
16         age:
17           - 0
18           - 100
19     compartments:
20       c1:
21         ratio: 0.1
22         triggering_status: Infected
23         type: node-stochastic
24       c2:
25         type: count-down
26         name: healing
27         iteration-count: 4
28     rules:
29       r1:
30         - Susceptible
31         - Infected
32         - c1
33       r2:
34         - Infected
35         - Recovered
36         - c2

```

In diffusion simulations, compartments and rules describing the simulation logic are also included in the *definitions* component. Compartments are rule-bits

that we specify the conditions of a node state change. NDLib facilitates compartments of several types that can be utilized for various simulations with different combinations. In cases where these compartments are not enough to describe a problem, modelers can switch to custom simulation networks and include their own methods for the task.



Figure 3.4: Rule addition in the Crowd GUI for the SIR example.

The first rule that uses compartment *c1* included in Listing 3.3 runs as follows: If a node is susceptible, execute *c1*. If this node has any Infected neighbors (triggering status), with a probability of 0.1 (ratio), the node switches from Susceptible to Infected state. Rule 2 describes the Infected to Recovered sequence with a countdown compartment to define that after 4 iterations, the node will switch states. Additionally, the process of constructing these rules in the GUI is illustrated in Fig. 3.4.

Listing 3.4: Defining model exploration in configuration file

```

1 model-exploration:
2   count:
3     path: structure.random.count
4     values:
5       - 100
6       - 200
7       - 300

```

To analyze the effect of specific variables in a simulation, users can provide a list of values in the *model-exploration* section, along with the path to those variables in the configuration file. In the example shown in Listing 3.4, the number of nodes in the network is explored, with the path to the variable *count* referencing its place in the definitions given in Listing 3.2. This configuration can also be set up through the GUI by entering multiple values in the input section instead of a single value. For instance, when configuring network settings as shown in Figure 3.3b, a list of integers representing node counts can be specified to explore different graph sizes.

3.2.3 Step 3: Define Custom Methods and Run Simulation

For simulations that require additional simulation-specific logic or collection of data, users can define custom methods in the Method Lab section of the GUI. These functions can call any Python library installed on the user’s system. The user interface supports this process with a code editor and allows the user to select the execution phase for each function. Listing 3.5 demonstrates the equivalent implementation using the library version of Crowd. All custom methods have access to the network object and must include it as a parameter. Detailed information for the rules that user-defined methods should follow is provided in Crowd’s documentation¹.

Listing 3.5: Define custom methods and run simulation

```
1 # Returns the % of infected nodes in every snapshot
2 def get_percentage_infected(network):
3     return (network.node_count[1] /
4             network.G.number_of_nodes()) * 100
5
6 # Run the simulation
7 my_project.lib_run_simulation(epochs=50,
8                               snapshot_period=5,
9                               curr_batch=1,
10                              after_iteration_methods=[
11                                  get_percentage_infected
```

¹<https://crowd.readthedocs.io>

The custom method we implement in 3.5 should collect data following the completion of the diffusion process in each iteration; therefore, it is passed to Crowd’s run method as an *after iteration* method. The calculated percentage returned by this function is automatically saved in each iteration. We run the simulation for 50 iterations (*epochs*), save the graph and data collectors every 5 iterations (*snapshot period*), for one time (*curr_batch* number).

3.2.4 Step 4: Analysis

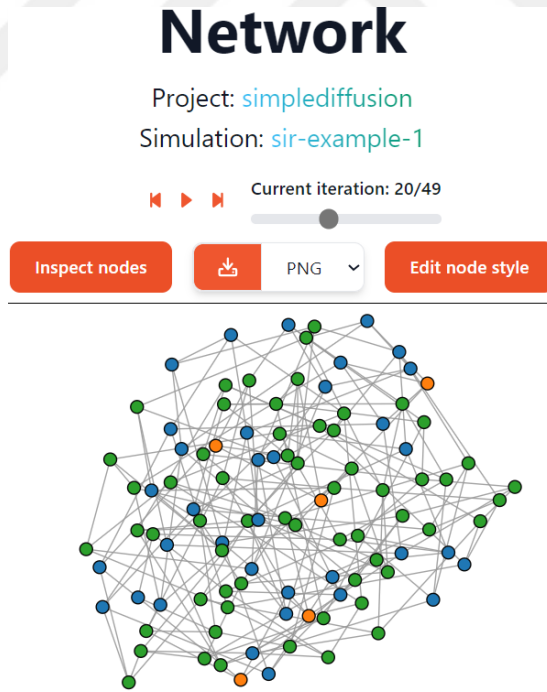


Figure 3.5: Network tab of Crowd’s GUI.

Upon completing the simulation, users are navigated to the Network tab in the GUI to examine the evolution of the graph at each snapshot period, as illustrated in Fig. 3.5. This interface supports interactions with the network visualization through moving the nodes, zooming, and adjusting the node visualization parameters. For more advanced network analysis and visualization, Crowd allows

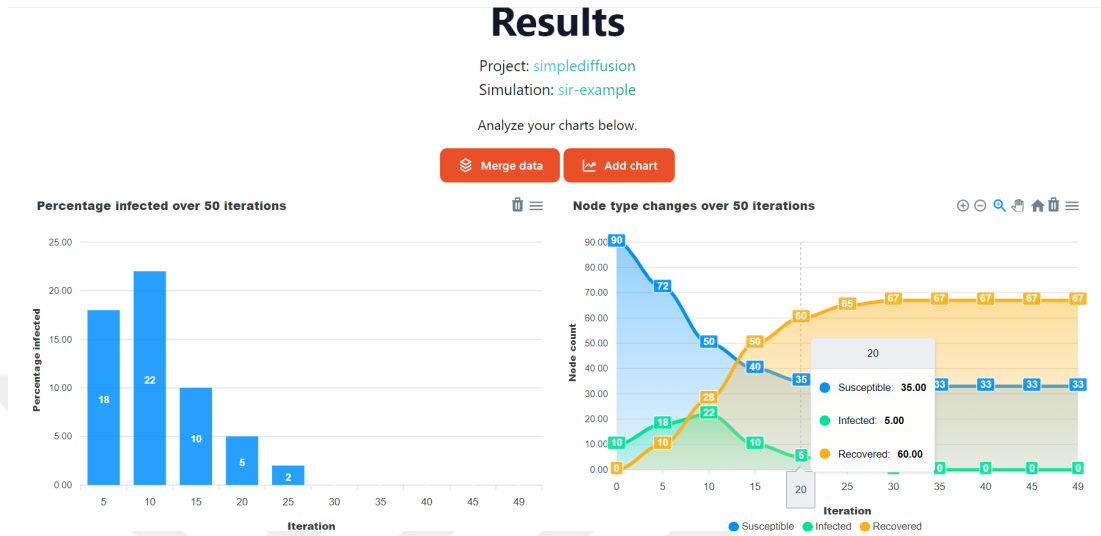


Figure 3.6: Results tab of Crowd’s GUI.

exporting the network state in the specified iterations to the common graph representation formats, such as GEXF, for tools like Gephi.

As described in Section 3.1.3, users can perform data aggregation and visualization by generating interactive charts within the Results tab (Fig. 3.6). The bar chart displays the information collected with our user-defined function, while the area chart utilizes node type counts automatically saved during each step of the simulation. In addition to the visual analysis provided by the GUI, researchers can use the simulation output files saved in JSON format for additional data analysis.

Chapter 4

Case Studies

In this chapter, we intend to illustrate how Crowd simplifies and streamlines the implementation of existing research across diverse and complex domains by presenting practical examples through various use cases. The first case study models a COVID-19 epidemic scenario using generative agents, based on the study by William et al. [40] within a network environment. Our second use case focuses on the influence maximization problem using a real social network from Facebook [41], which is widely utilized in social network research. The final scenario describes a trust game in which investor and trustworthy/untrustworthy trustee agents adapt their strategies each day based on neighbor payoffs, an implementation of the work by Chica et al. [42].

4.1 Scenario 1: Generative Agents in Social Simulations

In social simulations, agents commonly represent people in real life. However, these agents do not possess the common knowledge humans have, and their

decision-making processes are restricted by the information provided by the modeler. Recently, researchers have addressed this challenge by integrating large language models (LLMs) into agent-based models, utilizing their capabilities such as world knowledge and common sense reasoning for agents’ decisions [43]. While the LLM agents are utilized in social simulation to model virtual towns and communities [44] and to develop simulators [45], this approach has also been adopted in the fields of economy, natural sciences, and engineering [43]. A study in the field of epidemiology, GABM-Epidemic [40], shows that when given information about the daily cases, agent’s characteristics, and infection symptoms, LLM agents behave similarly to real people. This section provides an implementation of this study in Crowd to demonstrate its utility as a Python library, seamless integration with libraries such as Hugging Face Transformers for LLM inference, and compatibility with interactive environments such as Jupyter notebooks.

In this scenario, at the start of each simulation day, the agents choose between going out and staying home. This approach differentiates this study from a standard SIR model. Agents who choose to go out become a part of the epidemic network, allowing them to interact with the other agents in the environment. Contrarily, agents that stay home are excluded from the interactions and cannot infect the other agents or get infected. This decision is made by sending a prompt to the LLM, which is constructed according to the agent’s age, personality traits, and memories. Memories of an agent consist of their symptoms, information about the virus spread, and their reason to go out, which is defined as work for all agents. The prompt is concluded by asking the model if the agent should stay home. The output provided by the LLM includes a short answer, “yes” or “no,” and a reasoning of one to two sentences. In a rule-based simulation, agents’ personalities do not impact their decision to go out; however, with this approach, a person described as reckless may decide to go out regardless of the high number of daily new cases.

In the original study, agents reside on a grid environment without a network structure, and the agents to interact with are chosen randomly for each agent, up to five contacts. As Crowd uses a network environment for simulations, all agents are placed on a random regular network, where each node has the same degree.

In this setup, each node is connected to five other nodes. In each simulation step, agents interact with their neighbors instead of random agents. Due to these modifications in simulation settings, a direct comparison of simulation results may not be meaningful.

Moreover, William et al. [40] utilize ChatGPT-3.5 [46] for the task, while we chose the Mistral 7B – Instruct model [47] due to its open-source nature and its superior performance over other models with similar parameter sizes in the common-sense reasoning task, as observed at the time of implementing this case. We conduct the experiment on Google Colab using a T4 GPU. The model is imported from the HuggingFace transformers library with 8-bit quantization applied for faster inference and reduced costs. With each query to LLM taking approximately 10-10.5 seconds, we complete a single experiment of 50 days and 100 agents, consisting of 5000 queries, in 14 hours 17 minutes (857 minutes).

In the configuration file, we first define the node types (susceptible, infected, recovered) and then assign the node parameters (location, personality traits, and age).

Listing 4.1: Compartments in the generative agents simulation

```

1 compartments:
2     c1:
3         attribute: location
4         probability: 0.1
5         type: node-categorical
6         value: grid
7     c2:
8         type: count-down
9         name: healing
10        iteration-count: 6

```

For diffusion simulations that can be defined using NDLib’s compartment structure, the rules can be defined in this file without the need for any code for the basic logic of the diffusion process. The rules of this simulation can be written with only two compartments given in Listing 4.1. *c1* is a node categorical compartment, which means that the node will move from state 1 to state 2 if the categorical condition is satisfied. Hence, *c1* can be translated as the following:

If a node's location is *grid*, with 0.1 probability, *Susceptible* nodes will transfer to *Infected*. When a node is infected, it takes 6 days to recover. We implement this using a count-down compartment (*c2*), which holds a variable called *healing* for the infected agent. When the agent is infected, healing starts with the value of 6, and it decreases by 1 every day. When this number reaches 0, the agent will go to the *Recovered* state.

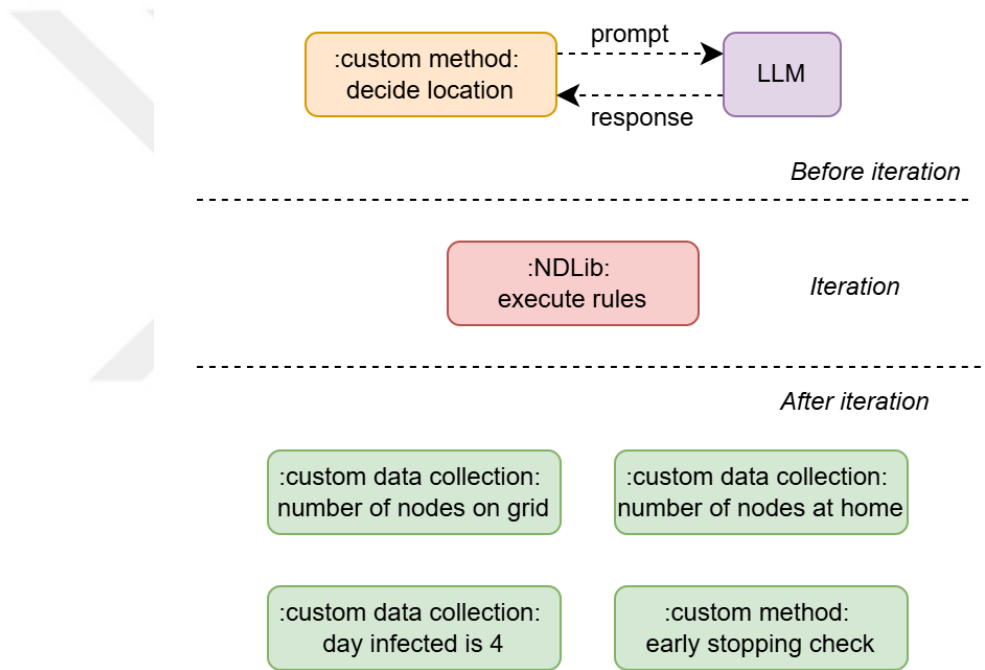


Figure 4.1: Execution order of a full iteration in the generative agents experiment.

Additional methods are implemented to execute before and after each iteration, which are passed to Crowd's run method as list parameters. As shown in Fig. 4.1, determining the location using prompts sent to an LLM is performed prior to executing the infection logic. Data collection methods are executed following the completion of an iteration. Values returned by these functions are saved by default in every snapshot period, allowing access to the iteration-specific information for further analysis.

Fig. 4.2 shows the number of agents staying home in each simulation iteration. This chart is generated in the results tab of the GUI by selecting the automatically

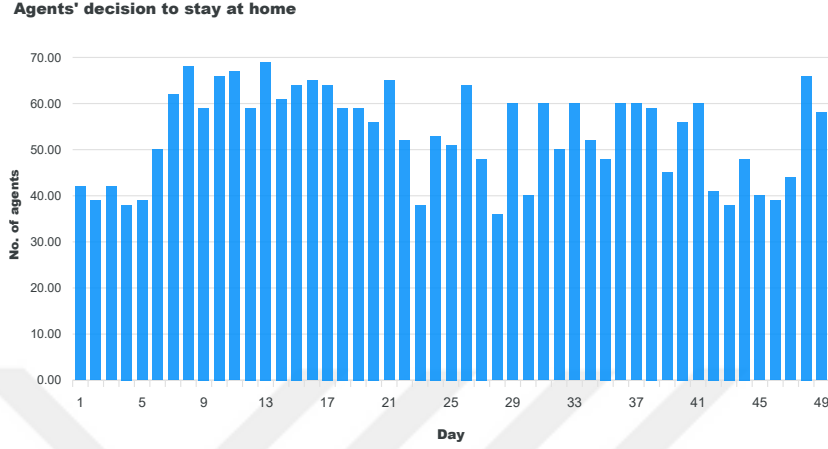


Figure 4.2: Agents staying at home in the generative agents experiment.

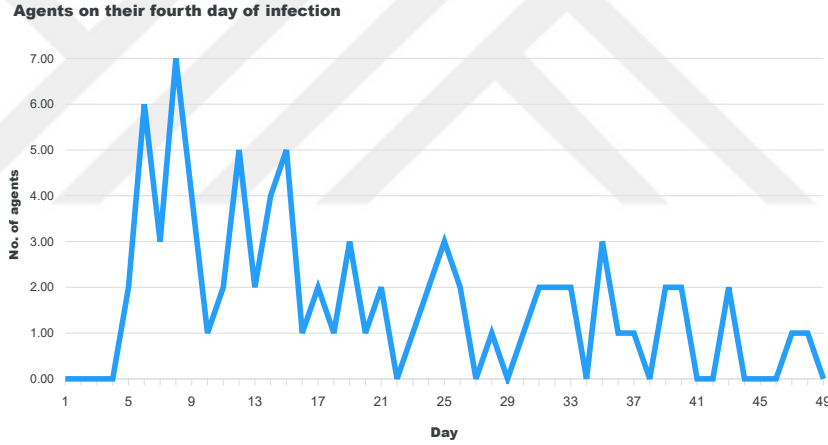


Figure 4.3: Agents on the fourth day of infection in the generative agents experiment.

saved file named after the custom data collection function. New daily active cases are recorded with a user-defined method that counts the agents on their fourth day of infection. The percentage of new daily cases relative to the total population is included in the prompt, contributing to the LLM agent’s decision to stay home or go out. As shown in Figures 4.2 and 4.3, the rises and falls in both graphs follow a similar pattern.

Previously mentioned analysis methods allow us to inspect the simulation from a cumulative point of view. For a node-specific view, we can utilize the interactive network visualization (first and last iterations given in Fig. 4.4) or the *inspect node* tab of the GUI, where the values of node parameters and neighboring nodes

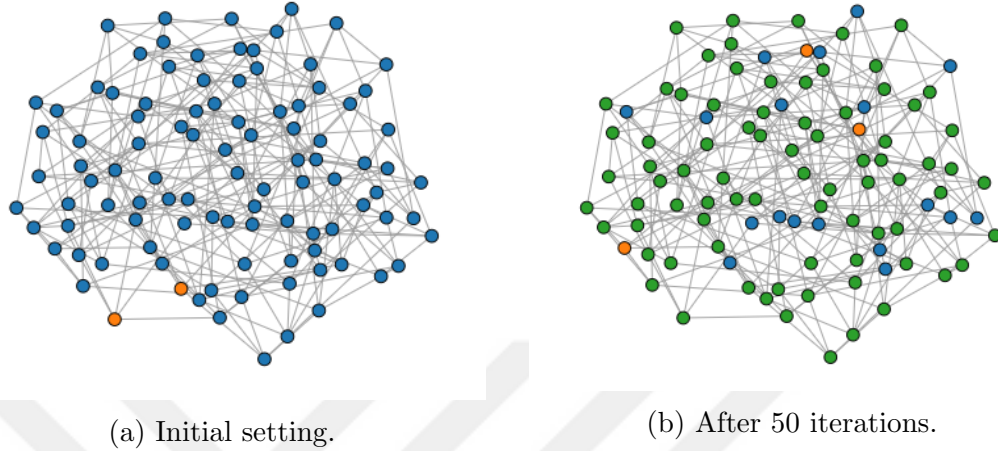


Figure 4.4: Network visualization of the generative agents experiment: $n=100$ and $R_0 = 3$ (blue: Susceptible, orange: Infected, green: Recovered).

are listed.

For simulations employing Generative Agent-Based Models (GABM), DeepMind Concordia [48] is an alternative to Crowd. Besides the actors of the scenario being represented by LLMs, the environment in Concordia is also represented and managed by an LLM. This approach greatly differs from rule-based simulations, where the rules defined by the modeler have full control of the simulation. While Concordia supports programmed logic for agent state updates, frameworks like Mesa are more practical when rule-based decision-making is the main methodology.

Although Crowd and Concordia are both Python-based simulation frameworks, they differ mainly in their focus (social networks vs. GABM) and simulation environment (network-based vs. LLM-managed). Crowd connects agents with links and allows interactions with the other agents (e.g., their neighbors) in a structured environment. Concordia’s simulations are conducted within a generative environment, where the agents to interact are determined based on the scenario rather than the underlying network topology.

Agent information in Concordia is represented as strings stored in an associative memory structure. While Crowd does not provide a built-in agent memory

structure, modelers can implement and integrate one as needed. Moreover, Crowd provides more visualization facilities, including interactive network visualization, node and edge inspection, and data merging for drawing charts, whereas Concoridia mainly supports chart generation.

Overall, Crowd eliminates the need to write any code for infection logic and visualization tasks for this study, only leaving the task-specific LLM prompting and data collection to the modelers. Furthermore, the code structure of this case study serves as a template for future studies employing generative agents for simulations within the framework.

4.1.1 Scenario 2: Influence Maximization

Influence maximization (IM) in social networks is vastly studied with a wide range of applications in economy, politics, and social sciences. The motivation of this task is to find a set of nodes that can spread information to the maximum number of users in a given network [49]. The selected (seed) nodes are marked as “active”, and they try to activate the other nodes that were marked as “inactive” in the beginning. In various studies, researchers combine their selection of seed nodes with a diffusion simulation with Independent Cascade (IC) [50] and Linear Threshold (LT) [51, 52] models, to show the effectiveness of their novel algorithms. This often requires a comparison with the other available methods, and running the simulation multiple times. Within our framework, we aim to simplify and fasten this process by allowing various initial node type selections and automated data collection. Besides selecting the nodes randomly with count and weight input, modelers can determine the top k nodes with PageRank, degree, in-betweenness, closeness, eigenvector, and Katz centralities. Reading from a file option is especially useful to load the seed nodes determined with the researcher’s own algorithm in the previous stage of the study or the other methods to compare that are not provided by Crowd.

In this study, we implement the IC model using an agent-based approach, where instead of looping over the edges, we iterate through each inactive agent

while their neighbors try to influence them. For this, we set the influence probability, which denotes the success rate of a neighboring node activating the currently inactive node v , as $1/(in_degree_of_v)$ [50]. An active node can try to influence its neighbors only once. Initially, we select 100 active nodes with various centrality metrics. For our experiments, we use a Facebook dataset of social circles [41], which has 4039 nodes and 88,234 edges, and is available on the Stanford Large Network Dataset Collection [53].

To satisfy the condition that an active node should try to spread information only once, we utilize a temporary node type, “Active Spreader,” besides the “Active” and “Inactive” states on which the model is based. With this extension, when a node is activated, it is marked as an “Active Spreader”. After a single iteration of trying to activate other nodes, it becomes Active and loses the ability to influence the neighboring nodes. The configuration to initialize the node types is given in Listing 4.2.

Listing 4.2: Definition of node types in the IM simulation.

```

1  nodetypes:
2      Active_Spreader:
3          choose_with_metric:
4              metric: pagerank
5              count: 100
6      Active:
7          random-with-count:
8              count: 0
9      Inactive:
10         random-with-count:
11             count: 3939

```

Following the network initialization, we store the influence probabilities for each edge as an edge parameter, calculated with the formula given in the previous paragraph. In each epoch, we iterate over the nodes. When an inactive node has an active spreader neighbor, we generate a number between 0 and 1. Each active spreader neighbor has the ability to activate the current node if the activation probability of the edge between them is larger than or equal to the current number.

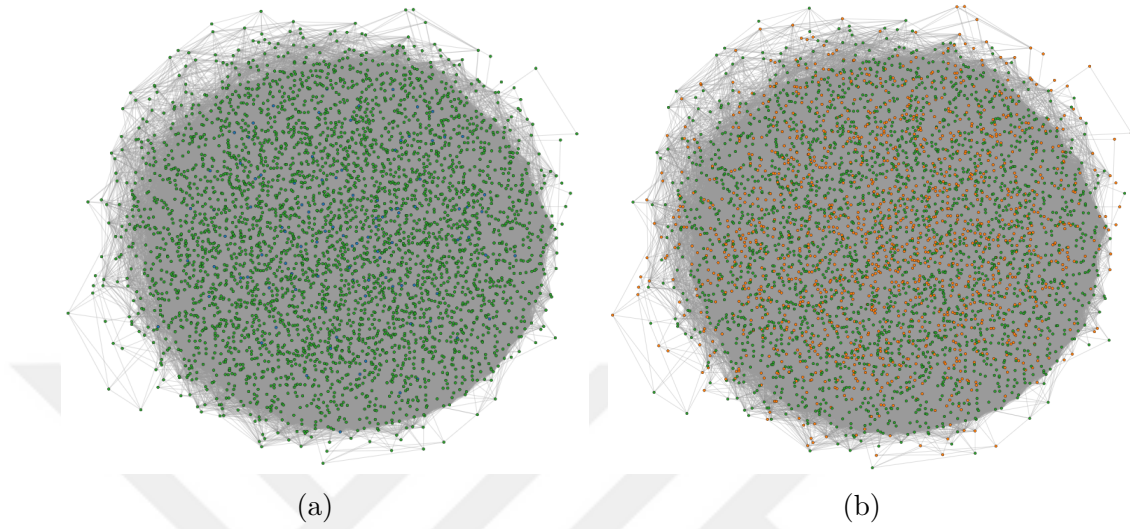


Figure 4.5: Network visualization of the IM experiment, $n = 4039$, $k = 100$, method = PageRank (blue: Active Spreader, green: Inactive, orange: Active). (a) Initial setting. (b) After 20 iterations.

We run four experiments with different methods for the selection of initial active nodes: PageRank, degree, betweenness, and eigenvector centralities. Fig. 4.5 shows the initial state of the network and the activations that occurred after 20 iterations in the PageRank experiment. Fig. 4.6 illustrates the number of nodes of each type in a scatter plot to demonstrate one of the available chart types in the tool. While this data is collected by the framework in each iteration, researchers may need the total number of active nodes to put their results into formats such as tables and charts. For this purpose, we can write a simple data

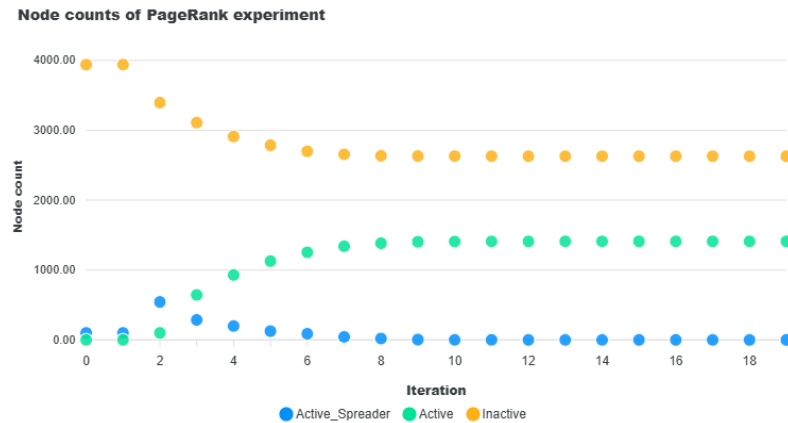


Figure 4.6: IM: Node counts (seed node selection with PageRank)

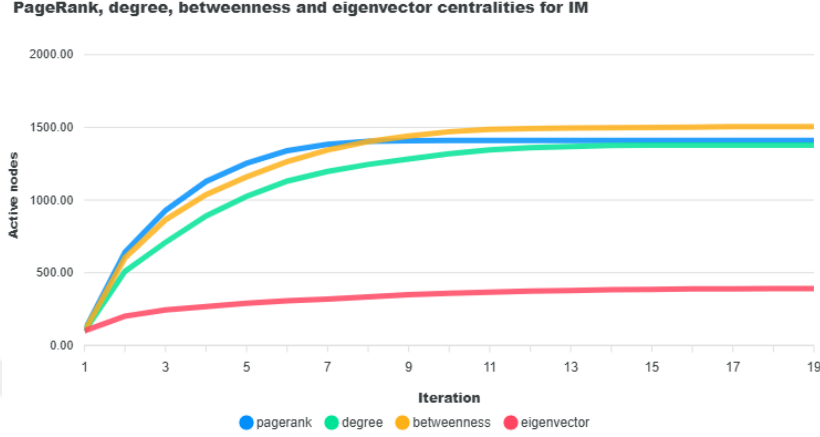


Figure 4.7: IM: Comparison of total activation over time (seed node selection with PageRank, Degree, Betweenness, and Eigenvector centralities).

collection method that we will set to execute after each iteration.

After conducting the PageRank experiment, we only change the selection metric in the configuration file, then re-run the simulation each time with the new setting. To display a comparison in a chart, we need to have a file containing the number of total active nodes in all four simulations, which we can do by using the merge methods provided by Crowd. The new file will show the values for each iteration labeled with the simulation name, which we utilize to draw Fig. 4.7. Comparison figures such as Fig. 4.7 are commonly used in IM papers to show the improvement provided by the novel methods described.

In this implementation, we provide a basic example to illustrate how Crowd can be utilized to test the influence of selected seed nodes with various methodologies, combine the simulation results, and draw charts for comparison. With Crowd’s simplified node initialization, data savers, merge methods, and built-in visualization, it provides an environment to run experiments with less implementation time.

4.1.2 Scenario 3: Trust Game on Networks

Cooperative games are commonly used to model the behavior of human society, where each player adopts a strategy that may result in varying benefits or costs. Each type of game can introduce different kinds of roles and strategies. A common example is the Prisoner’s Dilemma [54], which presents two strategies: being a cooperator who provides benefits to the other player by paying a cost, or a defector who denies benefiting another player, hence paying no cost and only benefiting from the cooperator’s actions [55]. The extension of this game to the populations can be inspected in structured and unstructured populations. In unstructured populations, the individuals can interact with any other individual with a similar possibility, while in structured populations such as networks, individuals interact with the people they are connected to, i.e., their neighbors. The payoffs received by a player depend on the structure of the population.

An extension of cooperative games in economics, trust games, is based on investors and trustees. Investors have an initial fund and they transfer a portion of it to the trustee. In return, the trustee gives back an amount of their earnings, which can also be zero. In this section, we inspect an N-player evolutionary trust game in a network setting, a study by Chica et. al [42], and show how Crowd can be utilized for the simulation of evolutionary cooperative games.

In this study, the agents represent the players, adopting one of the three strategies: Investor (I), trustworthy trustee (T), or untrustworthy trustee (U). The agents can change strategies based on the proportional imitation rule, where the current agent’s payoff is compared to that of a randomly selected neighbor. If the neighbor’s payoff is higher, the agent switches to the neighbor’s strategy with a probability that is set according to the expected improvement this change will provide. Similarly, the current payoff calculation for each agent is also done with respect to local factors, such as using a number of neighboring investors instead of global investors for the trustee’s payoff formulas.

In our implementation, we represent I, T, and U as node types, with strategy, current payoff, and previous payoff as node parameters. Global parameters, the

received fund multipliers for trustworthy and untrustworthy trustees (R_T and R_U , respectively), the temptation-to-defect ratio r_{UT} , and minimum and maximum possible payoffs, are stored as network parameters. Constants are defined in the settings file, while parameters depending on network properties, such as the maximum possible payoff based on average degree, are set in the Python script before running the simulation.

In our experiments, we use the Scale-Free (SF) network provided with the source code of the original study, which is generated using the Barabasi-Albert model with $m = 3$, and consists of 1024 nodes. While the original work experiments with various networks and agent strategies, we only analyze how the increase of the temptation-to-defect ratio r_{UT} affects the overall level of trust within the population. Following the settings provided in the original study, we increase r_{UT} from 0 to 1 in increments of 0.1 and run each configuration for 5000 iterations and 50 times.

We utilize Crowd’s `CustomSimNetwork` class, which allows the modelers to pass in the agent-level methods to be executed in each iteration. As mentioned in the previous paragraphs, agents adopt a proportional imitation strategy, comparing their payoffs with those of neighbors and changing their strategies probabilistically. The mathematical modeling of this strategy is further explained in [42]. We implement proportional imitation using a custom method, which is applied as an agent method during each iteration of the simulation.

After executing this method for each agent, we then call the after-iteration method written for data collection and parameter-setting purposes. In this method, we calculate the new payoffs for each agent with its current strategy and sum up the individual payoffs to obtain a global payoff. We return the global payoff variable, which Crowd automatically stores its value to a file with the name of the method. Following the conclusion of each simulation execution, we run data collection methods to store the value of the r_{UT} variable and node type counts in the same file that is generated by the framework.

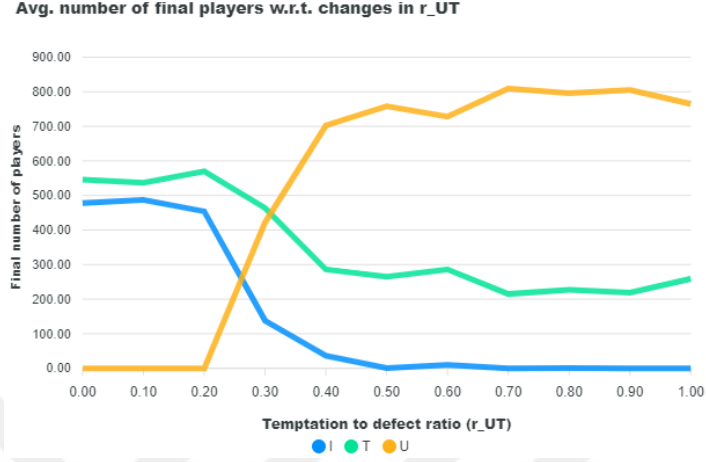


Figure 4.8: N-player trust game: Averaged number of investors (I), trustworthy trustees (T), untrustworthy trustees (U).

After completion of the runs, we merge the results by taking the average of selected values with Crowd’s “merge in parent directory” feature. To visualize the effect of r_{UT} , we further merge the results of simulations with varying r_{UT} values. With the data we obtained, we generate Figures 4.8 and 4.9 in Crowd’s GUI. For Fig. 4.8, we utilize the numbers collected with the after-simulation methods. In this figure, we can see the number of investors and trustworthy trustees decreasing as more people adopt the untrustworthy strategy. Even though the agents switch to the untrustworthy type to gain more benefits, Fig. 4.8 shows that the global net wealth declines as the number of untrustworthy agents increases.

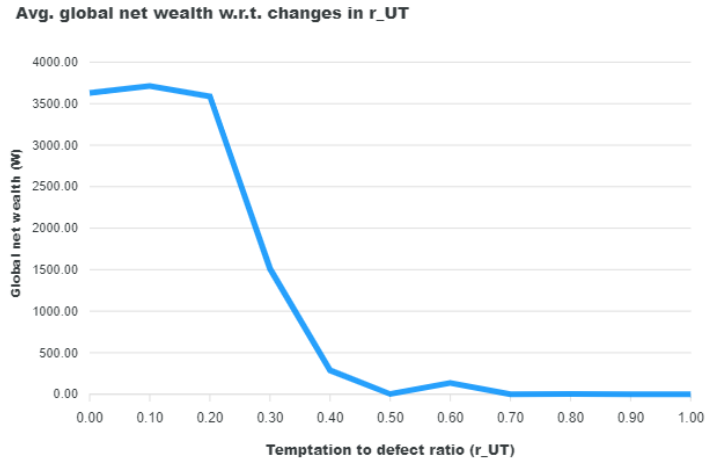


Figure 4.9: N-player trust game: Averaged global net wealth. Over 50 simulations for different temptation-to-defect ratios r_{UT} ($R_T = 6$).

In addition to these charts, Fig. 4.10 illustrates the network in the first and last iterations, highlighting the disappearance of investor agents and the takeover by untrustworthy agents across the network. Our results align with the findings of the original study, as Figures 4.8 and 4.9 show similar or nearly identical values, with slight differences likely due to the stochasticity of the system. Moreover, the dynamics observed in the simulation also follow the intertemporal sequential social dilemma definition, where short-term gains lead to choices that harm long-term collective outcomes [56].

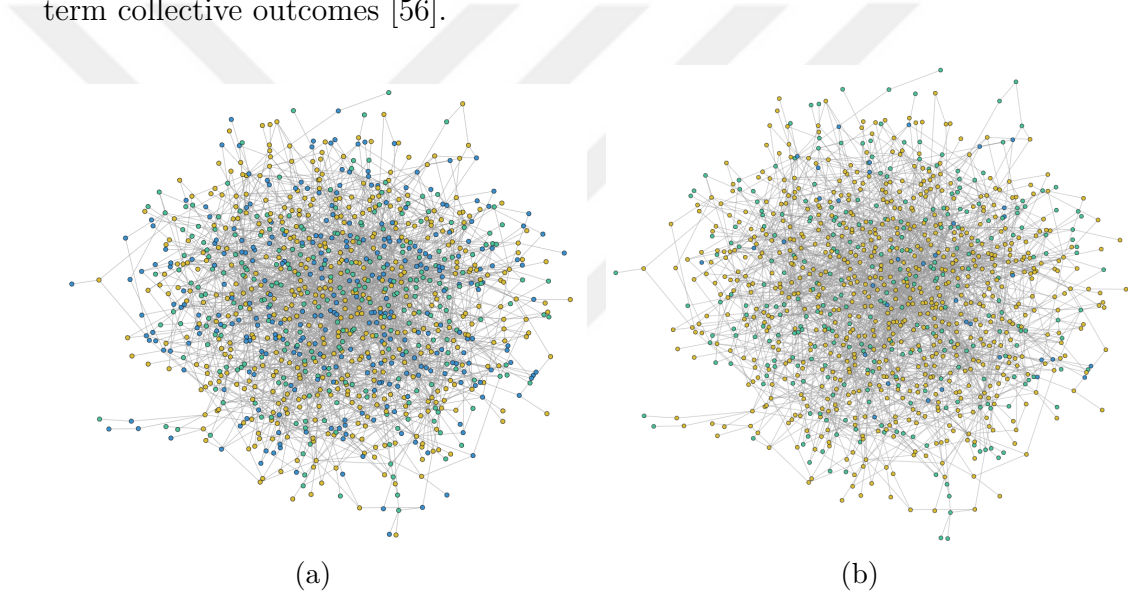


Figure 4.10: Network visualization of the trust game experiment: $n = 1000$ and $r_{UT} = 0.4$ (blue: Investors (I), green: Trustworthy trustees (T), yellow: Untrustworthy trustees (U)). (a) Initial setting. (b) After 5000 iterations.

This case study provides an example of the use of Crowd’s `CustomSimNetwork` class, showcasing how the framework can be utilized in a detailed analysis of cooperative games in structured populations represented as networks. Compared to other implementations of the scenario in different frameworks, Crowd enables quick modeling with minimal code.

Chapter 5

Personality-Driven Social Dynamics with LLM Agents in Shared Residential Settings

The effect of personality and preferences on building energy consumption has been previously explored in some studies [57, 58]. However, these works do not consider the integration of social network structures with generative agents for a personality-driven simulation of decision making in the context of building energy modeling. Our approach introduces a novel perspective on modeling the residents in households and aims to represent the human decisions in a more comprehensive manner. We choose building energy as our social decision context due to its multifaceted nature. Energy use is impacted by a mix of external factors (e.g., weather conditions), social ties, personal traits, and preferences. A simulation of decision-making in the building context allows the inspection of the relationship between these factors over time. On each day of our simulation, agents determine the temperature for the central heating system through both family-level and building-level polls. With this simulation scenario, we execute simulations in three different settings with varying percentage distributions of the positive personality traits. Through regression analyses at the network level, we examine the impact of this distribution on all our network-level variables, such

as average happiness level, cost, and average degree of friendships. At the node level, we determine the most significant factors for the individual happiness levels and degree decisions through panel data regression.

5.1 Methodology

5.1.1 Creating Families

We first load Zachary’s Karate Club network [59], which is an undirected real-world network widely utilized in social networks research. It consists of 34 nodes and 78 edges. We categorize these nodes as the “Family Representatives”, who will be presenting the family’s choice of degree to the building poll every day. The links between Family Representatives represent the friendship relationship. The weight of these edges is utilized in the simulation as “closeness level”.

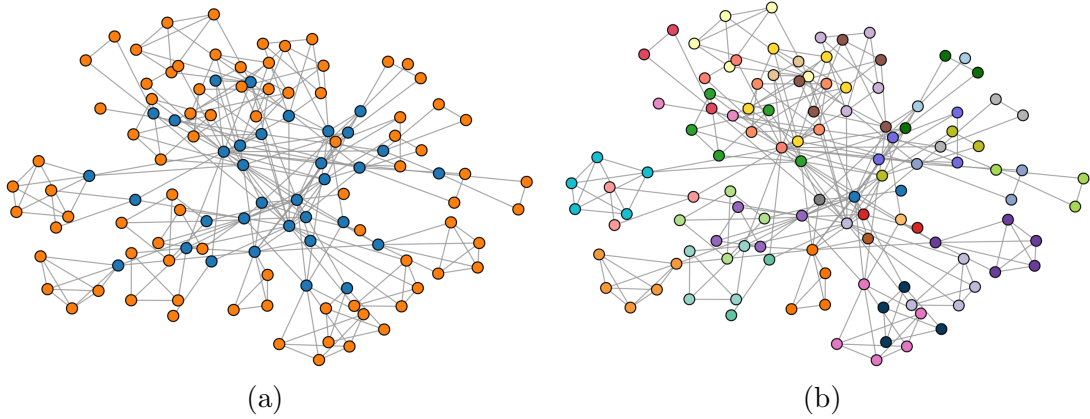


Figure 5.1: Network visualization of created families where nodes are colored (a) according to the roles (blue: Family Representative, orange: Family Member), (b) according to their family IDs.

For each family representative, we randomly add n family members, where n is selected from the range $[0, 4]$. All family members are connected to each other, creating a fully connected subgraph represented by the Family Representative’s ID. The new network consists of 116 nodes and 246 edges. Fig. 5.1a illustrates the created network according to nodes’ roles (Family Representative or Family

Member), and Fig. 5.1b shows the family relationships present in the network, where the fully connected family sub-graphs are observable through the coloring of the nodes.

Following this step, each node in the extended graph is assigned a name, age, and a set of personality traits. These traits are selected from the Big 5 Traits’ Facets [60], based on their relevance to the simulation’s scope. Facets such as *Angry Hostility* and *Feelings* are included to observe their influence on agents’ happiness levels, while *Impulsiveness*, *Assertiveness*, *Alturism*, *Compliance*, and *Deliberation* are expected to affect the temperature choices and closeness levels between friends. Instead of assigning numerical values to represent traits, we use a pair of opposing adjectives—one positive and one negative—for each facet. This approach simplifies the interpretation of the facets for the LLM. The assignment ratio of these facets to nodes is varied for the different simulation settings.

Table 5.1: Node parameters related to personality and associated adjectives

Parameter Name	Big 5 Trait (Facet)	Positive Adj.	Negative Adj.
Angry Hostility	Neuroticism (N2)	Easygoing	Easily-angered
Impulsiveness	Neuroticism (N5)	Self-controlled	Impulsive
Assertiveness	Extraversion (E3)	Assertive	Passive
Feelings	Openness to Experience (O3)	Emotional	Unemotional
Alturism	Agreeableness (A3)	Selfless	Selfish
Compliance	Agreeableness (A4)	Cooperative	Uncooperative
Deliberation	Conscientiousness (C6)	Cautious	Careless
Environmentalism	x	Environmentalist	Not environmentalist
Frugality	x	Frugal	Wasteful

In addition to the node parameters representing the personality types, agents are also labeled based on their care for the environment, spending habits, and heater preferences. A list of node parameters related to personality and the associated adjective pairs is given in Table 5.1. The node and edge parameters created in this stage are used to personalize the prompt sent to the LLM, which will act as a person with these traits and simulate human behavior with a broader range of possibilities compared to rule-based methods.

5.1.2 Extracting Weather Data

To observe the behavior of the generative agents from a broader perspective as much as possible within 30 days, we select a month of the year with fluctuating temperatures. This corresponds to a period between February 15 to March 16 from the Typical Meteorological Year (TMY) data obtained utilizing pvlib [61]. We select Ankara, the capital of Turkiye, to run our experiments, which can be extended in future studies.

Instead of using the hourly weather data provided, we take the average temperature for each day. While this approach does not consider the day and night temperature differences, it reduces the execution time by giving each day 1 iteration in the simulation, instead of 24 iterations. Therefore, we can utilize the data of a longer period (30 days) and run our simulations with different settings, while each simulation takes notably less time to complete. We execute our simulations on an L4 GPU through Google Colab, and a simulation of 30 days takes approximately 13 hours.

5.1.3 Simulation Scenario

Our simulation is based on a building with 34 apartments and a central heating and cooling system. Each day, families come together to decide on a temperature to set on this system. For this decision, they consider their own temperature preferences, their family members' preferences, and the degree outside. Each family member presents a degree, affected by these properties as well as their personality traits. The average of all the members' choices is taken and given to the Family Representative. If a person lives alone, only their own preferences are considered for this step.

Each person is also assigned a happiness value. This value is initially set to 100 at the beginning of the simulation, and each day, agents are asked about their level of happiness with respect to their previous day's choice and the final

temperature set in the building.

After the family’s decision, the next step for a Family Representative is to pass another degree choice to the building poll. In this stage, family representatives are also informed about their friends: Their closeness level, friend’s family’s degree choice for that day, and friend’s degree suggestions for the last three days. The family representatives make a second decision based on their friends’ and family members’ choices, their own temperature preference, and personality. Each day, these agents may decide to increase or decrease the closeness values with their friends, updating the weight of the edge between them.

5.1.4 Simulation Methodology With Crowd

For our simulations, we choose Crowd’s Custom Simulation Network as the environment. This network type requires the simulation logic to be defined by the modelers in Python methods, which are then passed to Crowd for execution. As explained in Section 3.1.1, Crowd provides four intervals for the execution of custom methods: *before-iteration* (can be iteration setup methods), *every-iteration-agent* (called for each agent in random order), *after-iteration* (can be data collection methods) and *after-simulation* (finalize the simulation or collect end results).

Our custom methods for this study, and their execution periods are provided in Fig. 5.2. Instead of utilizing *every-iteration-agent* method call interval of Crowd, we directly loop over the agents with respect to their family IDs, or directly perform model-level calculations at the same time in these functions.

Environment variables in Crowd are named as “network-parameters”. In our simulations, we store the current day and its temperature as network-level parameters and update them each day. Followed by this initialization, we have two phases of degree decisions, following the simulation scenario explained in the previous subsection. In Phase 1, we ask the LLM to act as the agents in the families

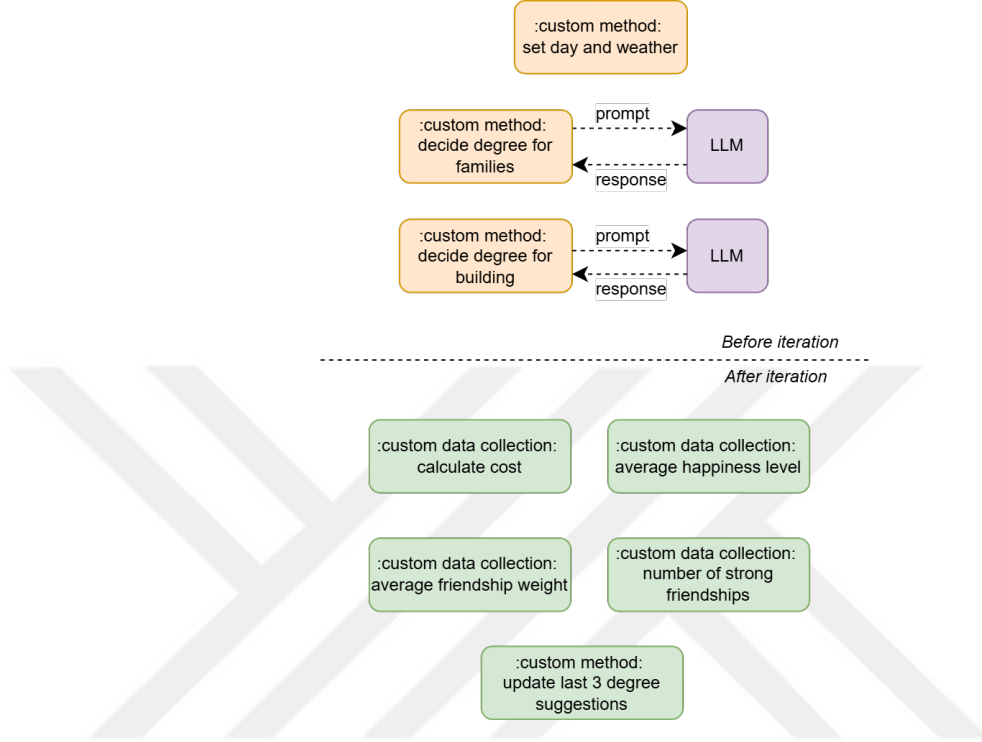


Figure 5.2: The methods passed to Crowd framework and their execution order

and provide its choices. In Phase 2, LLM makes decisions for the family representatives, and we set the building’s temperature as the average of their degree suggestions.

Each day, we calculate the cost using Eq. 5.1 given below:

$$cost = |d_{set} - d_{outside}| \times C \quad (5.1)$$

In this equation, C is a constant that represents the cost of increasing the building temperature by one degree. As the aim of this study is to present a simulation methodology, rather than to provide precise estimates of real-world energy consumption, we set $C = 1$ for simplicity.

The data for the average friendship weight and the number of strong friendships are also collected at the end of each iteration. For this calculation, only the nodes with the “friend” connection (i.e., the Family Representatives) are considered.

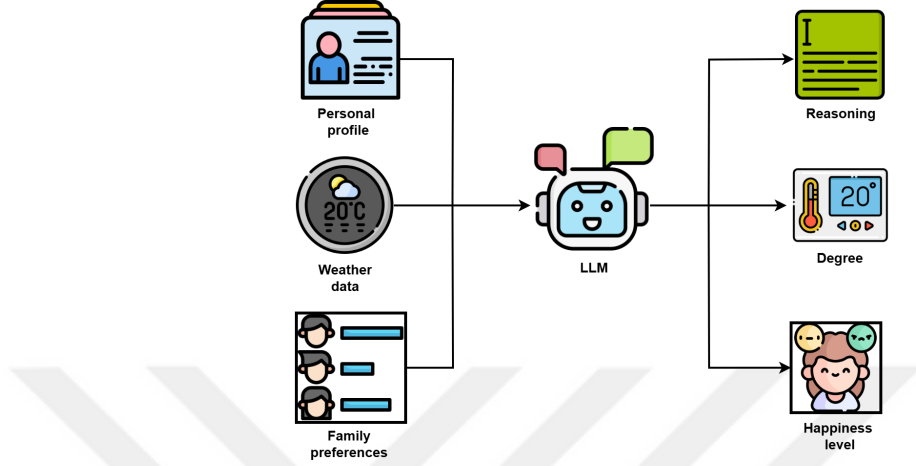


Figure 5.3: Inputs provided in the query to LLM and outputs obtained in the first phase of the simulation

The edge weights between friends are initially obtained from the Karate Club network, and they may change with respect to agents' decisions in each iteration. When the weight of an edge between two friends is larger than 3, it is counted as a strong friendship.

Lastly, before moving on to the next iteration, we call the *update last 3 choices* function, which updates the associated node parameter that is given to the LLM in the second phase prompt.

5.1.5 Agent Decisions With LLMs

In our simulations, we assign the role of decision-making for each agent to an LLM, instead of following a rule-based model. We choose Mistral 7B [47] and apply 8-bit quantization, aiming to complete the queries to LLM more quickly while maintaining the use of an open-source model.

An illustration of the first phase of the simulation is given in Fig. 5.3, and an example prompt is given in Fig. 5.4. In this stage, agents are given a list of family members' heater preferences, their current happiness levels, and the weather information. Moreover, a reference table for the heater preferences categories and

```

You are Jennifer. You are 61 years old.
Your personality traits are:
    Easily-angered, Impulsive, Passive, Unemotional, Selfish, Uncooperative, Careless, Not environmentalist, Wasteful.

You live in an apartment building and share a home with other family members. Every day, you come together and decide what you
should set the degree of the heater. Later, your family representative will submit this degree to the building-wide poll and
the final temperature will be set depending on everyone's choices.

Heater preferences of you and your family members include:
- Danielle prefers: cool
- Angela prefers: cold
- David prefers: warm
- You prefer: cold

Reference table:
    Cold: <18 degrees
    Cool: 18-20 degrees
    Neutral: 21-24 degrees
    Warm: 25-27 degrees
    Hot: >27 degrees

Your current happiness level is 80 (from 1 to 100). Happiness increases when the final heater degree is close to your
preference, and decreases otherwise.

Currently, it is -3 degrees outside.

**First task**: Decide the best heater degree **today** based on:
- Your personality traits,
- Heater preference of you and your family,
- The current outside temperature,
- Your current happiness level.

**Second task**: Set a new happiness for yourself (an integer from 1 to 100) based on:
- The building-wise degree set yesterday: 22.
- Your traits

**Response format (must follow strictly)**:
Reasoning: [Explain your reasoning in 1 paragraph. Mention how your traits influence your decision.]
Degree: [An **integer only**, e.g., 22]
Happiness level: [An **integer between 1 and 100**, no text or extra commentary]

Only provide **one** reasoning, **one** degree, and **one** happiness level. If multiple reasonings exist,
combine them into one paragraph.

Example (format only, not your situation):
Reasoning: It is 10 degrees outside. I like warmer temperatures, and others do too. I think 24 degrees is a good
balance. I felt okay yesterday, but I want to be a bit warmer today.
Degree: 24
Happiness level: 90

```

Figure 5.4: Phase 1 sample prompt

corresponding degree ranges is provided. In our initial experiments, we observe that without this reference, LLM may suggest nonviable degrees such as 1 or 100 as the output. To further enhance our prompt, we separate the tasks, itemize the decision factors, and add an example output within the instructions. These suggestions, provided by ChatGPT [62], improve the quality of the reasoning provided in each query's output and show that larger models can be utilized to polish the input for smaller models for these types of tasks.

The output provided by the LLM for the sample prompt is given in Fig. 5.5. As instructed, the LLM returns a reasoning for its decision based on the personality

Reasoning: Given my cold preference, the outside temperature of -3 degrees, and the fact that I am uncooperative and selfish, I would like to set the heater degree as low as possible to minimize the comfort of others while still maintaining a livable temperature for myself. However, considering my passive and careless traits, I might not push for the lowest possible temperature, as I might not want to cause a significant disagreement or discomfort within the household. Therefore, I propose a heater degree of 18 degrees (cool).

Degree choice: 18
Happiness level: 80

Figure 5.5: Phase 1 sample output

traits of the agent and preferences of the agent’s family. We extract the degree choice and happiness levels of each agent, and as explained in Section 5.1.3, provide each family’s average suggestions to the Family Representative in the second phase.

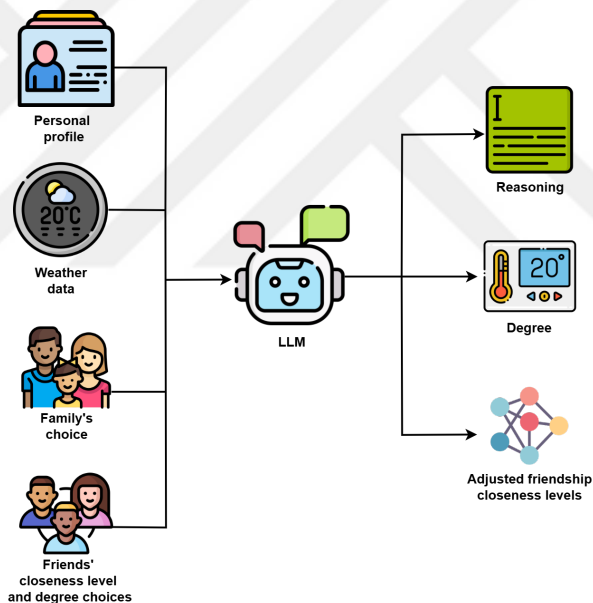


Figure 5.6: Inputs provided in the query to LLM and outputs obtained in the second phase of the simulation

Fig. 5.6 illustrates the information provided to the LLM and the requested output variables. Through our experiments, we observe that the order and structure of the friends’ information in the prompt given in Fig. 5.7 impacts the responses. When all the information related to a friend is given together without categorizing with respect to the contents, the LLM may evaluate any one of the categories and ignore the others. Hence, we categorize the information with respect to its contents, and specifically place the agent’s suggestions and preferences right after for better comparison. Since the number of closeness updates is not predetermined, we specifically instruct LLM to return a JSON object for the

You are Tim. You are 21 years old.
Your personality traits are:
Easygoing, Self-controlled, Assertive, Emotional, Selfless, Cooperative, Cautious, Environmentalist, Frugal.

You live in an apartment building, sharing a home with other family members. Every day, you come together and decide what you should set the degree of the heater.

You have 3 friends in this building. You have a closeness level between 1 to 5 with each friend, where a higher number indicates a stronger friendship. Below is a summary about your friends:

- Mike (ID: 25) - closeness level: 3
- Kim (ID: 27) - closeness level: 4
- Raul (ID: 31) - closeness level: 5

- Mike's last 3 degree suggestions were: [19, 19, 21]
- Kim's last 3 degree suggestions were: [25, 22, 22]
- Raul's last 3 degree suggestions were: [24, 24, 25]
- Your last 3 suggestions were: [26, 25, 25]

- Today Mike's family decided on: 20
- Today Kim's family decided on: 25
- Today Raul's family decided on: 24
- Today, your family decided on: 23

Note: Your friends also knew your family's decision before submitting the last degree suggestion to the building poll.

Also, your heater preference is hot.
Reference table:
Cold: <18 degrees
Cool: 18-20 degrees
Neutral: 21-24 degrees
Warm: 25-27 degrees
Hot: >27 degrees

Your current happiness level is 98 (from 1 to 100). Happiness increases when the final heater degree is close to your preference, and decreases otherwise.

Currently, it is -2 degrees outside.

You are your family's representative.

First task: Decide the best heater degree **today** based on:

- Your personality traits,
- Your heater preference,
- Your family's request,
- Your friends' choices,
- The current outside temperature

Second Task: You may choose to update your closeness levels to your friends based on:

- How aligned their decisions in the last 3 days were with yours
- Your personality traits

Your choice will be sent to the building poll and the degree will be set depending on everyone's choices.
Please also provide your reasoning.

Response format (must follow strictly):
Reasoning: [Explain your reasoning in 1 paragraph. Mention how your traits influence your decision.]
Degree: [An **integer only**, e.g., 22]
Updated closeness: { "friend_name(id)": new_level, "friend_name(id2)": new_level, ... }

- You must use **double quotes** for both keys and string values.
- The **Updated closeness** must be a valid **JSON object**.
- New closeness levels must be between [1, 5].

Only provide **one** reasoning and **one** degree. If multiple reasonings exist, combine them into one paragraph.
If you update any closeness levels, list them explicitly in the format above.
Any incorrect formatting will result in your response being ignored.

Example (format only, not your situation):
Reasoning: I want to support my family's request while staying close to my own preference. Clara's suggestions were far from mine, and I feel she doesn't care about my comfort, so I will reduce my closeness with her.
Degree: 20
Updated closeness: { "Clara(27)": 1, "David(40)": 5 }

Figure 5.7: Phase 2 sample prompt

```

Reasoning: "As an environmentalist and frugal individual, I aim to
conserve energy and reduce costs. Given the current outside temperature of
-2 degrees, I believe it's unnecessary to set the heater too high. I also
consider the closeness levels of my friends, and Raul's (ID: 31) consistent
choices align well with mine. However, Kim's (ID: 27) and Mike's (ID: 25)
choices have been less aligned with my preference, which indicates a lower
level of closeness. Therefore, I will set the heater degree to a neutral
level, balancing my preferences with energy conservation and the need for
a comfortable living environment.",
Degree choice: "22",
Updated closeness: { "Mike(25)": 2, "Kim(27)": 3 }

```

Figure 5.8: Phase 2 Output

ease of parsing.

The output corresponding to the example prompt is given in Fig. 5.8. In this sample, it can be seen that the agent decides to decrease the closeness levels with the friends with differing choices, while keeping the highest closeness level with the friend suggesting similar temperatures. Moreover, despite the agent's hot heater preference, family and close friends Kim and Raul's higher temperature choices, the agent ultimately suggests a degree falling into the lower neutral category to conserve energy and reduce costs because of his personality traits.

5.2 Results

To investigate the impact of personality traits on happiness levels, degree selections, and closeness levels between friends over time, we run our experiments in three different setups. While the initial network structure remains constant in all settings, the main difference is the distribution of personality traits. As previously explained in Section 5.1.1, we utilize adjectives derived from the Big 5 Traits' Facets and group them as positive or negative. The derived adjectives are grouped together with their opposites, and assigned to agents in each simulation based on the selected *positive trait percentage*. In our three setups, we select this *positive trait percentage* as 100%, 50%, and 0%, respectively, and collect

both network-level and node-level data to observe the effects of this distribution. While these experiment setups are limited and may not fully reflect the complexity of real-world societies, they provide an efficient way to observe the contrast considering the cost of running generative agents simulations with large networks and long iterations.

5.2.1 Network-level Analysis

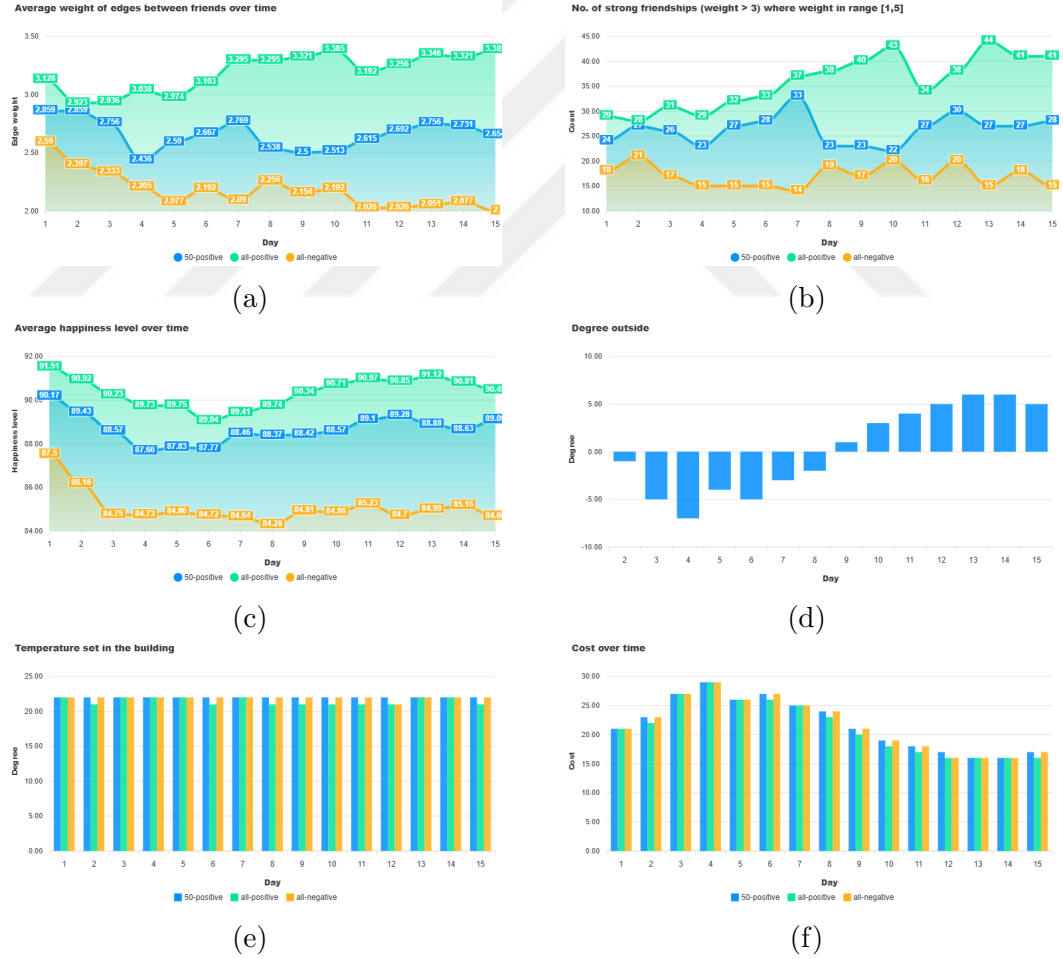


Figure 5.9: Charts generated with Crowd. (a) Average weight of edges between friends over time. (b) Number of strong friendships (weight > 3 where weight in range [1,5]). (c) Average happiness level over time. (d) Degree outside. (e) The temperature set in the building. (f) Cost over time.

At the network level, we examine how the structural features of the network,

temperature-related factors, and average happiness levels of agents evolve over time. For this analysis, we first generate charts with Crowd, given in Fig. 5.9. Then, to complement the visual representation of our results, we run the Ordinary Least Squares (OLS) method to compute the regression slopes for each metric given in these charts. The slope values presented in Table 5.2 quantify the rates of change and highlight the differences across our three types of personality trait distributions. While the charts illustrate the daily fluctuations occurring with respect to the changes in outer factors such as the weather conditions (degree outside), the numbers in Table 5.2 provide a broad perspective on the overall trends. The p-values in this table represent how well the regressed line is able to represent the general trend of the changes of the corresponding variable.

Table 5.2: OLS slope coefficients (with standard errors) over 15 days

Metric	All Positive	50% Positive	All Negative
Avg. friendship weight	0.0298 (0.0059)***	-0.0145 (0.0088)	-0.0295 (0.0059)***
No. of strong friendships	1.0786 (0.2021)***	0.0214 (0.1648)	-0.0464 (0.1381)
Cost	-0.8071 (0.1770)***	-0.7750 (0.1729)***	-0.7893 (0.1804)***
Temperature set	-0.0321 (0.0308)	0.0000 (0.0000)	-0.0143 (0.0155)

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Cohesion in networks refers to the existence and strength of ties, or the level of connectedness among nodes, from a structural perspective [63]. Since average friendship weight and the number of strong friendships are metrics that rely on tie strength, they can serve as indicators of network cohesion. Table 5.2, Figures 5.9a and 5.9b show that both metrics increase over time in the all-positive case, while they decline in the all-negative case. This suggests that these two opposing personality type cases can result in notably distinct dynamics of network cohesion.

As explained in Section 5.1.4, we calculate cost as the absolute difference between the temperature set and the degree outside. Hence, the calculated cost is highly dependent on the weather outside, which can be observed from Figures 5.9d and 5.9f. With the increase in the outside temperature, the cost decreases across all cases.

Fig. 5.9e illustrates that, regardless of individual agent traits or preferences,

the building temperature (determined by the average choice of the Family Representatives) consistently remains around 21–22 degrees across all simulations. This temperature falls under the Neutral range given as input to the LLM. This indicates that despite the agents do not communicate directly with each other to negotiate a temperature that is best for everyone, the collective decision still falls under a facilitative degree. While the building temperature is always set to 22 in the 50% positive case, we observe that both all-positive and all-negative agents occasionally vote for 21 degrees, which explains the differing coefficients in Table 5.2. In those days, the decrease in cost is slightly impacted by this metric as well.

Lastly, Fig. 5.9c illustrates the changes in the average happiness levels over time. The mean happiness over the 15 iterations is 91.01 for the all-positive case, 89.33 for the 50%-positive case, and 85.99 for the all-negative case. These values represent increases of 1.88% and 5.84% in the all-positive case compared to the 50%-positive and all-negative cases, respectively. While the difference is more apparent in comparison to the all-negative case, it is relatively small when compared to the 50%-positive case.

For another network-level analysis, we inspect the influence of the other variables on the average happiness level of agents, using OLS. For the regression, we combine the results obtained in the three settings and include the data of all 30 days collected for the 50% positive traits case, while each of the other cases contributes 15 observations, summing up to 60 observations.

Table 5.3: OLS Regression results to analyze the influence of network-level variables on average happiness level

Parameter	Coefficient (Std. Error)	<i>p</i> -value
Average friendship weight	4.723 (0.314)	< 0.001
Degree outside	0.076 (0.023)	< 0.001

Although we collect multiple network-related and temperature-related information in the simulation, we only utilize one of each category in this analysis to avoid correlation between the regressors. Hence, in Table 5.3, we present the best combination of regressors: average friendship weight and degree outside. Our results indicate that a 1-unit increase in the average friendship weight corresponds

to a 4.72-point increase in happiness, while a 1-degree rise in external temperature (degree outside) is associated with a 0.076-point increase in happiness.

In our study, the agents are prompted to keep their relationships closer with the friends whose choices align with theirs. Moreover, depending on their personality traits, agents may pay more attention to friends' choices with higher closeness levels and try to accommodate between their own preference, family's request, and friends' choices. This attempt at cooperation and higher levels of connectedness results in a higher level of average happiness level.

5.2.2 Node-level Analysis

Since our aim for the node-level analysis is to inspect the effects of different personality traits and node-level SNA metrics on individual happiness level and degree choices, we exclude the 100% and 0% positive settings for this task. With the remaining data, we use a Correlated Random Effects (CRE) [64] model to inspect both within-agent and between-agent variation in 30 days, for both happiness and degree choice analyses. We chose the CRE model over the standard random effects model as the latter was rejected by the Hausman test. However, the fixed effects model eliminates the time-invariant predictors, which are the main focus of our analysis.

During our simulations, we collect two time-variant types of continuous data: individual happiness levels and daily degree choices. On the other hand, the time-invariant categorical data includes node parameters, such as personality types (Big 5 Facets, environmentalism, frugality), non-weighted centrality metrics, and heater temperature preferences. While estimating the impact of the other variables on the happiness levels, we include another regressor required for the CRE model, labeled as *degree choice mean*. This is calculated for each agent separately by taking the average of their own degree choices, hence staying constant throughout 30 days. Similarly, we include the *happiness level mean* while analyzing the effects on the degree choice.

Table 5.4: Correlated random effects model to estimate happiness level on 30-day node level data

Variable	Estimate (Std. Error)	<i>p</i> -value
Degree choice	0.2468 (0.0601)	< 0.001***
Degree choice mean	3.8119 (0.0920)	< 0.001***
Assertive (E3)	3.2413 (0.9494)	< 0.001***
Emotional (O3)	1.0160 (0.9354)	0.2775
Selfless (A3)	-2.6886 (1.2999)	0.0387*
Environmentalism	2.0929 (1.2058)	0.0827†
Warm temperature pref.	-7.7128 (1.1907)	< 0.001***
Hot temperature pref.	-13.584 (1.2503)	< 0.001***

† $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Table 5.4 shows the results of the regression on individual happiness levels with the best-performing estimators. With 116 nodes and 30 iterations, we have 3480 observations for this analysis. The results indicate that the most significant regressors are degree choice, agents’ mean degree choice, assertiveness, and temperature preferences.

The degree choice (the temperature suggestion of the agent for the day) and the agents’ mean degree choice positively impact agents’ happiness levels, indicating that people with higher suggestions achieve higher happiness. On the other hand, warm and hot temperature preferences have a negative relationship with the happiness level. While this may seem contradictory on the surface, the β values/coefficients of these variables and the reference information given to the LLM in the prompt for the temperatures provide a different point of view.

As can be seen from the example prompts given in Figures 5.4 and 5.7, warm and hot preferences of nodes indicate a preference for temperatures above 25 degrees. However, Fig. 5.9e shows that the temperature finally set in the building is either 21 or 22 in all iterations. These values fall under the neutral range, and as the gap between the agents’ preferences and the neutral range increases, their dissatisfaction with this selection becomes more significant. The results from Table 5.4 illustrate that the agents having a warm temperature preference have happiness levels that are 7.7128 points lower, while those with a hot temperature preference experience a decrease of 13.584 points in happiness. This suggests that

the positive relation between the degree choice and happiness levels ($\beta = 0.2468$) holds true when the degree choice is not far from the neutral range or is still in the colder ranges.

In terms of personality traits, assertiveness and selflessness have the highest significance. While assertiveness is positively associated with happiness ($\beta = 3.2413$), selflessness has a negative impact ($\beta = -2.6886$). As assertive agents aim to align the outcome with their own preferences, they tend to suggest temperatures that match their ideal range, which keeps their happiness levels high. On the contrary, selfless agents may make decisions far from their choices, considering the preferences of their family and friends more than their own, resulting in lower happiness values.

Being environmentalist also affects the happiness positively ($\beta = 2.0929$), however it is not as significant as the other personal traits. Although it is the third most important personality facet, being emotional is shown to have less significance.

Table 5.5: Correlated random effects models to estimate degree choice on 30-day node-level data

Variable	Model 1		Model 2	
	Estimate (Std. Error)	<i>p</i> -value	Estimate (Std. Error)	<i>p</i> -value
Happiness level	0.0202 (0.0049)	< 0.001***	0.0202 (0.0049)	< 0.001***
Happiness level mean	0.2157 (0.0058)	< 0.001***	0.2273 (0.0056)	< 0.001***
Assertive (E3)	-0.5343 (0.2287)	0.0196*	-0.4616 (0.2682)	0.0854 [†]
Selfless (A3)	0.5556 (0.3073)	0.0707 [†]	0.5422 (0.3596)	0.1317
Degree centrality	3.7636 (4.2801)	0.3793	–	–
Warm temperature pref.	2.1054 (0.2745)	< 0.001***	–	–
Cool temperature pref.	–	–	-1.0937 (0.3323)	0.001***
Hot temperature pref.	3.4629 (0.2871)	< 0.001***	2.4978 (0.3282)	< 0.001***

[†] $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Table 5.5 shows the combined results for the CRE model on agents' degree choices with two models, differing in the regressors included. Similar to the estimates from Table 5.4, happiness level and degree choice are positively associated. On the other hand, the sign of the coefficient β changes for the personality traits assertiveness and selflessness. Assertiveness negatively impacts the degree choice ($\beta = -0.5343$), while being selfless increases the degree choice by 0.5556. This

Reasoning: Given my preference for cool temperatures, I would like to set the heater degree closer to the 'cool' range to align with my preference. However, considering the outside temperature is 5 degrees, setting the heater too low might make the apartment uncomfortably cold. Also, my family members prefer warmer temperatures, so I should compromise to maintain a livable environment. I choose 20 degrees as a reasonable compromise.

Figure 5.10: Example output showing the agents choosing the higher limit of their preference category considering their needs and other people’s comfort.

may be explained by the fact that assertive people tend to suggest lower temperatures to align with their own preferences, despite the needs and preferences of the other people in the building. In contrast, selfless people aim to keep the temperature in the neutral range for the comfort of their family and friends.

A key observation from the results presented in Table 5.5 is regarding the degree centrality. Degree centrality is a metric commonly employed in SNA to define a node’s importance with respect to its number of connections. In our study, the nodes have both family and friend ties, and our calculation for this metric is inclusive of both types of connections. Although the centrality metrics or the full network structure is never given to the LLM in any phase, the nodes with higher degrees appear in more people’s prompts as friends. Hence, these nodes are expected to influence more agents’ decisions, making degree centrality a significant regressor. However, with a p-value of 0.3793, our results indicate that the degree centrality of the nodes is not as influential as the other factors.

Similar to their impact on the happiness levels, the temperature preferences remain significant for the *degree choice*. Despite warm and cool temperature preferences have the same distance to the neutral category, their coefficients differ notably. While this may be a consequence of using different regressors in the model, we also observe this trend through the agents’ responses. As can be seen in Fig. 5.9d, the temperature outside changes between -7 and 6. In the responses from the LLMs, it is evident that the agents acknowledge the need for heating and consider other people’s needs. Hence, despite their cold or cool preferences, the agents may suggest a degree closer to the neutral range, going with the higher limit of their preference category. An example of this situation is given in Fig.

5.10. A similar trend can be followed for the hot preference in Model 1. Moreover, the positive sign of the coefficient for warmer degree preferences and the negative sign of the coefficient for the cool preference align with the logical expectation.

5.3 Discussion

In this chapter, we employed the Mistral 7B model to generate agent decisions for its efficiency and ease of use. Using larger and stronger models is expected to affect the decisions made by the agents, and hence the overall results. Future work would include doing a comparative analysis of utilizing different models and investigating how the social interactions take place with respect to them.

In the LLM-based generation of human daily activities and occupancy data for building energy modeling, the studies focus on hourly actions of the agents [65, 66]. The interactions of agents and the energy-consuming appliances are also calculated and visualized on an hourly basis. For instance, the heating and cooling systems are only turned on when there is at least a single family member at home. This approach, combined with the inclusion of the average cost of using certain appliances, allows for more realistic simulations. As our focus in this study is to show the impact of personality traits, preferences, and social network structure on the temperature decisions, we simplify the simulations by making each iteration represent a day, and by setting the cost of increasing the temperature of the building by a degree to a value of 1. By simply changing the value of the constant C in Eq. 5.1, researchers can achieve more realistic results for the given scenario.

In addition, expanding the scope of the simulation to include more iterations, locations with varying climates, and attributes of family structure could lead to valuable insights. The importance of family structures and culturally sensitive activity generation is explored in [65], which can also be extended to our simulation scenario. Our current simulation model also assumes no communication or friendship between family members from different households, which limits the

realistic representation of a community.

In [67], Park et al. show that simulation with generative agents also exhibits emergent behavior such as information diffusion. Therefore, the various studies focusing on the diffusion of energy-saving behavior over social networks and the effects of public interventions on such cases (e.g., [68, 69]) can be extended with generative agent-based modeling.



Chapter 6

Discussion

6.1 Modeling Other Social Network Problems With Crowd

With the case studies presented in the preceding two chapters, we aimed to demonstrate how Crowd can be useful for different research areas in the field of social networks. In this chapter, we briefly address how Crowd can also be applied to other social network problems.

For instance, from a company’s perspective to select influencers for product marketing [70] or from a regular user’s perspective to analyze methods to become an influencer [71], engagements of posts on social media platforms can provide valuable insights. To model relationships such as followers, retweets, and replies in a network, one option is to utilize multiple networks, where each user will have the same node ID across all networks. Alternatively, a single network can be used with different types of edges, which can be represented using edge parameters in Crowd. For example, if person 1 retweets person 2’s tweets five times, the “retweet” parameter of the edge between them will be set to 5.

Besides numerical parameters, Crowd also supports categorical parameters.

To represent relationships between users, an edge parameter such as “relationship” can be used, holding values like “colleague,” “family,” or “friend.” These parameters—both numerical and categorical—can be used to model the level of influence in trend analysis or misinformation propagation studies.

Categorical node parameters can include information such as “language,” “location,” or “gender,” which can be extracted from user profiles in online social networks. Numerical node parameters can be used to store values such as “age” (for epidemic simulations), “payoff” (for cooperative games), or “cluster ID” (after running community detection algorithms).

6.2 Comparison With a General-Purpose Framework for the Three Case Studies

To demonstrate the use of Crowd in comparison with existing general-purpose frameworks, this section provides a direct comparison with *Mesa* (v3.1.4) [16] for the three case studies presented in Chapter 4 of this thesis. The comparison focuses on the required effort for modeling, visualization facilities, and execution times. Mesa is chosen as the baseline for comparison as it is the most widely used Python-based ABMS framework. Since Crowd is designed for network-based simulations, the comparison is limited to experiments that utilize network environments.

6.2.1 Modeling Stage

Most ABMS frameworks require the user to implement Model and Agent classes. In the Model class, the network environment is initialized first, followed by setting up model parameters, creating agents, and specifying data collection settings. In Mesa, these steps must be explicitly defined in the code. In Crowd, however, these configurations are specified in YAML files or selected via the GUI, eliminating

the need for additional coding.

Moreover, in the `Model` class of Mesa, users create a *run* method, which calls a self-defined *step* function for a specified number of iterations. Within the *step* function, both agent-level and model-level methods execute the simulation logic, followed by data collection.

While this setup is not hard to implement, it is repetitive across simulations. Crowd automates this process by providing a *run_simulation* method within the `Project` class. Users can specify the number of iterations, snapshot periods, parameters for model exploration, and user-defined functions to execute at specific intervals. Unlike Mesa, data collection methods do not need to be explicitly passed to a data collector. If a method returns a value, Crowd automatically saves the results in a file. Further details on this process are discussed in Section 3.1.1, with an example in Section 3.2.3.

Additionally, Mesa does not provide methods to save the state of the graph or the collected data. Users must implement this manually to analyze the results of the simulation later or visualize the network using other tools. In Crowd, the graph is saved in the selected file formats at each snapshot period, and the collected data is saved at the end of the simulation.

6.2.2 Execution Times

We conduct the experiments under the following conditions: Each case study is implemented in both frameworks as closely as possible, and each experiment is repeated 5 times. The average execution times are presented in Tables 6.1, 6.2, and 6.3, with the better execution time shown in bold.

The first experiment was implemented on Google Colab and run on an L4 GPU. Due to the overhead introduced by LLM inference, a significant increase in the number of agents results in additional hours of runtime. Therefore, we conduct our experiments with 10, 50, and 100 agents. With each query to the

Table 6.1: Execution times of the generative agents case study in Mesa and Crowd frameworks.

Tool/Nodes	10	50	100
Mesa	25.52 min	127.9 min	249.33 min
Crowd	24.3 min	122.47 min	244.66 min

Table 6.2: Execution times of the influence maximization case study in Mesa and Crowd frameworks on Facebook, GitHub, and Twitch social network datasets.

Tool/Dataset	Facebook (4K)	GitHub (37K)	Twitch (168K)
Mesa	7.88 s	46.15 s	19.57 min
Crowd	4.76 s	24.23 s	10.97 min

LLM taking around 7.5 seconds on average, we complete a single experiment of 20 days within the times provided in Table 6.1. Since the number of nodes is relatively small, saving the network state in each iteration does not result in a significant overhead. Therefore, we save the network state in every iteration.

For the second and third case studies, we save the graph data only in the first and last iterations, with the data collector’s results being saved at the end of the simulation. These experiments are conducted on a system with an Intel i5-10210U CPU (1.60 GHz) and 32 GB RAM.

For the second case study, influence maximization, we run our experiments for 30 iterations on three real-world networks from Facebook (4039 nodes, 88,234 edges), GitHub (37,700 nodes, 289,003 edges) [72], and Twitch (168,114 nodes, 6,797,557 edges) [73]. We demonstrate the change in the execution times as the number of nodes and edges increase in Table 6.2. It is important to note that the significant increase in the number of edges from the GitHub dataset to the Twitch dataset also contributes to the execution time.

For the third case study, unlike in Section 4.1.2, where the temptation to defect ratio is varied, here we run the experiments for 6000 iterations with data from the same study (Scale-Free networks generated with the Barabasi-Albert model with $m = 3$) but with different network sizes. The execution times for experiments with 1024, 4900, and 14400 nodes are given in Table 6.3.

Table 6.3: Execution times of the networked trust game case study in Mesa and Crowd frameworks

Tool/Nodes	1024	4900	14400
Mesa	31.78 s	176.45 s	11.2 min
Crowd	29.32 s	141.63 s	8.68 min

As shown in Tables 6.1, 6.2, and 6.3, both frameworks perform similarly for these scenarios, while Crowd takes less time in most simulations. This illustrates that Crowd’s additional simplifications do not slow down the execution and can be preferred for Python-based simulations without any performance drawback.

6.2.3 Visualization Stage

Mesa uses Python-based tools for visualization, allowing users to customize the visualization components through code. With this approach, modelers have more flexibility in describing the appearance of graphs and charts. Crowd provides visualization primarily through its GUI, which does not require any coding. It utilizes JavaScript libraries and provides more interactive graphs and charts, allowing operations like zooming, dragging nodes and chart lines, and viewing node and link IDs.

Chapter 7

Conclusion

In this thesis, we presented *Crowd*, an open-source Python framework specifically developed to facilitate the modeling, simulation, and analysis of social networks. The network, agents, parameters, and various simulation settings are defined with a configuration file, allowing no-code diffusion modeling and a lower-entry barrier for social simulations. Further customization is enabled through user-defined Python methods that are executed in the time specified by the modeler. By focusing only on writing code for the specifics of their study and not requiring to handle the basics of simulation and visualization, researchers can quickly and simply address their research questions.

In addition to a Python library, we provide a graphical user interface to create projects and simulations, visualize the network saved in each step of the simulation, draw charts, and aggregate results. Taking advantage of the Python ecosystem, the researchers can integrate any machine learning or data analysis libraries within their simulation and utilize novel methodologies, such as generative agents for more realistic modeling of human behavior compared to rule-based approaches.

Additionally, we presented Crowd's applications in epidemiology, influence

maximization, evolutionary games, and building energy modeling. In our simulation of building energy usage, we integrate generative agents with social networks to simulate personality-driven social dynamics. Our results show that in the network-level, happiness of agents is positively impacted by the friendship weights and the weather outside. In the node-level, Assertiveness and Selflessness are the most significant personality traits, while temperature preferences notably impact the degree suggestions and happiness levels.

In future work, we aim to expand Crowd’s network generation capabilities with machine learning-based methods to synthesize data-driven network structures with user-specified requirements. Moreover, we plan to integrate an LLM-based code generator into the framework to assist both non-expert and experienced users with user-implemented agent and data collection methods. With these features, we intend to provide novel features to the agent-based and social network simulation spaces.

In the current version of Crowd, only single-threaded execution of simulations is allowed. Utilizing Python’s *multiprocessing* module, we aim to add parallel processing to the framework and reduce the time needed for parameter sweeps by executing multiple simulations simultaneously. To further enhance Crowd’s performance, we plan to experiment with compiling the library to C utilizing tools such as Nuitka. Additionally, we plan to add more scheduling options for custom network simulations and expand the library with additional visualization features to further extend the framework’s modeling and analysis capabilities.

Furthermore, we plan to add facilities to Crowd to support the validation of simulation results, such as formatters for real-world data to be included in the list of data files to draw charts in the GUI, or simplified integration into post-simulation functions to perform desired computations with both data. We plan to test these functionalities with real-life datasets that provide information over time.

Finally, we aim to establish a GitHub repository to share data and models that can be used within Crowd, supported by contributions from both our research

group and the broader community. Feedback and participation in this repository will support the ongoing development and enhancement of Crowd.



Bibliography

- [1] Q. Zhang, J. Liu, B. Liu, and G. Wang, “Market adoption simulation of electric vehicle based on social network model considering nudge policies,” *Energy*, vol. 259, 2022, pp. 124984.
- [2] M. A. Zaffar, R. L. Kumar, and K. Zhao, “Using agent-based modelling to investigate diffusion of mobile-based branchless banking services in a developing country,” *Decis. Support Syst.*, vol. 117, 2019, pp. 62–74.
- [3] R. Hou, J. Wu, and H. S. Du, “Customer social network affects marketing strategy: a simulation analysis based on competitive diffusion model,” *Physica A*, vol. 469, 2017, pp. 644–653.
- [4] N. Stroeymeyt, A. V. Grasse, A. Crespi, D. P. Mersch, S. Cremer and L. Keller, “Social network plasticity decreases disease transmission in a eusocial insect,” *Science*, vol. 362, no. 6417, 2018, pp. 941–945.
- [5] P. Block et al., “Social network-based distancing strategies to flatten the COVID-19 curve in a post-lockdown world,” *Nature Human Behav.*, vol. 4, no. 6, 2020, pp. 588–596.
- [6] P. Lv et al., “Agent-based campus novel coronavirus infection and control simulation,” *IEEE Trans. Comput. Social Syst.*, vol. 9, no. 3, 2021, pp. 688–699.
- [7] B. Ross, L. Pilz, B. Cabrera, F. Brachten, G. Neubaum and S. Stieglitz, “Are social bots a real threat? An agent-based model of the spiral of silence to analyse the impact of manipulative actors in social networks,” *Eur. J. Inf. Syst.*, vol. 28, no. 4, 2019, pp. 394–412.

- [8] Y. Gao and Z. Zhang, “Consensus reaching with non-cooperative behavior management for personalized individual semantics-based social network group decision making,” *J. Oper. Res. Soc.*, vol. 73, no. 11, 2021, pp. 2518–2535.
- [9] A. Devari, A. G. Nikolaev and Q. He, “Crowdsourcing the last mile delivery of online orders by exploiting the social networks of retail store customers,” *Transp. Res. E*, vol. 105, 2017, pp. 105–122.
- [10] A. Trivedi, S. Rao, “Agent-based modeling of emergency evacuations considering human panic behavior,” *IEEE Trans. Comput. Social Syst.*, vol. 5, no. 1, 2018, pp. 277–288.
- [11] C. M. Macal, “Everything you need to know about agent-based modelling and simulation,” *J. Simul.*, vol. 10, no. 2, 2016, pp. 144–156.
- [12] A. Antelmi, G. Cordasco, G. D’Ambrosio, D. De Vinco and C. Spagnuolo, “Experimenting with agent-based model simulation tools,” *Appl. Sci.*, vol. 13, no. 1, 2022, pp. 13.
- [13] S. Tisue and U. Wilensky, “Netlogo: A simple environment for modeling complexity,” in *Proc. Int. Conf. Complex Syst.*, 2004, vol. 21, pp. 16–21.
- [14] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, “Mason: A multiagent simulation environment,” *Simul.*, vol. 81, no. 7, 2005, pp. 517–527.
- [15] M. J. North et al., “Complex adaptive systems modeling with Repast Symphony,” *Complex Adaptive Syst. Model.*, vol. 1, 2013, pp. 1–26.
- [16] D. Masad and J. L. Kazil, “Mesa: an agent-based modeling framework,” in *SciPy*, 2015, pp. 51–58.
- [17] G. Rossetti, L. Milli and S. Rinzivillo, “NDlib: a python library to model and analyze diffusion processes over complex networks,” in *Companion Proc. Web Conf.*, 2018, pp. 183–186.

- [18] S. Abar, G. K. Theodoropoulos, P. Lemarinier and G. M. O'Hare, "Agent based modelling and simulation tools: a review of the state-of-art software," *Comput. Sci. Rev.*, vol. 24, 2017, pp. 13–33.
- [19] J. Chen and U. Wilensky, "ChatLogo: a large language model-driven hybrid natural-programming language interface for agent-based modeling and programming," 2023, *arXiv:2308.08102*.
- [20] M. Resnick, "StarLogo: an environment for decentralized modeling and decentralized thinking," in *Conf. companion on Human factors in computing systems*, pp. 11-12, 1996.
- [21] J. Foramitti, "AgentPy: a package for agent-based modeling in Python," *J. Open Source Softw.*, vol. 6, no. 62, 2021, pp. 3065.
- [22] K. Sullivan, M. Coletti and S. Luke, "GeoMason: geospatial support for MASON," Dept. of Computer Science, George Mason Univ., Fairfax, VA.
- [23] R.D. Chiara, A. Mancuso, D. Mazzeo, V. Scarano and C. Spagnuolo, "Bringing together efficiency and effectiveness in distributed simulations: the experience with D-MASON," *Simulation*, vol. 89, no. 10, 2013, pp. 1236-1253.
- [24] N. Collier and M. North, "Repast HPC: A platform for large-scale agent-based modeling", *Large-Scale Comput.*, 2011, pp. 81–109.
- [25] N. Collier and J. Ozik, "Distributed agent-based simulation with Repast4Py," in *Proc. Winter Simul. Conf. (WSC)*, 2022, pp. 192–206.
- [26] A. Chopra et al., "DeepABM: scalable and efficient agent-based simulations via geometric learning frameworks-a case study for COVID-19 spread and interventions," in *Winter Simul. Conf. (WSC)*, 2021, pp. 1–12.
- [27] G. Datseris, A. R. Vahdati and T. C. DuBois, "Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity," in *Simul.*, 2022.
- [28] A. Antelmi et al., "Reliable and efficient agent-based modeling and simulation," *J. Artif. Soc. Social Simul.*, vol. 27, no. 2, 2024.

- [29] D. Blanco-Moreno, R. Fuentes-Fernández and J. Pavón, “Simulation of on-line social networks with Krowdix,” in *2011 Int. Conf. on Comput. Aspects of Social Netw. (CASoN)*, IEEE, 2011, pp. 13-18.
- [30] K. Ryczko, A. Domurad, N. Buhagiar and I. Tamblyn, “Hashkat: large-scale simulations of online social networks,” *Social Netw. Anal. Mining*, vol. 7, 2017, pp. 1–13.
- [31] X. Ye, L. Dang, J. Lee, M. H. Tsou and Z. Chen, “Open source social network simulator focusing on spatial meme diffusion,” in *Human Dyn. Res. Smart and Connected Communities*, 2018, pp. 203-222
- [32] J. M. Sánchez, C. A. Iglesias and J. F. Sánchez-Rada, “Soil: An agent-based social simulator in Python for modelling and simulation of social networks,” in *Proc. PAAMS*, 2017, pp. 234–245.
- [33] C. C. Kerr et al., “Covasim: an agent-based model of COVID-19 dynamics and interventions,” *PLOS Comput. Biol.*, vol. 17, no. 7, 2021, pp. e1009149.
- [34] I. Mahmood et al., “FACS: A geospatial agent-based simulator for analysing COVID-19 spread and public health measures on local regions,” *J. Simul.*, vol. 16, no. 4, 2022, pp. 355-373.
- [35] IDMOD, “SynthPops: Synthetic Population Data,” GitHub repository, Available: <https://github.com/InstituteForDiseaseModeling/synthpops>, Accessed: Jun. 8, 2025.
- [36] T. Yilmaz, “Towards modeling and mitigating misinformation propagation in online social networks,” Ph.D. dissertation, Dept. Comp. Eng., Bilkent Univ., Ankara, Turkey, 2023.
- [37] W. O. Kermack, and A. G. McKendrick, “A contribution to the mathematical theory of epidemics,” in *Proc. of the Roy. Soc. of London, Series A, Containing papers of a mathematical and physical character*, vol. 115, no. 772, pp. 700-721, 1927.

- [38] A. Hagberg, P. J. Swart and D. A. Schult, “Exploring network structure, dynamics, and function using NetworkX,” Los Alamos Nat. Lab. (LANL), Los Alamos, NM, USA, Rep. LA-UR-08-5495, 2008.
- [39] G. Csardi and T. Nepusz, “The igraph software,” *Complex syst*, vol. 1695, pp. 1-9, 2006.
- [40] R. Williams, N. Hosseinichimeh, A. Majumdar and N. Ghaffarzadegan, “Epidemic modeling with generative agents,” 2023, *arXiv:2307.04986*.
- [41] J. McAuley and J. Leskovec, “Learning to discover social circles in ego networks,” in *Proc. Neural Inf. Process. Syst. (NIPS)*, 2012.
- [42] M. Chica, R. Chiong, M. Kirley and H. Ishibuchi, “A networked N -player trust game and its evolutionary dynamics,” in *IEEE Trans. Evol. Comput.*, vol. 22, no. 6, pp. 866-878, 2017.
- [43] L. Wang, et al., “A survey on large language model based autonomous agents,” *Frontiers of Comput. Sci.*, vol. 18, no. 6, pp. 186345, 2024.
- [44] J. Lin, H. Zhao, A. Zhang, Y. Wu, H. Ping and Q. Chen, “Agentsims: An open-source sandbox for large language model evaluation,” 2023, *arXiv:2308.04026*.
- [45] C. Gao, et al., “S³: social-network simulation system with large language model-empowered agents,” 2023, *arXiv:2307.14984*.
- [46] OpenAI, “ChatGPT 3.5 Turbo,” Accessed: Jun. 8, 2025. Available: <https://platform.openai.com/docs/models/gpt-3.5-turbo>
- [47] A. Q. Jiang et al., “Mistral 7B,” 2023, *arXiv:2310.06825*.
- [48] A. S. Vezhnevets, et al., “Generative agent-based modeling with actions grounded in physical, social, or digital space using Concordia,” 2023, *arXiv:2312.03664*.
- [49] Y. Li, J. Fan, Y. Wang and K. L. Tan, “Influence maximization on social graphs: a survey,” in *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 10, pp. 1852-1872, 2018.

- [50] D. Kempe, J. Kleinberg, and E. Tardos, “Maximizing the spread of influence through a social network,” in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pp. 137-146, 2003.
- [51] M. Granovetter, “Threshold models of collective behavior,” *Amer. J. Sociology*, vol. 83, no.6, 1978, pp. 1420-1443.
- [52] T. C. Schelling, *Micromotives and Macrobehavior*, New York, NY, USA: Norton, 1978.
- [53] J. Leskovec and A. Krevl, “SNAP datasets: Stanford large network dataset collection,” 2014. Accessed: Jun. 8, 2025. [Online]. Available: <https://snap.stanford.edu/data>
- [54] G. Szabó and G. Fáth, “Evolutionary games on graphs,” *Phys. Rep.*, vol. 446, nos. 4–6, pp. 97–216, 2007.
- [55] H. Ohtsuki, C. Hauert, E. Lieberman, and M. A. Nowak, “A simple rule for the evolution of cooperation on graphs and social networks,” *Nature*, vol. 441, pp. 502–505, 2006.
- [56] Y. Du, J. Z. Leibo, U. Islam, R. Willis, and P. Sunehag, “A review of cooperation in multi-agent learning,” 2023, *arXiv:2312.05162*.
- [57] K. Kaspar et al., “Effects of occupant thermostat preferences and override behavior on residential demand response in CityLearn,” *Energy and Buildings*, vol. 324, pp. 114830, 2024.
- [58] X. Liu, Q. C. Wang, I. Y. Jian, H. L. Chi, D. Yang, and E. H. W. Chan, “Are you an energy saver at home? The personality insights of household energy conservation behaviors based on theory of planned behavior,” *Resour., Conservation and Recycling*, vol. 174, pp. 105823, 2021.
- [59] W. W. Zachary, “An information flow model for conflict and fission in small groups,” *J. of Anthropological Res.*, vol. 33, no. 4, pp. 452-473, 1977.
- [60] P. T. Costa Jr, and R. R. McCrae, “Domains and facets: Hierarchical personality assessment using the Revised NEO Personality Inventory,” *J. of Personality Assessment*, vol. 64, no. 1, pp. 21-50, 1995.

- [61] A. Jensen, K. Anderson, W. Holmgren, M. Mikofski, C. Hansen, L. Boeman, R. Loonen, “pvlib iotools — Open-source Python functions for seamless access to solar irradiance data,” *Solar Energy*, vol. 266, pp. 112092, 2023.
- [62] OpenAI, “ChatGPT,” Available: <https://chatgpt.com/>, Accessed: Jun. 8, 2025.
- [63] J. Martí, M. Bolívar, and C. Lozares, “Network cohesion and social support,” *Social Netw.*, vol. 48, pp. 192-201, 2017.
- [64] Y. Mundlak, “On the Pooling of Time Series and Cross Section Data.” *Econometrica*, vol. 46, no. 1, pp. 69–85, 1978.
- [65] M. Takroui, N. M. Cuadrado, and M. Takáč, “Knowledge distillation from large language models for household energy modeling,” 2025, *arXiv:2502.03034*.
- [66] M. Almashor, Y. Miyashita, S. West, and T. V. D. Dong, “Can Private LLM Agents Synthesize Household Energy Consumption Data?” in *Proc. of the 15th ACM Int. Conf. on Future and Sustain. Energy Syst.*, pp. 664-668, 2024.
- [67] J. S. Park, J. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M.S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” in *Proc. of the 36th Annu. ACM Symp. on User Interface Softw. and Technol.*, pp. 1-22, 2023.
- [68] H. Li, Z. H. Wang, and B. Zhang, “How social interaction induce energy-saving behaviors in buildings: Interpersonal & passive interactions vs public & active interactions,” *Energy Econ.*, vol. 118, pp. 106515, 2023.
- [69] M. Yang, H. Chen, R. Long, and J. Yang, “The impact of different regulation policies on promoting green consumption behavior based on social network modeling,” *Sustain. Prod. and Consumption*, vol. 32, pp. 468-478, 2022.
- [70] R. R. Mallipeddi, S. Kumar, C. Sriskandarajah, and Y. Zhu, “A framework for analyzing influencer marketing in social networks: selection and scheduling of influencers,” *Manage. Sci.*, vol. 68, no. 1, pp. 75-104, 2022.

- [71] A. Adesokan, M. S. Siraj, A. B. Rahman, E. E. Tsiropoulou, and S. Papavasiliou, “How to become an influencer in social networks,” in *ICC 2023-IEEE Int. Conf. on Commun.*, pp. 5570-5575, 2023.
- [72] B. Rozemberczki, C. Allen, and R. Sarkar, “Multi-scale attributed node embedding,” 2019, *arXiv:1909.13021*.
- [73] B. Rozemberczki, and R. Sarkar, “Twitch gamers: a dataset for evaluating proximity preserving and structural role-based node embeddings,” 2021, *arXiv:2101.03091*.