MultiRepast4py: A Framework for Agent Based Simulations on Multilayer Networks

Keng-Lien Lin
University of California, San Diego
San Diego, United States
kel030@ucsd.edu

Parinaz Naghizadeh
University of California, San Diego
San Diego, United States
pnaghizadeh@ucsd.edu

ABSTRACT

Agent-based Simulations (ABS) offer a powerful approach for analyzing how individual agents' decisions and interactions within networked systems lead to system outcomes. ABS have been widely used across various fields, including for the simulation of disease spread, misinformation, and population dynamics. However, traditional ABS platforms, such as NetLogo and Repast, simplify models by assuming a single network of interactions between agents. In reality, agents' interactions are typically multi-layered (i.e., involve multiple interconnected networks that influence agents' decisions). To address this limitation, we have developed `MultiRepast4py`, a multilayer simulation tool that extends Repast4py's core simulation capabilities. While Repast4py excels at large-scale simulations, it, like other simulation platforms, lacks multilayer functionalities. Our framework bridges this gap by integrating MultinetX (a multilayer network generation package), together with agent identification and synchronization techniques, to enable agent-based simulations across multiple network layers. Through simulation examples, we showcase how `MultiRepast4py` can enable more comprehensive agent-based simulations, guiding better predictions and interventions in areas such as opinion dynamics and epidemiology.

KEYWORDS

Agent Based Simulations, Multilayer Networks, Repast.

ACM Reference Format:
Keng-Lien Lin and Parinaz Naghizadeh. 2025. MultiRepast4py: A Framework for Agent Based Simulations on Multilayer Networks. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 8 pages.

1 INTRODUCTION

Agent-based models and simulations have become a widely used computational approach to modeling the behavior and interactions of autonomous agents, and understanding how these translate into more complex, system-level outcomes [21]. In an agent-based simulation (ABS), agents are first endowed with a set of attributes and decision-making heuristics. The simulation then allows these agents to interact repeatedly over time, with their interactions governed by an *interaction topology* (referred to here as a *network*). By running large-scale experiments in a cost-effective way, ABS enable researchers to analyze how various "micro"-scale assumptions on agents (e.g., their movement patterns, their propensity to catch/pass a disease, or adopt an opinion), as well as different interventions that can change agents' behavior or their network (e.g., limiting agents' exposure to content on social networks, or isolation/vaccination strategies), can (collectively) alter the "macro" outcomes (e.g., spread of a disease or misinformation).

There currently exist several commonly used simulations platforms for ABS, built on different programming languages and offering different levels of customization; these include NetLogo [30], Repast [8], MASON [20], FLAME [17], and Swarm [22]; we refer the interested reader to [24] for a comparison of (some of) these platforms. The versatility of ABS and the accessibility of these platforms has led to the wide use of ABS by researchers in many fields (e.g., biology, ecology, and across social sciences, including in economics, political science, and sociology). For instance, ABS have been leveraged to gain insights into COVID-19 dynamics and interventions [16], healthcare resource management [7], spatio-temporal dynamics of crime [26], supply chain management [15], and policy design in labor markets [10]. The capabilities of these platforms have also been extended in several directions, including by integrating micro-findings from human subject experiments into agents' decision models [29], advocating for data-driven ABS [25], and developing strategies for scaling-up to accommodate data-intensive simulations [4].

The gap. Despite their widespread use and recent advances, existing ABS platforms assume that agents' interactions occur within a *single* interaction topology/network (e.g., they account for the spread of misinformation over one social network, or the spread of a disease over one grid). However, there exist many settings, including in the study of disease spread and information dynamics, where this approach overlooks the complexities of intersectionality of humans - how an individual's experiences and outcomes can be shaped by their multiple social identities and networks. For instance, individuals with accounts on two social media platforms (e.g., TikTok and Instagram) follow, and are followed by, different accounts on each platform, can cross-post the same content on both platforms, and may further use these platforms at differing frequencies and for different purposes; studying the spread of (mis)information on any one of these platforms in isolation would fail to capture these nuances. Similarly, a family network and a workplace network may differ in size, interaction frequency, and types of interactions, yet both can significantly influence an individual's exposure to a disease; isolation or vaccination policies that do not explicitly account for these differences may therefore be suboptimal.

Multilayer networks. For the situations discribed above, and other similar contexts, *multilayer networks* have been proposed as a model

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), A. El Fallah Seghrouchni, Y. Vorobeychik, S. Das, A. Nowe (eds.), May 19 – 23, 2025, Detroit, Michigan, USA.

to simultaneously account for the multiple modalities of interactions between agents; see [1, 5, 18, 27] for surveys of this field. The study of multilayer networks, as opposed to the study of their constituent *single-layer* networks in isolation, can offer a nuanced understanding of how the "micro" differences between the various modalities of interactions (ranging from the agents' attributes on each network, to the differences in each network's topology, to the frequency with which agents are actively interacting with others in each network) impact macro outcomes. Existing works have used the formalism of multilayer networks to study game theoretical decision making over multiple interaction/information modalities (e.g., [3, 11, 13, 28]), to evaluate the resilience of networks of networks against failures or attacks, often by studying percolation (e.g., [5, 6, 12, 23, 32–34]), and to analyze dynamical processes on interacting networks (such as diffusion and spreading processes) to identify the critical thresholds for an outbreak (in epidemic modeling) or consensus (in the study of opinion dynamics) on these networks (e.g. [5, 31, 35]). Despite the wide applicability of multilayer network models and the growing research on them, to the best of our knowledge, there currently exists no agent-based simulation platform for multilayer networks, limiting existing and potential future works to (smaller-scale) custom simulation environments; the technical expertise required to develop these in turn limits the wider adoption of these tools across disciplines.

Our contributions. To address these limitations, we have developed `MultiRepast4py`, a multilayer agent-based simulation framework. This framework builds on an existing ABS simulation platform (specifically, Repast4py [8]), enabling researchers to model complex, multilayered interactions without needing extensive programming skills, making it accessible to researchers across various disciplines. The framework's design maintains Repast4py platform's flexibility and scalability, which can enable leveraging High-Performance Computing (HPC) resources for large-scale (multilayer) simulations. It is also prepared for integration with data-driven approaches, allowing researchers to easily incorporate real-world data into their models. Researchers can define and implement custom inter-layer interaction rules. In more detail, our framework makes the following key contributions:

(1) **Multilayer Agent-Based Simulation Capability:** `MultiRepast4py` introduces the ability to run Multilayer Agent-Based Simulations by building on the established Repast4py platform. This bridges a crucial gap in ABS technology, allowing for more nuanced and realistic modeling of complex systems where agents interact across multiple interconnected networks. This advancement can be particularly crucial in fields such as epidemiology and social sciences, where interactions often occur across various contexts or environments simultaneously.
(2) **Scalability and Flexibility:** `MultiRepast4py` retains all the features that make the Repast suite powerful, including its renowned scalability and flexibility. This ensures that researchers can tackle large-scale, complex simulations without sacrificing computational efficiency.
(3) **A Graphical Interface:** Going beyond the capabilities offered by Repast4py, we have equipped `MultiRepast4py` with a graphical interface, which lets users track the evolution of the multilayer ABS over time steps. (This is similar to capabilities offered by platforms such as NetLogo).
(4) **Opportunities for Customization:** `MultiRepast4py` is developed in Python, and remains open to additional features and enhancements. This includes the potential for incorporating data-driven models, further increasing the framework's relevance and applicability in an era of big data.

Use cases. To illustrate potential use cases for `MultiRepast4py`, this paper presents two simulation studies: the spread of rumors across different social networks, and the spread of a disease when agents have differnet interaction modalities. In the rumor spreading case, the layers represent different social media platforms, each with unique network structures and usage patterns (reflected in differing interaction frequencies). In the second case study, the two grids represent weekdays (e.g. workplace networks) and weekends (e.g., friends and family networks), with varying grid sizes leading to differences in population density, agent travel distances, and interaction frequencies. These adjustable parameters, which can vary across layers, are not feasible in a single-layer network simulation; not accouting for them could lead to oversimplified or inaccurate results. In certain scenarios, such limitations could have severe consequences —misinformation might incite social unrest, and ineffective quarantine policies could cost lives. The multilayer approach offers a more realistic and nuanced modeling framework, allowing for deeper insights and more reliable outcomes in these critical contexts.

Paper organization. We begin by providing an overview of multilayer network models in Section 2. In Section 3, we describe `MultiRepast4py`, providing the rationale behind our choices for the platform, our framework's main components and their implementation details, and the main challenges addressed when developing it. We illustrate `MultiRepast4py` through simulations of rumor spreading across multiple social networks, and the spread of diseases across multiple interaction modalities, in Section 4. We conclude with potential directions of future work in Section 5.

2 MULTILAYER NETWORKS

We model *multilayer networks* as structures consisting of multiple *single-layer networks* connected together, with each layer corresponding to a particular type of social or economic relation, a mode of interaction, or an information channel, between the agents.

That is, each layer α of a multilayer network is itself a network, represented by a graph $\mathcal{G}^\alpha =< \mathcal{N}^\alpha, \mathcal{A}^\alpha >$. In this graph, $\mathcal{N}^\alpha$ denotes the set of agents in layer α and $\mathcal{A}^\alpha$ denotes the *intra-network* adjacency matrix. An agent $m \in \mathcal{N}^\alpha$ could be, e.g., an individual in a social network, or a firm in a market. An edge $a^\alpha_{mn} \in \mathcal{A}^\alpha$ represents the dependency between agents m and n in $\mathcal{G}^\alpha$, and can capture, e.g., the exchange of information (in-person or virtual), friendships, or a business agreement. Network $\mathcal{G}^\alpha$ is in general directed (allowing for unilateral dependencies) and weighted (to capture different strengths of dependencies). In this work, we focus on undirected (but still weighted) interactions in each layer.

In addition, as these layers do not operate in isolation, there exist some connections between the nodes in different layers, captured using an *inter-network* adjacency matrix $\mathcal{B}^{\alpha,\beta} \in \mathbb{R}^{\mathcal{N}^\alpha \times \mathcal{N}^\beta}$. An edge

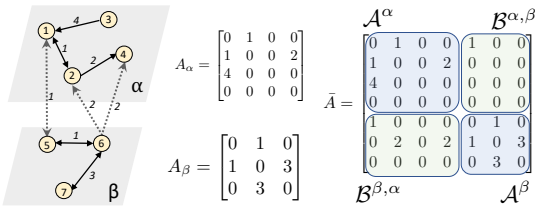

Figure 1: An example of a multilayer network, and its intra-layer and inter-layer adjacency matrices.

$b_{mn}^{\alpha,\beta} \in \mathcal{B}^{\alpha,\beta}$ indicates that the decisions made by agent m in $\mathcal{G}^\alpha$ are linked to those of agent n in $\mathcal{G}^\beta$. In this work, we focus on the case of an *identity* inter-network adjacency matrix. These matrices capture a special case of multilayer networks in which the different layers consist of the *same* set of nodes/agents, but where the nature of the relation between the nodes being different in each layer; these are also sometimes referred to as *multiplex networks* in the literature [5]. Multiplex networks are primarily used when agents have access to different communication or interaction modalities. Examples include the spread of social influence campaigns between social networks (e.g. Twitter in layer α and Facebook in layer β), or the spread of diseases as individuals interact with others in both their family (layer α) and work (layer β) networks.

An example of multilayer networks is shown in Figure 1. As illustrated in the figure, the inter-network and intra-network adjacency matrices can be collected into a single "supra-adjacency" matrix $\bar{A}$. One might then propose that the interactions of agents can be viewed as happening over a single-layer network with adjacency matrix $\bar{A}$. We note, however, that the multilayer network is different from this single-layer network with adjacency matrix $\bar{A}$. First, a multilayer model can impose different structural properties on the adjacency matrices in each layer, and enables us to investigate their impact on emergent phenomena accordingly. For example, real-world data can be used to learn the different structural properties of two social networks separately, and allow for each to be reflected independently in the ABS environment. Further, disturbances or information from a node may spread within each layer following a different process, and at a different time scale. For instance, individuals may interact with their co-workers primarily during the week, and with their extended family and friend group primarily over the weekend; they may also engage in different activities with each group, therefore impacting the likelihood of the spread of a disease between them in each context differently. Lastly, multilayer network simulations allow us to distinguish how regulators can propose and enact interventions in each layer.

3 `MULTIREPAST4PY`: OUR PROPOSED MULTILAYER ABS FRAMEWORK

Given the limitations of traditional ABS platforms and the importance of incorporating multilayer functionalities, our objective is to create ABS tools that enable agent-based simulations over multilayer networks. In this section, we first outline our evaluation of existing simulation platforms to identify those best suited for developing and implementing a multilayer ABS framework (Section 3.1). We then provide details on our implementation of `MultiRepast4py`, and outline the main challenges that were addressed (Section 3.2).

3.1 Platform Selection and Rationale

Existing ABS platforms. As noted in the introduction, there exist several widely used platforms for ABS, built on different programming languages; these include NetLogo [30], Repast [8], MASON [20], FLAME [17], and Swarm [22], and their extensions. Among these, NetLogo is the highest-level platform, with a relatively simple programming language and a well-developed graphical interface; however, while NetLogo offers robust features for certain applications (e.g., grid-based ABMs), it presents limitations for highly specialized or complex simulations. The remaining platforms (e.g., Repast, MASON, Swarm) provide a *framework* (a set of concepts for describing an agent-based model and the required simulation components) along with a *software library* (implementations of the framework using customized simulation tools). While these require additional programming skills, they facilitate customization, as well as integration with a wide range of data analysis and machine learning libraries to enable data-driven ABS. Our focus in this work is similarly on developing a framework and library for multilayer ABS, while maintaining the flexibility for customization and data-driven implementations.

Repast4py [9]. In selecting a platform for our multilayer ABS framework, we initially considered both high-level ABS platforms (specifically, NetLogo) and more flexible, programmable solutions (such as Repast). Our requirements necessitated a framework that could support large-scale models, offer dynamic simulation capabilities, and facilitate data-driven approaches. Repast4py, a Python-based member of the Repast Suite, was chosen as the development platform for the following reasons:

(1) *Scalability*: Repast4py simplifies the construction of large-scale ABMs that can be distributed across multiple processing cores using MPI, enabling efficient execution of complex simulations.
(2) *Flexibility*: The platform's dynamic simulation step capabilities, facilitated by the scheduling feature, enables a high degree of customization, ensuring the seamless implementation of our multilayer approach.
(3) *Extensibility*: Built on Python, Repast4py inherently supports integration with a wide range of data analysis and machine learning libraries, facilitating data-driven modeling and future expansions.

Repast4py provides a foundation for future scalability and adaptability. This choice enables researchers to extend our framework for increasingly complex simulations, incorporate diverse data sources, and potentially leverage high-performance computing resources as research needs evolve.

NetworkX [14] and MultinetX [2]. To address the limitations of single-network simulations and create customized multilayer networks, we integrated MultinetX and NetworkX into our framework. The official demo example of Repast4Py utilizes NetworkX for generation of single-layer networks. MultinetX is an extension

of NetworkX; it is a Python package designed for the manipulation and visualization of multilayer networks. At its core is the MultilayerGraph class from NetworkX, which inherits all properties from NetworkX's networkx.Graph() for network generation. The integration of these two libraries allows for the effective creation of multilayer networks in `MultiRepast4py`. In our framework, each layer is created through Networkx, and then combined into a multilayer network using MultinetxUsers from MultinetX. In summary, NetworkX and MultinetX were integrated into our framework for the following reasons:

(1) *Flexible Network Creation*: NetworkX and MultiNextX offer customization for creating networks. Users can define nodes and both inter-layer and intra-layer adjacency matrices. (However, the current version is limited to undirected networks.)
(2) *Visualization Support*: The packages allow for the visualization of dynamical processes through node and link coloring, enhancing the interpretability of simulation results.
(3) *Compatibility with Repast4py*: Crucially, MultinetX allows for the creation of multilayer networks that can be exported to .pb files using Repast4py's writenetwork function, ensuring seamless integration with our chosen platform.

3.2 Implementation Details

The implementation of the multilayer functionality in `MultiRepast4py` consists of three key steps. First, during the network creation process, it is crucial to label the nodes according to their respective layers. Second, when creating the agents and the model, these labels are used to categorize agents into sets corresponding to the different layers. Lastly, during the simulation, layer-specific steps can be executed, with agents synchronized across layers at each time step.

3.2.1 Network Creation. In this step, we create networks using MultinetX. We can utilize any of the provided function calls or customize our own by creating nodes and specifying edges. It is important to ensure that both networks are undirected and of the same size. Subsequently, we label the nodes and define the inter-layer connections. We show the pseudo-code for this step in Algorithm 1.

This algorithm constructs a multilayer graph from two networks, $G1$ and $G2$, by assigning node layers and identifiers, initializing an adjacency block for inter-layer connections, and then creating and interconnecting the layers within a multilayer graph structure. The algorithm iterates until the network creation is complete, ensuring each node is properly configured and interconnected across layers.

3.2.2 Agent-Based Model Creation. We next modify the `create_agent` function from Repast4py to incorporate the attributes we defined in the first step. Then, we group the nodes by layers using the tags we created. This grouping allows for layer-specific operations and interactions. We show the pseudo-code for this step in Algorithm 2.

This algorithm defines a function to create an agent with layer and number attributes, including node ID, agent type, and rank. It also initializes two sets to categorize agents by their layers. As it iterates through the agents in the model context, it assigns each agent to either 'layer1' or 'layer2' based on its layer attribute. The output consists of the sets containing agents from each layer, enabling simulation of multilayer networks.

Algorithm 1: Network Creation

Input: Number of layers L, Number of agents per layer N
while *not complete* **do**
  for *node* $i \in [0, N-1]$ **do**
    Assign layer to nodes:
      $G1.\text{nodes}[i].\text{layer} \leftarrow 1$
      $G2.\text{nodes}[i].\text{layer} \leftarrow 2$
    Set node identifiers:
      $G1.\text{nodes}[i].\text{number} \leftarrow i$
      $G2.\text{nodes}[i].\text{number} \leftarrow i$
  end
  Initialize adjacency block:
   $adj\_block \leftarrow$ mx.lil_matrix(np.zeros($(2N, 2N)$))
  Create interconnections:
    $adj\_block[0 : N, N : 2N] \leftarrow$ np.identity(N)
  Make adjacency block symmetric:
    $adj\_block \leftarrow adj\_block + adj\_block^T$
  Create multilayer graph $mg \leftarrow$ mx.MultilayerGraph()
  Add layers to the graph:
    mg.add_layer($G1$)
    mg.add_layer($G2$)
  Interconnect layers using adjacency matrix:

   mg.layers_interconnect(inter_adjacency_matrix = adj_block)
end
Output: Multilayer graph mg, Layer assignments $G1$, $G2$

Algorithm 2: Model Creation

Function: Create Agent
Input: Node ID *nid*, Agent Type *agent_type*, Rank *rank*, Additional Parameters *kwargs*
Output: Agent
layer $\leftarrow$ *kwargs.get*('*layer*', None)
number $\leftarrow$ *kwargs.get*('*number*', None)
return RumorAgent(*nid*, *agent_type*, *rank*, *layer*, *number*)
In Model Class:
Initialize layer sets:
  layer1 $\leftarrow$ set()
  layer2 $\leftarrow$ set()
for *agent* ***in*** *self.context.agents()* **do**
  if *agent.layer == 1* **then**
    layer1.add(agent)
  end
  else if *agent.layer == 2* **then**
    layer2.add(agent)
  end
end
Output: Layer sets layer1, layer2

Remark *(Agent identification challenges).* Agent identification and synchronization is one of the key elements of our framework, as it allows us to create and track agents over multiple layers. Initially, layer categorization was attempted using agent IDs. However, this approach proved unreliable within the MPI (Message Passing Interface) system of Repast4py, as agent IDs might not be unique when processes run on different ranks. This observation led to the development of our current approach, which utilizes explicit layer and number attributes for each agent. The current method of categorizing agents by layer and number has proven to be more robust and easier to implement. With the agent type left unmodified, users can define it according to their specific needs. Layer and number attributes were added as additional fields through JSON, ensuring no conflicts with the original features provided by Repast4py, resulting in a cleaner overall design.

3.2.3 Synchronizing agents. With agents grouped into layers, we can customize the simulation steps for each layer and control their frequency using the schedule runner. After changes are made within a layer, the corresponding agents in other layers can be updated through a synchronization process that is tailored to specific scenarios. For example, in simulating the spread of rumors, when the agent attribute receive_rumor is binary and irreversible, we can override the attribute in other layers at the end of the timestep when it is set to true. However, in other scenarios where the attribute is a dynamic value, the synchronization function may need to be modified to ensure the simulation runs as intended.

4 EXPERIMENTAL DEMONSTRATION

To demonstrate the capabilities of our multilayer simulation framework, we present two case studies focused on information dynamics and epidemiology – specifically, rumor propagation and disease spread. These topics are commonly explored in agent-based simulations and serve as excellent examples to highlight the advantages of our multilayer approach.

Both case studies are based on modifications of existing demos from Repast4py. Case 1 (Rumor Spreading on Social Networks) is adapted from the Rumor Spreading example in Repast4py, while Case 2 (Spread of a Disease through Multiple Interaction Modalities) combines aspects of the Random Walk and Zombie models of Repast4py. We selected these examples to illustrate the process of transforming a single-layer ABS into a multilayer ABS.

4.1 Case Study 1: Rumor Spreading on Social Networks with Cross-Posting

Our first case study aims to demonstrate that multilayer simulations can provide insights that traditional single-layer approaches may overlook as they focusing on the spread of information on social networks in isolation, and do not account for cross-posting between them. Figure 2 shows a snapshot of the graphical illustration of the rumor spreading process provided by `MultiRepast4py` during this experiment.

4.1.1 Experiment Setup. This model consists of two layers, each comprising 15 nodes. These layers are intended to represent 15 individuals active on two distinct social platforms. The rumor is initially seeded with one individual. In the simulation, this is represented

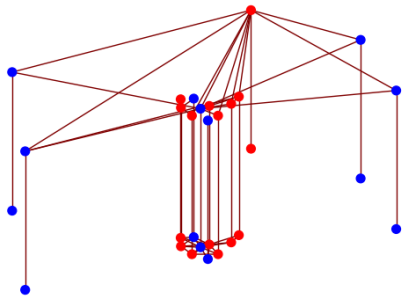

Figure 2: Visualization of the Rumor Spreading process (at time step 14 out of 50) in `MultiRepast4py`.

by two nodes on different layers, where one node can be viewed as the original agent, and the other node viewed as its "shadow clone".[1] As detailed in Section 3.2, our proposed agent identification and synchronization framework ensures that these "two" agents maintain synchronized states throughout the simulation.

Network Structure. Both layers are selected to be small-world networks, with customized edge structure, as detailed in Table 1.

People	Edges in G1	Edges in G2	Combined Edges
1	9	0	9
2	2	0	2
3	3	0	3
4	4	1	5
5	4	2	6
6	4	2	6
7	3	2	5
8	8	8	16
9	3	3	6
10	2	4	6
11	0	3	3
12	0	2	2
13	0	2	2
14	0	2	2
15	0	9	9

Table 1: Edges in G1, G2, and Combined Edges for each agent in Case Study 1 (Rumor Spreading)

Agent Attributes. Agents are characterized by a single binary attribute and two integers labeling layer and number:

- Received_rumor: Set to 1 when the agent has received the rumor, 0 otherwise.
- Layer: Which layer the node is located in.
- Number: Which number it is given

Simulation Step. The simulation proceeds as follows:

(1) At each time step, agents who have received the rumor check their neighbors on both layers.

[1]The term "shadow clone" is a reference from the anime Naruto, where the Shadow Clone Jutsu is a clone technique that allows the user to create one or more copies of themselves, as introduced by Tobirama Senju.

(2) For each uninformed neighbor, there is a 2% chance of transmitting the rumor.
(3) The interaction frequency and transmission probability are identical across layers, but the network structures differ (these parameters can be customized).

The simulation is run for 50 time steps on each replication and seed combination. We select 15 different seeds, and run the experiment 100 times for a given seed. Together, this results in 1500 total simulations (each lasting 50 time steps). We report averages over these runs, to remove the effects of randomness in the experiments.

4.1.2 Results. Figure 3 illustrates the spread of the rumor (measured by the number of nodes who have heard the rumor) after 50 time steps, when starting from three different seed nodes. From this figure, it is evident that Person 8 exhibits a dominant spreading process. While this may seem intuitive given its highest number of combined edges across the two networks (see Figure 2), the significance lies in the fact that such dominance would not be apparent if only a single layer were considered. In fact, Person 1 and Person 15 are the nodes with the highest degree centrality in Layers 1 and 2 respectively, a measure commonly used to identify the most important node for seeding a rumor.

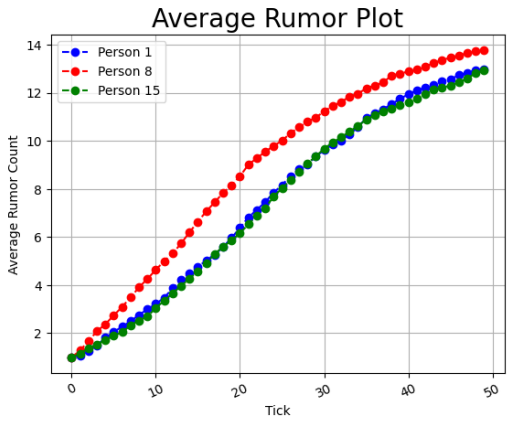

Figure 3: The spread of a rumor (measured by the number of nodes who have heard the rumor) after 50 time steps, when starting from different seed nodes.

This result highlights the importance of multilayer network analysis, made possible by the `MultiRepast4py`. By integrating multiple network layers, our framework enables a more comprehensive examination of agent interactions and network modeling, revealing critical insights that may otherwise remain hidden in single-layer ABS analyses.

4.2 Case Study 2: Spread of a Disease through Multiple Interaction Modalities

The study of the spread of diseases has also been a common topic in agent-based simulations (e.g., [16, 19] on ABS studies of the spread of COVID-19). This type of simulation differs from our earlier case study of information dynamics simulations: the spread of diseases typically takes place in a geographic information system (GIS) environment, as disease spread is closely tied to spatial distance. As a result, this case study requires a modeling environment that can account for spatial relationships. The demo provided by Repast4py uses a grid-based environment, which we have modified to create a multilayer version.

4.2.1 Experiment Setup. This case study demonstrates a multilayer simulation approach to analyzing disease spread in a *two-layer* grid-based environment.

Network Setup. Our two proposed grid layers are intended to distinguish the interactions among individuals on weekdays (Layer 1) and weekends (Layer 2). For instance, Layer 1 can be viewed as each individual's workplace (capturing the spread of the disease during work-related interactions), while Layer 2 can be viewed as their extended family and friends network (capturing the spreading processes due to socializing with them over the weekend). The two grid layers are set up to differ in size, frequency of interactions, and population density, as detailed below:

- Two layers, each modeled as a discrete grid, with the same set of 1,000 agents on both layers
- Layer 1: 200x200 grid (Weekday)
- Layer 2: 100x100 grid (Weekend)

The larger grid on Layer 1 creates lower population density, making the disease less likely to spread. Conversely, the smaller grid in Layer 2 results in a higher population density, increasing the likelihood of disease spread.

Agent Attributes. Agents are characterized by the following attributes:

- Infected: Boolean indicating disease status
- Infect_Count: Integer representing the number of infections (may exceed one).
- Recover_Count: Integer representing the number of recoveries (may exceed one).
- Recovered: Boolean indicating recovery status
- Disease probability: Likelihood of receiving the disease
- Infected duration: Countdown for the infection period
- Layer: Integer identifying the current layer (1 or 2)
- Number: Unique identifier within each layer

Simulation Dynamics. To simulate real-life scenarios, agents are initially assigned a higher probability of contracting the disease. After infection and recovery, this probability decreases to represent immunity, although there remains a chance of reinfection. The duration of infection is extended to allow for a thorough observation of system patterns. Specifically, we set the following parameters:

- Initial seed: 100 individuals (200 agents) infected at the start
- Transmission: Probabilistic based on agent's disease probability, initially set at 50%, reduce to 5% after recovery
- Recovery: After a random duration between 80-100 ticks

Movement Patterns. Agents move randomly with varying ranges to reflect the differing human movement at work vs. when socializing with friends. In Layer 1, agents move 1 unit per step, simulating shorter movements typically seen on weekdays (a random integer between -1 and 1 in both the x and y axes). In Layer 2, agents move up to 4 units per step (a random integer between -4 to 4 in both the x and y axes), simulating longer-distance travel often

associated with weekends. Movement occurs in cardinal directions in both layers.

Simulation Process. The simulation is scheduled with five steps for weekdays (Layer 1) and two steps for weekends (Layer 2). These layers are mutually exclusive, meaning they do not run simultaneously within the same time step. Agents are given "shadow clones" on each layer (as detailed in Section 3.2), with disease status synchronized between layers after each step.

During each simulation step, the following events occur:

(1) Agents move within their respective layers.
(2) Disease transmission occurs within each layer.
(3) Agent states are synchronized across layers.
(4) Global infection and recovery counts are logged.

The simulation is run for a duration of 500 time steps, and is replicated 100 times.

4.2.2 Results. We illustrate the results of this simulation in Figure 4. The figure shows two sets of lines: red lines represent cumulative cases, while blue lines represent active cases at each time step. Three different scenarios are illustrated: the control group (normal movement), quarantine on weekends when infected, and full quarantine (quarantine every day) when infected. As expected, we observe approximately 33% fewer active cases at the peak of the spread (tick 125) when comparing full quarantine to the control group. The weekend quarantine policy, while not as comprehensive as full quarantine, still shows significant impact. Considering that it only covers a portion of the week, weekend quarantine proves to be surprisingly effective. It contributes to both delaying the peak of infections and reducing the overall number of cases, albeit to a lesser extent than full quarantine.

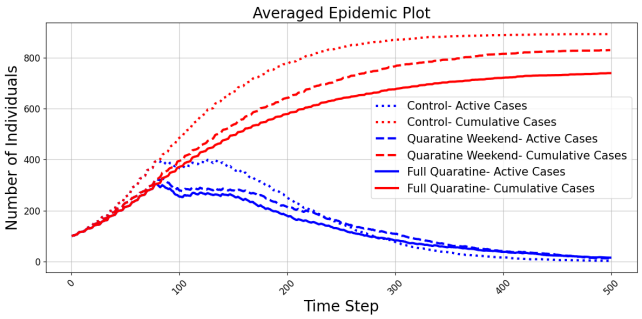

Figure 4: Disease spread over time under different quarantine scenarios: control (no quarantine), weekend-only quarantine, and full quarantine. Cumulative and active cases plotted for each scenario.

Using the multilayer model, we are able to simulate a range of complex scenarios that a single-layer model could not handle as effectively. For instance, the multilayer approach allows us to model different movement behaviors and policies across distinct time periods— such as differentiating weekday and weekend interactions— which more accurately reflects real-world human dynamics. Unlike a single-layer model, which would force us to homogenize behavior across all interactions, the multilayer model captures variations in mobility, social contacts, and policy impacts across different contexts. This leads to greater precision in simulating quarantine policies and their outcomes. By isolating interactions in separate layers, we can observe the direct and indirect effects of interventions within specific contexts, such as workplaces versus social settings, yielding more detailed insights into how infections spread and how they can be controlled. The single-layer model lacks this flexibility and would likely fail to capture the nuanced effects of policies that vary by time or setting.

5 CONCLUSION

We have developed `MultiRepast4py`, a multilayer agent-based simulation framework. This framework builds on an existing ABS simulation platform (specifically, Repast4py [8]) and Python-based multilayer network generation package (specifically, MultinetX [14]), and equipping it with a graphical interface, enabling researchers to model complex, multilayered interactions without needing extensive programming skills, making it accessible to researchers across various disciplines. We used simulations studies on rumor spreading across social networks and spread of diseases across multiple interaction modalities to highlight the advantages of our multilayer approach. Our use cases were selected from existing ABS on single-layer networks, to further illustrate the process of transitioning from single-layer to multilayer ABS using our proposed platform. As noted in the introduction, our framework has the potential to be integrated with data analysis techniques to enable data-driven (multilayer) ABS; we view this as an important directions of future work. Other future work directions include expanding our framework to allow for directed interactions, and non-identity inter-layer interaction matrices.

REFERENCES

[1] Alberto Aleta and Yamir Moreno. 2019. Multilayer networks in a nutshell. *Annual Review of Condensed Matter Physics* 10 (2019), 45–62.
[2] Roberta Amato, Nikos E Kouvaris, Maxi San Miguel, and Albert Díaz-Guilera. 2017. Opinion competition dynamics on multiplex networks. *New Journal of Physics* 19, 12 (2017), 123019.
[3] Federico Battiston, Matjaž Perc, and Vito Latora. 2017. Determinants of public cooperation in multiplex networks. *New Journal of Physics* 19, 7 (jul 2017), 073017. https://doi.org/10.1088/1367-2630/aa6ea1
[4] Parantapa Bhattacharya, Saliya Ekanayake, Chris J Kuhlman, Christian Lebiere, Don Morrison, Samarth Swarup, Mandy L Wilson, and Mark G Orr. 2019. The matrix: An agent-based modeling framework for data intensive simulations. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 1635–1643.
[5] Stefano Boccaletti, Ginestra Bianconi, Regino Criado, Charo I Del Genio, Jesús Gómez-Gardenes, Miguel Romance, Irene Sendina-Nadal, Zhen Wang, and Massimiliano Zanin. 2014. The structure and dynamics of multilayer networks. *Physics reports* 544, 1 (2014), 1–122.
[6] Sergey V Buldyrev, Roni Parshani, Gerald Paul, H Eugene Stanley, and Shlomo Havlin. 2010. Catastrophic cascade of failures in interdependent networks. *Nature* 464, 7291 (2010), 1025–1028.
[7] Eduardo Cabrera, Manel Taboada, Ma Luisa Iglesias, Francisco Epelde, and Emilio Luque. 2011. Optimization of healthcare emergency departments by agent-based simulation. *Procedia computer science* 4 (2011), 1880–1889.
[8] Nick Collier. 2003. Repast: An extensible framework for agent simulation. *The University of Chicago's social science research* 36 (2003), 2003.
[9] Nicholson Collier and Jonathan Ozik. 2022. Distributed agent-based simulation with Repast4Py. In *2022 Winter Simulation Conference (WSC)*. IEEE, 192–206.
[10] Nicolas De Bufala and Jean-Daniel Kant. 2019. An evolutionary approach to find optimal policies with an agent-based simulation. In *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019)*,.
[11] Raman Ebrahimi and Parinaz Naghizadeh. 2023. United we fall: On the nash equilibria of multiplex network games. In *2023 59th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 1–8.
[12] Jianxi Gao, Sergey V Buldyrev, H Eugene Stanley, and Shlomo Havlin. 2012. Networks formed from interdependent networks. *Nature physics* 8, 1 (2012), 40–48.

[13] Jesús Gómez-Gardeñes, Irene Reinares, Alex Arenas, and Luis Mario Floría. 2012. Evolution of Cooperation in Multiplex Networks. *Scientific Reports* 2, 1 (31 Aug 2012), 620. https://doi.org/10.1038/srep00620

[14] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[15] Nirupam Julka, Rajagopalan Srinivasan, and I Karimi. 2002. Agent-based supply chain management—1: framework. *Computers & Chemical Engineering* 26, 12 (2002), 1755–1769.

[16] Cliff C Kerr, Robyn M Stuart, Dina Mistry, Romesh G Abeysuriya, Katherine Rosenfeld, Gregory R Hart, Rafael C Núñez, Jamie A Cohen, Prashanth Selvaraj, Brittany Hagedorn, et al. 2021. Covasim: an agent-based model of COVID-19 dynamics and interventions. *PLOS Computational Biology* 17, 7 (2021), e1009149.

[17] Mariam Kiran, Paul Richmond, Mike Holcombe, Lee Shawn Chin, David Worth, and Chris Greenough. 2010. FLAME: simulating large populations of agents on parallel hardware architectures. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. 1633–1636.

[18] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. 2014. Multilayer networks. *Journal of complex networks* 2, 3 (2014), 203–271.

[19] Fabian Lorig, Emil Johansson, and Paul Davidsson. 2021. Agent-based social simulation of the COVID-19 pandemic: A systematic review. *Journal of Artificial Societies and Social Simulation* 24, 3 (2021).

[20] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. 2005. Mason: A multiagent simulation environment. *Simulation* 81, 7 (2005), 517–527.

[21] Charles M Macal and Michael J North. 2009. Agent-based modeling and simulation. In *Proceedings of the 2009 winter simulation conference (WSC)*. IEEE, 86–98.

[22] Nelson Minar, Roger Burkhart, Chris Langton, Manor Askenazi, et al. 1996. The swarm simulation system: A toolkit for building multi-agent simulations. (1996).

[23] Marzieh Parandehgheibi and Eytan Modiano. 2013. Robustness of interdependent networks: The case of communication networks and the power grid. In *2013 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2164–2169.

[24] Steven F Railsback, Steven L Lytinen, and Stephen K Jackson. 2006. Agent-based simulation platforms: Review and development recommendations. *Simulation* 82, 9 (2006), 609–623.

[25] William Rand. 2019. Theory-interpretable, data-driven agent-based modeling. *Social-behavioral modeling for complex systems* (2019), 337–357.

[26] Raquel Rosés, Cristina Kadar, Charlotte Gerritsen, and Chris Rouly. 2018. Agent-Based Simulation of Offender Mobility: Integrating Activity Nodes from Location-Based Social Networks.. In *Aamas*. 804–812.

[27] Mostafa Salehi, Rajesh Sharma, Moreno Marzolla, Matteo Magnani, Payam Siyari, and Danilo Montesi. 2015. Spreading processes in multilayer networks. *IEEE Transactions on Network Science and Engineering* 2, 2 (2015), 65–83.

[28] Ebrahim Moradi Shahrivar and Shreyas Sundaram. 2017. The game-theoretic formation of interconnections between networks. *IEEE Journal on Selected Areas in Communications* 35, 2 (2017), 341–352.

[29] Edward Bishop Smith and William Rand. 2018. Simulating macro-level effects from micro-level observations. *Management Science* 64, 11 (2018), 5405–5421.

[30] Seth Tisue and Uri Wilensky. 2004. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, Vol. 21. Citeseer, 16–21.

[31] Osman Yağan and Virgil Gligor. 2012. Analysis of complex contagions in random multiplex networks. *Physical Review E* 86, 3 (2012), 036103.

[32] Osman Yağan, Dajun Qian, Junshan Zhang, and Douglas Cochran. 2012. Optimal allocation of interconnecting links in cyber-physical systems: Interdependence, cascading failures, and robustness. *IEEE Transactions on Parallel and Distributed Systems* 23, 9 (2012), 1708–1720.

[33] Yingrui Zhang, Alex Arenas, and Osman Yağan. 2018. Cascading failures in interdependent systems under a flow redistribution model. *Physical Review E* 97, 2 (2018), 022307.

[34] Yingrui Zhang and Osman Yağan. 2019. Robustness of interdependent cyber-physical systems against cascading failures. *IEEE Trans. Automat. Control* 65, 2 (2019), 711–726.

[35] Yong Zhuang and Osman Yağan. 2019. Multistage complex contagions in random multiplex networks. *IEEE Transactions on Control of Network Systems* 7, 1 (2019), 410–421.