MultiRepast4py: A Framework for Agent-Based Simulations on Multilayer Networks

Keng-Lien Lin and Parinaz Naghizadeh

University of California San Diego, San Diego CA 92092, USA
{kel030, parinaz}@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 in the study of the spread of diseases and information. Existing platforms for ABS, such as NetLogo, Repast, and Mesa, typically focus on agents' interactions over a single network. In reality, however, agents' interactions are typically multi-layered (i.e., involve multiple interconnected networks that influence agents' decisions); existing ABS tools offer limited or no direct interaction across multiple networks of interactions. Researchers who require multilayer dynamics often rely on workarounds, such as creating custom implementations in NetworkX, or integrating multiple network representations, which can become highly specialized, difficult to generalize, and technically demanding to reproduce. To address this, we propose `MultiRepast4py`, a multilayer simulation tool extending the simulation capabilities of Repast4py. Our framework enables simulations on multilayer networked systems by efficiently reconstructing network data and utilizing agent attributes, allowing agents to dynamically access multilayer connections during simulation. By maintaining Repast4py's scalability and minimizing memory overhead, `MultiRepast4py` ensures high performance for large-scale simulations. Through simulation examples on the spread of information in social networks, we showcase how `MultiRepast4py` can enable more comprehensive agent-based simulations, guiding improved predictions and interventions.

Keywords: Agent Based Simulations · Multilayer Networks · Repast.

1 Introduction

Agent-based models and simulations have become a widely used computational approach for analyzing how the behavior and interactions of autonomous agents give rise to complex, system-level outcomes [21]. In an agent-based simulation (ABS), agents are first endowed with a set of unique 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 a disease, or adopt an opinion), as well as different interventions to 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 information).

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 [10], MASON [20], FLAME [17], and Swarm [22]; we refer the interested reader to [19] 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. For instance, ABS have been leveraged to gain insights into COVID-19 dynamics and interventions [16], healthcare resource management [8], spatio-temporal dynamics of crime [26], supply chain management [15], and policy design in labor markets [11]. 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], demonstrating the advantages of data-driven ABS [25], and developing strategies for scaling-up to accommodate data-intensive simulations [3].

The gap. Despite their widespread use and recent advances, most 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 mode of interaction). While Python-based libraries such as Mesa (equipped with NetworkX) and custom code allow the use of multiple networks, these approaches usually treat networks as parallel or isolated, rather than providing native support for dynamic interactions across layers. As a result, researchers often develop context-specific simulation environments (e.g., [31, 23, 16, 34]), which can be difficult to generalize beyond their intended context, and/or scale. This limitation is especially consequential in settings such as disease spread or information dynamics, where the multi-modality of human experiences plays a central role. 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 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. We also note that, while sharing some commonalities, our view of multilayer ABMS differs from *multilevel* ABMS (see [5] for a survey), which primarily focuses on hierarchies (e.g., micro-, meso-, and macro-levels of agent groupings, nested communities, and group dynamics).

Multilayer networks. For the situations described above, and other similar contexts, *multilayer networks* have been proposed as a model to simultaneously account for the multiple modalities of interactions between agents; see [4, 18, 1, 27, 6] 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., [28, 14, 2, 12]), to evaluate the resilience of networks of networks against failures or attacks, often by studying percolation (e.g., [7, 13, 33, 24, 35, 36, 4]), 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., [32, 37, 4]). Despite the broad applicability of multilayer network models and the increasing body of research dedicated to them, most existing implementations of these models are either theoretical or rely on custom simulation environments (e.g., those developed in some of the works in the literature above) that treat each layer separately, limiting the ability to explore interactions between layers in a scalable, generalizable manner.

Our contributions. To address this limitation, we have developed `MultiRepast4py`, a multilayer agent-based simulation framework. This framework builds on a commonly used, existing ABS simulation platform, Repast4py [10], making it accessible to researchers (especially those already familiar with Repast4py) who want to model complex, multilayered interactions without needing extensive programming skills. In more detail, our framework makes the following contributions:

1. **Multilayer Agent-Based Simulation Capability:** `MultiRepast4py` extends the established Repast4py platform to support multilayer agent-based models (ABMs). This bridges a gap in ABM technology, enabling more nuanced and realistic simulations of complex systems where agents interact across multiple interconnected networks.
2. **Scalable Multilayer Simulation:** `MultiRepast4py` retains all the features that make the Repast suite powerful, including its scalability and flexibility. This ensures that researchers can tackle large-scale, complex simulations without sacrificing computational efficiency.
3. **Accessible Integration:** `MultiRepast4py` requires only minimal adjustments to existing models. By reconstructing network files and adding a single agent attribute, users can incorporate multilayer interactions while preserving their existing Repast4py logic and workflows.
4. **Extensibility for Future Research:** Developed in Python, `MultiRepast4py` is open to customization and extension. For instance, researchers may adapt it to incorporate data-driven network models.

Our proposed framework is made publicly available at https://github.com/KengLL/MultiRepast4py.

Illustration through simulation studies. To demonstrate the practical utility of `MultiRepast4py`, we present a case study analyzing rumor propagation across interconnected social networks. This is done through two complementary simulations: a reduced-scale proof-of-concept and a full-scale validation. The study builds upon Repast4py's single-layer rumor model, explicitly demonstrating how to enhance conventional agent-based simulations with multilayer capabilities. The layers represent distinct social media platforms with unique network topologies (Erdős-Rényi random graphs) and interaction patterns. Through parameterized layer configurations, we model critical real-world phenomena like cross-platform connectivity and varying interaction frequencies – features impossible to capture in single-layer ABS. Our small-scale experiment reveals how nodes with balanced cross-layer connectivity (combined degree centrality = 7) outperform those with superior single-layer positions, while the large-scale extension demonstrates persistent multilayer effects in more realistic networks (50,000 nodes per layer). By comparing single-layer versus multilayer seeding strategies, we empirically validate that ignoring platform interdependence leads to suboptimal diffusion predictions. The seamless scaling from 25 to 50,000 agents per layer further demonstrates our framework's computational feasibility, executing efficiently on consumer-grade hardware. This case study exemplifies how `MultiRepast4py` enables researchers to 1) convert existing single-layer models into multilayer ones, and 2) identify emergent phenomena arising from cross-layer interactions.

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 simulation studies in Section 4, and 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 relation, mode of interaction, or information channel, between agents.

Formally, each layer α of a multilayer network is a network represented by a graph $\mathcal{G}^\alpha =< \mathcal{N}^\alpha, \mathcal{A}^\alpha >$, where $\mathcal{N}^\alpha$ 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. 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). We assume that interactions are undirected and weighted (reflecting mutual dependencies, but with potentially different strengths).

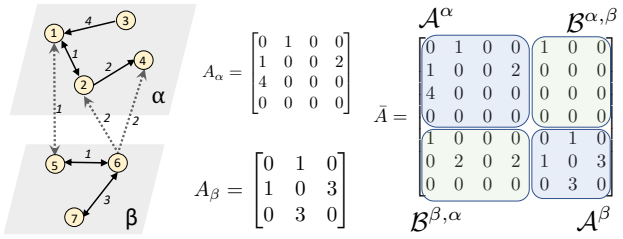

Fig. 1. An example of a multilayer network, and its intra-layer and inter-layer adjacency matrices.

In addition, as these layers do not operate in isolation, there exist connections between 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 $b_{mn}^{\alpha,\beta} \in \mathcal{B}^{\alpha,\beta}$ 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 paper, 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 referred to as *multiplex networks* in the literature [4]. 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 illustration of a multilayer network 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 during the week, and with their extended family and friends 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`: A Multilayer ABS Framework

Given the limitation 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

As noted in the introduction, there exist several widely used platforms for ABS, built on different programming languages; these include NetLogo [30], Repast [10], 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.

Accordingly, Repast4py [9], 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.

3.2 Implementation Details

The multilayer functionality in `MultiRepast4py` is implemented through a structured process that embeds each agent's multilayer edges into a unified data structure. Specifically, connections from the network files are mapped using the node

identifiers (`node_id`) defined in the first file, ensuring consistent agent identification across all layers.

Each agent is represented by an `Agent` dataclass containing its node_id, agent_type, rank, and a list of `LayerData` objects. Each `LayerData` object encapsulates the layer_id and a list of `Edge` objects, where each edge contains the target node_id and weight. This structured approach provides better type safety and cleaner data management compared to raw dictionaries.

The implementation is organized into four sequential stages, each addressing a critical aspect of our multilayer simulation framework. First, multilayer network files are parsed and merged into a unified structure, creating a cohesive representation of all layers. Second, edge data is transformed (compressed and encoded) to satisfy the Repast4py's network file format requirements while enhancing memory efficiency. Third, agents are initialized by reconstructing their multilayer connections from the processed data, enabling a decentralized setup that supports parallel execution. Finally, the simulation step function is adapted to allow layer-specific interactions, ensuring that each simulation cycle accurately reflects the dynamics of the corresponding layer. Together, these stages facilitate efficient memory usage and scalable parallel processing while preserving the essential semantics of multilayer networks. The following sections provide detailed explanations of each component.

Network File Parsing. In this stage, multiple network files are provided as input, each representing a layer. The file contains agent information in the form of unique identifiers (UIDs): (`node_id`, `agent_type`, and `rank`) and the edge connections between nodes. These network files can be generated using Repast4py's `write_network` function. The parsing process constructs a structured data model where each agent is represented as an `Agent` object with the following attributes:

Agent(node_id: int, agent_type: int, rank: int, layers: List[LayerData])

Each `LayerData` instance encapsulates the connections of an agent within a specific layer and stores them as a collection of `Edge` objects, each defined by a `target_id` and `weight`. This object-oriented approach improves code maintainability, provides clear semantic structure, and facilitates efficient access and updates to layer-specific edges. By storing outgoing edges as agent attributes, the model achieves greater decentralization, thereby reducing the need for message passing between ranks.

Data Compression & Encoding. Outgoing edge lists for agents cannot be directly stored in Repast4py network files, as the framework's network file format prohibits nested data structures and requires string-based keys. To ensure compatibility with Repast4py's network serialization requirements, the compression logic is encapsulated within the `CompressionHandler` class. This class first converts the structured agent data into a JSON-serializable format before applying a four-step transformation:

Algorithm 1 Multi-Layer Network Reconstruction

Require: List of network file paths $file\_paths$
Ensure: Unified network file with compressed multi-layer edges

Data Structure Initialization
procedure ProcessMultilayerNetwork($file\_paths$)
  $network\_data \leftarrow$ NetworkData($num\_layers = \text{len}(file\_paths)$)
  $network\_data.agents \leftarrow \{\}$ ▷ Dict[int, Agent]
Node Parsing Phase
  if $file\_paths$ is not empty **then**
    $lines \leftarrow$ ReadFile($file\_paths[0]$)
    $network\_data.is\_directed \leftarrow$ ParseHeader($lines[0]$)
    for $line$ **in** $lines[1:]$ **do**
      if $line$.strip() = "EDGES" **then**
        Break
      end if
      Parse $node\_id, agent\_type, rank \leftarrow line$
      $agent \leftarrow$ Agent($node\_id, agent\_type, rank$)
      if compressed data exists in $line$ **then**
        $agent.layers \leftarrow$ DecompressAgentData($line.attributes$)
      end if
      $network\_data.agents[node\_id] \leftarrow agent$
    end for
  end if
Edge Parsing Phase
  for $layer\_id, file\_path$ **in** enumerate($file\_paths$) **do**
    $lines \leftarrow$ ReadFile($file\_path$)
    for $line$ **in** $lines$ **do**
      if $line$.strip() = "EDGES" **then**
        Continue
      end if
      if $found\_edges$ **then**
        Parse $(source\_id, target\_id, weight) \leftarrow line$
        $agent \leftarrow network\_data.agents[source\_id]$
        $layer \leftarrow agent.$AddLayer($layer\_id$) ▷ Get or create LayerData
        $layer.$AddEdge($target\_id, weight$) ▷ Append Edge object
        if not $network\_data.is\_directed$ **then**
          $target\_agent \leftarrow network\_data.agents[target\_id]$
          $target_layer \leftarrow target_agent.$AddLayer($layer\_id$)
          $target_layer.$AddEdge($source\_id, weight$)
        end if
      end if
    end for
  end for
Compression and Output Phase
  Copy base file to output path
  for each node line in base file **do**
    if line is header **then**
      $output\_lines$.append($line$)
    else if line before "EDGES" marker **then**
      $node\_id \leftarrow$ ParseNodeId($line$)
      $agent \leftarrow network\_data.agents[node\_id]$
      $compressed \leftarrow$ CompressAgentData($agent$)
      $attributes \leftarrow$ json($\{'data' : compressed\}$)
      Append node info with $attributes$ to $output\_lines$
    end if
  end for
  Write $output\_lines$ to output file ▷ Exclude original EDGES section
  return $network\_data$
end procedure

1. **Serialization**: Convert `LayerData` to dictionaries via `to_dict()` methods.
2. **JSON Encoding**: Serialize the dictionary structure with compact formatting.
3. **zlib Compression**: Apply compression to the UTF-8 encoded JSON string.
4. **Base64 Encoding**: Convert compressed bytes to ASCII-safe string

The final encoded string is stored in the following format:

{"data" : "[Base64 string]"}

Agent Initialization. During the agent initialization phase, the `read_network` function is invoked to load the modified network file. The method `CompressionHandler .decompress_agent_data()` is then applied to reconstruct the list of `LayerData` objects from the Base64-encoded string. Each agent independently deserializes its layer data using the `LayerData.from_dict()` method, which recreates the structured edge information. This design preserves scalable parallel initialization across ranks while providing a clean, object-oriented interface for managing agent connectivity across layers.

Modify Step Function. The multilayer simulation is facilitated through Repast4py's scheduling mechanism. To enable multilayer simulations, a modified step function is defined in `MultiRepast4py`, which accepts a layer parameter to specify the active layer for a given simulation step. This function leverages agent-specific data structures to manage interactions dynamically. For instance, the agent's layer-specific edges are accessed via:

```
agent.get_layer(layer_id).edges  # Returns list of Edge objects
```

For compatibility, a list of neighbor identifiers can also be obtained via:

```
[edge.target_id for edge in agent.get_layer(layer_id).edges]
```

The `Agent.get_layer()` method returns the corresponding `LayerData` object for the specified layer, or `None` if the agent has no connections in that layer. This provides a cleaner and more expressive API for custom propagation mechanisms compared to direct dictionary access, while maintaining scalability across layers.

4 Experimental Demonstration

To demonstrate the capabilities of our multilayer simulation framework, we present a case study focused on information dynamics – specifically, rumor propagation. This topic is commonly explored in agent-based simulations and serves as an excellent example to highlight the advantages of our multilayer approach.

This case study is based on modifications of existing demo, Tutorial 2 - The Rumor Network Model from Repast4py. We selected this example to better illustrate the process of transforming a single-layer ABS into a multilayer ABS. The simulation was executed on an Apple M2 chip with 8GB RAM, showing that our framework is accessible on standard computing platforms.

4.1 Multilayer Analysis of Rumor Propagation Through Cross-Platform Interaction

This case study demonstrates how multilayer network simulations reveal propagation dynamics that conventional single-layer analyses cannot capture, particularly through the mechanism of cross-platform information diffusion.

Model Configuration. We implement a two-layer multiplex network model with 25 unique agents. Each agent is represented as a node in both layers, corresponding to the same individual active on two distinct social platforms. The experimental configuration employs a reduced-scale network to enable clear demonstration of multilayer interaction effects, with larger-scale validation presented in Section 4.2.

- **Network Topology:** Both layers' connections are generated via Erdős-Rényi random graphs($G(n,p)$ model, $n = 25, p = 0.1$), producing sparse networks with average degree $k = 2.5$. This model was selcted for its simplicity, controllability, and reproducibility.
- **Agent State Model:** Each agent has the following attribute
  - `received_rumor`: Binary state (0=uninformed, 1=informed)
  - `layers`: List of `LayerData` objects containing `Edge` lists

Propagation Dynamics. The rumor dissemination process operates as:

1. Both layer activates at each time step
2. Within the activate layer, Informed nodes attempt transmission to adjacent uniformed neighbors
3. Per-contact infection probability $\beta = 0.005$
4. New informed nodes participate in next spreading cycle

Simulation Protocol. We execute 100 Monte Carlo replications for each of 25 network seeds, yielding 2,500 independent simulations (100 time steps each). All results present ensemble averages with stochastic effects mitigated through this extensive sampling.

Combined Degree Centrality. For a multilayer network with L layers, the **combined degree centrality** C_D^{combined} of an agent m is defined as:

$$C_D^{\text{combined}}(m) = \sum_{\gamma=0}^{L-1} \frac{1}{T_\gamma} \cdot C_D^{\gamma}(m), \tag{1}$$

- $C_D^{\gamma}(m)$ denotes the degree centrality of agent m in layer γ,
- T_γ is the execution interval of layer γ, specifying how frequently the layer is active; it must be strictly positive ($T_\gamma > 0$).

Empirical Findings. Figure 2 demonstrates the propagation patterns for three strategic seed nodes, quantified through cumulative adoption curves. Table 1 provides structural context through degree centrality measures.

Table 1. Node Centrality Measures Across Network Perspectives. Combined degree centrality is calculated as the sum of intra-layer degree centralities, weighted by the reciprocal of each layer's activation interval ($T_0, T_1 = 1$).

	Degree Centrality		
Seed Node	Layer 1	Layer 2	Combined
Agent 2	3	4	**7**
Agent 23	**4**	1	5
Agent 24	1	**5**	6

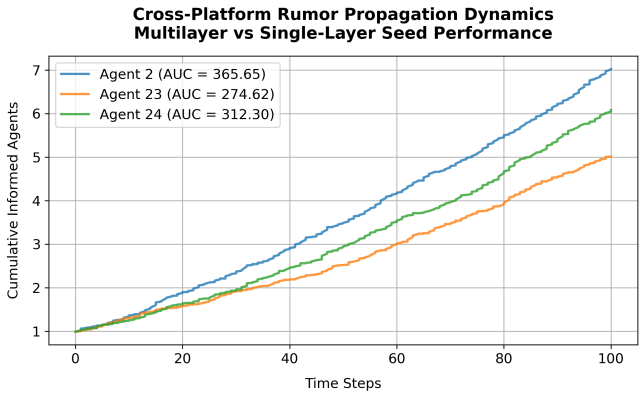

Fig. 2. Rumor propagation across multilayer networks. Curves show mean adoption rates (N=100 simulations) for three seed nodes. The Area Under the Curve (AUC) is used as a quantitative measure of overall dissemination efficiency.

The propagation dynamics reveal several critical insights. Despite Agents 23 and 24 exhibiting maximal intra-layer centrality in their respective platforms ($Layer1 : C_D = 4, Layer2 : C_D = 5$), Agent 2 demonstrates superior global propagation efficiency (AUC = 365.65 vs 274.62 and 312.30) due to its cross-platform connectivity (Combined $C_D = 7$). This emergent property remains invisible to single-layer analyses - traditional centrality metrics might identify Agents 23 and 24 as optimal seeds within their respective platforms, yet such single-layer analyses risk underestimating the influence of inter-layer connectivity in shaping system-scale diffusion. This highlights the importance of multilayer network analysis, made possible by the `MultiRepast4py`, in accounting for the impacts of different interacting networks on global outcomes.

4.2 Large-Scale Validation of Multilayer Propagation Dynamics

To evaluate the scalability and robustness of our framework, we extend the analysis presented in Section 4.1 by conducting a full-scale simulation of cross-platform information diffusion. This experiment quantifies the persistence of multilayer interaction effects in realistic network configurations and demonstrates the computational feasibility of large-scale multilayer agent-based simulations.

Model Configuration. We implement a two-layer multiplex network with 50,000 agents per layer, preserving the structural consistency of the small-scale demonstration while scaling the network parameters to reflect real-world social platforms.

- **Network Topology:** Each layer is instantiated as an Erdős-Rényi random graph($G(n,p)model, n = 50000, p = 0.0005$), yielding networks with an average degree $k = 25$.
- **Agent State Model:** As described in Section 4.1.

Propagation Dynamics: As detailed in Section 4.1, we set $\beta = 0.004$ to decelerate the spread, enhancing clarity and interpretability in the plot.

Simulation Protocol: We perform 100 Monte Carlo replications for each of three network seeding strategies, resulting in 300 independent simulations (each spanning 100 time steps). Seed nodes are selected according to three criteria:

- Top 10 degree centrality nodes in Layer 1 (single-layer perspective)
- Top 10 degree centrality nodes in Layer 2 (single-layer perspective)
- Top 10 combined degree centrality nodes (multilayer perspective)

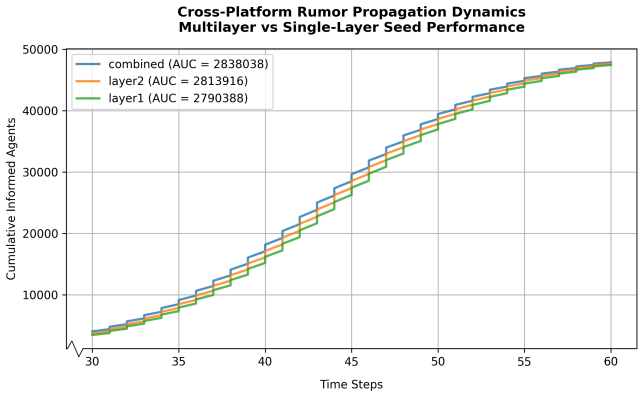

Fig. 3. Large-scale rumor propagation across Multilayer Networks. Curves depict the mean adoption rates (N=100 simulations) across three seeding strategies.

Empirical Findings. Figure 3 illustrates the propagation trajectories for each seeding strategy, with the Area Under the Curve (AUC) used to quantify overall dissemination efficiency.

Although the absolute differences in AUC values are modest relative to those observed in the small-scale demonstration, statistical analyses (t-test, Combined

vs. Layer 1, p-value: 0.0007) confirm that the disparities are significant. Notably, the combined centrality seeds achieve approximately 10% greater population penetration at key diffusion milestones compared to the single-layer strategies. These findings underscore the importance of cross-platform connectivity in enhancing global diffusion efficiency in large networks.

5 Conclusion

We have developed `MultiRepast4py`, a multilayer agent-based simulation framework. This framework builds on an existing ABS simulation platform (specifically, Repast4py [10]) enabling researchers to model complex, large-scale, multilayered interactions without needing extensive programming skills, making it accessible to researchers across various disciplines. We used simulations studies on rumor spreading to highlight the advantages of our multilayer approach. Our use case 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.

Code Availability The source code for `MultiRepast4py` is available at https://github.com/KengLL/MultiRepast4py.

Acknowledgments. This work is supported by the University of California San Diego Summer Research Internship program, and in part by the NSF under award CCF-2416311.

Disclosure of Interests. The authors declare no competing interests to declare that are relevant to the content of this article.

References

1. Aleta, A., Moreno, Y.: Multilayer networks in a nutshell. Annual Review of Condensed Matter Physics **10**, 45–62 (2019)
2. Battiston, F., Perc, M., Latora, V.: Determinants of public cooperation in multiplex networks. New Journal of Physics **19**(7), 073017 (jul 2017). https://doi.org/10.1088/1367-2630/aa6ea1, https://dx.doi.org/10.1088/1367-2630/aa6ea1
3. Bhattacharya, P., Ekanayake, S., Kuhlman, C.J., Lebiere, C., Morrison, D., Swarup, S., Wilson, M.L., Orr, M.G.: The matrix: An agent-based modeling framework for data intensive simulations. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1635–1643 (2019)
4. Boccaletti, S., Bianconi, G., Criado, R., Del Genio, C.I., Gómez-Gardenes, J., Romance, M., Sendina-Nadal, I., Wang, Z., Zanin, M.: The structure and dynamics of multilayer networks. Physics reports **544**(1), 1–122 (2014)
5. Brugière, A., Nguyen-Ngoc, D., Drogoul, A.: Handling multiple levels in agent-based models of complex socio-environmental systems: A comprehensive review. Frontiers in Applied Mathematics and Statistics **8**, 1020353 (2022)
6. Bródka, P., Musial, K., Jankowski, J.: Interacting spreading processes in multilayer networks: A systematic review. IEEE Access **8**, 10316–10341 (2020). https://doi.org/10.1109/ACCESS.2020.2965547
7. Buldyrev, S.V., Parshani, R., Paul, G., Stanley, H.E., Havlin, S.: Catastrophic cascade of failures in interdependent networks. Nature **464**(7291), 1025–1028 (2010)
8. Cabrera, E., Taboada, M., Iglesias, M.L., Epelde, F., Luque, E.: Optimization of healthcare emergency departments by agent-based simulation. Procedia computer science **4**, 1880–1889 (2011)
9. Collier, N., Ozik, J.: Distributed agent-based simulation with repast4py. In: 2022 Winter Simulation Conference (WSC). pp. 192–206. IEEE (2022)
10. Collier, N.: Repast: An extensible framework for agent simulation. The University of Chicago's social science research **36**, 2003 (2003)
11. De Bufala, N., Kant, J.D.: An evolutionary approach to find optimal policies with an agent-based simulation. In: 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), (2019)
12. Ebrahimi, R., Naghizadeh, P.: United we fall: On the nash equilibria of multiplex and multilayer network games. IEEE Transactions on Control of Network Systems (2025)
13. Gao, J., Buldyrev, S.V., Stanley, H.E., Havlin, S.: Networks formed from interdependent networks. Nature physics **8**(1), 40–48 (2012)
14. Gómez-Gardeñes, J., Reinares, I., Arenas, A., Floría, L.M.: Evolution of cooperation in multiplex networks. Scientific Reports **2**(1), 620 (Aug 2012). https://doi.org/10.1038/srep00620, https://doi.org/10.1038/srep00620
15. Julka, N., Srinivasan, R., Karimi, I.: Agent-based supply chain management—1: framework. Computers & Chemical Engineering **26**(12), 1755–1769 (2002)
16. Kerr, C.C., Stuart, R.M., Mistry, D., Abeysuriya, R.G., Rosenfeld, K., Hart, G.R., Núñez, R.C., Cohen, J.A., Selvaraj, P., Hagedorn, B., et al.: Covasim: an agent-based model of covid-19 dynamics and interventions. PLOS Computational Biology **17**(7), e1009149 (2021)
17. Kiran, M., Richmond, P., Holcombe, M., Chin, L.S., Worth, D., Greenough, C.: 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. pp. 1633–1636 (2010)

18. Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J.P., Moreno, Y., Porter, M.A.: Multilayer networks. Journal of complex networks **2**(3), 203–271 (2014)
19. Kravari, K., Bassiliades, N.: A survey of agent platforms. Journal of Artificial Societies and Social Simulation **18**(1), 11 (2015). https://doi.org/10.18564/jasss.2661, http://jasss.soc.surrey.ac.uk/18/1/11.html
20. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: A multiagent simulation environment. Simulation **81**(7), 517–527 (2005)
21. Macal, C.M., North, M.J.: Agent-based modeling and simulation. In: Proceedings of the 2009 winter simulation conference (WSC). pp. 86–98. IEEE (2009)
22. Minar, N., Burkhart, R., Langton, C., Askenazi, M., et al.: The swarm simulation system: A toolkit for building multi-agent simulations (1996)
23. Murdock, I., Carley, K.M., Yağan, O.: An agent-based model of cross-platform information diffusion and moderation. Social Network Analysis and Mining **14**(1), 145 (2024)
24. Parandehgheibi, M., Modiano, E.: Robustness of interdependent networks: The case of communication networks and the power grid. In: 2013 IEEE Global Communications Conference (GLOBECOM). pp. 2164–2169. IEEE (2013)
25. Rand, W.: Theory-interpretable, data-driven agent-based modeling. Social-behavioral modeling for complex systems pp. 337–357 (2019)
26. Rosés, R., Kadar, C., Gerritsen, C., Rouly, C.: Agent-based simulation of offender mobility: Integrating activity nodes from location-based social networks. In: Aamas. pp. 804–812 (2018)
27. Salehi, M., Sharma, R., Marzolla, M., Magnani, M., Siyari, P., Montesi, D.: Spreading processes in multilayer networks. IEEE Transactions on Network Science and Engineering **2**(2), 65–83 (2015)
28. Shahrivar, E.M., Sundaram, S.: The game-theoretic formation of interconnections between networks. IEEE Journal on Selected Areas in Communications **35**(2), 341–352 (2017)
29. Smith, E.B., Rand, W.: Simulating macro-level effects from micro-level observations. Management Science **64**(11), 5405–5421 (2018)
30. Tisue, S., Wilensky, U.: Netlogo: A simple environment for modeling complexity. In: International conference on complex systems. vol. 21, pp. 16–21. Citeseer (2004)
31. Xue, J., Park, S., Mondal, W.U., Reia, S.M., Yao, T., Ukkusuri, S.V.: Supporting post-disaster recovery with agent-based modeling in multilayer socio-physical networks. arXiv preprint arXiv:2307.11464 (2023)
32. Yağan, O., Gligor, V.: Analysis of complex contagions in random multiplex networks. Physical Review E **86**(3), 036103 (2012)
33. Yağan, O., Qian, D., Zhang, J., Cochran, D.: Optimal allocation of interconnecting links in cyber-physical systems: Interdependence, cascading failures, and robustness. IEEE Transactions on Parallel and Distributed Systems **23**(9), 1708–1720 (2012)
34. Yi, C., Yang, Q., Scoglio, C.M.: Multilayer network analysis of fmd transmission and containment among beef cattle farms. Scientific Reports **12**(1), 15679 (2022)
35. Zhang, Y., Arenas, A., Yağan, O.: Cascading failures in interdependent systems under a flow redistribution model. Physical Review E **97**(2), 022307 (2018)
36. Zhang, Y., Yağan, O.: Robustness of interdependent cyber-physical systems against cascading failures. IEEE Transactions on Automatic Control **65**(2), 711–726 (2019)
37. Zhuang, Y., Yağan, O.: Multistage complex contagions in random multiplex networks. IEEE Transactions on Control of Network Systems **7**(1), 410–421 (2019)