Phantom - An RL-driven framework for agent-based modeling of complex economic systems and markets

Leo Ardon
J.P. Morgan AI Research
Jared Vann
J.P. Morgan AI Research
Deepeka Garg
J.P. Morgan AI Research
Tom Spooner
Sutter Hill Ventures
Sumitra Ganesh
J.P. Morgan AI Research

ABSTRACT
Agent based modeling (ABM) is a computational approach to modeling complex systems by specifying the behavior of autonomous decision-making components or agents in the system and allowing the system dynamics to emerge from their interactions. Recent advances in the field of Multi-agent reinforcement learning (MARL) have made it feasible to learn the equilibrium of complex environments where multiple agents learn at the same time - opening up the possibility of building ABMs where agent behaviors are learned and system dynamics can be analyzed. However, most ABM frameworks are not RL-native, in that they do not offer concepts and interfaces that are compatible with the use of MARL to learn agent behaviors. In this paper, we introduce a new framework, Phantom, to bridge the gap between ABM and MARL. Phantom is an RL-driven framework for agent-based modeling of complex multi-agent systems such as economic systems and markets. To enable this, the framework provides tools to specify the ABM in MARL-compatible terms - including features to encode dynamic partial observability, agent utility / reward functions, heterogeneity in agent preferences or types, and constraints on the order in which agents can act (e.g. Stackelberg games, or complex turn-taking environments). In this paper, we present these features, their design rationale and show how they were used to model and simulate Over-The-Counter (OTC) markets.

CCS CONCEPTS
- Software and its engineering → Application specific development environments;
- Theory of computation → Multi-agent reinforcement learning; Market equilibria.

KEYWORDS
Reinforcement Learning, Agent-based Model, Multi-agent, Simulation Framework

1 INTRODUCTION
Agent based modeling (ABM) is a computational approach to modeling complex systems in a bottoms-up manner by specifying the behavior of autonomous decision-making components in the system (or agents); and allowing the system dynamics to emerge from their interactions. Drawing upon their real-world counterparts they seek to model, agents assess the state of the world and make decisions that will affect the rest of the system inducing the emergence of non-trivial phenomena. ABM offers several advantages over traditional differential equations modeling often used to study system dynamics. First, the description of problems is more natural because the real world is composed of autonomous entities. Second, it offers flexibility in the way the agents are modeled, with the option to replicate the heterogeneity of behaviors observed in real life.

Recent advances in the field of Reinforcement Learning (RL) have brought another dimension to the study of complex multi-agent systems with the introduction of an autonomous learning component to the ABM paradigm. This line of research seeks to study the equilibrium of such non-stationary environments where multiple agents learn at the same time, by playing against each other. Multi-agents reinforcement learning (MARL) techniques have been applied to autonomous vehicles, cooperative agents systems and trading simulators [1]. However, most frameworks for agent-based modeling are not RL-native, in that they do not offer concepts and interfaces that are compatible with the use of MARL to learn agent behaviors in a specified ABM. Our goal with Phantom is to bridge the gap between ABMs and MARL. Phantom is an RL-driven framework for agent-based modeling of complex multi-agent systems such as economic systems and markets. It leverages the power of MARL to automatically learn agent behaviors or policies, and the equilibria of complex general-sum games. To enable this, the framework provides tools to specify the ABM in MARL-compatible terms - including features to encode dynamic partial observability, agent utility / reward functions, heterogeneity in agent preferences or types, and constraints on the order in which agents can act.

While Phantom is agnostic to the particular domain being modeled, the modeling philosophy is especially suited to the design of economic systems, markets and other multi-agent problems within the financial domain. These multi-agent systems are typically populated by self-interested agents with a well-defined space of possible utility functions. Having a common, standardized framework to model and represent multi-agent problems in this space, also opens up the possibility of creating a suite of benchmark problems related to finance and markets (similar to how MuJoCo enabled the creation of benchmark environments for robotics / continuous control).

In this paper, we elaborate on the architecture and design of the Phantom framework and provide details about the main features and their rationale. Finally, we show how this framework has been used to model Over-The-Counter (OTC) markets and helped uncover interesting emergent behaviors.

2 PRINCIPAL FEATURES
2.1 Partial Observability
The agents in an ABM interact by sharing information with each other, that can affect their behavior and eventually lead to uncovering interesting phenomena. However, in many real-world applications not all the information shared across the system is
available for all the agents to consume e.g. a bidder entering an auction does not know how much its competitors are willing to bid, a market maker might only be able to observe the pricing inquiries it receives and its own transactions, a customer using a ride-sharing app might only see local drivers.

Most real-world problems have a strong component of partial observability and it was therefore crucial for our framework to support partially observable environments seamlessly and with the guarantee that there will be no information leakage among the agents. We propose in our framework, a customizable network model to design complex relationships between the different agents in the system and we offer a safe mechanism to ensure that only the specified information is shared with the other agents, guaranteeing true partial observability.

2.1.1 Network Model.

In Phantom, we model the relationship between agents in the system as a network or graph where each vertex / node represents an agent and each edge represents an open line of communication between two agents. One of our main desiderata for the framework was the ability to support complex and dynamic connectivity patterns between the agents. For this reason, we decided to treat the network component as a first-class citizen of the framework. The network can be seen as the physical layer on which the information is sent through, which means that two agents will only be able to communicate if an edge exists between the two vertices representing them. This property of the framework turns out to be particularly powerful to express partial observability.

The network being a component on its own, it is possible to encapsulate logic to update the network dynamically and replicate as closely as possible real-world interactions. For example, in a global currency (FX) market, agents might enter and exit the market at different times depending on their time-zone. In a ride-sharing market, the connectivity of a customer to drivers depends on geographical proximity which might vary with time as the agent moves. In over-the-counter financial markets, we might not have perfect knowledge of the actual network and relationships between liquidity providers and takers. These examples require dynamic or stochastic networks which can be implemented in Phantom by extending a well-defined network interface. One can implement their logic in a custom Network class and update the network topology at any point during the experiments, with the guarantee that two agents will be able to share information if and only if there are connected.

As part of the framework, we provide two different implementations of the network that already cover a range of use cases [1, 6, 12]. The first one is a static network where the connectivity between the agents is defined upfront and remains static throughout training and simulation. The second implementation, more robust, is a stochastic network where each edge connecting two agents, is associated with a probability of existing. The network can be ‘re-sampled’ during RL-training, between episodes, to yield a new structure which can impact the behaviors of the agents in the system (Figure 1). Adding stochasticity in the connectivity among agents helps prevent the MARL algorithms from overfitting to a specific network topology and is particularly useful to generalize the learned policies over a range of possible connectivity patterns when the actual graph is not known.

2.1.2 Messages and Views.

In Phantom, we offer two mechanisms to share information with a neighbour agent. The first one, which we qualify as ‘active’, take the form of a Message intentionally sent at a time $t$ from one agent to another. This active way of sharing information ensures that the information has been consumed by the receiver of the message. A message is triggered by an event in the system, such as a new time step or the reception of another message. The emission of a new message is often associated with the decision making process choosing the information the agent wants to share.

On the other hand, the second mechanism to share information is referred to as ‘passive’. The agent simply exposes specific information for others to consume but does not actively send it. We use Views to encapsulate the data to be shared. Each agent generates a customized view for each of its neighbors with only the information required. The views are regularly updated but no notification is sent to the other agents. It is entirely up to them to decide if and when they consume that information. The collection of views from all the neighbors of a given agent represents the context of that agent at a given time $t$ and can be used to make decisions. Views are particularly useful when the data exposed changes frequently and does not necessarily require an action from the other agent; instead of sending a message for every change the View will be updated without much processing from the system leading to higher overall performances.
As opposed to some other frameworks using message busses to expose an agent’s state to the other participants in the system, Phantom enforces the communication to only occur through the edges of the underlying network characterizing the connectivity between agents. In effect, Messages and Views can only be shared with an agent’s neighbors (i.e., where there exists an edge connecting the two nodes representing the agents). This aspect of the framework, as well as the ability to have different Views for different neighbors, were designed to easily encode the partial observability associated with many real-world problems. Implementing such a property in a subscription based model over a message bus would require ad-hoc validation logic to evaluate whether an agent can subscribe to a particular topic. The direct use of the underlying network to pass Message and to access Views greatly simplifies the implementation of the multi-agent system for the end user of the framework.

To make the concepts of Messages and Views more concrete, we provide a simple example (Figure 2). Let us consider a system where multiple vendors are trying to sell a product and a buyer wants to buy this product at the lowest possible price. Each vendor, due to their own internal finances, offers the product at a different price. The price and the quantity of product available is typically the kind of information that will be exposed in a View by the vendor agent, for connected buyers to consume during their decision making process. To decide whom to trade with, the buyer accesses the view exposed by each of the vendors and evaluates the price and the quantity of product available. Based on this information, the buyer makes an educated decision on which vendor to buy the product from and sends him a Message with the quantity of product required and the payment. The vendor replies by returning another message approving the transaction.

2.2 Heterogeneity of Learned Agent Behaviors

Specifying the behaviors of agents in the system, and how they evolve, is one of the crucial tasks in specifying an ABM for a domain and often requires hand-coding of known strategies in classical ABM approaches. While Phantom supports taking actions from a hand-crafted (fixed or evolving) policy, it is also natively geared towards supporting MARL as an approach to train the policies of the agents.

A MARL-driven approach to building an agent-based model requires specification of agents’ reward or utility functions; the agent behaviors emerge from the learning process as each agent tries to maximize its reward in the presence of other learning agents. In Phantom, the Agent definition includes specification of its observation and action spaces, as well as a reward function.

2.2.1 Types.

The agent reward or utility function is typically parameterized by a vector of values referred to as the Type - which in effect, implies that the agent class is associated with a space of possible reward functions rather than a single fixed one. Each Type value specifies a particular instance of the reward formulation and an associated learned behavior. For example, a market maker agent in a financial market might want to maximize profit and loss (PnL) while minimizing risk - this could be encoded as a parameterized reward function $PnL - \gamma \cdot Risk$ where the Type parameter $\gamma$ encodes the agent’s preferences regarding the trade-off (or its risk-aversion).

There are two advantages to this approach: (1) A single agent class could be used to learn a range of behaviors depending on the instantiation of Type parameters, making the ABM specification compact. (2) It is also often the case, that while the modeler might have sufficient domain knowledge about the general form of the reward function, she might not have direct knowledge of the exact form for each agent. Phantom allows the modeler to specify the general, parameterized form of the reward function - treating the Types as hyper-parameters that could be specified or calibrated.

2.2.2 Supertypes.

While the Types construct is powerful, it is still difficult to scale for a large number of agents. We argue that it is usually easier to think of families of agents sharing the same average behavior or “persona”, as it is often named in the industry. Of course, each individual is unique and has its own characteristics but agents in the
We build upon the, now standard, Open-AI gym [2] paradigm where agents would need to manually specify the environment through the intermediary of a centralized controller. Observations are collected to decide on the next action to take. The associated with the performed action is computed and the current state of the environment is updated. Finally, the reward is calculated, and the process repeats. This complex logic is encapsulated under the ‘step’ method of the environment interface. We present in Figure 3a the breakdown of what is happening at every step of the simulation. First the actions coming from the agents’ policy are processed to create both the messages that will need to be sent to the other agents and the mutations that will need to be performed to update the agent’s internal state. The messages are dispatched through the network via a customizable Message Resolver, deciding the ordering in which the messages will be sent. The messages are posted on a queue which is consumed until exhaustion allowing multiple messages back and forth between agents. Once all the messages have been processed, the mutations are executed to update the agent’s internal state. Finally, the reward associated with the performed action is computed and the current observations is collected to decide on the next action to take.

2.3 Modeling Complex Games

We build upon the, now standard, Open-AI gym [2] paradigm where a learning agent interacts in discrete time with the rest of the system via the intermediary of a centralized environment. The multi-agent setting adds a certain level of complexity to the environment component who now plays the role of orchestrator of the simulation. It is in charge of deciding when and in which order the agents get to act in the environment, when to send the messages between agents and when to update the agents’ internal state. This complex logic is encapsulated under the ‘step’ method of the environment interface. We present in Figure 3a the breakdown of what is happening at every step of the simulation. First the actions coming from the agents’ policy are processed to create both the Messages that will need to be sent to the other agents and the Mutations that will need to be performed to update the agent’s internal state. The messages are dispatched through the network via a customizable Message Resolver, deciding the ordering in which the messages will be sent. The messages are posted on a queue which is consumed until exhaustion allowing multiple messages back and forth between agents. Once all the messages have been processed, the mutations are executed to update the agent’s internal state. Finally, the reward associated with the performed action is computed and the current observations is collected to decide on the next action to take.

With multiple agents at play, the complexity of the ‘step’ method can rapidly increase and it becomes more and more difficult to design complex problems. To alleviate this, Phantom provides a simple and modular way to implement complicated sequences of stages where only a subset of the agents act. It uses the Finite State Machine formalism to define the order in which the agents should execute their actions in the environment. We show on Figure 3b, different examples of environment going from a simple one-step environment to a more sophisticated one involving a complex sequence of stages. For instance, the “Turn Based Env” characterizes a Stackelberg game where the agents are categorized into two groups playing alternatively. This type of game can be used to evaluate how one group of agents react to the actions performed by the agents from the other group [1].

2.4 Leveraging MARL for Scale

Phantom is setup to leverage advances in MARL and enable the use of RL-driven agent-based modeling at scale. The framework provides a direct integration with the distributed RL library RLLib [11], providing scalable implementations of state of the art RL algorithms and allowing the study of large multi-agent systems.

The framework also offers a built-in implementation of the Shared Policy learning technique presented in Vadori et al. [19], that can easily be configured via the framework’s API. Phantom automatically augments the observation space of an agent with its Type parameters for each episode, making it seamless to train policies that generalize across the range of values characterizing a family/supertype. It also allows the agents from the same family to share the same policy, considerably limiting the number of models to train.

There is a tremendous push towards learning the equilibria of complex general sum games with techniques that combine elements of game theory and RL. This is a rapidly advancing field and our hope is that with a RL-driven ABM framework like Phantom, the ABM community can leverage these advances to realistically simulate complex financial domains.

3 APPLICATION TO FINANCE

Phantom has been successfully put to contribution in Ardon et al. [1], Ganesh et al. [6], Vadori et al. [19] to study financial Over-The-Counter (OTC) markets. Studying this type of environment is particularly challenging for two main reasons. First, the fact that these markets traditionally involve a large number of participants, each with their own strategy and behavior. Second, the partial observability setting of OTC markets: a trade is only observable by the two parties involved (the buyer and the seller). The use of Phantom has eased the implementation of the OTC market environment and helped study emergent market participant behaviors.

Ardon et al. [1] trained 25 OTC market participants in parallel by dividing them into 2 families: the Liquidity Providers (LP) and the Liquidity Takers (LT). All the agents from the same family share the same parametrized reward formulation, leading to different behaviors for each set of parameters or “type”. To simulate the
Phantom - An RL-driven framework for agent-based modeling of complex economic systems and markets

![Diagram of the Phantom framework](image)

Figure 3: On the left, a breakdown of the different phases happening in an environment 'step'. On the right, some examples of environment designs, ordered by complexity. The State Machine Representation shows how the environment will orchestrate the simulation. For example, in the 'Turn Based Env' the group of 'Odd' and 'Even' Agents will act in the environment alternatively characterizing a Stackelberg Game.

The heterogeneity of the agents’ behavior, multiple supertypes are specified, each of which has different distribution parameters driving the strategy the agents will adopt. For example, one LP’s supertype features a higher risk sensitivity, causing the agents from this supertype to limit the size of their inventory and reduce the risk associated with a sudden price move. Another supertype example is to have a group of LTs who solely focus on maximizing their PnL and will decide not to trade if it will likely incur a loss. On the other end of the spectrum, more institutional traders focusing on trading a certain quantity optimally, are characterized with yet another supertype putting more weight on the quantity component of the reward function. The agents belonging to the same family share their policy, that is parametrized with the agent’s type in order to learn the full range of behaviors this family can adopt. The complexity of the problem is then drastically reduced, as only two policies are being trained.

The stochastic network implementation of Phantom offers a realistic configuration of the OTC market, where an agent is not always connected to the same set of market participants. By sampling a new network at the beginning of every episode, the learnt policy is able to generalize to various network topologies making the agent more robust to different market structures. Finally, the Message construct of the framework is used to execute trades between the agents of the systems. The Message payload contains the quantity of the asset traded and the associated cashflow. The Views on the other hand, are used by the LPs to publish their quoted prices. The LP regularly adjusts them to satisfy its own objective, while the LT will query the Views to decide with whom to trade.

We present in Figure 4b an example of the behavior the LP adopts under different market conditions. The results were obtained by varying the connectivity between one LP and the group of institutional LTs in the network, simulating days where the easy flow coming from institutional traders is reduced. This analysis was possible thanks to the use of the stochastic network provided by the framework. By being exposed to different variation of the network configuration the policy is able to support a wide range of scenarios with different levels of connectivity. In this particular analysis, we see that the less connected to flow LTs, the longer the LP will need to close its position. On the other hand, when the LP is well connected to flow LTs, it becomes easier to get rid of its inventory and the holding time is therefore reduced. Similarly, Figure 4a shows the impact of the risk aversion of the LP on its inventory. The more risk averse the LP is, the smaller its average inventory will be. This shows that the agent has learnt to limit the risk associated with a sudden price move by reducing its average absolute position.

4 RELATED WORK

Despite having been around since the 70’s [17], the notion of Agent Based modeling is an area that really started to grow in the 90’s. This sudden expansion can in part be attributed to the development of multi-agents frameworks such as SWARM [13], NetLogo [18] and others, making ABM more accessible to practitioners, reducing the
was developed with performance in mind to be able to support a large number of agents exchanging many messages. Although, Java is a proven language to deal with low-latency systems, its verbosity and its complexity prevent a broad adoption from the AI research community. On the other hand, the Python language, owing to its simplicity and the ease to build and import new libraries, has established itself as the go-to language for Machine Learning and other AI sub-fields. Among ABM frameworks, ABIDES [3] is a discrete event simulator that has been successfully used to model limit-order books in finance, but it relies on hand-coded agent policies.

In most recent years, we have seen an increase in the development of RL-frameworks designed for fast code iteration and rapid experimentation. In 2018, TF-Agents was created as an additional module to the TensorFlow framework to "make implementing, deploying, and testing new Bandits and RL algorithms easier" [7]. The concept of agent is introduced as a core element of the module. However, although it supports a multi-agent setting, the framework was not designed with this in mind making the implementation of multi-agents system more convoluted.

MARL frameworks such as WarpDrive [10] and MAVA [15] are designed to enable easier and more efficient implementation of MARL algorithms. The former innovates by focusing on performance with the use of GPU and their parallelization power. The multi-agent RL framework runs the simulation on a single GPU avoiding transferring data between the rollout workers and the policy trainer.

MAVA also proposes a new distributed framework for multi-agent RL. It leverages many of Deepmind’s open source components such as Acme the distributed single agent RL framework [9], Reverb for data management [5] and Launchpad for distributed processing orchestration [20]. Like Phantom, MAVA offers the options to specify network configuration to model the agents communication, however unlike in Phantom the network configuration remains fairly basic and stays static throughout the simulation and therefore does not allow the study of systems with stochastic connectivity.

More broadly, in contrast to the RL and MARL frameworks, Phantom’s goal is to provide the tools to develop the multi-agent environment for a problem or domain (as an ABM), rather than MARL algorithm itself. Indeed, our hope is that Phantom enables the ABM community to easily leverage the power of new MARL frameworks to learn agent behaviors at scale.

5 CONCLUSION

In this paper, we introduced a new framework, Phantom that leverages the power of MARL to automatically learn agent behaviors or policies, and the equilibria of complex general-sum games. To enable this, the framework provides tools to specify the ABM in MARL-compatible terms - including features to encode dynamic partial observability, agent utility / reward functions, heterogeneity in agent preferences or types, and constraints on the order in which agents can act. We presented the rationale behind the implementation of the main features built to ease the development of complex multi-agent systems. We also provided some examples where the framework has helped analyze a system such as the financial market, well-known for its complexity. Our hope is that Phantom enables the ABM community to easily leverage the power of MARL algorithms and advances to learn agent behaviors at scale, and can be used to create benchmark environments for financial domains.

DISCLAIMER

This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates (“J.P. Morgan”), and is not a product of the Research Department of J.P. Morgan. J.P. Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful. © 2021 JPMorgan Chase & Co. All rights reserved.

REFERENCES

[1] Leo Ardon, Nelson Vadori, Thomas Spooner, Mengda Xu, Jared Vann, and Sumitra Ganesh. 2021. Towards a Fully RL-Based Market Simulator. In Proceedings of the Second ACM International Conference on AI in Finance (Virtual Event) (ICAF ’21). Association for Computing Machinery, New York, NY, USA, Article 7, 9 pages. https://doi.org/10.1145/3490554.3494372
[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv arXiv:1606.01540 (Jun 2016). https://doi.org/10.48550/arXiv.1606.01540 [cs].
[3] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. 2019. ABIDES: Towards High-Fidelity Market Simulation for AI Research. CoRR abs/1904.12066 (2019). arXiv abs/1904.12066 http://arxiv.org/abs/1904.12066
Phantom - An RL-driven framework for agent-based modeling of complex economic systems and markets

[4] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. 2021. Multi-agent reinforcement learning: A review of challenges and applications. Applied Sciences 11, 11 (2021), 4948.

[5] Albin Cassier, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sotiauas, and Manuel Kroiss. 2021. Reverb: A Framework For Experience Replay. https://doi.org/10.48550/ARXIV.2102.04736

[6] Sumitra Ganesh, Nelson Vadovi, Mengda Xu, Hao Zheng, Prashant Reddy, and Manuela Veloso. 2019. Reinforcement learning for market making in a multi-agent dealer market. arXiv preprint arXiv:1911.05892 (2019).

[7] Sergio Guadarrama, Anoop Korattikara, Oscar Ramírez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Eid Kokko, Luciano Sbaiz, Jamie Smith, Gabriel Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. 2018. TF-Agents: A library for Reinforcement Learning in TensorFlow. https://github.com/tensorflow/agents

[8] Dirk Helbing, Illes J. Farkas, and Tamás Vicsek. 2000. Simulating dynamical features of escape panic. Nature 407 (2000), 487–490.

[9] Matthew W. Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal M. P. Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassier, Matthew Hoffman, Siqi Liu, Manuel Kroiss, Aedan Pope, and Alban Rrustemi. 2021. Launchpad: A Programming Model for Distributed Machine Learning Research. arXiv preprint arXiv:2106.04516 (2021). https://arxiv.org/abs/2106.04516