FABS: An Extensible and High-Performance Digital Twin Framework of AI-Driven Financial Systems

Angus Tze Lap Leung
Imperial College London
United Kingdom
angus.leung22@imperial.ac.uk

Ce Guo
Imperial College London
United Kingdom
c.guo@imperial.ac.uk

Wayne Luk
Imperial College London
United Kingdom
w.luk@imperial.ac.uk

Abstract

The study of complex AI systems in finance requires scalable digital twins, virtual laboratories for model verification, stress-testing, and counterfactual analysis. Agent-based modeling (ABM) is a primary tool for creating these digital twins, offering a bottom-up approach to capturing market-wide emergent phenomena. However, a major computational bottleneck hinders this work: existing simulation tools cannot efficiently model the large populations of interacting agents required for a high-fidelity digital replica of a financial system. The state-of-the-art simulator, MAXE, for instance, cannot accelerate a single, large-scale simulation. To address this, we introduce the Financial Agent-Based Simulator (FABS), an open-source C++ platform specifically designed as a high-performance engine for constructing and operating these large-scale digital twins. FABS combines three key features: a fine-grained parallel architecture to accelerate a single simulation, a dynamic graph-based optimization to reduce communication overhead, and an extensible callback system for simplified AI model integration. When benchmarked against MAXE in a large-scale fire sale scenario, FABS achieves a runtime speed-up of up to 12.52x. We validate its fidelity as a digital twin by showing that its synthetic data reproduces key stylized facts of financial markets, such as volatility clustering and fat-tailed returns. FABS provides a foundational tool for the financial AI community to build and analyze the complex agent-based digital twins needed to understand modern markets, at a scale that was previously computationally prohibitive.

CCS Concepts

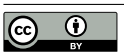
• **Computing methodologies** → **Instance-based learning**; • **Applied computing** → **Multi-criterion optimization and decision-making**; • **Information systems** → **Expert systems**; **Data analytics**.

Keywords

Agent-Based Modeling, Simulation, Systemic Risk, High-Performance Computing, Synthetic Data Generation, Graph Partitioning

ACM Reference Format:

Angus Tze Lap Leung, Ce Guo, and Wayne Luk. 2025. FABS: An Extensible and High-Performance Digital Twin Framework of AI-Driven Financial Systems. In *6th ACM International Conference on AI in Finance (ICAIF '25), November 15–18, 2025, Singapore, Singapore.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3768292.3770369



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICAIF '25, Singapore, Singapore
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2220-2/2025/11
https://doi.org/10.1145/3768292.3770369

1 Introduction

The growing use of artificial intelligence in finance requires new methods for conducting computational experiments. To understand the effects of AI, researchers and practitioners need to verify the safety of autonomous agents, test new strategies in a variety of market scenarios, and analyze the collective behaviors that result from the interaction of many AI systems. In such analyses, many what-if tests are difficult to perform using only historical data, because historical data is a passive record of one path the market has taken and does not allow for testing actions that were never chosen [6]. This limitation has encouraged the development of high-fidelity simulators, which we term digital twins [1, 21, 31], to serve as scalable laboratories for financial AI. We use this term to emphasize the platform's role as a virtual counterpart of the financial market, designed for the specific purpose of verifying and testing AI-driven agents in counterfactual scenarios. While the concept of a digital twin is sometimes associated with the definition of being tightly coupled to a real-time data stream, in the context of agent-based modeling, it broadly represents a reliable environment that enables researchers to perform 'what-if' tests and explore actions that were never chosen in reality.

Simulating large populations of AI-driven agents creates a major computational bottleneck. This challenge persists even with performance-oriented simulators. For instance, the state-of-the-art MAXE framework [3] achieves high performance through highly optimized, single-threaded code, making it two to three times faster than its predecessor, ABIDES [5], in benchmarks with 416 agents. However, MAXE lacks native support for multi-threading or distributed computing. Parallel execution is therefore only possible by having the operating system launch multiple, independent instances of the simulator. While this approach is effective for running separate experiments, it cannot accelerate a single, large-scale simulation. This architectural constraint creates a hard ceiling for studying systemic behaviors that emerge from thousands of interacting agents, which is the specific problem our work addresses.

To address this computational bottleneck, we introduce the Financial Agent-Based Simulator (FABS), an open-source[1] platform designed for large-scale computational experiments with AI agents in finance. FABS is designed to be both fast and easy to use for developing new models. Our main contributions are the following:

- A parallel architecture designed for fine-grained, agent-level parallelism, which distributes the workload of a single simulation across multiple processor cores to reduce runtime.

[1] Code repository: https://github.com/sonnets-project/fabs-twin

- A performance optimization that uses graph-based analysis of agent communication patterns and spectral clustering to dynamically improve parallel efficiency during a simulation.
- An extensible, callback-based framework that simplifies the integration of diverse AI agent models and ensures type safety at compile time, which helps improve programming productivity.

Our results show that FABS achieves a runtime speed-up of up to 12.52x over the MAXE framework. This improvement makes large-scale computational experiments with many interacting AI agents computationally feasible. We also confirm that the simulated market successfully reproduces key stylized facts, such as volatility clustering and fat-tailed returns [8]. This result validates FABS as a trustworthy platform for generating realistic data. By removing a key performance barrier, the platform provides a foundational tool for the financial AI community to build, test, and analyze the advanced multi-agent systems that are increasingly shaping modern markets.

2 Background and Related Work

2.1 Digital Twins in Financial AI

The validation and verification of AI systems in finance present a significant challenge. Historical data, while valuable, is a passive record of a single path taken by the market. It is insufficient for stress-testing AI agents, understanding their counterfactual impact, or preparing for rare but critical black-swan events [6, 14]. This has led to a growing interest in computational experimentation using agent-based models as digital twins of real-world financial systems [2, 10]. These simulated environments serve as virtual laboratories where researchers can actively develop AI systems, ask what-if questions, and examine extreme market conditions.

This approach is particularly vital for reinforcement learning (RL), where agents learn through interaction. An agent-based digital twin can provide the millions of interactive episodes needed for training, something impossible to obtain from real markets [25]. Beyond RL, these digital twins are important in verifying the robustness of complex AI strategies and for studying emergent market dynamics, such as flash crashes or herding behaviors. A primary bottleneck for this entire research agenda remains the computational cost of building and running simulations with large populations and complex control logic. Our work directly addresses this scalability bottleneck.

2.2 Fast Agent-Based Financial Simulation

Agent-based models (ABMs) are a primary tool for creating digital twins, offering a bottom-up approach to modeling market phenomena that can reproduce key stylized facts of financial markets [8, 13, 29]. However, the available general-purpose ABM toolboxes are often ill-suited for the specific demands of financial AI. For instance, platforms like NetLogo [28], SWARM [19], and the more recent Repast Simphony [20] have two main drawbacks for this application. First, they lack the specialized financial components and high performance needed for large-scale AI experiments. Second, many are implemented in higher-level languages like Java, which can create performance bottlenecks in high-frequency simulations. Even high-performance, general-purpose platforms like FLAME GPU [23] or Distributed MASON [9] are not primarily designed for the complex, high-frequency, and asynchronous communication patterns inherent in financial market microstructure. Consequently, adapting these general-purpose tools requires significant design and programming effort, and they often fail to meet the computational demands of large-scale financial models.

Financially-oriented simulators like ABIDES [5] provide realistic market structures and are widely used, but being implemented in Python, its Global Interpreter Lock (GIL) prevents true multi-core parallelism for accelerating a single simulation. This limits its use to problems that can be solved on a single core. The current state-of-the-art for high-performance financial simulation is MAXE [3], an open-source C++ framework. While its performance is a significant step forward, MAXE has two fundamental limitations that motivate our work. First, its rigid, tightly-coupled messaging system, which relies on runtime type casting, complicates the integration and rapid prototyping of novel AI models [27]. Second, and more critically, its parallelism is coarse-grained, limited to running multiple independent simulations (e.g., for parameter sweeps). It cannot accelerate the workload of a *single*, large-scale simulation, creating the performance bottleneck that our work overcomes for the financial AI community. This architectural context is why we focus our direct performance comparison on MAXE in this study.

2.3 Parallel Simulation and Adaptive Performance Optimization

A key challenge in accelerating a single simulation is the communication overhead that arises when splitting workloads across multiple processor cores [12, 15]. Assigning agents that interact frequently to different cores can negate any speed gains, a well-known NP-hard challenge in parallel computing known as the mapping problem [4]. A common strategy to address this is to model the system as a graph and use partitioning algorithms to group frequently interacting components and minimize inter-core communication [30]. Established techniques for this include spectral clustering and libraries like METIS [16, 24]. Our work builds upon these optimization principles, as detailed in Section 3.3.

3 The FABS Architecture

To address the challenges of building and operating high-fidelity financial digital twins, we designed the Financial Agent-Based Simulator (FABS). FABS is a C++ platform engineered to support large populations of interacting, intelligent components. Its architecture is based on three core principles: an extensible messaging framework for rapid AI model integration, a fine-grained parallel execution model for performance at scale, and an adaptive optimization strategy to manage communication-heavy workloads. This section details the methodology behind these principles.

3.1 Extensible and Type-Safe Messaging Framework

A primary goal for any digital twin platform is to provide a flexible framework that simplifies the development and integration of complex, intelligent components. Many existing platforms use a centralized message queue with a generic message format. This design often requires developers to use error-prone techniques like

predefined identifiers and runtime type checks to manage different interactions [3, 27]. The resulting system components can be tightly coupled, making it difficult to modify or add new AI-driven behaviors without risking runtime failures. This rigidity can slow the iterative cycle of designing, training, and testing new AI models within the digital twin.

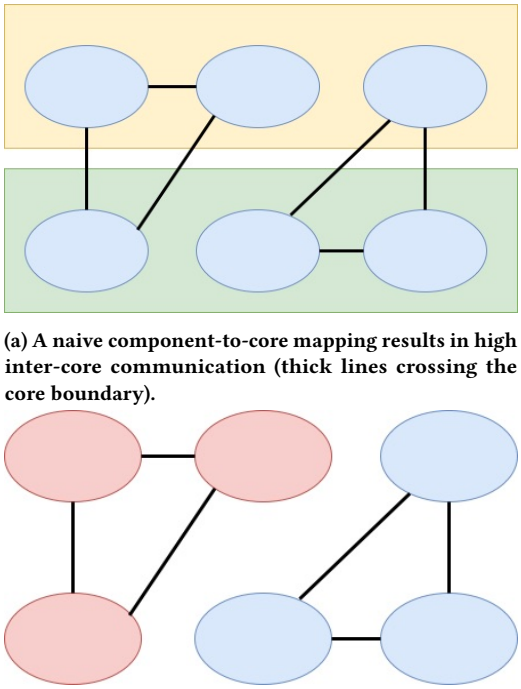
FABS addresses this challenge with a decentralized, callback-based messaging architecture. Instead of a single global queue, each component in the digital twin manages its own queue of incoming messages. A message is a lightweight object containing an execution time and a function object, or callback. This callback contains the complete logic for an interaction, including any necessary data from the sender. When it is time for the message to be processed, the receiving component executes this callback. This design offers several architectural advantages for developing and operating digital twins with AI components:

(1) Enabling Rapid Model Prototyping. The callback approach achieves greater decoupling between components. For example, an AI researcher can design a new trading agent and define its interactions with the market without requiring modifications to the exchange component's source code. This modularity helps developers focus on the agent's decision-making logic, such as a reinforcement learning policy, rather than platform-specific communication protocols, which can accelerate the design-test-refine cycle for new AI models.
(2) Efficient Data Exchange for Complex Models. Data can be passed efficiently by being packaged directly with the callback function. This structure helps reduce the need for repetitive, custom data structures for each new type of interaction. The process also leverages compiler optimizations to avoid needless memory allocations, which is important for performance when AI models must exchange complex information like state vectors or market observations.
(3) Enhanced System Robustness. Since an interaction is defined as a direct function call encapsulated in the callback, the validity of that call can be checked at compile time. This shifts error detection from runtime, where a crash might occur mid-operation, to the development stage. This approach contributes to a more stable and reliable digital twin, which is crucial for complex environments with many interacting AI components.

3.2 Fine-Grained Parallel Execution Model

The core performance contribution of FABS is its fine-grained, component-level parallel execution model, designed to fully utilize modern multi-core processors for a single, large-scale digital twin. This is a direct departure from systems like MAXE that only support coarse-grained parallelism, such as running multiple independent simulations at once.

At the start of an operation, the total component population is partitioned among a set of worker threads. Each worker thread is responsible for the lifecycle of its assigned components, including processing their individual message queues. The operation progresses in discrete, synchronized time steps, which we call simulation ticks. We must note that in this context, 'tick' refers to a discrete simulation time step, not the financial market's 'tick size' or minimum price fluctuation.



(a) A naive component-to-core mapping results in high inter-core communication (thick lines crossing the core boundary).

(b) Spectral clustering identifies optimal partitions (colored groups) by grouping frequently interacting components.

Figure 1: Illustration of the clustering-based optimization. Edge weight represents communication frequency. By remapping components based on the optimal partition, inter-core communication is minimized.

During a simulation tick, all worker threads execute their components' logic in parallel. To ensure correctness and a globally consistent state, a custom barrier synchronization is employed. When all threads have reached the barrier, a designated completion function is executed by the final arriving thread. This function is responsible for atomically advancing the global clock to the timestamp of the next scheduled event. This design guarantees that no component acts on a future simulation tick before all events in the current simulation tick have been fully processed. To further enhance performance, FABS explicitly pins each worker thread to a dedicated CPU core, which improves cache affinity by minimizing thread migration and reduces context-switching overhead [18].

3.3 Adaptive Performance Optimization

To accelerate a single simulation, its workload is split across multiple processor cores [12]. The performance of this parallel approach is often limited by the cost of inter-core communication and cache coherence traffic [15]. Assigning agents that interact frequently to different cores creates high communication overhead, as shown in Figure 1(a). This overhead can cancel out any speed gains from

parallelization. This is a well-known NP-hard problem in parallel computing, referred to as the mapping problem [4].

We address this mapping problem by modeling the agent system as a graph partitioning task. Agents are represented as nodes in a graph, and the communication frequency between agents determines the weight of the edges. The goal is to partition this graph in a way that minimizes the total weight of the edges that are cut [30]. This approach effectively groups frequently interacting agents onto the same core, which improves data locality and reduces communication traffic.

We solve this partitioning problem using spectral clustering, as shown in Figure 1(b). This technique identifies well-connected clusters by analyzing the eigenvectors of the graph's Laplacian matrix [24]. Since the standard algorithm has a time complexity of $O(n^2)$, which is too slow for large graphs, we employ the Nyström method, a much faster approximation. The speed of the Nyström method allows us to dynamically re-partition agents during the simulation as their communication patterns change [7, 22]. While other tools like the METIS library are fast [16], we chose spectral clustering because it is effective at finding the community structures common in agent interaction graphs [17]. To our knowledge, FABS is the first financial agent-based model to use this dynamic optimization strategy.

3.4 Architectural Comparison

To ground our design choices, we present a direct architectural comparison between FABS and MAXE, a financial ABM simulator [3]. Both platforms are discrete-event simulators, but they differ fundamentally in their design philosophy, particularly regarding extensibility and parallelism for accelerating a single, large-scale workload. Table 1 summarizes these key architectural distinctions.

The most fundamental difference lies in their approach to parallelism. MAXE is structured for coarse-grained parallelism, which is effective for running multiple independent simulations concurrently (e.g., for parameter studies) but does not accelerate a single, large-scale workload. FABS, in contrast, is built for fine-grained parallelism, which partitions the workload of a single simulation across all available cores.

This core design choice leads to the other architectural differences. For instance, MAXE's sequential execution for a single run is compatible with a centralized message queue, but this design would create a significant bottleneck for the parallel execution in FABS. Therefore, FABS employs decentralized, per-component message queues to maintain scalability. Similarly, for extensibility, MAXE's reliance on runtime casting and message identifiers creates a tightly coupled system that can impede the rapid integration of new AI agents. FABS addresses this by adopting a callback system that decouples interacting components and uses the type system to check for errors at compile time, improving both robustness and modularity. Finally, the performance optimization in FABS is a direct solution to the challenges introduced by its fine-grained parallel design, using adaptive clustering to manage the inter-core communication overhead that MAXE's sequential model does not encounter or address.

4 Evaluation

We conduct a series of experiments to evaluate FABS on two primary aspects: (1) its computational performance and scalability against a state-of-the-art simulator, and (2) its effectiveness as a data synthesis engine for producing realistic market phenomena that establish it as a digital twin environment.

4.1 Experimental Setup

The evaluation focuses on the performance of the core simulation architecture. To this end, we employ agents with simple, predictable, rule-based logic derived from an established financial model. This approach ensures that measured performance differences are directly attributable to the simulation engine, rather than being confounded by the computational overhead of complex AI models. Using AI-driven agents would obscure the analysis of the underlying platform, as performance would be influenced by the specific workload of the AI model rather than the simulator's efficiency.

In the absence of a standard benchmark suite for agent-based model (ABM) simulators, we adopt a large-scale fire sale scenario based on the model of the 2010 Flash Crash by Gao et al. [13]. This scenario serves as a challenging and realistic test case, providing the high volume of agent interactions necessary for a rigorous evaluation. The simulation comprises a single exchange agent managing a price-time order book, a population of four distinct types of trading agents, and a single 'spiking' agent designed to introduce a temporary liquidity shock that triggers the fire sale dynamics.

We benchmark FABS against MAXE [3], the state-of-the-art C++ platform for financial ABM, as our framework is designed specifically to overcome its core single-threaded architectural limitation.

GPU-based platforms like JAX-LOB [11] offer massive parallelism, but a direct comparison with FABS is inappropriate. The two frameworks have fundamental differences in hardware, parallelization strategy, and intended application. JAX-LOB is a highly specialized simulator built for GPUs. It runs many independent Limit Order Book simulations at once using inter-simulation parallelism, making it ideal for reinforcement learning. In contrast, FABS is a general-purpose framework designed for multi-core CPUs. It uses intra-simulation parallelism to accelerate a single, complex agent-based simulation.

Given these distinct approaches to hardware, parallelism, and problem scope, MAXE remains the most relevant benchmark.

To ensure a fair and direct comparison, we implemented identical agent logic and behavioral parameters across both platforms. All experiments were conducted on a machine equipped with an Intel Core i7-10700 8-core (16-thread) CPU and 16GB of RAM, running a Linux operating system. Both simulators were compiled with GCC using the `-O3` flag for maximum optimization. To provide a precise benchmark of the execution engines, the reported runtimes are the average of three trials and measure only the core simulation loop, excluding any initialization overhead.

4.2 Performance and Scalability

Our evaluation compares the parallel performance of FABS against the single-threaded execution of MAXE. This is because FABS is designed for intra-simulation parallelism, accelerating a single, complex model run, a capability MAXE's architecture does not address.

Table 1: Architectural Design Comparison: FABS vs. MAXE

Architectural Feature	MAXE [3]	FABS (Our Approach)
Core Execution Model	Sequential per simulation. A single thread executes all component logic, creating a performance bottleneck for large models.	Parallel per simulation. Component logic is distributed across multiple worker threads, enabling concurrent execution to accelerate a single model.
Parallelism Support	Coarse-grained only. Useful for parameter sweeps, but cannot accelerate a single simulation. This model leads to inefficient hardware use, as parallelism is limited by the number of simulations, not available cores.	Fine-grained (component-level). Designed to accelerate a single, large-scale simulation by parallelizing its internal workload, fully utilizing all available CPU cores for the primary research task.
Messaging Paradigm	Centralized global message queue. The kernel is a bottleneck responsible for dispatching all messages.	Decentralized per-component queues. Each component manages its own incoming messages, eliminating a central bottleneck and improving scalability.
Message Handling	Runtime casting with identifiers. Relies on dynamic casting and hardcoded message IDs, making it brittle and error-prone.	Type-safe function callbacks. Uses C++ lambdas for decoupled, compile-time checked interactions, enhancing robustness and extensibility.
Performance Optimization	None for a single simulation. Performance is static and depends on raw C++ execution speed.	Clustering-based. Dynamically re-maps components to cores using spectral clustering to minimize communication overhead and adapt to the simulation's changing dynamics.

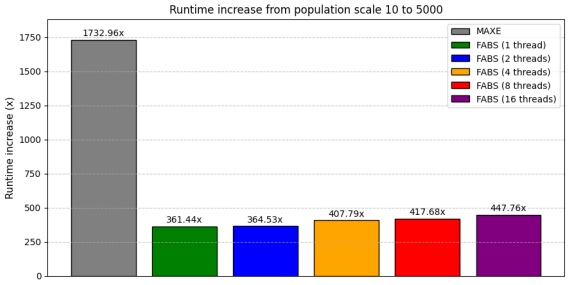


Figure 2: Relative runtime increase from population scale 10 to 5000. FABS's runtime grows at a much slower rate than MAXE's.

A direct, tick-by-tick numerical comparison is deliberately avoided, as forcing FABS's parallel execution to match MAXE's sequential event path would require synchronization overhead that nullifies the performance benefits. Instead, we validate the comparable quality of results by confirming that FABS successfully reproduces key emergent market phenomena, specifically fat-tailed returns and volatility clustering. This approach ensures the simulation's fidelity is maintained while achieving the desired speed-up.

As shown in Figure 3, FABS consistently outperforms MAXE across all agent population scales, with the performance gap widening as the simulation size increases. At the largest scale with 16 threads, FABS is 12.52x faster, making large-scale experiments computationally feasible. Notably, FABS is significantly faster even in a single-threaded configuration due to superior architectural design. It employs an $O(1)$ hashmap for agent lookups and decentralized message queues, which avoids the bottlenecks present in MAXE's $O(\log N)$ binary search and centralized global priority queue. The full runtime data is available in Table 2.

Table 3 details the internal scalability of FABS, which shows positive, albeit sub-linear, scaling with the number of cores, achieving up to a 1.53x speed-up with 16 threads. This sub-linear performance is an expected consequence of our benchmark's star-like communication topology, where all trading agents interact with a single exchange agent. This structure inevitably creates a computational bottleneck on the thread managing the central exchange, limiting overall throughput as more parallel resources are added. We anticipate that market models with more decentralized structures, such as those with multiple exchanges, would distribute the workload more evenly and thus exhibit stronger parallel speed-up.

4.3 Efficacy of Adaptive Clustering Optimization

We next evaluate the performance impact of our adaptive clustering optimization. Experiments were run on the 16-thread FABS configuration, comparing the baseline (no clustering) to static clustering (re-clustering at a fixed interval) and adaptive clustering (with an exponential backoff threshold of $k = 0.5$).

The results, shown in Figure 4 and Figure 5, reveal a clear performance trade-off dependent on scale. For smaller populations, the computational overhead of constructing the interaction graph and performing spectral clustering outweighs the communication benefits, leading to a net performance loss. This indicates that the optimization is not suitable for small-scale or less interaction-heavy simulations.

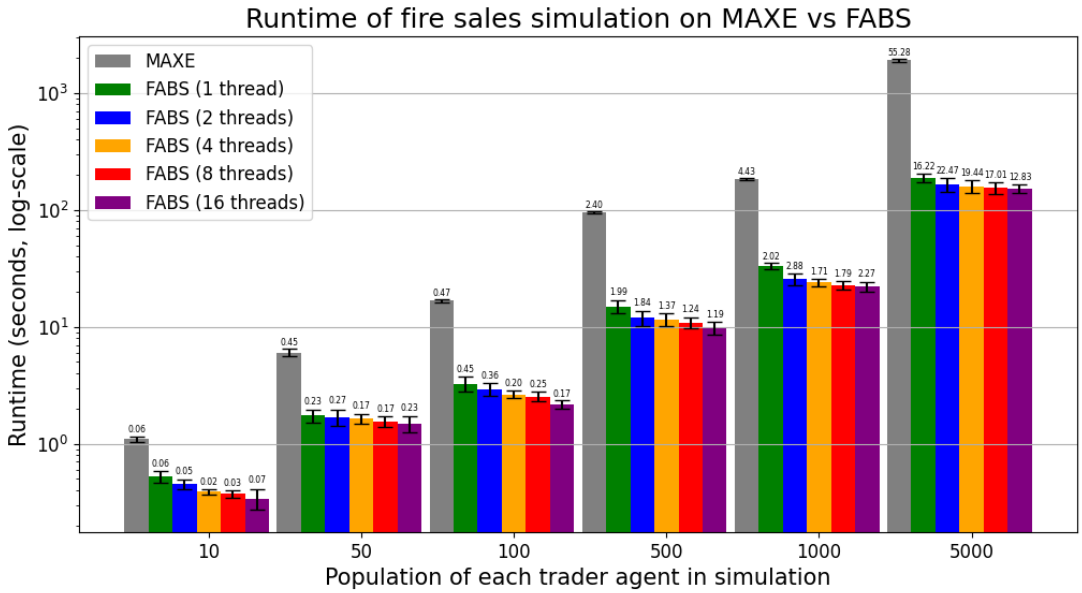


Figure 3: Runtime vs. Population Scale. FABS demonstrates significantly better scalability compared to the single-threaded MAXE.

Table 2: Runtime (s) and Speed-up of FABS vs. MAXE across different thread counts (T). FABS demonstrates consistent performance gains that increase with agent population and thread count.

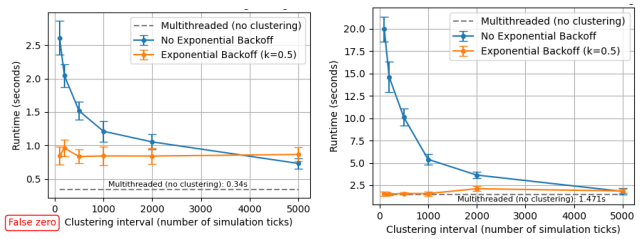
Population Scale (n)	MAXE Runtime (s)	FABS (T=1)		FABS (T=2)		FABS (T=4)		FABS (T=8)		FABS (T=16)	
		Runtime	Speed-up	Runtime	Speed-up	Runtime	Speed-up	Runtime	Speed-up	Runtime	Speed-up
10	1.10	0.52	2.12×	0.45	2.44×	0.39	2.82×	0.37	2.97×	0.34	3.24×
50	6.03	1.74	3.47×	1.70	3.55×	1.64	3.68×	1.55	3.89×	1.47	4.10×
100	16.77	3.27	5.13×	2.94	5.70×	2.64	6.36×	2.54	6.60×	2.17	7.73×
500	95.52	14.98	6.38×	11.90	8.03×	11.60	8.23×	10.88	8.78×	9.78	9.77×
1000	184.70	33.10	5.58×	25.66	7.20×	24.06	7.68×	22.82	8.09×	22.06	8.37×
5000	1906.26	187.95	10.14×	164.04	11.63×	159.04	11.98×	154.54	12.34×	152.24	12.52×

Table 3: Speed-up of multithreaded FABS relative to its single-threaded (T=1) execution.

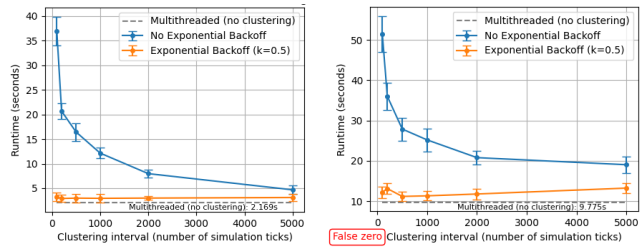
Population Scale	T=2 Speed-up	T=4 Speed-up	T=8 Speed-up	T=16 Speed-up
10	1.16×	1.33×	1.41×	1.53×
50	1.02×	1.06×	1.12×	1.18×
100	1.11×	1.24×	1.29×	1.51×
500	1.26×	1.29×	1.38×	1.53×
1000	1.29×	1.38×	1.45×	1.50×
5000	1.15×	1.18×	1.22×	1.23×

However, as the agent population grows, this trade-off inverts. At the largest scales, the benefits of reducing inter-core communication become significant enough to offset the computational cost. For instance, at a population scale of 5000 agents, the adaptive clustering configuration achieves a speed-up of 1.07x over the unoptimized parallel baseline.

Our benchmark scenario has a star-like communication graph, where all agents communicate only with a single central exchange agent. This topology represents the centralized microstructure of traditional financial exchanges, which creates a computational bottleneck on the exchange component. We deliberately chose this scenario as it represents the most constrained and challenging case for any parallelization and optimization strategy. The resulting speed-up, although modest, thus confirms that our dynamic topological approach provides a measurable benefit even in a worst-case parallel topology. As we note, a more decentralized model, such as one with multiple exchanges, would be better suited for demonstrating the full benefit of this optimization.

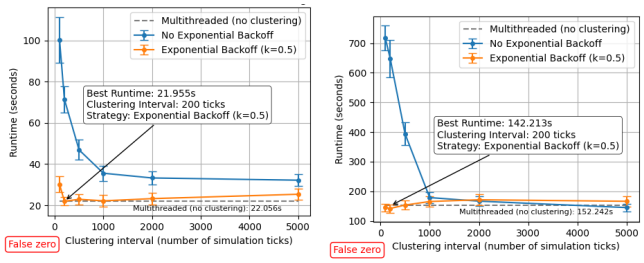


(a) Population size = 10 (b) Population size = 50



(c) Population size = 100 (d) Population size = 500

Figure 4: Effect of clustering optimization for small to medium population scales. The optimization overhead leads to a net performance loss.



(a) Population size = 1000 (b) Population size = 5000

Figure 5: Effect of clustering optimization for large population scales. Adaptive clustering (Exponential Backoff) outperforms the baseline.

4.4 Validation as a Data Synthesis Engine

To validate FABS as a high-fidelity digital twin of a financial market, we assess its ability to reproduce well-known emergent phenomena from simple agent rules. The primary goal is to establish a trustworthy foundation, confirming that the simulation engine can accurately mirror the complex dynamics of its real-world counterpart. As a first step, we subject the digital twin to a high-stress fire-sale scenario, designed to replicate the conditions of a flash crash. The resulting price trajectory, shown in Figure 6, exhibits a sharp, non-equilibrium price drop. This behavior is an intended and crucial validation, confirming that the model correctly captures the high-volatility dynamics of a market in crisis.

Beyond mirroring the qualitative market trajectory, the synthetic time series generated by FABS quantitatively reproduces two fundamental stylized facts of real financial markets [8]. First,

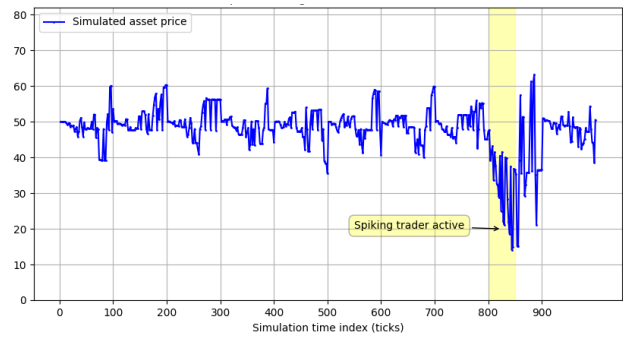
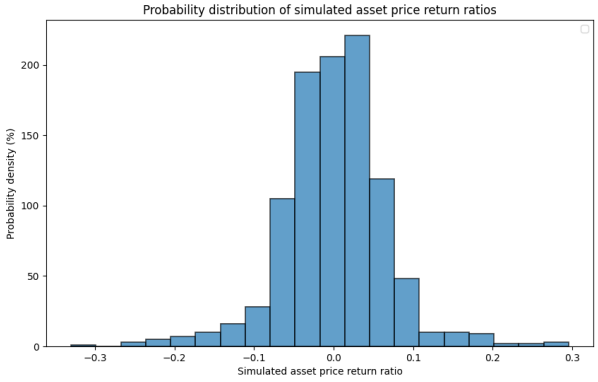
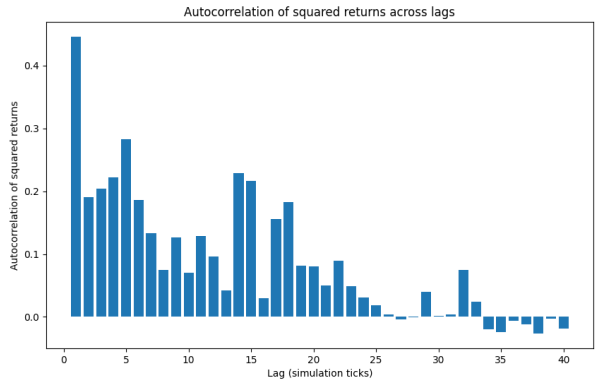


Figure 6: Simulated asset price history during the fire sale scenario.



(a) Fat-tailed distribution of asset returns.



(b) Autocorrelation of squared returns.

Figure 7: FABS successfully reproduces two key stylized facts of financial time series, validating its use for data synthesis.

the distribution of asset returns in Figure 7(a) exhibits significant fat tails (kurtosis = 4.25), capturing the real-world tendency for extreme events to occur more frequently than predicted by a Gaussian

model. Second, Figure 7(b) shows the slow decay in the autocorrelation of squared returns, the defining characteristic of volatility clustering. By successfully replicating these emergent properties, FABS demonstrates its potential as a digital twin environment for generating qualitatively realistic synthetic data to explore complex what-if scenarios.

5 Conclusion and Future Work

This work introduces FABS, an open-source, high-performance engine for constructing and operating large-scale agent-based financial digital twins. The engine's fine-grained parallel architecture fully utilizes the potential of modern multi-core processors for a single simulation, while its dynamic graph-based optimization is specifically designed to meet the computational demands of hosting many interacting AI agents. Our results show a runtime speed-up of up to 12.52x over the state-of-the-art MAXE platform, an improvement that removes a critical bottleneck, enabling the creation of more complex and realistic digital twins for studying market-wide emergent phenomena. By successfully reproducing key stylized facts of financial markets, we have also established FABS's fidelity as a reliable virtual laboratory for testing AI strategies and synthesizing realistic training data. FABS provides the foundational tool for building and analyzing digital twin environments required for financial AI systems.

Future work will focus on enhancing the digital twin's ecosystem and integrating it with live data. We plan to increase fidelity by incorporating complex entities like central banks to study systemic risk [26], and by populating the environment with adaptive learning agents to test AI strategies against emergent dynamics. Furthermore, integrating FABS with real-time market data will create the tight coupling necessary for a true digital twin, allowing the simulation to be continuously informed by its live counterpart for more powerful predictive analysis.

Acknowledgments

The support of the United Kingdom EPSRC (grant numbers UKRI256, EP/V028251/1, EP/N031768/1, EP/S030069/1, and EP/X036006/1), Altera, Intel, AMD, SiliconFlow is gratefully acknowledged. Google Gemini Pro was utilized for polishing the language and proofreading of this manuscript.

References

[1] Muhammad Anshari, Mohammad Nabil Almunawar, and Masairol Masri. 2022. Digital twin: Financial technology's next frontier of robo-advisor. *Journal of risk and financial management* 15, 4 (2022), 163.

[2] Stefano Battiston, J Doyne Farmer, Andreas Flache, Diego Garlaschelli, Andrew G Haldane, Hans Heesterbeek, Cars Hommes, Carlo Jaeger, Robert May, and Marten Scheffer. 2016. Complexity theory and financial regulation. *Science* 351, 6275 (2016), 818–819.

[3] Peter Belcak, Jan-Peter Calliess, and Stefan Zohren. 2021. Fast agent-based simulation framework with applications to reinforcement learning and the study of trading latency effects. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*. Springer, Virtual Event, 42–56.

[4] Shahid H Bokhari. 1981. On the mapping problem. *IEEE Transactions on computers* 100, 3 (1981), 207–214.

[5] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. 2020. ABIDES: Towards high-fidelity multi-agent market simulation. In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, Virtual Event, 11–22.

[6] François Challet. 2018. Opinion: To build trustworthy AI, here is what financial services need. *Proceedings of the National Academy of Sciences* 115, 50 (2018), 12571–12573.

[7] Anna Choromanska, Tony Jebara, Hyungtae Kim, Mahesh Mohan, and Claire Monteleoni. 2013. Fast Spectral Clustering via the Nyström Method. In *Lecture Notes in Computer Science*. Springer, Berlin, 367–381. doi:10.1007/978-3-642-40935-6_26

[8] Rama Cont. 2001. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance* 1 (02 2001), 223–236. doi:10.1080/713665670

[9] Gennaro Cordasco, Vittorio Scarano, and Carmine Spagnuolo. 2018. Distributed MASON: A scalable distributed multi-agent simulation environment. *Simulation Modelling Practice and Theory* 89 (2018), 15–34.

[10] J Doyne Farmer and Duncan Foley. 2009. The economy needs agent-based modelling. *Nature* 460, 7256 (2009), 685–686. doi:10.1038/460685a

[11] Sascha Yves Frey, Kang Li, Peer Nagy, Silvia Sapora, Christopher Lu, Stefan Zohren, Jakob Foerster, and Anisoara Calinescu. 2023. JAX-LOB: A GPU-Accelerated limit order book simulator to unlock large scale reinforcement learning for trading. In *Proceedings of the Fourth ACM International Conference on AI in Finance*. ACM, Brooklyn, NY, 583–591.

[12] Richard M Fujimoto. 2000. *Parallel and distributed simulation systems*. John Wiley & Sons, New York City.

[13] Kang Gao, Perukrishnen Vytelingum, Stephen Weston, Wayne Luk, and Ce Guo. 2024. High-Frequency Financial Market Simulation and Flash Crash Scenarios Analysis: An Agent-Based Modelling Approach. *Journal of Artificial Societies and Social Simulation* 27 (01 2024), 1–39. doi:10.18564/jasss.5403

[14] Martin D. Gould, Mason A. Porter, Stacy Williams, Mark McDonald, Daniel J. Fenn, and Sam D. Howison. 2013. Limit order books. *Quantitative Finance* 13 (11 2013), 1709–1742. doi:10.1080/14697688.2013.803148

[15] Mark D. Hill and Michael R. Marty. 2008. Cache coherence techniques for multicore processors. *Commun. ACM* 51, 12 (01 2008), 72–77.

[16] George Karypis and Vipin Kumar. 1997. *METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*. Technical Report. University of Minnesota Twin Cities.

[17] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[18] Abdelhafid Mazouz, Sid-Ahmed-Ali Touati, and Denis Barthou. 2011. Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of SPEC OMP applications on intel architectures. In *2011 International Conference on High Performance Computing & Simulation*. IEEE, Istanbul, Turkey, 273–279. doi:10.1109/hpcsim.2011.5999834

[19] Nelson Minar, Roger Burkhart, Chris Langton, and Manor Askenazi. 1996. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations.

[20] Michael J North, Nicholson T Collier, Jonathan Ozik, Eric R Tatara, Charles M Macal, Mark Bragen, and Pamela Sydelko. 2013. Complex-adaptive-systems modeling with Repast Simphony. *Complex adaptive systems modeling* 1, 1 (2013), 1–26.

[21] Ioannis Passas. 2025. From Digital Twins to Digital Triplets in Economics and Financial Decision-Making. *Encyclopedia* 5, 3 (2025), 87.

[22] Farhad Pourkamali-Anaraki. 2020. Scalable Spectral Clustering With Nyström Approximation: Practical and Theoretical Aspects. *IEEE Open Journal of Signal Processing* 1 (2020), 242–256. doi:10.1109/OJSP.2020.3039330

[23] Paul Richmond, Robert Chisholm, Peter Heywood, Mozhgan Kabiri Chimeh, and Matthew Leach. 2023. FLAME GPU 2: A framework for flexible and performant agent based simulation on GPUs. *Software: Practice and Experience* 53, 8 (2023), 1659–1680.

[24] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.

[25] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. Market making with reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. ACM, Stockholm Sweden, 1097–1105.

[26] Anjan V. Thakor. 2015. The Financial Crisis of 2007–2009: Why Did It Happen and What Did We Learn? *Review of Corporate Finance Studies* 4 (04 2015), 155–205. doi:10.1093/rcfs/cfv001

[27] The MAXE Team. 2020. GitHub - maxe-team/maxe: Multi-Agent eXchange simulator developed at Oxford-Man Institute. https://github.com/maxe-team/maxe

[28] Seth Tisue and Uri Wilensky. 2004. NetLogo: Design and Implementation of a Multi-Agent Modeling Environment.

[29] Arthur Turrell. 2016. Agent-based models: understanding the economy from the bottom up. *Bank of England Quarterly Bulletin* 56 (01 2016), 173–188.

[30] Ulrike von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and Computing* 17 (08 2007), 395–416. doi:10.1007/s11222-007-9033-z

[31] Lili You and Mingli Zhu. 2023. Digital twin simulation for deep learning framework for predicting solar energy market load in trade-by-trade data. *Solar Energy* 250 (2023), 388–397.