Scholar Publishing Group
International Journal of Neural Network
https://doi.org/10.38007/NN.2026.050104
ISSN 2790-2692 Vol. 5, Issue 1: 33-41

Research on the Design and Consistency Verification of Deterministic Replay Mechanism for Event-Driven Engines in Quantization Backtesting Platforms

Ren Yan

Guangzhou College of Commerce, School of Information Technology & Engineering, Guangzhou, 511363, China

Keywords: Quantitative backtesting; Event-driven engine; Deterministic replay; Consistency verification; Replay auditing; Low-latency system

Abstract: Quantitative backtesting platforms serve as "offline decision-making testbeds" for strategy evaluation, parameter exploration, and risk verification. However, traditional backtesting systems suffer from issues such as event sequence drift, similar matching semantics, unstable concurrent scheduling, and backtesting results failing to truly reflect market paths. This results in significant deviations in strategy returns, drawdowns, and transaction details. To address this pain point, this paper presents a technical solution for deterministic replay using an event-driven engine on a quantitative backtesting platform. This framework uses a unified event clock, sequence number adjudication, state snapshots, incremental logs, matching semantic encapsulation, and audit verification chain as its core, mapping changes in market data, orders, transactions, fees, risk control triggers, and account status into a recordable, replayable, and verifiable event stream. Based on publicly available research on event replay benchmark data and statistical results from high-performance event stream systems, the system is designed from four aspects: event modeling, execution consistency, low-latency transmission, and deviation measurement. Furthermore, a backtesting replay consistency metric is constructed. Analysis shows that deterministic replay can not only improve the interpretability and accountability of backtesting results, but also greatly improve the engineering reliability of parameter optimization, regression testing, and strategy pre-deployment verification.

1 Introduction

Quantitative strategy research, implementation, and risk control review all rely on historical backtesting results. However, the market microstructure has obvious event density and state coupling. Small changes such as order book updates, transaction returns, order cancellation confirmations, fee deductions, and risk control threshold triggering will directly affect changes in account equity. As long as there are slight differences in the event arrival order, matching time, concurrent execution, or state recovery of the backtesting platform, the final profit curve will show path dependence deviation. Therefore, how to ensure that the same historical dataset, the same

strategy code, and the same parameters maintain a consistent state development after multiple executions has become a problem of building a high-quality quantitative foundation. [1][3]

In the past three years, the quantitative research community has paid more and more attention to the consistency of reproducible experiments and deployments. FinRL Contests uses standard environments, consistent evaluation protocols and open-source starter kits as benchmark conditions to set them as the standard for financial reinforcement learning evaluation. FinRL-X goes further and proposes that research, backtesting and broker execution should maintain consistent interfaces and execution semantics to reduce the gap between offline evaluation and online implementation. [3][5] PredictionMarketBench makes historical order books, transactions and settlement events into portable episodes, and then uses a deterministic event-driven simulator to achieve repeatable comparisons, indicating that deterministic replay is becoming the main infrastructure for quantitative evaluation from a debugging tool. [4]

This paper discusses the engine design issues for a quantitative backtesting platform, using event-driven, deterministic replay, and consistency verification as its logical framework. It primarily addresses the following questions: first, where does the nondeterminism in the backtesting engine originate? Second, how to build a unified event model to support stable replay? Third, how to find a balance between low latency and high throughput while maintaining sufficient audit information? Fourth, how to quantify the consistency of results to support regression testing and deployment review?

2. Current Status Analysis of the Research Topic

Current research primarily focuses on three areas. The first category is research on recording and replaying for distributed system debugging. Galstyan proposed aiRR, suggesting that embedding recording logic into message-driven applications reduces synchronous recording overhead by obtaining necessary sequence information from communication boundaries. He concluded that application-integrated replay is more suitable for high-concurrency service systems than black-box tracing. This approach directly impacts quantitative backtesting platforms, as order-driven logic inherently possesses clear message boundaries and a centralized state machine kernel, enabling replayability even when thread-level details cannot be traced.

The second category focuses on research into high-performance event stream processing systems. Dorier et al. proposed that Mofka has significantly lower event processing overhead than traditional messaging systems in high-performance computing. Their results show that batch size, event size, and the number of partitions affect end-to-end latency, and a well-designed data plane can control the overhead per event to the microsecond level. For quantized backtesting, the replay system must faithfully reproduce the order of historical events without allowing logs and the transmission path itself to become new performance bottlenecks. Therefore, overhead control at the event stream layer is particularly important.

The third category is integrated framework research centered on quantitative research platforms. Research on FinRL Contests, FinRL-X, PredictionMarketBench, and AI platforms for trading simulation all emphasize the importance of unified data specifications, standardized evaluation, execution realism, and audit output. In addition, research on TRADES and deep order book prediction shows that high-frequency markets have strong data time dependence and microstructure sensitivity. Even if the order of events and matching assumptions are roughly considered fixed, it will reduce the backtesting results' understanding of the real market. [7][8]

In summary, existing research has addressed sub-questions such as how to replay data with low overhead, how to efficiently process event streams, and how to establish reproducible experimental platforms. However, for quantitative backtesting platforms, there is still no solution that unifies

historical market data replay, matching semantics, account status recovery, risk control verification, and audit trail into a single event-driven engine.

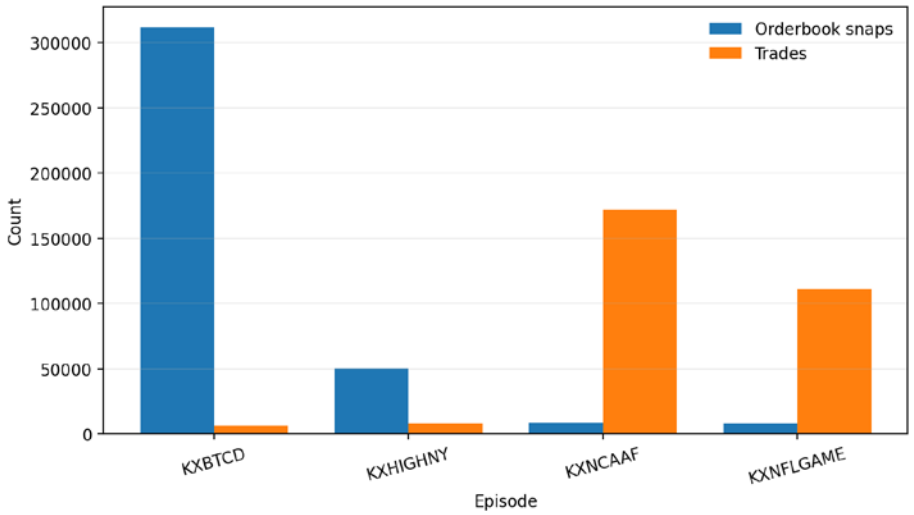

Figure 1 shows the order book snapshots and transaction event sizes for different episodes in the public event-driven replay benchmark.

Figure 1 is redrawn based on publicly available data from PredictionMarketBench. As can be seen from the figure above, the number of order book snapshots and the number of transactions vary greatly in different episodes. In the cryptocurrency scenario, the order book updates faster, while in sports events, it shows a higher concentration of transactions. Therefore, it can be concluded that quantitative backtesting engines cannot simply use a fixed step size to simulate event progression. They should drive state transitions according to the real market structure and provide the same scheduling scheme for different market structures. [4]

3. Raise questions

Based on the current engineering issues of the platform, deterministic replay for quantization backtesting platforms will encounter at least the following four problems.

First, there's the randomness of the event sequence. Traditional backtesting frameworks typically use timestamps for coarse-grained sorting. When several market updates, order cancellation reports, or transaction reports have the same timestamp, the system implicitly determines the execution order using thread scheduling, data source order, or container traversal order. This results in different state paths during repeated executions, which is more pronounced in high-frequency strategies and market-driven strategies.

Second, there is a disconnect between matching semantics and market semantics. Many backtesting systems use "execute at the next bar" or "execute at the snapshot price" to simplify the execution logic, which cannot truly reflect the constraints of microstructures such as maker/taker, queue priority, partial execution, and fee deduction. PredictionMarketBench's research found that if the modeling of fees, execution, and settlement paths is not considered, the strategy will be overestimated during backtesting. [4]

Third, the engineering costs of logging and state recovery are too high. If a complete snapshot is saved for each state, it will not only put a lot of pressure on storage, but also greatly increase the latency of backtesting. If only the last account equity value is saved, detailed debugging and difference finding will not be possible. How to find a compromise between snapshots and

incremental logs is the main contradiction between availability and performance. [1], [2]

Fourth, backtesting results lack consistency criteria. Most platforms only compare cumulative returns, ignoring indicators such as holding paths, fees, transaction rates, drawdown trajectories, and risk control trigger sequences. When strategies have similar returns but different risk exposure paths, simple return comparisons cannot meet the requirements of institutional-level audits. Therefore, multiple consistency measures should be applied to event flows and account status.

Table 1 Comparison of Design Dimensions Related to Deterministic Replay

Design dimension	Conventional engine	Replay requirement	Proposed mechanism
Event ordering	Timestamp only	Stable tie-breaking	Time + sequence + partition key
Matching semantics	Price shortcut	Execution realism	Maker/taker queue model
State recovery	Full rerun	Fast restore	Snapshot + incremental log
Cost accounting	End-period summary	Bar-level reconciliation	Per-fill fee and slippage log
Parallel execution	Thread-dependent	Deterministic scheduling	Single logical clock
Audit output	Equity only	Traceability	Event hash and invariant checks

Table 1 shows that deterministic replay is not simply saving the logs and running it again; it requires ensuring strict consistency in all aspects, including event sequencing, matching semantics, state recovery, cost calculation, and audit output. Especially for institutional-level quantitative backtesting platforms, without stable same-second adjudication rules and transaction-by-transaction cost logs, even similar revenue curves do not necessarily indicate that the execution path can be repeated.

4. Problem Solving/Strategies

This paper presents a deterministic replay technology architecture guided by a quantitative backtesting platform to address the above issues. The core idea is to build upon a unified event model, and on this basis, connect data access, event sorting, matching execution, account evolution, snapshot recovery, and audit verification into a reusable pipeline.

4.1 Unified Event Clock and Stable Sequencing Strategy

The system abstracts all inputs into events e_i, including market data, order requests, fill reports, risk triggers, fee accruals, and settlements. Each event has a ternary sorting key: physical time t_i, source sequence number s_i, and partition key p_i. For events at the same timestamp, the order of thread scheduling is no longer followed; instead, a stable decision is made using the sequence number and partition key, ensuring that multiple runs of events achieve a complete and ordered result.

$$e_i \prec e_j \Leftrightarrow (t_i, s_i, p_i) <\text{lex} (t_j, s_j, p_j) \quad (1)$$

Equation (1) gives the relationship of event sorting. It means that physical time is compared first, then the same source sequence number is compared, and finally the partition key is compared. If the

sort key in the original log remains unchanged, then the total order of events will not change during the playback process. It draws on the constraints of the event-driven replay benchmark on "fixed episode file + fixed configuration + fixed seed to obtain a definite trajectory". [4]

4.2 State Machine-Driven Account Evolution and Matching Execution

The engine uses a single logic clock state machine. Only one event is submitted to the account state transition function at any given time to prevent non-repeatable results from concurrent writes. Market events only update the order book view, order events enter the matching queue, and trade and cancellation events drive the synchronous updates of positions, cash, frozen margin, and fee accounts. This mechanism is consistent with the idea of "single-threaded core state machine and parallel parsing of peripherals" in message-driven systems, which locks the core state transition order while ensuring high-throughput peripheral processing. [1]

$$S_{k+1} = F(S_k, E_k, A_k) \tag{2}$$

In equation (2), the state of the account before the k-th event is represented by S_k, the external event by E_k, the internal action performed by the policy or matchmaker by A_k, and the state transition function by F(·). Because the state transition depends only on the explicit input and not on the thread scheduling, it is suitable for use in historical replay, regression testing, and difference localization.

4.3 Snapshot-Incremental Hybrid Recording Mechanism

To achieve a balance between recoverability and storage costs, this paper designs the system using periodic snapshots combined with event-by-event incremental logging. The system generates a snapshot of the account and order book only after processing N events or after a fixed time window. Within the window, only event hashes, order book differences, transaction confirmations, fee changes, and risk control flags are retained. When a replay request occurs, the system first loads the most recent snapshot and then applies incremental events sequentially, significantly reducing the cost of full replay.

$$T_r = T_1 + T_2 + T_3 \tag{3}$$

Equation (3) shows that the total replay time consists of snapshot loading time, incremental application time, and consistency verification time. Compared with pure full replay, the snapshot plus incremental approach changes the complexity of recovery from a linear relationship with the total history length to a relationship proportional to the "length of the event segment after the most recent snapshot", making it more suitable for high-frequency engineering scenarios such as parameter search and nighttime regression.

4.4 Low-latency event channels and batch control

The fidelity of the replay system cannot be achieved at the expense of all performance. Public statistics from Mofka show that low-overhead event systems have an order-of-magnitude advantage in single-event overhead compared to traditional message middleware, and the overhead curves also differ depending on the batch size, event size, and number of partitions. [2] Therefore, this paper argues that metadata should be processed separately from large-volume payloads, with fields required for sorting and auditing going through the control channel, and depth of the order book or large feature snapshots going through the data channel. Furthermore, an upper limit should be set on the batch size to prevent batch processing of large events from causing increased latency.

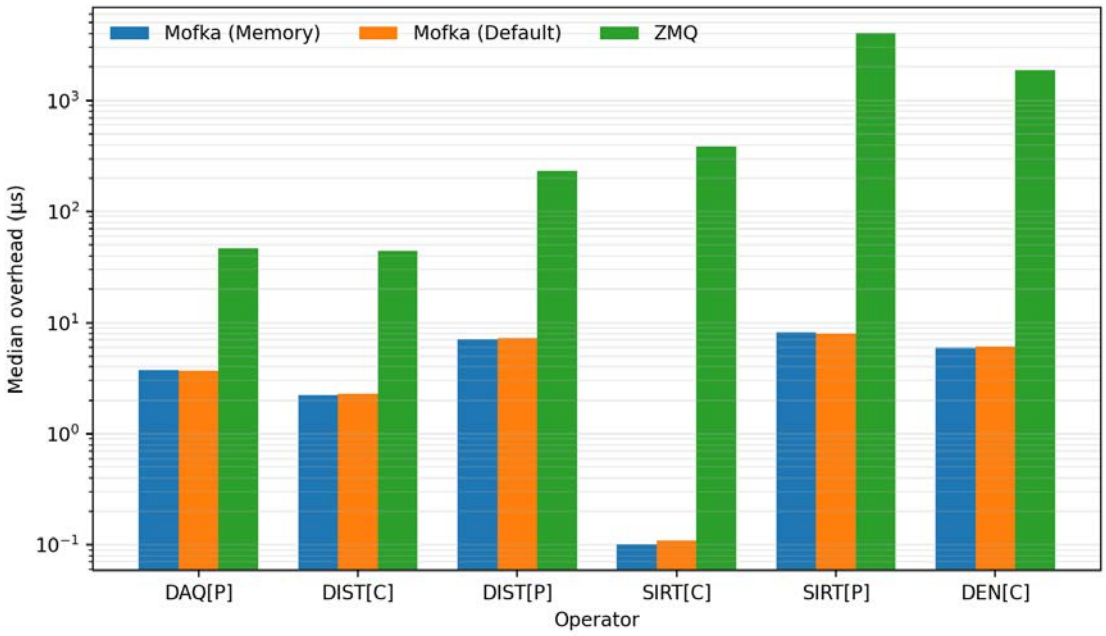

Figure 2 compares the single-event overhead of a high-performance event streaming system and a traditional messaging system under different operations.

Figure 2 is redrawn based on the median data in Table 3 of the Mofka paper and represented on logarithmic coordinates. As can be seen from Table 4-2, the median overhead of Mofka in operations such as DAQ, DIST, SIRT, and DEN is generally lower than that of ZMQ, and the advantage is more obvious in the case of large events. Therefore, if the replay engine uses low-overhead event channels and separates the sorted metadata from the large payload, it can reduce the overhead of log writing and replay execution while ensuring repeatability. [2]

4.5 Consistency Verification and Deviation Measurement

To prevent audit blind spots due to similar returns but different paths, this paper uses multidimensional consistency loss to compare the replay trajectory and the benchmark trajectory layer by layer, including positions, cash, expenses, drawdown, turnover rate and risk control trigger sequence.

$$L_r = \Sigma_k \|\hat{x}_k - x_k\|_1 + \alpha \Sigma_k (\hat{o}_k \neq o_k) \tag{4}$$

In equation (4), x_k is the account state vector at step k, o_k is the label of key discrete events (risk control trigger, margin call protection, forced liquidation signal, etc.), and the cap number is the result of replay. This loss considers both continuous state error and discrete path error, and can identify pseudo-consistency phenomena where returns are similar but event paths are different.

$$D = \beta_1 |P^b - P^r| + \beta_2 |M^b - M^r| + \beta_3 |F^b - F^r| \tag{5}$$

Equation (5) gives the backtest-replay deviation indices PnL, DD, Fill, and β_i for project acceptance. These indices can be used for version upgrade regression, matchmaker replacement testing, and difference verification before strategy deployment.

Table 2 Core Modules and Key Parameters of the Prototype Engine

Module	Parameter	Value	Function

Event clock	Ordering key	time+seq+partition	Stable total ordering
Replay store	Snapshot interval	50k events	Fast state restore
Matcher	Execution mode	maker/taker	Microstructure fidelity
Audit layer	Trace hash	SHA-256	Regression verification
Risk engine	Trigger granularity	per event	Path-level control
Scheduler	Core mode	single logical thread	Deterministic transition

Table 2 shows the core modules and key parameter settings of the prototype engine. As can be seen from the table, the parameters are not only designed for throughput, but also for replayability, auditability, and regressability. For example, the snapshot interval ensures recovery speed, event-by-event risk control ensures path consistency, and the single logical thread does not reduce performance; it simply places the non-deterministic parts in the external controllable modules.

5. Conclusion

This paper discusses the basic requirements of "results reproducible, paths interpretable, and execution auditable" under the quantitative backtesting platform, and conducts a systematic study on the deterministic replay of event-driven engines. According to the research, the nondeterminism of quantitative backtesting is not only caused by random seeds, but is more fundamentally related to factors such as the adjudication of events in the same second, the similarity of matching semantics, the difference in concurrent scheduling, the small log granularity, and the lack of unified audit indicators. [1][4][5]

To address the above issues, this paper proposes a scheme that integrates a unified event sorting key, a single logic clock state machine, a hybrid snapshot and incremental recording system, a low-overhead dual-channel event stream, and a multi-dimensional deviation metric. The paper also uses publicly available benchmark event scale statistics and overhead data from high-performance event systems to illustrate that the scheme has both engineering feasibility and platform-level scalability. [2], [4]

In terms of institutional quantitative research and development, the deterministic value of replay is not simply about repeatedly drawing the same return curve. More importantly, it provides a common evidence for strategy regression testing, parameter search, brokerage interface replacement, risk management migration, and compliance verification. Future research can combine deterministic replay with order book generation models, transaction cost learning models, and cross-market joint replay frameworks to transform the backtesting platform from a return evaluation tool into an integrated infrastructure for research, verification, and auditing. [7][8][9][10]

References

[1] *Liu, X., & Yang, D. (2025, March). LLM Data Strategy: Improving Data Availability and Efficiency. In Doctoral Symposium on Computational Intelligence (pp. 425-437). Singapore: Springer Nature Singapore.*

[2] *Zhang, Q. (2025, October). Application of Reinforcement Learning in Dynamic Advertising Content Generation. In 2025 2nd International Conference on Software, Systems and Information Technology (SSITCON) (pp. 1-5). IEEE.*

[3] *Zhang, Q. (2026). Security Improvement and Application of Identity and Access Management in Saas Platform.*

[4] *Wu, Y. (2025, October). Multi-Level Belief Rule Base Modeling Architecture and Intelligent Optimization Technology for Decision Support Systems. In 2025 2nd International Conference on Software, Systems and Information Technology (SSITCON) (pp. 1-8). IEEE.*

[5] *Chen, M. (2026). Research on Privacy-Preserving AI Model Training and Validation Methods Based on Federated Learning.*

[6] *Yiting Hong. Differentially Private High-Dimensional Business Data Publishing and Analysis Algorithm. International Journal of Business Management and Economics and Trade (2026), Vol. 7, Issue 1: 28-35.*

[7] *Xu, D. (2026). Analysis of the impact of video infrastructure optimization on large-scale content quality improvement.*

[8] *Pan, H. (2025, March). Research on Efficient Computing Model of Hartree Fock and Density Functional Theory Based on GPU Acceleration. In Doctoral Symposium on Computational Intelligence (pp. 485-496). Singapore: Springer Nature Singapore.*

[9] *Wu, W. (2025, June). Construction and optimization of intelligent gateway software management platform based on jenkins cluster management under cloud edge integration architecture in industrial internet of things. In International Conference on 6G Communications Networking and Signal Processing (pp. 633-645). Singapore: Springer Nature Singapore.*

[10] *Wu Y. Software Engineering Practice of Microservice Architecture in Full Stack Development: From Architecture Design to Performance Optimization[J]. 2025.*

[11] *Wu Y. Optimization of Generative AI Intelligent Interaction System Based on Adversarial Attack Defense and Content Controllable Generation[J]. 2025.*

[12] *Sun J. Quantile Regression Study on the Impact of Investor Sentiment on Financial Credit from the Perspective of Behavioral Finance[J]. 2025.*

[13] *Wang Y. Application of Data Completion and Full Lifecycle Cost Optimization Integrating Artificial Intelligence in Supply Chain[J]. 2025.*

[14] *Chen M. Research on Automated Risk Detection Methods in Machine Learning Integrating Privacy Computing[J]. 2025.*

[15] *Sun, Q. (2026). Research on a Robotic Natural Language Intelligent Decision-Making Framework Based on Large Language Models, Thinking Chain Reasoning, and Multi-Agent Collaboration.*

[16] *Liu, H. (2026). Research on Dynamic Price Prediction of E-commerce Based on Time Series Modeling.*

[17] *Yu, X. (2026). Strategy Models and Practical Research of Growth Marketing under the Background of Digital Transformation.*

[18] *Hou, Y. (2026). Research on Server Performance Stability Assurance Mechanisms during Cross-Generation Computing Platform Upgrades.*

[19] *Han, X. (2026). Research on Automotive Manufacturing Process Optimization Methods for Multi-Supplier Collaboration.*

[20] *Huang, J. (2025, August). Research on Multi-Model Fusion Machine Learning Demand Intelligent Forecasting System in Cloud Computing Environment. In 2025 2nd International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS) (pp. 1-7). IEEE.*

[21] *Huang, J. (2025, September). Performance Evaluation Index System and Engineering Best Practice of Production-Level Time Series Machine Learning System. In 2025 International Conference on Intelligent Communication Networks and Computational Techniques (ICICNCT) (pp. 01-07). IEEE.*

[22] Hou, Y. (2026). Research on Heterogeneous Server Upgrade Strategies and Resource Utilization Efficiency Oriented Toward Green Computing Objectives. Advances in Computer and Communication, 7(1).

[23] Zhang, C., Han, J., Zou, Y., Dong, K., Li, Y., Ding, J., & Han, X. (2024, April). Detecting the anomalies in LiDAR pointcloud. In WCX SAE World Congress Experience. SAE Technical Paper.

[24] Ding, J. (2025). Research On CODP Localization Decision Model Of Automotive Supply Chain Based On Delayed Manufacturing Strategy. arXiv preprint arXiv:2511.05899.

[25] Wu, Y. (2026). Federated Learning-based Algorithm Design for Privacy Preservation in Cross-domain Data Sharing. Engineering Advances, 6(1).

[26] Sun, J. (2025). Research on Business Data-driven Risk Prediction Methods Based on Machine Learning. Advances in Computer and Communication, 6(4).

[27] Yanchun Wang. (2025) Research on Enhancing ERP System Efficiency Through AI in Cross-border Supply Chain Environments. Advances in Computer and Communication, 6(5), 268-273.

[28] Zhou, Y. (2026). Energy efficiency and sustainability strategies for data centers. European Journal of Engineering and Technologies, 2(1), 46-53.

[29] Lu, Z. (2025). Design and Practice of AI Intelligent Mentor System for DevOps Education. European Journal of Education Science, 1(3), 25-31.

[30] Wu Y. Software Engineering Practice of Microservice Architecture in Full Stack Development: From Architecture Design to Performance Optimization[J]. 2025.