Scholar Publishing Group
Socio-Economic Statistics Research
https://doi.org/10.38007/SESR.2026.070105
ISSN 2790-2722 Vol. 7, Issue 1: 37-47

Research on Optimization of In-Memory Database Index Structure for Microsecond-Level Matching and Market Data Processing

Lizhi Wang

School of Economics, Wuhan Donghu University, Wuhan, 430000, Hubei, P.R. China

Keywords: financial transactions; in-memory database; index structure; tail latency; adaptive optimization

Abstract: In scenarios such as high-frequency trading, tick-by-tick market data distribution, and risk control linkage, in-memory databases are involved in order status storage, market snapshot queries, risk control key value operations, and important tasks such as time-series data statistics. Compared to financial trading loads, conventional internet OLTP business has stronger time locality, bursty updates, and P99 tail latency requirements. In recent years, research on adaptive radix trees, hybrid tree hash indexes, split in-memory indexes, and compact perfect hashes has continued to develop, providing new avenues for index improvement in microsecond-level trading systems. This paper, based on English literature from the past three years, designs an index optimization framework suitable for financial trading in-memory databases, focusing on four aspects: low tail latency, low write amplification, low memory consumption, and controllable range query performance. A composite design, mainly composed of upper-layer ordered routing, leaf-layer segmented hashing, cache-friendly node layout, version-aware concurrency control, and range rollback mechanism, achieves better performance in balanced point lookups, updates, and range scans. This paper reconstructs statistical graphs based on publicly available experimental results and provides a multi-objective optimization model and engineering implementation suggestions. The research results show that, for financial transaction loads, it is not necessary to choose only one type of tree or hash structure. It is better to determine the layered collaborative design method based on access distribution, key growth pattern and tail latency.

1. Introduction

Electronic trading in the capital markets has evolved from millisecond-level competition to microsecond-level competition. In key pathways such as matching engines, order routing, risk verification, and market data replay, databases are no longer consistently presented using traditional

disk-based relational databases. Instead, they increasingly favor high-performance data services that primarily utilize memory and support a mix of key-value storage and ordered retrieval workloads. For tasks such as order book updates, account position verification, risk control threshold lookup, and transaction queries within a time period, index structures are often more efficient than the computational logic itself. Recent studies have shown that index tree height, node location, concurrency control granularity, and remote access rounds directly impact average access time and tail latency, thereby affecting the stability and on-time delivery of the trading system.

Public research shows that traditional B+ trees have mature advantages in ordered range queries, but they are prone to problems such as write hotspots, network amplification and lock conflicts in the writing of monotonically increasing keys, the deployment of memory distribution, and the high-concurrency update path [2][4]. In contrast, adaptive radix trees such as ART use variable nodes and path compression to reduce the access overhead of tree height, and have better cache friendliness in read-intensive scenarios. Hybrid indexes combine the ordered leaf layer and hash leaf nodes of B-link trees to reduce monotonically inserted hotspots while ensuring scanning capability [1]. In the past two years, new indexes based on separate memory and RDMA network have raised the number of RTTs required for each access and the number of bytes moved by each node to the core of the design, making index optimization expand from a single-machine caching problem to a problem involving caching, network and concurrency together [4][7][9].

However, financial transaction load is not a general YCSB load. Order number, timestamp, transaction sequence number, account number and security code are all access methods of composite keys. In addition, in the transaction window, the traffic pulse will cause the system to suddenly rise from low load to near saturation, causing the P99 latency to deteriorate sharply. Existing studies have proposed improvement schemes from tree index, hash index, learning index and persistent memory index [3][5][6][8], but it is still necessary to conduct a comprehensive analysis to transform these research results into an implementable index framework suitable for microsecond-level matching and market data processing. This paper conducts a comprehensive study on the current status, problems and optimization strategies of in-memory database index structure.

2. Current Status Analysis of the Research Topic

2.1 Main technical routes for memory index optimization in the past three years

The first category is the improvement of tree structure. Blink-hash uses hash nodes in the leaf layer to alleviate the hot spot problem of monotonic insertion, and switches back to tree structure when scanning, achieving a balance between insertion, point lookup and range scan. SMART is a concurrency and caching strategy for reconstructing ART for the scenario of separate memory. It improves throughput and latency under write-intensive and read-intensive tasks by locking internal nodes, modifying leaf nodes in place, and writing and merging [2]. dex and deft also showed that logical partitioning, lightweight caching and cost-aware offloading can significantly improve the scalability of ordered indexes in remote memory [4][10].

The second category is the optimization of hash structure. For objects that use point queries, such as order status, account key value, and risk control tag, hash index has constant-level lookup efficiency. However, traditional scalable hash will generate a lot of synchronization costs and additional remote read and write when expanding. Outback uses the idea of dynamic minimum perfect hash to decouple the computationally intensive part and the storage-intensive part, and realizes one round-trip data

access on the RDMA network [7]. Shard is optimized around the correctness of duplicate keys, lazy expansion, synchronization frequency control, etc., and has better throughput than the existing DM hash index under different YCSB loads [9]. Sphinx uses compact perfect hash as the core, pursues the synchronous reduction of query, update and memory usage, and believes that low memory usage is the premise of low tail latency.

The third approach is to optimize learned indexes and persistent indexes. Although learned indexes have good lookup times for stable test sets, they still have the cost of retraining and unpredictable tail latency in transaction scenarios with frequent updates and large key distribution drift. Related research has also improved learned indexes from aspects such as disk adaptation and serial correlation compression [6]. Systems like FluidKV and SPIRIT, which are designed for persistent memory or ultra-high-speed storage, have demonstrated that the high-performance write index in the foreground and the low-occupancy persistent index in the background work together to maintain recoverability while controlling DRAM costs [5][6]. The results have direct guiding significance for the question of whether transaction memory banks need hot and cold tiered indexes.

Table 1 shows the mapping relationship between low-latency financial transaction load and indexing requirements, providing a basis for subsequent experiments and optimizations.

Workload	Key pattern	Primary access	Latency focus	Preferred index trait
Order book update	Monotonic + composite	Insert/Update	P99 write	Hotspot mitigation
Risk rule lookup	Random	Point read	P50/P99 read	Low tree height / O(1)
Position check	Random + skewed	Read/Update	Tail stability	Cache locality
Market replay	Time-ordered	Range scan	Throughput	Ordered leaves
Session recovery	Bulk sequential	Scan/Rebuild	Recovery time	Compact metadata

Table 1 shows that in a financial trading environment, there is no "universal index" that can achieve optimal results for point queries, monotonic writes, and long-range scans. Order book updates prioritize hotspot dispersion and write tail latency, while market replay and post-meeting audits require leaf layers to maintain orderly connections. Therefore, index optimization cannot be based on tree, hash, or learning methods. Instead, it should identify the sources of operational bottlenecks along the critical path. For hotspot writes, lock contention and node splitting should be reduced; for point queries, tree height and cache invalidation should be minimized; and for range accesses, sequential traversal capabilities should be preserved.

2.2 Implications of the Public Experimental Results for the Boundaries of Existing Indexing Capabilities

Figure 1 shows an approximate data recovery using the histogram published in the Blink-hash paper. The figure shows that ART has the highest peak value during monotonic insertion, indicating that adaptive node expansion is beneficial for absorbing consecutive key writes. However, in scanning scenarios, B-link trees with ordered leaf chains and hybrid variants generally outperform B-link trees without ordered leaf chains. The significance of Blink-hash is not that it is the strongest in all scenarios, but rather that it alleviates write hotspots without sacrificing range queries. For financial transaction databases, this result indicates that order inflow paths and after-hours replay paths need to be considered separately, and the leaf structure should be switchable.

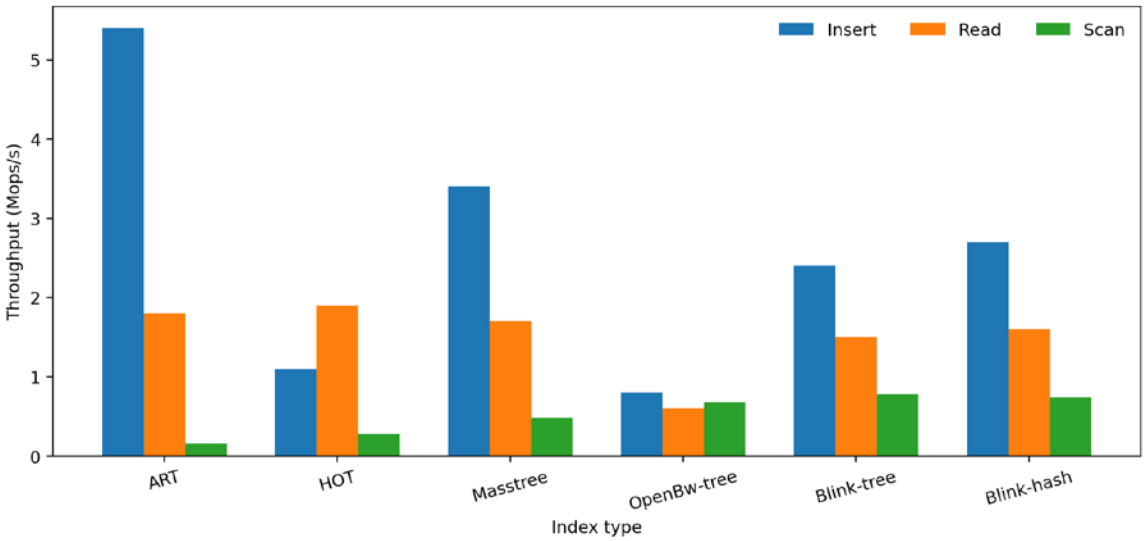

Figure 1 shows a comparison of single-threaded index throughput redrawn from the publicly available experimental graph of Blink-hash (the data is an approximate reconstruction of the graph, from reference [1] Figure 6).

Based on the system-level results from the literature in the past three years, SMART is much better than Sherman in typical write-intensive loads, achieving the highest throughput (6.1 times) and the lowest latency (1.4 times) [2]; DEX has a greater advantage over Sherman in split memory range indexing, with an advantage of up to 1.7-56.3 times [4]; FluidKV has the highest improvement of 9 times, 3.8 times and 90% in write throughput, read throughput and DRAM usage, respectively [5]; Outback has the highest throughput improvement of 5.03 times under data search load [7]; Shard is 6.7 times better than previous methods for YCSB workloads [9]. These three sets of results show that modern index optimization has changed from the original "comparing which is better, tree or hash" to the current "targeted elimination of bottleneck factors", that is, reducing the number of remote round trips, reducing the size of metadata, preventing expansion pauses, and compressing cache miss rates, etc.

In the context of financial trading systems, the above trend offers two methodological advantages. The core of low-latency indexing is not the theoretical complexity of a single operation, but rather the product of cross-layer resource consumption—that is, exactly which cache mismatches, remote accesses, locks, or CAS retries are triggered in a single lookup. The stability of a system is not determined by its average latency, but by the tail latency inflection point when the load suddenly increases. Therefore, when designing in-memory indexes for trading systems, reducing the P99 latency degradation slope should be considered an equally important objective as increasing peak throughput.

3. Raising Questions

Based on existing research and the characteristics of trading systems, the problem of in-memory database indexing for low-latency financial trading can be summarized into four levels. First, leaf layer write hotspots caused by monotonic or locally ordered keys can lead to node splitting, lock contention, and remote cache jitter. Second, point queries and range queries coexist, so traditional pure hash

indexes and pure tree indexes are difficult to achieve optimal performance globally. Third, when concurrency increases within the matching window, average latency and P99 latency tend to diverge sharply when the system is close to saturation. Fourth, trading systems require online recovery and audit replay, so indexes cannot only pursue instantaneous performance but also need to consider memory usage, expansion costs, and recoverability.

Therefore, the issue addressed in this paper is not whether a particular index is the best, but rather how to construct a low-level, low-contention, low-amplification, and rollback-capable composite index. Theoretically, lookup latency, queuing, memory usage, and write amplification need to be considered within the same analytical framework; from an engineering perspective, coordination is required between upper-layer ordered routing, leaf-level hotspot distribution, cache layout, and concurrency control. Therefore, this paper presents the following analytical model.

$$E[T_{lookup}] = \sum_{l=1}^{h}(p_l^{L1}t_{L1} + p_l^{L2}t_{L2} + p_l^{L3}t_{L3} + p_l^{M}t_{M} + p_l^{R}t_{R}) \tag{1}$$

Equation (1) gives the hierarchical expression for the average search latency. The number of index access path layers is represented by h, the probability of hitting different storage levels (L1, L2, L3, local memory, and remote memory) in the l-th layer is represented by p, and the access cost is represented by t. For transaction systems, the real factor determining search time is not the height of the tree, but the combined result of the tree height multiplied by the memory access cost of each layer.

$$M_{idx} = \frac{N(k+p+\delta)}{\eta} \tag{2}$$

The index memory footprint is as shown in equation (2), including the number of records N, key metadata length k, pointer or offset length p, concurrency control and version information overhead δ, and node or bucket space utilization η. This indicates that increasing the node fill rate and compressing redundant data will reduce the number of cache misses and lower memory costs.

$$W_q \approx \frac{\rho}{1-\rho} \cdot \frac{c_a^2+c_s^2}{2} \cdot E[S], \qquad L_{tail} \approx E[S] + W_q \tag{3}$$

Equation (3) uses queuing analysis to characterize the trend of tail latency under high load. System utilization ρ = arrival process variation coefficient c_a, service process variation coefficient c_s, average service time E[S]. When the transaction period is in a burst traffic segment, and the index service time fluctuates greatly, the tail latency will tend to 1 according to ρ and thus increase rapidly.

4. Problem Solving/Strategies

4.1 Constructing a composite index of "ordered routing + hotspot-distributed leaf layers"

For situations where order book updates and risk point queries coexist, this paper proposes a composite index structure using an upper-layer ordered routing and a lower-layer hotspot-distributed leaf layer. The upper layer employs a lightweight ordered directory, which can use ART or compressed B-link directory nodes to store only route boundaries and minimal necessary metadata, ensuring entry point location capabilities for range queries and interval replay. The lower layer uses segmented hash leaf nodes in hotspot key segments, transforming monotonically increasing writes from being concentrated in the rightmost leaf to being distributed across multiple buckets within the same segment, thereby reducing lock contention and splitting frequency. For cold segments and scan-sensitive segments, the leaf chain is maintained, and only ordered access is performed.

This strategy aligns with the principles of Blink-hash, but places greater emphasis on "key-segment awareness" scheduling logic for financial scenarios. When a security code, account group, or time period experiences a surge in writes, the corresponding hotspot segments are hashed. After the system analysis or playback phase concludes, the system automatically performs a background reshaping task to restore the ordered leaf layout. Therefore, while ensuring overall interpretability, it achieves phased optimization of the write and query paths.

4.2 Optimize node layout and metadata representation with caching behavior as the core

The real bottleneck of the index structure in a transaction system is not the number of comparisons, but the waste of cache lines and pointer jumps. The following three principles guide the design of nodes: First, control the size of directory nodes so that frequently used fields fall within one or two cache lines; second, separate fixed-length metadata and variable-length data for leaf nodes to prevent large value objects from affecting the key index; third, use offsets instead of long pointers to reduce Midx in single-machine or shared address space scenarios. For a distributed memory environment, the number of bytes read from the far end during a lookup and the number of RTTs should also be included in the design as equivalent targets.

Existing research shows that SMART, DEX, Outback, and Shard all prioritize the number of keys a single node can store over the entire system. Instead, they focus on which bytes are actually moved and how many rounds of interaction are performed per access. This approach is well-suited for financial trading systems. Due to microsecond-level budgets, a single unnecessary cache miss or remote read is more expensive than multiple integer comparisons.

4.3 Reducing update path tail latency based on version-aware concurrency control

Write operations in trading databases are characterized by high frequency, small objects, and strong bursts. Using coarse-grained locks or global synchronization during expansion can easily lead to tail latency spikes during opening, closing, and abnormal market conditions. This article recommends a version-aware concurrency control approach: optimistic reads and local CAS updates at the directory level, and bucket-level or segment-level version numbers combined with write merging at the leaf level. This allows readers to proceed lock-free most of the time, while writers only incur the cost of retries during local conflicts.

It's also necessary to distinguish between "updates that immediately affect correctness" and "structural updates that can be postponed." Insertions and version increments within buckets should be immediately visible, while node rebalancing, empty slot reclamation, and hot/cold segment migration can be handled by background asynchronous threads. After this separation, the system transforms critical paths into the shortest atomic operation chains, minimizing the impact of P99 latency on structural maintenance tasks.

$$A_u = \frac{B_{meta}+B_{data}+B_{reorg}}{B_{payload}} \tag{4}$$

In equation (4), the update amplification factor B_meta is the number of bytes caused by metadata modification plus the number of bytes written for the payload, plus the additional overhead caused by splitting, reorganizing, expanding, etc., that is, B_meta = B_data + B_reorg. Financial trading systems should minimize the number of B_reorg operations, and most updates should become local in-situ

operations or small-scale version changes.

4.4 Introducing multi-objective optimization and runtime adaptive switching mechanism

$$\min J = \alpha \overline{L} + \beta L_{99} + \gamma M_{idx} + \lambda A_u - \mu X \quad (5)$$

Equation (5) gives the multi-objective objective function for index optimization, where $\overline{L}$ is the average latency, L99 is the P99 latency, M_idx is the memory usage, A_u is the write amplification, X is the throughput, and α, β, γ, λ, and μ are the weights. The trading system dynamically adjusts the weights according to the needs of each stage, such as intraday trading, post-market replay, and disaster recovery, thereby achieving the goal of adaptive operation during runtime.

From an operational perspective, the system continuously collects the following metrics: hotspot segment write concentration, leaf layer CAS failure rate, cache hit rate, P99 latency slope, interval scan ratio, and expansion wait time. If the write concentration is detected to be continuously increasing while the scan ratio is decreasing, the hotspot segment will be converted into a segmented hash leaf layer. If the scan ratio increases or an audit task begins execution, the hotspot segment will gradually be converted into an ordered leaf chain. Therefore, the index structure can be transformed from a static design object into a schedulable resource.

Table 2 summarizes the results of three representative English-language publications, organized based on the original abstracts or conclusions.

Method	Year	Index family	Reported result	Source
Blink-hash	2023	Hybrid tree-hash	Up to 91.3x on 32-thread insert scaling	[1]
SMART	2023	ART on DM	Up to 6.1x throughput, 1.4x lower latency	[2]
DEX	2024	B+-tree on DM	1.7x-56.3x faster than baseline	[4]
FluidKV	2024	Multi-stage KV index	9x write, 3.8x read, 90% less DRAM	[5]
Outback	2025	Perfect-hash decoupled index	1.06x-5.03x higher throughput	[7]
Sphinx	2025	Succinct perfect hash	2x lower query/update latency and memory	[8]
Shard	2025	Resize-optimized hash	Up to 6.7x over prior DM hashes	[9]
Deft	2025	Scalable tree index	2.4x-9.5x over ordered indexes	[10]

As shown in Table 2, the advantages of in-memory index optimization in the past three years mainly come from three sources: reducing additional interaction rounds on the critical path (Outback, DEX, Shard); mitigating update hotspots and structural maintenance costs (Blink-hash, SMART); and compressing the structure size to achieve better caching and memory performance (Sphinx, FluidKV). In financial trading systems, latency, stability, and cost are the three engineering goals that correspond precisely to these three benefits, thus exhibiting strong transferability.

Figure 2 compares different optimization directions using the best return multiples given in publicly available literature. Although the hardware platforms, datasets, and load configurations of the various papers differ, making direct horizontal ranking impossible, an important pattern emerges: targeted improvements to the main bottlenecks lead to significant improvements in the index structure. For financial transaction databases, this means improvements cannot be limited to minor parameter adjustments but should involve systemic innovation addressing fundamental issues such as hotspots, RTT, caching, scaling, and metadata size.

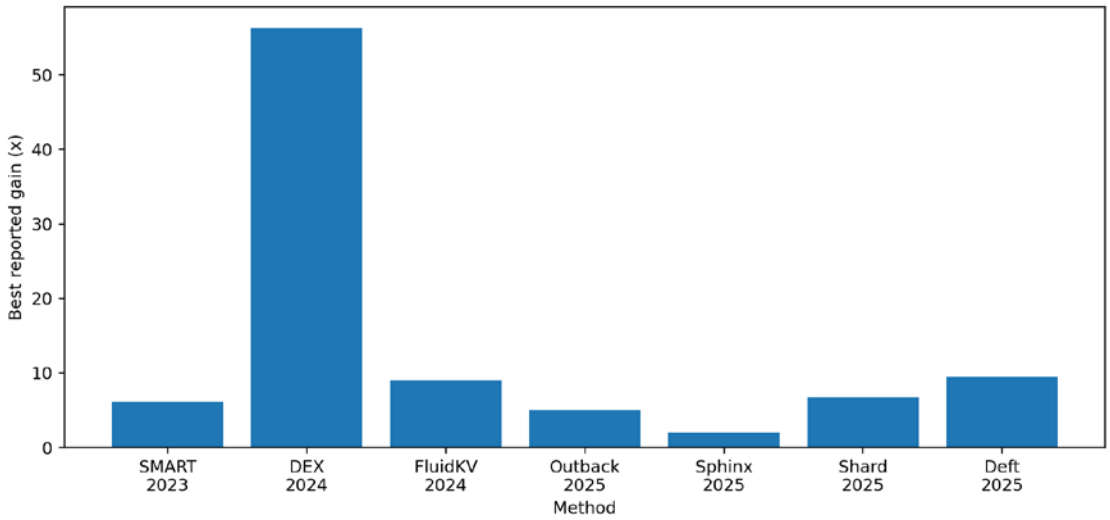

Figure 2 is a comparison table of the best return multiples in three representative indexing method publications (compiled from references [2][4][5][7][8][9][10]).

5. Conclusion

This paper focuses on "Optimization of In-Memory Database Index Structure for Low-Latency Financial Transactions," and elaborates on the topic in five parts: introduction, current status analysis, problem statement, solution strategy, and conclusion. It also provides a systematic review of English literature from the past three years. The research suggests that the goal of index optimization in financial transaction scenarios is not simply to maximize average throughput, but also to control average latency, P99 latency, write amplification, memory usage, and interval scan performance within a very limited time budget.

Building upon this foundation, this paper proposes a composite optimization framework characterized by ordered routing, hotspot-distributed leaf layers, cache-friendly layout, version-aware concurrency, and runtime adaptability. This framework incorporates key ideas from research in Blink-hash, SMART, DEX, FluidKV, Outback, Sphinx, Shard, and Deft, transforming them into engineering recommendations applicable to sub-scenarios of financial trading such as order book updates, risk verification, and market replay. For low-latency financial trading, in-memory database indexes are not fixed data structures but rather an infrastructure that should be dynamically and runtimely adjusted based on conditions such as load phase, key distribution, and tail latency budget. Subsequently, empirical calibration of index switching thresholds is performed using real matching logs for different securities, different trading periods, and different abnormal market conditions.

References

[1] *Bukun Ren. Multimodal Learning Method for Cross-Modal Data Alignment and Retrieval. International Journal of Multimedia Computing (2026), Vol. 7, Issue 1: 1-8.*

[2] Yixian Jiang. Performance Optimization and Improvement of Advertising Machine Learning Platform Based on Distributed Systems. International Journal of Big Data Intelligent Technology (2026), Vol. 7, Issue 1: 9-17

[3] Yiting Gu. Application and Optimization Strategies of Cloud Services in Front end Engineering. International Journal of Big Data Intelligent Technology (2026), Vol. 7, Issue 1: 1-8

[4] Jin Li. Performance Analysis of Efficient Microservice Architecture in the Financial Industry. Machine Learning Theory and Practice (2026), Vol. 6, Issue 1: 1-9.

[5] Qifeng Hu. Optimization and Upgrade Path of Tax Management Software System Based on Cloud Platform. Socio-Economic Statistics Research (2025), Vol. 6, Issue 2: 194-200.

[6] Yilin Fu. Data-driven Optimization of Capital Market Trading Strategies and Risk Management. International Journal of Business Management and Economics and Trade (2025), Vol. 6, Issue 1: 221-228.

[7] Shuang Yuan. Research on Abnormal Detection and Transaction Risk Management Based on Machine Learning. International Journal of Social Sciences and Economic Management (2026), Vol. 7, Issue 1: 10-18

[8] Linwei Wu. Application and Development of Blockchain Technology in Financial Infraustructure Innovation. International Journal of Business Management and Economics and Trade (2025), Vol. 6, Issue 1: 213-220.

[9] Linwei Wu. The Application of Quantitative Methods in Project Management and Actual Effect Analysis. International Journal of Business Management and Economics and Trade (2025), Vol. 6, Issue 1: 204-212.

[10] Yuanjing Guo. The Practical Impact of an International Perspective on Promoting Financial Education. International Journal of Business Management and Economics and Trade (2025), Vol. 6, Issue 1: 196-203.

[11] Wang, C. (2026). Research on the Control of Uncertainty Risks in Investment Decision-making by Financial Modeling.

[12] Xinran Tu. Resource Allocation Optimization and Cost Saving Analysis Based on Data Mining. International Journal of Business Management and Economics and Trade (2026), Vol. 7, Issue 1: 1-9

[13] Linwei Wu. Data Visualization and Decision Support Analysis Based on Tableau. Socio-Economic Statistics Research (2026), Vol. 7, Issue 1: 10-18

[14] Yilin Fu. Research on the Application of Innovative Financial Technologies in Capital Market Risk Management. Socio-Economic Statistics Research (2026), Vol. 7, Issue 1: 1-9

[15] Yuhan Zhou. Green Bonds and Sustainable Financing Models in Energy Finance. International Journal of Social Sciences and Economic Management (2026), Vol. 7, Issue 1: 28-35

[16] Zhengle Wei. Research on Innovative Design of Financial Derivatives and Market Risk Management Strategies. International Journal of Social Sciences and Economic Management (2026), Vol. 7, Issue 1: 19-27

[17] Huijie Pan. Discussion on Low-Latency Computing Strategies in Real-Time Hardware Generation. International Journal of Neural Network (2025), Vol. 4, Issue 1: 57-64.

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

[19] Liu, D., Shen, Q., & Liu, J. (2026). The Health-Wealth Gradient in Labor Markets: Integrating Health, Insurance, and Social Metrics to Predict Employment Density.

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

[21] Shen, D. (2026). Application of Large Language Model in Mental Health Clinical Decision Support System. International Journal of Engineering Advances, 3(1), 23-30.

[22] Wang, Y. (2026). Research on Optimization of Neuromuscular Rehabilitation Program Based on Physiological Assessment. European Journal of AI, Computing & Informatics, 2(1), 21-30.

[23] Ding, J. (2026). Optimization Strategies for Supply Chain Management and Quality Control in the Automotive Manufacturing Industry. Strategic Management Insights, 3(1), 17-23.

[24] Zhang, Q. (2026). How to Improve Marketing Efficiency and Precision through AI-Driven Innovative Products. Strategic Management Insights, 3(1), 1-8.

[25] Liu, Y. (2026). The Promoting Role of Fintech and Product Innovation in the Context of the Digital Economy. Strategic Management Insights, 3(1), 9-16.

[26] Lu, C. (2026). Research on 3D Reconstruction Methods of Remote Sensing Images Combined with Deep Learning and GIS. International Journal of Engineering Advances, 3(1), 15-22.

[27] Cai, Y. (2026). Design and Implementation of System Extensibility under High Concurrency Environment. International Journal of Engineering Advances, 3(1), 31-37.

[28] Liu, Y. (2026). The Application of Data-Driven Financial Risk Management in Multinational Enterprises. Economics and Management Innovation, 3(1), 20-26.

[29] Huang, J. (2026). Practice of Public Space Optimization and Functional Enhancement in Cultural Architecture. European Journal of Engineering and Technologies, 2(1), 9-21.

[30] Xu, D. (2026). AI-Driven Video Content Optimization Strategies for Immersive Media. European Journal of Engineering and Technologies, 2(1), 1-8.

[31] Qi, Y. (2026). AI Driven Payment System Security Improvement and User Privacy Protection Mechanism. Journal of Computer, Signal, and System Research, 3(1), 35-41.

[32] Qi, Y. (2026). High Reliability Architecture and Compliance Design of Enterprise Level Financial Infrastructure. International Journal of Engineering Advances, 3(1), 8-14.

[33] Zheng, H. (2025). Research on Lifecycle Configuration and Reclamation Strategies for Edge Nodes Based on Microservice Architectures. Advances in Computer and Communication, 6(5).

[34] Zheng, H. (2025). Research on Delay-aware Scheduling Algorithms for Edge Task Migration in High-concurrency Environments. Engineering Advances, 5(4).

[35] Hui, X. (2026). Research on the Design and Optimization of Automated Data Collection and Visual Dashboard in the Medical Industry. Journal of Computer, Signal, and System Research, 3(1), 27-34.

[36] Shen, D. (2024, November). Application of Machine Learning Algorithms Based on Magnetic Resonance Imaging in the Diagnosis of Knee Joint PVNS. In International Conference on Cognitive based Information Processing and Applications pp. 401 – 411. Singapore: Springer Nature Singapore.

[37] 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.

[38] Zhou, Y. (2024, November). Construction of a Multi-factor Quantitative Stock Selection System for the New Energy Industry Based on Microservices Architecture and Machine Learning Components. In International Conference on Cognitive based Information Processing and Applications (pp. 163-174). Singapore: Springer Nature Singapore.

[39] Sun, Q. (2026). Research on Lightweight Intelligent Dialogue Systems Based on Semantic Entity Enhanced Intention Recognition and Rule Retrieval Generation Hybrid Models.
[40] 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.
[41] Wang, Y. (2026). Research on the Application of Artificial Intelligence in Supply Chain Risk Early Warning.
[42] Liu, H. (2026). Research on the Application of Causal Reasoning Method in Content Compliance Experimental Evaluation.
[43] Liu, H. (2026). Research on Dynamic Price Prediction of E-commerce Based on Time Series Modeling.
[44] Yu, X. (2025). Digital Transformation Empowers Growth Marketing with Marketing Data Analysis Integration and Real-Time Display Strategy.
[45] Yu, X. (2026). Strategy Models and Practical Research of Growth Marketing under the Background of Digital Transformation.