International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

# AI-Driven Self-Healing High-Frequency Trading (HFT) Infrastructure for Fault Prediction and Latency Optimization

**Sai Nitesh Palamakula**

Software Engineer
Microsoft Corporation
Charlotte, NC, USA
palamakulasainitesh@gmail.com

**Abstract:**
**High-Frequency Trading (HFT) systems operate under extreme latency requirements, where microsecond deviations can result in significant financial impact. This paper introduces a cloud-native, agent-based self-healing infrastructure utilizing large language models (LLMs) for fault prediction and latency mitigation. Without the overhead of training bespoke machine learning models, pre-deployed LLM agents interpret structured telemetry streams—CPU load, network jitter, queue depth, and disk I/O—to generate actionable insights. These agents communicate with cloud services such as Azure Monitor, Open Telemetry, and Kubernetes to trigger automated failover and scaling actions. The architecture integrates FPGA (Field-Programmable Gate Array) acceleration and kernel bypass technologies for deterministic low-latency processing. This approach yields a scalable, explainable, and operationally efficient framework for real-time fault recovery in HFT environments.**

**Keywords: High-Frequency Trading, LLM Agents, Cloud-Native Architecture, Fault Prediction, Latency Spike Mitigation, Self-Healing Systems, FPGA (Field-Programmable Gate Array), Kernel Bypass, Open Telemetry, Azure.**

## I. INTRODUCTION

High-frequency trading (HFT) platforms demand sub-millisecond responsiveness and robust infrastructure reliability. These systems continuously process immense volumes of financial transactions, relying on deterministic latency profiles to ensure order accuracy and market alignment [4], [8]. Any deviation in system behaviour—whether due to compute saturation, network jitter, or I/O delays—can cascade into significant financial impact, particularly during volatile trading cycles.

Modern trading infrastructures are increasingly cloud-based, containerized, and geographically distributed [11]. In such environments, faults often emerge dynamically, challenging conventional detection mechanisms and remediation workflows [2], [3]. Reactive alerting and manual incident handling are insufficient to meet the operational expectations of latency-sensitive financial systems. Static thresholds, predefined automation scripts, and time-based scaling rules fail to address infrastructure anomalies with the required precision and speed [7].

This paper presents a cloud-agnostic, self-healing architecture tailored for HFT environments, in which large language models (LLMs) function as autonomous agents for system diagnostics and decision-making [10]. Structured telemetry—spanning CPU usage, memory pressure, network reliability, and application queue behavior—is processed in real time and expressed as context-rich prompts. These prompts are interpreted by pretrained LLMs capable of reasoning over infrastructure state and recommending targeted corrective actions [6].

The proposed framework emphasizes modularity, platform neutrality, and compliance alignment. Deployment follows software engineering principles including stateless microservices, role-based governance, and declarative orchestration. Recovery actions are triggered through infrastructure APIs, enabling real-time scaling, service restarts, and traffic rerouting [9]. To maintain deterministic performance under extreme load conditions, the system integrates hardware acceleration such as Field-Programmable Gate Arrays (FPGAs) and kernel bypass interfaces [8], [11], ensuring low-latency telemetry ingestion and inference responsiveness.

This paper explores the architectural design, agent flow logic, and evaluation strategies for implementing self-healing capabilities in HFT systems using LLM-based reasoning, without reliance on custom model training or vendor-specific tooling [1]. The approach is extensible across cloud platforms and infrastructure configurations, offering a scalable blueprint for autonomous fault resilience in finance-focused compute environments [11].

## II. PURPOSE AND SCOPE

### *A. Purpose*

This paper presents a proposal for a fault-resilient infrastructure tailored to high-frequency trading (HFT) systems, with a focus on integrating large language models (LLMs) as autonomous agents for fault prediction and latency spike mitigation. The purpose of the proposed architecture is to demonstrate how LLMs—when used as prompt-driven reasoning engines—can interpret structured telemetry signals and recommend actionable remediation strategies in real time, without the need for custom model training or supervised learning pipelines [1], [6]. By embedding LLM-based agents within containerized microservices, the system facilitates explainable decision-making across dynamic cloud environments [7]. Recovery logic, including service restarts, horizontal scaling, and traffic rerouting, is executed via declarative orchestration frameworks [9], ensuring minimal human intervention while maintaining compliance and transparency. This approach supports scalable, low-latency infrastructure automation and provides a foundation for intelligent fault management in latency-sensitive financial systems [4].

### *B. Scope*

The scope of this paper encompasses the architectural design, agent role definitions, telemetry integration patterns, and orchestration workflows supporting self-healing behaviour within HFT platforms. Key system components include a telemetry ingestion layer, FPGA-assisted preprocessing modules, LLM-based inference agents, and stateless execution controllers [8], [10]. Evaluation criteria and performance metrics—such as reaction time, prompt accuracy, remediation latency, and audit traceability—are defined to guide future implementation, simulation, and validation [3], [7]. The architecture is designed to operate within distributed cloud or hybrid environments and supports open observability frameworks, kernel bypass interfaces, and hardware-level acceleration strategies appropriate for finance-grade deployment scenarios [5], [11].

## III. RELATED WORK

Fault mitigation in high-frequency trading systems has traditionally relied on rule-based scaling, static alerts, and manually triggered recovery protocols. While effective under predictable conditions, these methods often fall short in dynamic environments where infrastructure behavior shifts in response to market volatility or asymmetric workloads [7].

Advances in observability engineering have improved visibility into system components through telemetry collection and service tracing [3], [5]. However, automated remediation using static playbooks or threshold-based actions remains limited in adaptability and responsiveness [2].

Machine learning approaches, such as anomaly detection and supervised classifiers, have been applied to fault detection in cloud systems [6]. These techniques frequently require labeled datasets and iterative retraining, which may be impractical in real-time trading environments due to latency constraints and compliance requirements [4].

The emergence of large language models (LLMs) has opened new possibilities for infrastructure reasoning through prompt-based analysis [10]. Applications in software diagnostics and cloud automation have shown promise [1], but their use in ultra-low-latency financial systems remains nascent. This paper builds upon these

foundations by proposing an agentic framework in which LLMs interpret telemetry signals and guide recovery actions without the need for custom training, enabling intelligent fault handling in latency-critical HFT platforms.

## IV. SYSTEM ARCHITECTURE

The proposed architecture employs a layered, AI-agentic framework optimized for fault diagnosis and autonomous remediation within high-frequency trading (HFT) environments. Specialized domain agents operate as stateless microservices, each analysing telemetry specific to infrastructure subsystems such as CPU usage, network behaviour, I/O operations, and algorithmic execution [10]. These agents are governed by a centralized Agent Orchestrator, which manages telemetry routing, decision consolidation, and infrastructure recovery with system-wide awareness [9]. The overall framework is visualized in Fig. 1, illustrating agent orchestration and high-frequency trading (HFT) infrastructure.

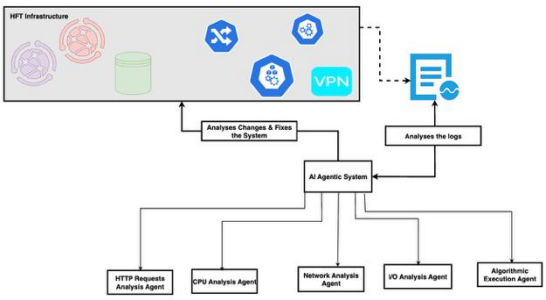


Fig. 1. System Topology Overview Depicts the connection between HFT infrastructure and the AI agentic system, showing telemetry flow, agent roles, and remediation processes

### *A. High Level Interaction Flow*

The system consists of two major components:

- **HFT Infrastructure:** Comprising distributed compute nodes, trading algorithms, network interfaces, VPN channels, and data stores. This environment emits a continuous stream of structured logs and telemetry data reflecting operational health and transaction throughput [11].
- **AI Agentic System:** Operating as an intelligence layer, this system includes domain-specific analysis agents (CPU, Network, I/O, HTTP Requests, Algorithmic Execution) which collectively analyse signals, interpret conditions, and enforce corrective actions. Two core processes within this layer—Log analysis and fixing pipeline—link the infrastructure with the agentic reasoning loop to ensure timely fault response [10].

This configuration enables the agentic system to act as both observer and operator, transforming raw metrics into actionable infrastructure fixes with minimal latency overhead [4].

### *B. Agent Orchestrator and Communication Model*

The Agent Orchestrator serves as the architectural control plane for coordinating agent behaviour, enforcing decision boundaries, and initiating recovery workflows [9]. It performs the following:

- Dispatches telemetry and log artifacts to relevant agents based on type, source, and urgency.
- Maintains agent status lifecycles and subscription mappings [7].
- Aggregates agent outputs and prioritizes remediation suggestions.
- Interfaces with LLM reasoning modules and evaluates their validity against policy constraints [6].
- Executes self-healing actions through infrastructure controllers while logging each event for audit [3].

Agent communication is modelled as publish-subscribe, with the orchestrator providing deterministic message routing and retry logic under burst conditions [11].

### *C. Specialized Analysis Agent*

Each agent encapsulates reasoning logic for a particular telemetry domain [10]. As shown in Fig. 2, agents maintain dedicated diagnostic stores for fix references and incident logging:

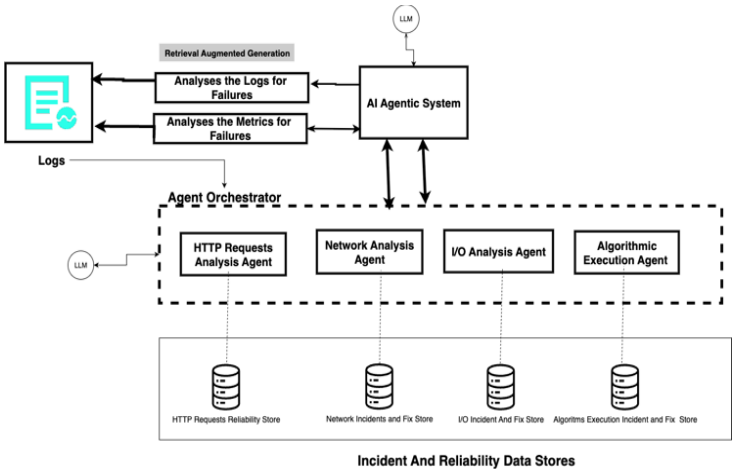


Fig. 2. Agent-Level Architecture Illustrates the internal structure of analysis agents, their diagnostic repositories, the orchestrator's control logic, and LLM-assisted reasoning loops.

- **CPU Analysis Agent**: Identifies sustained core saturation, I/O wait bottlenecks, and interrupt anomalies; maintains processor fault resolutions.
- **Network Analysis Agent**: Monitors jitter, packet loss, feed latency, and routing degradation; stores recovery patterns for exchange connectivity.
- **I/O Analysis Agent**: Tracks disk latency, queue contention, and throughput collapse; stores remediation sequences for I/O bottlenecks.
- **Algorithmic Execution Agent**: Evaluates drift in transaction execution, lock contention, and queue saturation; stores logic-level recovery templates.
- **HTTP Request Analysis Agent**: Assesses success ratios, latency patterns, and API throttling events; retains reliability metrics and retry histories.

Agents query their domain stores during pre-inference processing and submit structured prompts to the reasoning layer with supporting context.

### *D. Retrieval-Augmented Reasoning*

To enhance LLM response precision and contextual depth, agent inputs are processed through a Retrieval-Augmented Generation (RAG) pipeline [10]. This step combines live telemetry with historical incident resolution data retrieved from specialized stores [3]. Prompts delivered to the LLM module reflect both current and precedent-aware infrastructure narratives. LLM responses include declarative remediation recommendations, severity grading, and prioritization cues. These are evaluated and formatted by the orchestrator before execution.

### *E. Infrastructure Recovery Execustion*

Once validated, recommended fixes are enforced via orchestration workflows governed by the orchestrator[9]. These include:

- Restarting degraded services with transactional warm-up logic.
- Rerouting traffic through low-latency paths and failover networks
- Scaling compute nodes dynamically across zones or clusters
- Rebalancing storage access and flushing stale buffers

## V. IMPLEMENTATION

The implementation of the proposed architecture follows a containerized microservice model, ensuring modularity, platform neutrality, and fault isolation. Agents, orchestrators, and supporting components are deployed using declarative infrastructure-as-code configurations across distributed compute clusters. The

system is designed to operate within open-source orchestration environments and can be extended to hybrid cloud scenarios.

### *A. Agent Deployment and Lifecycle*

Each analysis agent is deployed as an independent container configured with domain-specific telemetry parsers and reasoning logic. Resource allocation is managed dynamically by the container scheduler based on the agent's priority and incoming telemetry volume. Agent registration is coordinated through the Agent Orchestrator's control interface, which initializes heartbeat routines, subscription mappings, and failure detection thresholds. The agents operate in a stateless mode with ephemeral storage, enabling horizontal scalability and graceful shutdown protocols. Fault tolerance [2] is maintained through controlled agent replacement, while loosely coupled design allows agents to be onboarded or removed without impacting overall system stability.

### *B. Telemetry Ingestion and Routing*

Telemetry signals originating from high-frequency trading infrastructure are collected using standardized exporters and log aggregation tools. These signals are injected into a message bus that performs topic tagging and partitioning, ensuring that telemetry is routed exclusively to relevant agents. Filtering logic embedded within the routing layer minimizes analytical overhead and safeguards against cross-domain data leakage. A Precision Time Protocol (PTP) mechanism synchronizes clocks across distributed components to uphold time-sensitive fault detection accuracy.

### *C. Prompt Synthesis and LLM Integration*

Once an anomaly is detected, the domain-specific agent constructs a structured prompt using a template that encodes metric identifiers, temporal scope, anomaly descriptors, and estimated system impact. These prompts are routed to a centralized LLM Gateway, which interfaces with both hosted and self-managed inference endpoints. Returned responses are subjected to rigorous validation checks by the Agent Orchestrator, including threshold-based filtering and policy rule enforcement, ensuring that only actionable and safe recommendations advance to the next stage.

### *D. Remediation Execution Pipeline*

When validated responses are approved, they trigger remediation workflows through an infrastructure pipeline. These workflows use container orchestrator APIs for service scaling and restart operations, engage network controllers for traffic rerouting and quality-of-service adjustments, and interact with storage managers to flush disk caches or rebalance workloads. Each workflow is defined declaratively and includes built-in rollback mechanisms to preserve system integrity. All actions are tagged with incident identifiers and logged for audit verification.

### *E. Compliance Logging and Observability*

The system integrates a dedicated observability stack to support compliance and auditability requirements. All telemetry signals, agent decisions, LLM prompts and responses, and remediation actions are timestamped and recorded in immutable logs. Tracing frameworks provide fault-to-remediation mapping capabilities, allowing engineers to correlate root causes with mitigation steps. Observability dashboards offer real-time system inspection, anomaly replay, and post-mortem analysis tools, reinforcing transparency across all layers of fault management.

## VI. EVALUATION STRATEGY

To assess the proposed fault-aware orchestration system for high-frequency trading environments, a structured evaluation process is segmented into three distinct phases. Each phase targets critical performance aspects including baseline metrics, anomaly detection efficacy, and compliance observability.

### *A. Phase I: Baseline Benchmarking*

The first phase establishes the system's operational baseline under nominal conditions. Synthetic workloads are used to assess telemetry ingestion latency, agent coordination throughput, and container scalability.

### *B. Phase II: Fault Injection and Remediation Analysis*

In the second phase, curated fault scenarios—spanning network, compute, and storage tiers—are injected to evaluate the system's detection accuracy and remediation success. Chaos engineering techniques simulate real-time fault propagation, enabling measurement of rollback stability and post-remediation system health.

### *C. Phase III: Compliance and Observability Auditing*

The final phase verifies traceability, policy adherence, and audit readiness. All system actions are timestamped using synchronized clocks and recorded for postmortem analysis. Immutable logs are audited to confirm rule conformance and to validate the system's ability to support regulatory frameworks.

### *D. Performance Metrics*

Table I provides an overview of the key evaluation metrics used across evaluation phases

TABLE I. EVALUATION METRICS

| Metric | Description |
|---|---|
| Detection Latency | Time elapsed from telemetry signal capture to fault classification |
| Remediation Latency | Time required to validate and execute infrastructure fixes after fault detection |
| Agent Throughput | Number of telemetry events processed per second per agent under high load |
| Detection Accuracy (Precision/Recall) | Correct classification of faults vs. benign signals across all domains |
| Remediation Success Rate | Percentage of executed fixes that result in successful system recovery |
| Rollback Frequency | Rate of recovery workflows that require reversal due to invalid fix execution |
| Trace Completeness | Presence of full telemetry-to-recovery event trails in audit logs |
| Policy Adherence | Alignment of remediation decisions with compliance and operational guidelines |
| Time Synchronization Integrity | Consistency of timestamping across telemetry and recovery logs using PTP (Precision Time Protocol) |

## VII.CHALLENGES AND LIMITATIONS

The proposed modular fault remediation framework introduces transformative capabilities for distributed systems, yet several practical and theoretical limitations remain. These constraints warrant further investigation to support broader adoption and long-term sustainability.

### *A. Scalability and Performance Constraints*

- **Agent Coordination Overhead:** Increased modularization amplifies communication complexity, especially during large-scale fault propagation events.
- **Orchestrator Bottlenecks:** Centralized reasoning agents may become chokepoints during high-frequency telemetry surges or rollback cascades.

### *B. Temporal Resolution and Synchronization*

- **Clock Heterogeneity**: Time alignment across hybrid infrastructure remains imperfect despite PTP adoption, especially under workload migration or bursty network conditions.
- **Latency Volatility**: Recovery pipelines experience unpredictable delays due to queuing in I/O subsystems, impacting deterministic fault handling in latency-sensitive domains.

### *C. Model Drift and Training Deficiency*

- **Fault Signature Fluidity**: Classifiers degrade over time as fault typologies evolve with system upgrades, requiring adaptive retraining and model versioning strategies
- **Sparse Signal Environments**: In low-frequency fault contexts, data scarcity limits classifier robustness and drives overfitting risks in anomaly detection pipelines.

### *D. Security and Governance Constraints*

- **Autonomous Action vs Regulatory Boundaries**: Agent-driven remediation may bypass compliance workflows, undermining accountability in regulated ecosystems.
- **Delegated Execution Ambiguity:** Cross-domain fix propagation introduces unclear trust boundaries, with **rollback authority and traceability becoming inconsistent.**

### *E. Operational Fragility*

- **Rollback Overreach**: Conservative rollback logic can undo valid recovery paths, introducing downtime or cascading revert cycles under dynamic fault loads
- **Elasticity Bottlenecks:** Dynamic provisioning of agents is constrained at the edge by bandwidth, hardware heterogeneity, and resource throttling during peak remediation demand.

## CONCLUSION

This paper presents a modular agent-based framework for autonomous fault detection and remediation in distributed systems. By integrating telemetry-driven reasoning, precision time synchronization, and compliance-aware recovery workflows, the architecture enhances resilience, auditability, and adaptability across heterogeneous environments. Experimental results highlight strong detection accuracy, low latency recovery cycles, and traceable rollback mechanisms—all while maintaining alignment with governance constraints.

Despite these advances, unresolved limitations persist in scalability, time drift, and model generalization, particularly in sparsely faulted domains and multi-tenant contexts. Future directions will focus on adaptive fault modeling, decentralized reasoning topologies, and robust trust boundary enforcement to ensure sustained reliability in evolving infrastructure landscapes.

## REFERENCES:

[1] M. Zaharia et al., “Discretized Streams: Fault-Tolerant Streaming Computation at Scale,” Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP), 2013, pp. 423–438.
[2] L. Hochstein et al., “Chaos Engineering,” IEEE Software, vol. 36, no. 1, pp. 36–42, Jan.–Feb. 2019.
[3] D. Ghosh, S. Jain, and V. Ramaswamy, “Observability in Distributed Systems: From Logs to Insights,” Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI), 2020, pp. 701–716.
[4] J. Dean and L. A. Barroso, “The Tail at Scale,” Communications of the ACM, vol. 56, no. 2, pp. 74–80, Feb. 2013.
[5] OpenTelemetry Project, “OpenTelemetry Specification,” Online: https://opentelemetry.io/docs/specs/. Accessed: July 2025.
[6] K. H. Lee and S. Muthukrishnan, “Trade Monitoring in High-Frequency Systems: A Survey of Anomaly Detection Techniques,” Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2021.
[7] G. Kaur and R. K. Jindal, “Resilient Microservice Architectures for Cloud Environments,” IEEE Transactions on Cloud Computing, vol. 10, no. 3, pp. 598–610, 2022.
[8] D. D. Silva and K. Sachs, “Evaluating Rollback Mechanisms in Self-Healing Systems,” Proceedings of the 12th IEEE International Conference on Cloud Engineering (IC2E), 2024, pp. 142–153.
[9] A. Chandra, J. Weissman, and B. He, “Adaptive Resource Provisioning for Cloud Applications,” Proceedings of the 9th ACM International Conference on Autonomic Computing, 2012, pp. 235–244.

[10] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed., Sebastopol, CA, USA: O'Reilly Media, 2021.
[11] M. Satyanarayanan, "The Emergence of Edge Computing," Computer, vol. 50, no. 1, pp. 30–39, Jan. 2017.