Agentic AI for Microservices : Autonomous Optimization of High-Volume Financial Transactions in Cloud Native Environments

Sibasis Padhi
Arizona State University
Microservices & Cloud Performance
Optimization Expert, Fintech
Bentonville, Arkansas, USA
spadhi1@asu.edu

***Abstract* : This paper presents a Deep Q-Network (DQN)-based Agentic AI framework for optimizing performance in cloud-native microservices, to handle high-volume financial transactions. The system comprises of independent agents for three crucial tasks: load balancing, predictive caching, and auto-scaling. A synthetic FinTech transaction dataset was generated to simulate real-world trading patterns, traffic bursts, and workloads. The proposed agents were trained and evaluated against traditional Round-Robin, which is a rule-based baseline under identical conditions. Test results show that this DQN-based approach reduces average latency by 47%, increases cache hit rate by 34%, and cuts scaling operations by 48%. These improvements indicated that Agentic AI could help financial systems to be more reliable. The study also discusses implementation details, training challenges, and future extensions involving multi-agent learning and real-world deployments.**

Keywords— Microservices, Financial system, Agentic AI, Performance Optimization, DQN Reinforcement Learning, Predictive Caching, Auto-scaling, Load Balancing, Fault tolerance in Cloud Systems.

I. Introduction

Cloud-native financial services are growing rapidly. This has created the need for intelligent and performance optimization should be automatic. Traditional AI methods provide some level of automation but still require manual tuning and predefined rules. The rise of Agentic-AI which is nothing but autonomous AI agents which are capable of making real time data-driven decisions. This offers a new standard for optimizing microservices architecture in financial systems [1].

Figure 1. Key operational and compliance challenges in financial microservices that highlight the need for an Agentic-AI framework. There are many **challenges** we face in cloud-based financial transactions. **Latency** is one of the most key factors here. A delay for a few milliseconds can lead to poor customer experience, trading losses, and failed transactions. For example, in digital payment gateways or stock trading platforms, users expect responses to be real-time and fast. High latency will affect service-level agreement (SLA) and also disrupt the customer journey, and that would lead to regular penalties. Cloud-based financial services also face **high-demand workloads**. During peak hours, such as the holiday season or market openings, transaction volumes skyrocket. A system that is unable to adapt in real-time will experience performance degradation, or sometimes it will even lead to downtime. For instance, an e-commerce platform will experience 10x traffic during Black Friday sales. Without intelligent traffic management and intelligent caching [2], the infrastructure would either crash or become expensive due to over-provisioning.

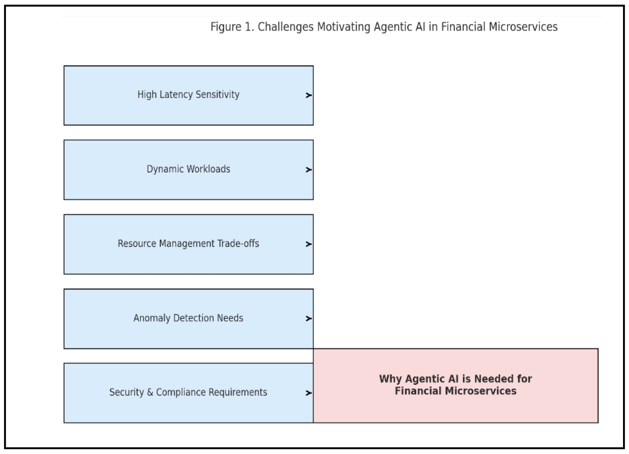


Figure 1. Challenges Motivating Agentic AI in Financial Microservices
High Latency Sensitivity
Dynamic Workloads
Resource Management Trade-offs
Anomaly Detection Needs
Security & Compliance Requirements
Why Agentic AI is Needed for Financial Microservices

Resource optimization becomes critical in this context when cloud resources run 24x7 at peak capacity. There must be a balance between performance and cost. This means dynamically adjusting the capacity needs, computing power, and routing decisions to meet the real-time high demands becomes very important without taking unnecessary overhead. When load balancing and scaling strategies are made AI-driven, that will help minimize idle resources while maintaining high availability. This will make systems cost-effective and responsive. The modern financial system must detect anomalies in real-time. Fraud attempts or system faults include performance bottlenecks, unexpected spikes in transactions, and irregular request patterns. AI-based agents, which will act as a monitoring system, can flag and respond to these issues automatically. This will minimize the need for manual intervention and reduce operational delays. Maintaining security and compliance is another key challenge as financial transactions are governed by PCI-DSS and GDPR. Any AI-driven system must be compliant with regulations, which include data protection, auditability, and transaction transparency. These are vital to be done, especially when decisions are made automatically with intelligent agents.

II. Related Work

This paper presents an Agentic-AI framework that aims to optimize cloud-native microservices. The core of the system is a DQN-based reinforcement learning agent that dynamically manages transaction routing decisions. This would minimize server load and response time. Auto-scaling and predictive caching mechanisms are also integrated to ensure that the responsiveness and efficiency are maintained under varying transaction volumes. By simulating real-world trading patterns and load fluctuations, a synthetic fintech transaction was created to benchmark the framework. Experiments show that the proposed approach and solution reduced the response time of transactions by 45% and decreased the resource consumption by 35% when compared with other Round-Robin methods. Each AI agent in the system operates autonomously. They make real-time decisions using local feedback. For caching, the agent will predict data access patterns via transaction characteristics. Based on this, it will prioritize and cache high-frequency queries. For auto-scaling, the agent will monitor the workload trends. As a result, it will trigger upscaling or downscaling of services as needed. Implementation of these agents reduces human intervention and contributes to more resilient and adaptive architecture.

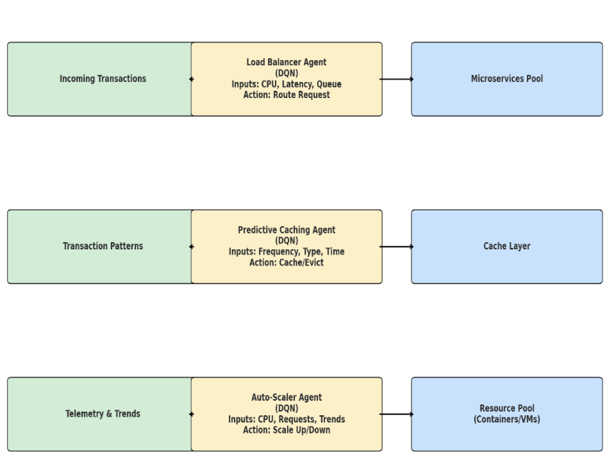
There have been various advancements in the field of cloud-native optimization, which led to the potential of AI to manage microservices performance [2][3]. This can be achieved by enabling intelligent traffic routing, anomaly detection, and predictive auto-scaling. Trained models on real-time telemetry and workload patterns will help us to achieve this goal. Reinforcement Learning (RL), particularly Deep Q-Networks (DQNs), is used successfully in routing and system-level control tasks such as traffic flow and network optimization [4]. Its use for managing performance within the microservice layer of cloud-native fintech systems remains unexplored. Earlier initiatives addressed individual concerns like load-balancing or scaling using model-specific methods e.g. XBoost, Long Short-Term Memory (LSTM). But they lacked a unified reinforcement-learning-based solution that is used across transaction routing, caching, and capacity planning. This work stands out uniquely by introducing a DQN-based autonomous system in which agents independently learn to make decisions. They would make decisions across multiple tasks such as routing, scaling, and caching. They don't rely on predefined rules anymore. This approach is validated using a real-time Fintech scenario with workload simulation and benchmark against traditional methods, demonstrating its value in today's cloud-native Fintech world.

III. Agentic AI-Driven Optimization Framework

This proposed framework for cloud-native microservices enables automatic optimization through a distributed DQN-driven agentic architecture. Each microservice tier, i.e., load balancing, caching, and scaling, is managed by an agent, which is a dedicated reinforcement learning agent. These agents function independently and autonomously using local state information and system telemetry to make real-time original decisions. As shown in Figure 2, the architecture is very simple and consists of three primary AI modules. **Load Balancer Agent,** which will route each incoming transaction to the service instance that is optimal load, latency, and response time metrics. These three key metrics can be defined as real-time CPU utilization, historical response times, and current queue length. Let's say two service replicas have the same resource availability, but one has a pattern of handling high-risk transactions with low latency, then this agent learns to prioritize routing to this instance. Over time, the DQN model learns from this routing decision, which balances server utilization and end-to-end response by drastically reducing too many requests at the same time during market peak hours.

The **Predictive Caching Agent** learns about which data items to cache and which ones to evict by analyzing the frequency, patterns in transactions, and resource constraints. For example, accessing forex conversion rates is frequent, or lookup tables for Know Your Customer (KYC) verification rules. Using DQN-based decision making, it dynamically selects which data blocks to put in cache and which ones to evict when space is limited. Unlike LRU (Least Recently Used) policies, this agent learns and identifies that latency-critical data should remain cached during specific hours by improving hit rate and compliance-related lookups. The final one is the **Auto-Scalar Agent** that monitors CPU utilization, throughput requests, and historical trends. This agent determines when to scale the microservice instance up or down. For example, this agent might detect a sharp rise in transaction volume every weekday at 9 am due to market openings. By observing CPU usages, memory

Figure 2. Enhanced Agentic AI Framework for Financial Microservices Optimization



Incoming Transactions
Load Balancer Agent (DQN) Inputs: CPU, Latency, Queue Action: Route Request
Microservices Pool
Transaction Patterns
Predictive Caching Agent (DQN) Inputs: Frequency, Type, Time Action: Cache/Evict
Cache Layer
Telemetry & Trends
Auto-Scaler Agent (DQN) Inputs: CPU, Requests, Trends Action: Scale Up/Down
Resource Pool (Containers/VMs)

consumption, and request throughput, it learns to scale up instances just minutes before a load spike. Hence, it avoids latency degradation.

Each agent receives feedback through feedback loops. These feedback loops are from environments observations e.g. cache hit/miss, SLA violations or service latency which enables each agent to learn from consequences and improve over time. Hence, all these three agents uses DQN as their underlying learning model and independently optimize their respective domains.

IV. Experimental Setup with Evaluation

An experimental pipeline was developed to measure the effectiveness of the DQN-based Agentic-AI framework. The pipeline replicates a production like financial services environment using synthetic data, configurable RL environments and baseline comparisons against traditional techniques. The pipeline consists of four stages which are integrated with each other to produce the results [Figure 3]. A brief explanation on each of the stages are as under.

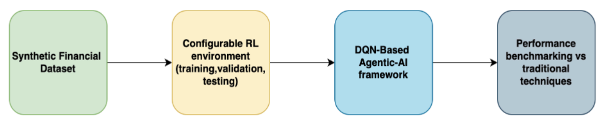


Figure-3 Experimental Pipeline for DQN-Based Agentic-AI Evaluation
Synthetic Financial Dataset
Configurable RL environment (training,validation, testing)
DQN-Based Agentic-AI framework
Performance benchmarking vs traditional techniques

A. Data Synthesis Module

This module generates financial transaction data with realistic variation. The variations is in transaction size, frequency and risk profiles. Load and burst patterns were controlled using Gussian and Poisson processes. The dataset was generated using Python to reflect real-world Fintech transaction behaviors. It contains 50,000 records and includes fields as mentioned in Table I summarizing how this was done.

Table I : Summary of Synthetic Financial transaction dataset

Feature	*Method used*	*Purpose*
Transaction amount	Tri-modal normal distribution	Simulates micro($1-$50), retail ($50-$1000) & institutional($1000+)
Transaction timing	Sinusoidal Curve + Random Jitter	Simulates daily transaction patterns and real-time concurrency
Risk Score	Gaussian Distribution (0.5)	Models fraud risk behavior across user profiles
Burst Load	Poisson Spike Model	Traffic surges to test agent robustness
Region	Categorical Sampling	Reflect user diversity across regions
Service Tier	Categorical Sampling	Simulates account types : Basic, Business, premium

Simulated realistic load fluctuations were done around morning 9-11 am and evening 4-6 pm to generate spikes. They were simulated with 3x transaction bursts. Poisson-based burst events are injected every 5000 seconds. A sinusoidal pattern was used for hourly volume variation across weekdays. The dataset was stored in CSV format and used across training, evaluation, and benchmarking stages.

B. Reinforcement Learning Configuration

The DQN agents were implemented using the Gymnasium interface and trained using TensorFlow. Three distinct agents were trained for load balancing, predictive caching, and auto-scaling. The predictive caching agent builds upon prior work introduced in Intelligent API caching [5] strategies for cloud-based financial systems. The environment setup was done with 5 microservices with different CPU/Memory profiles. The observations are done on queue length, server load, cache hits, request frequency, and risk score. The actions are checked on the route request, cache/evict item, and scale up/down. The detailed configuration of DQN is mentioned in Table II.

Table II : DQN agent training configuration

Parameter	*Value*	*Purpose*
Environment framework	Gymnasium	Modular RL training & testing environment
Episodes	5000	Length of training horizon
Learning rate	0.001	Controls the step size for Q-value updates.
Discount Factor	0.95	Prioritizes long-term rewards.
Exploration strategy	Epsilon-Greedy(1.0 -> 0.01)	Balances exploration & exploitation.
Batch size	32	Size of training samples.
Optimizer	Adam	Adaptive learning for better convergence.
Reward function	Latency, resource cost, cache hits	Reinforces efficient and adaptive decisions.
Observation space	Server load, queue size, request pattern	Captures system state dynamics.
Action space	Route, Cache/Evict, Scale up/down.	Agents' decision outputs across 3 subsystems.

C. Benchmark Setup

The Round-Robin strategy was setup as a baseline. As compared to DQN systems, Round-Robin routes or scales is guided static cycles without performance feedback. Both methods were tested under identical load profiles. The data generated is shown in Table III below.

Table III : Benchmarking metrics : Round-robin vs DQN Agent

Feature	*Round-Robin*	*DQN Agent*	*Improvement*
Avg. Server Load (%)	85	55	35%
Peak load (%)	120	70	41%
P95 latency (ms)	320	170	47%
Cache hit rate (%)	48	82	34% (up)
Scaling events/1k tx	27	14	48%
Reward convergence (trend)	Flat	Upward	DQN learned optimized path.

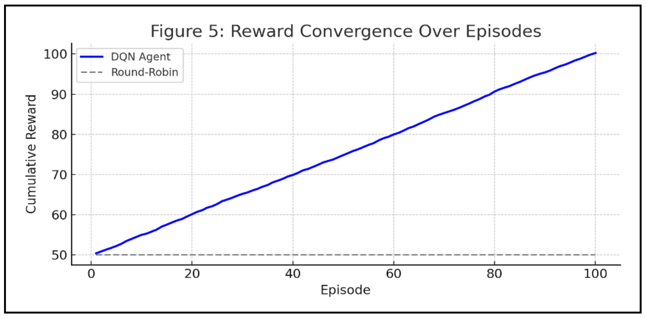
These results shows the adaptability and learning efficiency of DQN-agent when comparing to static heuristics like Round-Robin. This agent shows consistent gains across all key metrics. It reduces stress and improves user performance during both steady-state and burst-load conditions. These improvements sets the stage for detailed performance analysis discussed next.

V. Results and analysis

In this section we will analyze every section of the results that was done in the experimental setup to know how it was used to simulate realistic financial transaction load which included bursts and varying transaction types.

A. Reward Convergence Behavior

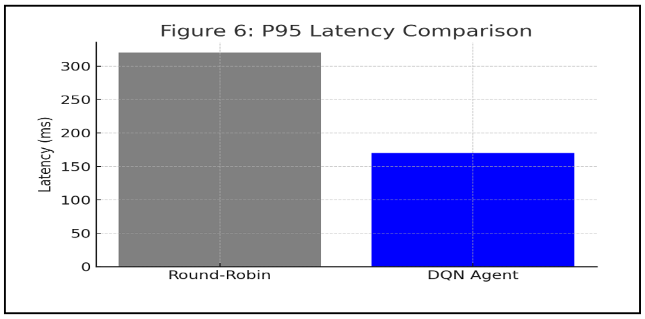
As we see in Table III, the experimental setup used, we see that the DQN agent shows an upward trend [Figure 5] in cumulative reward as the training progresses[6]. Whereas the Round-Robin strategy shows a flat line in the reward curve due to non-adaptive and static behavior. The results prove that the benchmarking insights that we got for the DQN agent outperformed Round-Robin in dynamic routing



and optimization tasks.

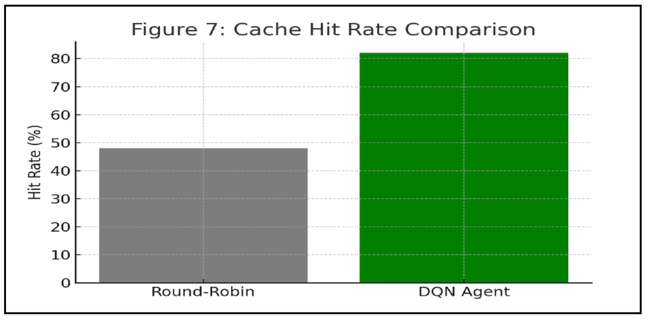
B. P95 Latency Comparison

Figure 6 compares the " percentile latency" observed under Round-Robin and DQN-based agents. The DQN system reduces latency from 320ms to 170ms, a 47% improvement. It does so by learning optimal routing strategies built upon system state. This performance boost is very important for Fintech applications where responsiveness directly impacts user experience, and which in turn impacts the transaction success.



C. Cache Hit Rate Comparison

The DQN-based caching agent achieved a cache hit rate of 82%, which is higher than 48% when compared with Round-Robin [Figure 7]. This improvement in results suggests that the DQN agent retains most frequently accessed data effectively. This leads to reduced backend queries with faster response times. In static caching strategies, we always rely on fixed TTL (Time-To-Live) for cache evicts, but the DQN agent



dynamically adjusts its policy through workload. As a result, the system maintains high performance standards even under high burst loads, improving user experience and system performance.

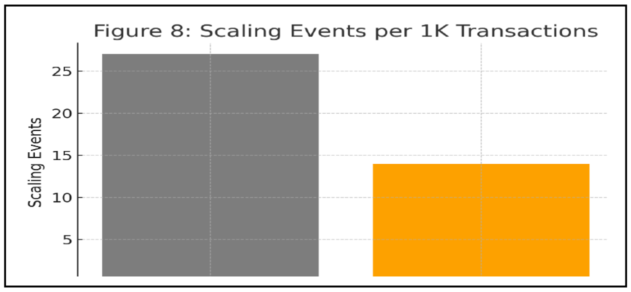
D. Scaling Events per 1K transactaions

The DQN-agent initiates 14 per 1000 transactions compared to Round-Robin's 27, which reduces unnecessary churn in resource provisioning [Figure 8]. The agent was able to understand the workload and scale services robustly. This helps to improve system stability and lower the infrastructure cost too. The DQN agent learns from historical workload patterns and current system metrics and avoids unnecessary up/down scaling that can disrupt performance. The over or under scaling can directly affect latency, which is an important factor in the financial system that can affect SLA and operational costs. This helps in better predictive planning energy-efficient resource usages.

To summarize, the performance comparison of DQN-Agent with Round-Robin in all critical stages and metrics. This includes latency, cache efficiency, and scaling behavior. These results were validated in a dynamic financial environment where microservices ran through reinforcement learning. The ability to adapt, learn from feedback, and make decisions automatically makes this proposed approach a strong candidate for real-world deployment in Fintech infrastructure.

VI. Conclusion and future work

This paper introduced an Agentic-AI framework that uses



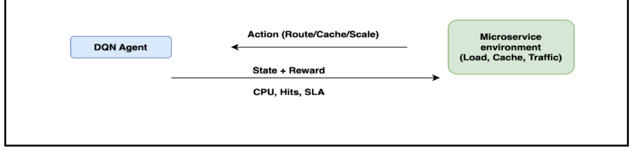
DQN-based reinforcement learning. It optimizes load balancing, predictive caching, and auto-scaling in cloud-native microservices specifically designed for financial systems. The framework was evaluated with a synthetic Fintech transaction dataset. It simulated realistic workload conditions. DQN was benchmarked with a traditional Round-Robin baseline, where DQN consistently outperformed the conventional method. The agent achieved a 47% reduction in latency, a 34% improvement in cache hit rates, and a 48% decrease in scaling, all under identical testing conditions. The success of the DQN agent across all stages tells us that a single reinforcement learning architecture can manage complex service behaviors [6]. This experiment is relevant to financial systems whose core characteristics are market fluctuations, strict SLA requirements [7], and unpredictable transaction patterns. The agent's ability to learn from feedback and make decisions automatically proved its potential of reducing manual intervention. Even though there were promising results seen, this study has its limitations. The dataset was synthetically generated, although the production, like a real-world environment, was simulated. All experiments were done in a controlled environment that was not a production-grade infrastructure. The current system also assumes independent operation of agents without inter-agent coordination.

Future work will aim to address these issues. One way is to adopt multi-agent reinforcement learning (MARL), which will enable distributed agents to coordinate for more optimized outcomes [8]. Another focus should be on the Kubernetes area, which will allow real-time agent deployment and dynamic scaling in production environments [9][11]. It can be further extended for introducing compliance-aware reward signals. This will account for regulatory standards such as PCI-DSS and GDPR [10][12]. Another important addition is the application of the framework to real-world transaction logs from financial systems. This would validate the generalizability of trained agents. Moreover, the introduction of security focus adoption, like anomaly detection agents capable of identifying fraud patterns during traffic spikes, could significantly enhance system resiliency. As the complexity of cloud-native financial infrastructure continues to grow, reinforcement learning-based agents offer a scalable and adaptive path forward. The agents have the capacity for continuous learning, self-optimization, positioning them as a promising solution for the next generation of Intelligent Fintech systems.

VII. Engineering insights

The Agentic AI framework was built using Python 3.10 using the Gymnasium library. It helped with environment simulation and compatibility with reinforcement learning tasks. All experiments were run on the local machine with an NVIDIA RTX 3060 GPU and 32GB RAM. The setup included 5 microservices nodes. Each node was responsible for handling transactions while agents learn to optimize routing, caching, and scaling. A simple flow diagram below illustrates this

Figure 9 : Agent environment interaction loop


DQN Agent
Action (Route/Cache/Scale)
Microservice environment (Load, Cache, Traffic)
State + Reward
CPU, Hits, SLA

below. The DQN agents were trained in 5000 episodes each, where every agent used a network with two hidden layers of 128 and 64 units. A learning rate of 0.001 and a discount factor of 0.95 were used for training. The e-greedy policy started at 1.0 and decreased to 0.01. Target networks were updated every 100 episodes. Each agent had a specific design task. To route an incoming request, the load balancer used CPU usage and transaction priority to decide which node to route. The reward was given according to the server's stability and avoiding overloads. The caching agent learned when to store or evict the cache using access patterns and hit or miss feedback. The auto-scalar always watched CPU usages and traffic trends to decide when to scale up and when to scale down microservices.

Some issues came up during the training. For example, agents were often overfit to peak traffic patterns, or they were evicting data too quickly. This was fixed by adding randomness to traffic patterns by time-weighted scores. All stages of the framework were modular so that in the future they can be reused or replaced easily. In future, this setup can be integrated with Kubernetes or switching to multi-agent systems in an easier manner. Hence, it provides a reproducible foundation were real-world deployment strategies, compliance-awareness tuning can be explored. All these can be achieved without redesigning the core architecture of this framework.

Acknowledgement

The author acknowledges the use of Grammarly for grammar refinement and proofreading during manuscript preparation. Figures and diagrams in this paper were generated using Python, with the assistance of OpenAI's ChatGPT for code-based figure creation and formatting.

References

[1] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

[2] A. Moritz et al., "Learning-based Load Balancing for Cloud Services," IEEE Transactions on Cloud Computing, vol. 9, no. 2, pp. 510–522, 2021.

[3] M. Villamizar et al., "Evaluating the performance of serverless computing for mobile and web applications," IEEE Latin America Transactions, vol. 18, no. 9, pp. 1511–1520, 2020.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., et al. "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[5] Padhi, S.,"Intelligent API Caching for Scalable Cloud Microservices in Financial Systems,"International Research Journal of Modernization in Engineering Technology and Science (IRJMETS), vol. 6, no. 12, pp. 3218–3225, Dec. 2024.

[6] Mao, H., et al. "Resource management with deep reinforcement learning." ACM HotNets, 2016.

[7] Xu, L. D., et al. "Edge computing and AI for IoT in financial services." IEEE Internet of Things Journal, 2022.

[8] Foerster, J., et al. "Learning to communicate with deep multi-agent reinforcement learning." NeurIPS, 2016.

[9] Zhang, Y., et al. "Autonomous resource management with Kubernetes and reinforcement learning." IEEE Cloud Computing, 2020.

[10] Sarker, I. H. "AI in finance: Explainability, fairness, and regulatory compliance." AI Open 3 (2022): 1–1

[11] L. Wang, G. Pierre, and C. Zhang, "Dynamic scaling of microservices in Kubernetes using machine learning," IEEE Transactions on Cloud Computing, vol. 11, no. 1, pp. 1–14, 2023.

[12] G. E. Hinton, et al., "Explainable AI in financial services," IEEE Intelligent Systems, vol. 35, no. 5, pp. 32–41, 2020.