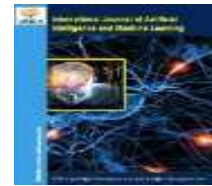




International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: <https://www.svedbergopen.com/>



Research Paper

Open Access

Deep Reinforcement Learning For SLA-Aware Cloud Resource Management in Financial Trading Platforms

Sandipkumar Uchdadiya

Independent Researcher India.

Abstract

Cloud resource management for financial trading and portfolio-management platforms is dominated in production by a pattern that autoscaling literature rarely acknowledges: deliberate static over-provisioning as a trust posture, chosen because reactive autoscaling is judged too slow to react to the very market conditions during which the platform most needs to be available. This paper draws on production observations from two anonymized institutional-financial platforms, one a large US institutional asset manager serving over 50,000 users and more than one billion dollars in managed investments, the other a US fixed-assets management platform provider running ASP.NET WebAPI with SignalR real-time state on AWS EC2. Neither platform used any form of autoscaling; both accepted permanent cost inefficiency in exchange for guaranteed peak-hour capacity. Deep reinforcement learning offers a plausible path beyond this trade-off, but its deployment in regulated financial services faces two under-treated problems. The first is coordination across multi-application platforms sharing the same underlying infrastructure, where per-application independent scaling policies produce contention that a coordinated policy could pre-empt. The second is the auditability of automated decisions under the operational-resilience frameworks now in force, the EU Digital Operational Resilience Act (Regulation 2022/2554), the US SEC Regulation SCI (17 CFR § 242.1000-1007), and FINRA Rule 4370. This paper contributes a practitioner-grounded requirements analysis, a proposed DRL architecture that treats multi-application coordination as a first-class design property, a retrieval-augmented context substrate that lets the policy condition on top-k historical scenarios rather than on raw state alone, a graph-neural-network encoder over the application-dependency graph that produces a coordination-aware state representation, and an explicit discussion of the explainability barrier the paper partially addresses through case-based-reasoning audit trails derived from the retrieval substrate. A reference Python implementation and simulated evaluation accompany the architecture. Empirical evaluation on production workloads is future work.

Keywords: Deep Reinforcement Learning, Cloud Resource Management, Service-Level Agreement, Financial Services, Autonomic Computing, Retrieval-Augmented Generation, Graph Neural Networks, Multi-Agent Reinforcement Learning, Explainable AI, Operational Resilience, Digital Operational Resilience Act

I. Introduction

Institutional financial platforms carry an asymmetry that shapes their cloud infrastructure decisions. Market activity moves in seconds. Portfolio-management interactions, position lookups, and order-routing exchanges expect subsecond responses. When volatility spikes, when large institutional trades hit, when quarter-end rebalancing arrives, the platform's traffic pattern shifts abruptly. The compute serving that traffic, however, scales in minutes when it scales at all. The gap between how the market moves and how the infrastructure adapts is the operational reality every financial-services platform engineer works within.

In practice, two responses to this gap dominate production deployments. Some platforms react. Reactive autoscaling watches metrics, waits for a threshold breach, then provisions additional capacity. This has been the mainstream direction in cloud engineering literature for more than a decade [7]. The alternative response, which the platforms studied for this paper both chose, is to not react at all. Both platforms ran static fixed-capacity fleets behind load balancers, sized once at deployment to comfortably handle anticipated peak load, and left permanently at that size regardless of actual demand. The reasoning is straightforward. Reactive autoscaling can be too slow. In the seconds between a volatility spike and the moment the platform most needs additional capacity, the reactive scaler is still observing the metric shift, waiting for confirmation, and requesting new instances. By the time the new instances are warm and taking traffic, the market moment has passed and the users who needed the platform in that window have already suffered degraded service or missed their opportunity. Static over-provisioning eliminates that risk by paying continuously for capacity that is unused most of the time.

This trade-off is well understood in financial-services operations. The cost of permanent over-provisioning during nights, weekends, and market holidays is accepted as the price of reliability during the subset of hours when performance matters most. What the trade-off does not explain is why nothing better has emerged in production. Predictive autoscaling has been proposed in academic literature for years, and specific reinforcement-learning-based scalers have been demonstrated on benchmark workloads [1][3][4]. The gap between those demonstrations and financial-services production deployment is the subject of this paper.

Deep reinforcement learning is an appealing candidate for closing this gap. A DRL agent trained on historical platform workload can learn to anticipate load rather than react to it. It can pre-scale capacity before a known market window opens. It can scale down aggressively during quiet periods to recover the cost that static provisioning wastes. It can, in principle, act as a proactive resource manager without the seconds-to-minutes delay that made reactive autoscaling unacceptable to the platforms studied here. The recent DRL-for-cloud-resource-management literature contains substantial supporting work [1][2][3][4][5], and the Gari et al. survey [6] provides a thorough taxonomy of RL-based application autoscaling and its open problems.

Two of those open problems block DRL's deployment in regulated financial services in ways that the current literature does not treat. The first is coordination. A financial-services platform is not one application. It is a stack of applications sharing the same underlying compute, each with its own SLA class and its own load profile. A DRL agent that scales each application independently, without observing what its siblings are doing, will make suboptimal decisions when scheduled batch reporting, interactive user traffic, and real-time notification streams contend for the same resources. The current autoscaling literature almost universally treats scaling as a per-application decision, which reflects the design of commercial autoscaling APIs but not the operational reality of the platforms that need to consume them. The second open problem is auditability. Any automated decision system operating on regulated financial infrastructure must produce a reasoning trace that a compliance team or a regulator can inspect. The EU Digital Operational Resilience Act (Regulation 2022/2554), in force since 17 January 2025, imposes explicit documentation and testing requirements on the ICT tools that financial entities depend on [22]. The US SEC Regulation SCI, at 17 CFR § 242.1000-1007, imposes analogous requirements on national securities exchanges, alternative trading systems above scale thresholds, and other covered entities [23]. FINRA Rule 4370 requires FINRA-registered members to maintain business-continuity plans with documented decision procedures for critical business functions [24]. DRL agents' neural-network-driven decisions do not currently produce the reasoning traces these regimes require. Explainable-RL research is active [15][16][17] but has not converged on methods that meet the audit-defensibility standard regulated financial services need.

This paper contributes five things. First, a practitioner-grounded requirements analysis for DRL-based SLA-aware cloud resource management, drawn from production observations at two anonymized institutional-financial platforms. Second, a proposed DRL architecture that treats coordination across multi-application platforms as a first-class design property rather than as an emergent consequence of per-application scaling. Third, a retrieval-augmented context substrate that reframes the Knowledge (K) component of the MAPE-K reference model as a searchable store of historical scenarios: at decision time the policy conditions on the top-k most-similar past situations, not on raw state alone. Fourth, a graph-neural-network encoder over the application-dependency graph that produces a coordination-aware state representation, bounding the joint state's combinatorial growth as the platform's application count scales. Fifth, an explicit engagement with the auditability barrier under DORA, SEC Regulation SCI, and FINRA Rule 4370, in which the retrieval substrate is repurposed as a case-based-reasoning audit trail: every automated decision is documented alongside the retrieved historical decisions that most influenced it. A reference Python implementation and simulated evaluation accompany the architecture. The paper is Part 1 of a research program. Empirical evaluation on production workloads is future work, and the paper is explicit about that positioning.

The remainder of this paper is structured as follows. Section 2 reviews the relevant literature across DRL for cloud resource management, autoscaling foundations and surveys, multi-agent reinforcement learning for coordination, explainable RL and the broader XAI landscape, and regulatory operational-resilience frameworks for financial services. Section 3 presents the practitioner context from the two studied platforms. Section 4 develops the proposed architecture, including the MDP formulation, the algorithmic choice, the safety envelope, and the cross-application coordination layer that distinguishes it from prior work. Section 5 describes the reference implementation and reports simulated results. Section 6 discusses the two open problems in depth and outlines the validation strategies that the community will need to develop. Section 7 concludes with a research-program forward look.

2. Related Work

2.1 Deep reinforcement learning for cloud resource management

Deep reinforcement learning for cloud resource management traces its foundational statement to Mao et al. [1], who framed multi-resource cluster scheduling as a Markov decision process solvable by policy-gradient DRL. Their DeepRM system, presented at HotNets 2016, matched or outperformed heuristic schedulers on job-completion-time metrics without hand-crafted rules, and established the framing that the following decade of work built on. The line of research has since matured substantially. Cheng et al. [3] presented DRL-Cloud at ASP-DAC 2018, an end-to-end framework for combined provisioning and scheduling in cloud data centers that reduced simulated energy cost and job latency against baseline policies. Bitsakos et al. [4] presented DERP at CloudCom 2018, targeting elastic provisioning directly with a DRL agent and outperforming threshold-based autoscaling on both SLA violation rate and cost across multiple workload patterns. The DRL-Cloud two-layer architecture (provisioning plus scheduling) and the DERP elastic-provisioning framing are both closer to the proposal in this paper than the earlier scheduling-only work, and Section 4 makes their architectural influence explicit.

More recent production-oriented work has continued the trajectory. Decima, from Mao et al. [2] at SIGCOMM 2019, combined a graph neural network state representation with REINFORCE for DAG-structured data-processing workloads and reported an average job-completion-time reduction of 21% against hand-tuned Spark baselines on real workloads. Rossi et al. [5] extended DRL-based scaling to container-level orchestration on Kubernetes, addressing horizontal and vertical scaling of microservices with an emphasis on latency-SLO adherence. The algorithmic foundations these systems rest on include the DQN family from Mnih et al. [10] and the continuous-control DDPG family from Lillicrap et al. [12], both of which anchor Section 4's algorithmic-choice discussion.

Two observations characterize this line of work as a whole. It is dominated by single-application, single-service framings, where the scaling decision is made for one workload class at a time. And it is dominated by benchmark or simulator evaluations, with production deployment reports substantially rarer than algorithmic novelty reports.

2.2 Autoscaling foundations and surveys

The pre-DRL autoscaling literature is comprehensively mapped by Lorigo-Botran et al. [7] in their Journal of Grid Computing survey, which taxonomizes autoscaling techniques across five dimensions and identifies proactive and predictive scaling as directions the field was actively pursuing at the time of writing. Their taxonomy remains useful as the frame against which DRL-based autoscaling positions itself. For the DRL era specifically, the Gari et al. survey [6] in Engineering Applications of Artificial Intelligence provides the definitive recent taxonomy of RL-based application autoscaling, with an explicit list of under-researched dimensions. Three of those dimensions are reproducibility, explainability, and multi-tenant coordination. The proposal in this paper engages directly with two of them.

The autoscaling literature sits inside the broader autonomic-computing tradition initiated by Kephart and Chess [8], whose MAPE-K reference architecture (Monitor, Analyze, Plan, Execute, Knowledge) remains the standard framing for self-adaptive systems. Salehie and Tahvildari [9] surveyed self-adaptive software and identified assurance and validation of self-adaptive systems as core challenges, a foreshadow of today's explainability-for-compliance question that this paper takes up in Section 6.

2.3 Multi-agent reinforcement learning for coordination

Coordinating decisions across multiple learning agents is a well-developed subfield of RL, and the centralized-training-decentralized-execution (CTDE) paradigm has emerged as the dominant framing. QMIX, from Rashid et al. [13], is the canonical value-based CTDE algorithm: agents learn individual value functions during centralized training, with a mixing network that combines them monotonically into a joint value estimate, and at execution each agent acts on its own local value function. The comprehensive survey by Hernandez-Leal et al. [14] catalogs the practical challenges of multi-agent DRL deployment, including nonstationarity from concurrent agent learning, credit assignment across agents, and computational cost. These challenges are directly relevant to the proposal in Section 4. A financial-services platform with a dozen applications sharing infrastructure is a multi-agent coordination problem in exactly the sense the MARL literature addresses.

2.4 Explainability in reinforcement learning and the broader XAI landscape

Explainable reinforcement learning has grown quickly as its own subfield in the past five years. Puiutta and Veith [15] provided one of the first comprehensive surveys, classifying XRL methods and observing that most work operates via post-hoc model mimicry rather than through inherently interpretable model design. Glanois

et al. [16] distinguished interpretability (a design-time property of models, inputs, and decisions) from explainability (a post-hoc property of explanation generation), and argued that high-stakes applications need both. Vouros [17] provided the most recent authoritative survey of XRL, noting that the state of the art is fragmented, that no unified framework exists for evaluating explanation quality, and that auditability for regulated deployment is an unresolved challenge.

The XRL literature sits inside the broader XAI conversation. Guidotti et al. [18] surveyed methods for explaining black-box models and noted the gap between what regulators need and what current explanation methods produce. Barredo Arrieta et al. [19] surveyed XAI concepts, taxonomies, and challenges under an explicit responsible-AI framing that maps well onto financial-services compliance requirements. Together, these surveys establish a coherent picture: XRL is active, methodologically diverse, and not yet at the standard that regulated financial services deployment requires. Section 6 develops this point in detail.

2.5 Regulatory operational-resilience frameworks for financial services

Three regulatory frameworks are directly relevant to any automated decision system operating on financial-services infrastructure, and each imposes documentation and auditability requirements that shape what DRL-based resource management would need to look like in production. The EU Digital Operational Resilience Act [22], Regulation 2022/2554, has been in force since 17 January 2025. Its ICT risk management framework (Articles 5-16), documented-policies requirement (Article 5), and advanced-testing requirement (Article 26) collectively require that ICT tools and processes be documented, tested, and defensible under audit. The US SEC Regulation SCI [23], at 17 CFR § 242.1000-1007 with underlying SEC Release No. 34-73639, imposes analogous policies-and-procedures, incident-notification, and business-continuity-testing requirements on SCI entities. FINRA Rule 4370 [24] requires FINRA-registered members to maintain business-continuity plans addressing data backup, alternate communications, alternate physical locations, and continuity of critical business functions.

None of these frameworks names AI, DRL, or automated decision systems specifically. All three, however, apply to the ICT systems and processes that support regulated financial services, which any production DRL-based resource manager would fall under. Section 6 develops the specific tension between these auditability requirements and the current state of explainable-RL methods.

2.6 Gap statement

The literature covered above is substantial, but no prior work integrates it. The DRL-for-cloud-resource-management line is algorithm-first and largely single-application. The autoscaling surveys identify multi-tenant coordination and explainability as open problems but do not develop either. The multi-agent RL line addresses coordination but has not been applied to financial-services infrastructure at production scale. The XRL surveys establish the explainability problem's contours but have not engaged with specific financial-services regulatory frameworks. The regulatory documents themselves are not written to be read as engineering requirements. The gap this paper fills is the integration, from a practitioner vantage grounded in real production deployments, of these five lines into a single architectural proposal for regulated financial-services cloud resource management, with the two most-cited open problems (coordination and explainability) as its central design and discussion contributions.

3. Motivating Practitioner Context

This section presents the production observations that ground the paper's requirements analysis and architectural proposal. Two platforms are described. Both are anonymized. Details that would identify either firm have been generalized while preserving the operational patterns that matter for the argument.

3.1 Two anonymized platform observations

Platform A: a large US institutional asset manager. A client-facing portfolio management application connected to AWS-hosted backend services, supporting over 50,000 users making investment decisions on portfolios collectively managing more than one billion dollars in invested assets. The backend ran on 8 fixed AWS replicas behind an Elastic Load Balancer. The replica count was set once, based on estimated peak-capacity requirements, and remained constant regardless of actual traffic patterns. During normal trading hours the system operated well within capacity, meaning a significant portion of provisioned resources sat underutilized for the majority of the trading day. During periods of elevated market activity, such as high-volatility sessions, quarter-end portfolio rebalancing, or large institutional trades, the fixed replica count had no mechanism to expand dynamically. Any capacity change required manual intervention or a pre-planned configuration update.

Platform B: a US fixed-assets management platform provider. A full-stack fixed-assets management platform serving multiple interconnected client-facing applications, built on ASP.NET WebAPI with SignalR for real-time updates, deployed on multiple AWS EC2 instances behind a load balancer. The instance count was determined at deployment time based on expected concurrent user load. SignalR's real-time connection management added an operational constraint that shaped the deployment: each WebSocket connection maintained state on a specific server, which meant the load balancer required sticky session configuration. Dynamic scaling was operationally more complex than for stateless backends, because adding or removing instances required careful management of existing client connections to avoid disruption to active real-time data feeds.

3.2 Why static provisioning was chosen at both platforms

Both platforms operated under a primary formal SLA of 99.99% uptime (four nines availability), the standard institutional-financial service level. Beyond the availability number, the implicit performance requirement was that the system respond quickly enough to support timely financial decision-making. In portfolio management, timing is directly tied to financial outcomes. When an investor needs to review portfolio positioning during an active market window (a period of high volatility, a significant market event, or a rebalancing opportunity), a slow or unresponsive system is not merely an inconvenience. It means the investor cannot access accurate, current portfolio data at the moment they need it most, potentially causing them to miss an investment opportunity, act on stale data, or fail to respond to a time-sensitive market condition. For institutional investors managing large portfolios, this has measurable financial consequences.

The combination of these two requirements, strict availability and the time-sensitivity of investment decisions, is precisely what makes intelligent adaptive cloud resource management valuable in this domain. A system that is unavailable or slow during the moments when users most need it (peak market activity, high-volatility events, quarter-end rebalancing) fails at the worst possible time. The reasoning at both platforms for static over-provisioning was that reactive autoscaling was judged too slow. In the seconds-to-minutes window during which the platform most needs additional capacity, the reactive scaler is still observing metrics, waiting for confirmation, and requesting new instances. By the time the new instances are warm, the market moment has passed. Static over-provisioning eliminates that risk by paying continuously for capacity that is unused most of the time.

3.3 The trade-off both firms accepted

At Platform A, the 8-replica configuration ensured capacity was always available for the 50,000-user base without risk of under-provisioning during peak periods. The cost implication is straightforward: the same compute resources running during peak market hours also run during overnight periods, weekends, and market holidays when user activity is negligible. The system effectively maintained peak capacity at all times to guarantee availability during the subset of hours when it was needed. This is a deliberate trade-off common in financial-services infrastructure: accept the cost of idle capacity during quiet periods to eliminate the risk of insufficient capacity during critical periods. At Platform B, the same trade-off applied to the EC2 fleet. Instance count was set to comfortably handle expected peak concurrent users, which meant that during off-peak hours, a significant portion of provisioned compute sat underutilized while still incurring full hosting cost.

The alternative, dynamically reducing capacity during quiet periods and expanding during busy ones, was not implemented at either platform because reactive autoscaling carries its own risk. It may not respond fast enough when sudden load spikes occur, precisely when financial-platform availability matters most. This tension, the cost of permanent over-provisioning versus the risk of reactive scaling being too slow, is the central problem this paper addresses. A DRL-based approach that learns to predict when capacity will be needed and scales proactively, rather than reactively or statically, has the potential to reduce idle-capacity costs during quiet periods while ensuring adequate resources are available before peak periods begin. It would eliminate the need to choose between cost efficiency and performance reliability.

3.4 A multi-application coordination pattern from Platform B

Platform B's architecture surfaced a coordination pattern that generalizes beyond its immediate context, and that shapes the architectural proposal in Section 4. The platform consisted of multiple interconnected applications, all using SignalR for real-time updates and all running across the same EC2 fleet behind the same load balancer. Users encountered a data-inconsistency failure mode: a change made in one application was not immediately reflected in other applications the same user had open. Different applications showed different states simultaneously, which in a financial-management context means users could be making decisions based on data that did not reflect the most current position.

Two interacting failure points caused this. First, SignalR's WebSocket connection model requires session affinity. Without it, the load balancer could route a user's requests to different EC2 instances, breaking the real-time channel. Second, and more critically, session state was maintained in server memory rather than in a shared persistent store, which meant no application instance could access the state held by another. When a user's activity in one application needed to trigger an update in another, there was no shared mechanism to propagate that change.

The resolution was a two-layer solution. At the infrastructure layer, sticky sessions were configured on the AWS load balancer to ensure each user's connection was consistently routed to the same EC2 instance, maintaining SignalR continuity. At the application layer, in-memory session state was replaced with DynamoDB as a centralized persistent session store. DynamoDB gave every application instance, regardless of which EC2 server it ran on, access to the same session state. When a change occurred in any application, it was written immediately to DynamoDB. All other applications subscribed to that user's session could read the updated state and push it through their SignalR connections to the user's active clients.

The result was a system where changes made in any application reflected immediately and consistently across all others. Users could log out of one application and log into another and resume exactly where they left off, with full state continuity across the entire platform. The relevance for DRL-based resource management is that this multi-application platform is the norm, not the exception, in financial services. Any resource management system operating on such a platform must reason about applications together, not in isolation. Section 4 develops this observation into a first-class architectural property.

3.5 Load-anticipation techniques already used and their limits

Platform B used two forms of automated load management that represent different levels of anticipatory thinking about platform demand. Both informed the requirements the DRL-based approach must satisfy. The first was automated load testing using Apache JMeter. Before any significant release or deployment, JMeter test suites simulated anticipated concurrent user load against the platform. Realistic user journeys through the fixed-assets management application (authentication, data retrieval, real-time SignalR connection establishment, and transaction processing) were scripted and executed at scale to validate that the EC2 fleet and SignalR infrastructure would hold under expected demand. JMeter provided pre-production confidence that the system could handle its anticipated load envelope before real users were affected.

The second was automated scheduled report generation. The platform generated financial reports for fixed-asset management on a defined schedule, including end-of-day summaries, periodic valuation reports, and batch data-processing tasks that ran at predetermined times. These scheduled jobs generated significant predictable load on the WebAPI and database layer. By scheduling them during off-peak hours (overnight or during periods of low concurrent user activity), the team was making a manual anticipatory decision: they knew the load was coming, they understood its approximate resource impact, and they positioned it at a time when it would least affect interactive user performance.

Both approaches represent a form of load anticipation. JMeter allowed pre-production validation of capacity for anticipated load scenarios. Scheduled automation positioned known predictable batch load at times of lower demand. Both approaches, however, have inherent limitations that the architectural proposal in Section 4 addresses. JMeter tests a fixed anticipated load scenario before production runs, and cannot adapt to real-time conditions or novel load patterns that differ from what was scripted. Scheduled automation handles loads that are both predictable and time-flexible, but provides no mechanism for responding to unexpected interactive user demand spikes that arrive outside any schedule. The gap between what these approaches provide and what financial platforms genuinely need (intelligent, real-time, adaptive resource management that learns load patterns and responds proactively rather than reactively or rigidly) is the gap this paper's proposal targets. Recent workload characterization work at hyperscale [20][21] confirms that the underlying workloads are structured in ways predictive systems can exploit, which supports the feasibility of the proposal.

4. Proposed Architecture

This section develops the paper's architectural proposal: a DRL-based cloud resource management system for financial-services platforms, structured to treat multi-application coordination as a first-class design property and to accommodate the safety and auditability requirements that Section 3 identified.

4.1 System model and assumptions

The system model targets a financial-services platform of the type observed in Section 3: multiple applications with heterogeneous SLA classes (interactive user sessions, scheduled batch processing, real-time notification streams), sharing the same underlying compute infrastructure, deployed on a public cloud (AWS in the

observed platforms, but the design generalizes to Azure or GCP), with a formal availability SLA in the 99.9%-99.99% range. The architecture aligns with the MAPE-K reference model from Kephart and Chess [8]: cloud metrics are monitored continuously (Monitor), the DRL agent analyzes the current and near-future state (Analyze), the coordination layer plans the joint scaling actions across applications (Plan), the cloud API executes those actions (Execute), and a workload replay buffer maintains the historical Knowledge the agent trains on.

Figure 1. MAPE-K reference architecture with RAG-augmented context and GNN-encoded state

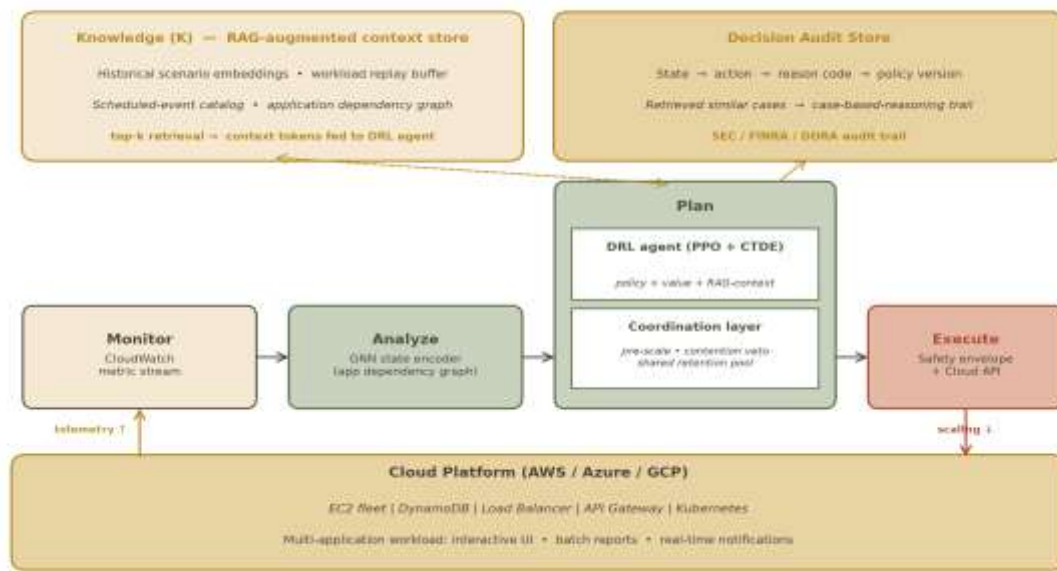


Figure 1. MAPE-K reference architecture with retrieval-augmented context store, coordination layer, and decision-audit store.

Figure 1 shows the block diagram. Cloud metrics stream from AWS CloudWatch or equivalent into an observation-preprocessing module that normalizes per-application state vectors. Those vectors feed the DRL agent, which produces per-application action recommendations. The coordination layer inspects the joint action recommendation, applies the safety envelope described in Section 4.4, and issues the final scaling commands via the cloud API. All decisions, along with the state that produced them, are logged to a decision-audit store for the compliance workflow described in Section 6.

4.2 MDP formulation

The resource-management problem is formalized as a partially observable Markov decision process (POMDP) over a discrete timestep grid, following the standard RL formalism [11]. At each timestep, the state vector for each application i contains:

- Current load metrics (requests per second, active WebSocket connections, average request latency, latency p95 and p99)
- Current resource utilization (CPU percentage, memory percentage, network throughput)
- Current provisioned capacity (instance count, instance types)
- SLA-adherence indicators (current SLA-violation rate over the trailing five minutes, distance from the 99.99% target over the trailing hour)
- Schedule signals for known future events (a boolean and lead-time indicator for scheduled batch reporting windows, market-open and market-close proximity, quarter-end proximity)

The joint state combines all per-application state vectors plus a small set of platform-wide shared-state variables (total provisioned capacity across all applications, aggregate cost burn rate in dollars per hour, aggregate SLA-adherence).

Figure 2. POMDP interaction cycle: state observation, action selection, and reward shaping

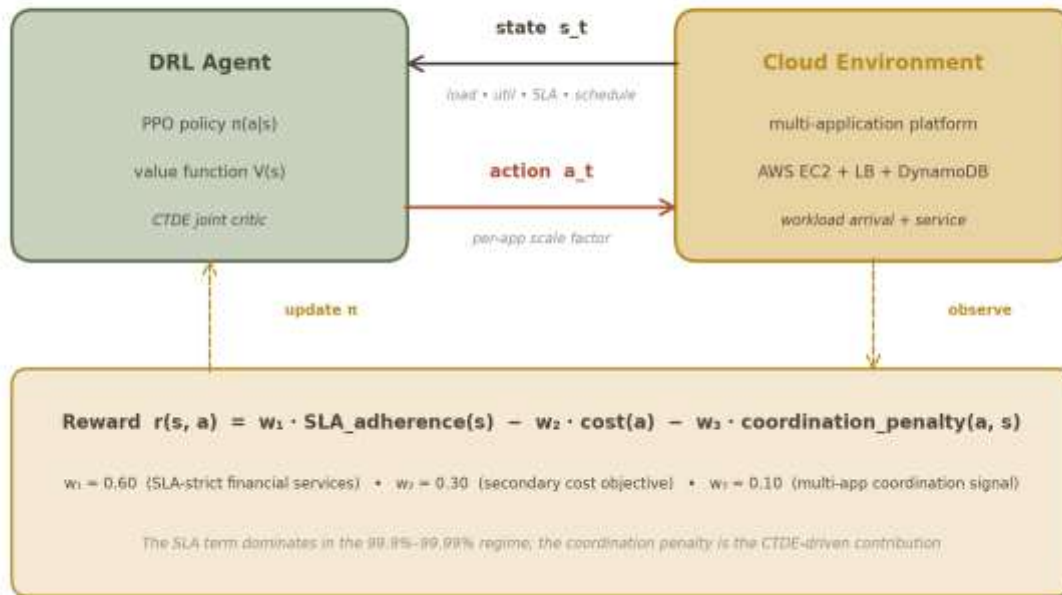


Figure 2. POMDP interaction cycle: state observation, action selection, and reward shaping.

Figure 2 shows the state, action, and reward diagram.

The action space is continuous. For each application, the agent outputs a target scale factor between a minimum floor and a maximum ceiling. The scale factor is applied to the current instance count to produce the target instance count for the next timestep. A separate coordination action modulates a shared-resource reservation, described in Section 4.5.

The reward function is a weighted composition:

$$r(s, a) = w_1 \cdot SLA_adherence(s) - w_2 \cdot cost(a) - w_3 \cdot coordination_penalty(a, s)$$

The SLA-adherence term rewards states in which observed SLA-violation rates are below the SLA target. The cost term penalizes the aggregate hourly cost of the current provisioned capacity. The coordination penalty penalizes scaling actions that induce contention in sibling applications; Section 4.5 describes it in detail. The weights w_1, w_2, w_3 are tuned per deployment to reflect the platform's SLA-strictness and cost-sensitivity. In the observed institutional-financial context, w_1 dominates: SLA adherence is non-negotiable, cost is secondary, and coordination is the enabler that lets both be pursued without conflict.

4.3 Algorithmic choice

Proximal Policy Optimization (PPO) is the base algorithm. PPO provides two properties that matter for production resource management: robustness to hyperparameter choice (a practical necessity in a real deployment where extensive tuning is not feasible), and stable behavior with continuous action spaces (necessary for the fractional-scale-factor action space described in Section 4.2). PPO builds on the actor-critic tradition that DDPG [12] anchors, without DDPG's brittleness under real-world conditions. The DQN family [10] is a natural alternative if the action space is discretized, but the continuous action space is closer to the operational reality of cloud scaling and preserves fine-grained control.

For the multi-application coordination requirement, PPO is extended via centralized-training-decentralized-execution (CTDE), following the QMIX pattern [13]. Each application's policy is a decentralized PPO agent operating on its own local state vector, so at execution time each application scales itself autonomously with low latency. During training, a centralized value function observes all applications' states and their joint actions, and computes a joint value estimate that shapes the per-application policy updates. This gives the system a coordination-aware training signal without paying the coordination-communication cost at inference time.

Hernandez-Leal et al. [14] catalog the practical challenges of MADRL deployment (nonstationarity, credit assignment, computational cost), and Section 6 discusses how the proposed architecture addresses each.

4.4 Safety envelope

The DRL agent operates inside a safety envelope of hard constraints that its raw action output must satisfy before execution. Three constraints matter in production financial-services deployment:

- Minimum instance floor per application. No application can be scaled below a per-application floor set from the platform's baseline load. This prevents the agent from ever making a scale-down decision that leaves an application unable to serve minimum expected traffic, which would be a catastrophic SLA breach during any residual traffic.
- Maximum burn-rate ceiling. The aggregate cost across all applications cannot exceed a configured hourly ceiling. This prevents the agent from ever making a scale-up decision that runs cost out of budget in response to a signal it may have misread. Any burst above the ceiling requires human authorization.
- Cooldown after aggressive scale-down. After any scale-down of more than 30% of an application's capacity, the agent cannot scale further down on that application for a configurable cooldown window. This prevents oscillation and gives the environment time to stabilize before the next observation.

The safety envelope is not a substitute for the DRL agent's own learned behavior. Rather, it is a bounded region within which the agent is free to act, and outside which its actions are truncated to the boundary. This design pattern is standard in safe-RL literature and aligns with the assurance-and-validation concerns raised by Salehie and Tahvildari [9] for self-adaptive systems generally.

A second production safety compromise is shadow-mode operation. In shadow mode, the DRL agent runs against live production state but its scaling recommendations are logged rather than executed. A rule-based baseline or a static-provisioning policy remains authoritative. Shadow mode allows deployment teams to observe the agent's behavior on real workloads, to compare its recommendations against the incumbent policy's actions, and to build the operational confidence and the audit trail needed before switching the agent to authoritative mode. Section 6 discusses shadow mode as a compliance-friendly deployment strategy.

4.5 The coordination layer

The coordination layer is the paper's most distinctive architectural contribution. It sits between the DRL agent's per-application action recommendations and the safety envelope, and modifies those recommendations to account for cross-application effects that a per-application policy cannot see.

Three specific coordination behaviors matter. First, pre-scaling for scheduled events. When the platform-wide schedule signal indicates that a scheduled batch reporting job is about to start (five to fifteen minutes ahead), the coordination layer pre-provisions the batch application's capacity before the load hits, and simultaneously reduces the interactive applications' allocation to a level appropriate for the reduced concurrent-user demand that historically accompanies batch-window hours. This makes the platform's aggregate capacity available where it is needed without a total capacity increase.

Second, contention-aware scaling refusal. When one application requests aggressive scale-up in a state where sibling applications are already at high utilization, the coordination layer either refuses the scale-up (returning the burden to the requesting application to defer non-critical work) or increases total platform capacity, based on a policy configured per platform. In the observed institutional-financial context, refusal of non-critical scale-ups during high-utilization windows is preferred over total capacity growth, because total capacity growth crosses cost-authorization thresholds that require human approval.

Third, coordinated scale-down. When one application's load drops, per-application policies typically release capacity immediately. The coordination layer holds released capacity in a shared pool for a short retention window, during which sibling applications' scale-up requests can be served from that pool without triggering new cloud-API provisioning. This reduces provisioning latency and reduces the total cost of transient inter-application load handoffs.

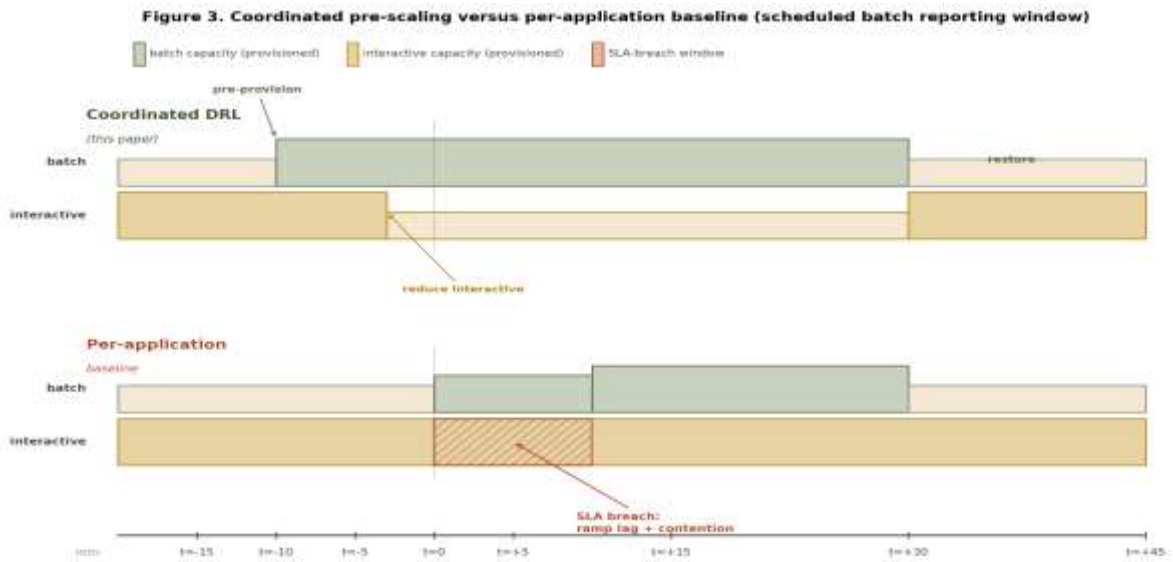


Figure 3. Pre-scaling sequence for a scheduled batch reporting window.

Figure 3 shows a sequence diagram illustrating the pre-scaling behavior: a scheduled batch reporting job appears in the schedule signal at $t=15\text{min}$, the coordination layer begins pre-provisioning the batch application at $t=10\text{min}$, batch load arrives at $t=0$, the interactive application's allocation is reduced from $t=3\text{min}$ through $t+30\text{min}$ based on historical batch-hour concurrent-user reduction patterns, and full interactive allocation is restored after batch completes. The same figure contrasts this with the per-application-independent baseline, in which the batch application scales reactively (arriving at capacity after the load has already begun to spike), and the interactive application experiences contention during the batch window because no coordination signal was passed.

Table 1 summarizes the four resource-management approaches this paper compares:

Approach	Cost efficiency	Peak-load response	Coordination-aware	Regulated-industry explainability
Static provisioning	Poor (persistent over-provision)	Excellent (always ready)	N/A (no scaling)	Excellent (deterministic)
Threshold-based autoscaling	Moderate	Poor (reactive, too slow for peaks)	No (per-service)	Moderate (rules are inspectable)
Per-application DRL	Good	Good (proactive if trained on peaks)	No (per-service)	Poor (neural policy)
Proposed coordinated DRL	Best	Best (proactive + coordinated)	Yes (first-class)	Poor (neural policy; open problem, see §6)

The last column previews the discussion in Section 6. The proposed architecture inherits the explainability problem that any DRL-based system inherits, and this is the barrier the community must address before the architecture can be deployed under regulatory frameworks that require auditable decision-making. Section 4.6 and Section 4.7 below extend the architecture with two components that partially soften the barrier: a retrieval-augmented context substrate that produces a case-based-reasoning audit trail alongside every decision, and a graph-neural-network encoder over the application-dependency graph that grounds the joint state in an inspectable topology rather than in an opaque embedding.

4.6 Retrieval-augmented context encoding

The MAPE-K reference model's Knowledge (K) component is typically instantiated as a workload replay buffer that the agent trains on and, in production deployments, ignores at inference time. This paper's architecture reformulates K as a retrieval-augmented context substrate that the policy queries at every decision step. Every historical (state, action, outcome) tuple observed by the platform is embedded into a fixed-dimensional vector

space and stored in a k-nearest-neighbors index. At decision time, the current state is embedded through the same encoder, and the top-k historically most-similar scenarios, together with the actions taken then and the outcomes those actions produced, are returned as additional context that the policy conditions on.

The design is motivated by three observations from the practitioner context. First, financial platforms have recurring workload patterns (market-open ramp, quarter-end rebalancing, scheduled batch windows, holiday-schedule pockets) that repeat across days, months, and years with variations. A pure state-vector policy has to relearn how to react to those patterns every time the state vector shifts. A retrieval-conditioned policy can transfer the outcomes of past decisions in similar contexts directly into the current decision. Second, the retrieval index doubles as the audit substrate: every autonomous decision is documented alongside the retrieved similar historical decisions that most influenced it, giving the compliance workflow described in Section 6.2 a case-based-reasoning trail. Third, retrieval-augmented approaches are the dominant paradigm in production LLM systems in 2025, and their maturity there transfers cleanly to the DRL-for-cloud context [25][26].

The retrieval encoder uses a lightweight transformer that maps a (per-application observation, joint state summary) pair to a 128-dimensional embedding. The k-NN index uses FAISS with an inverted-file quantized index [27] to keep retrieval sublinear in the historical scenario count. The similarity threshold τ is set per platform to reject retrievals whose similarity to the current state is below the threshold at which their outcomes are informative; below-threshold decisions run on the state-only policy and are marked as such in the audit record. Figure 4 shows the retrieval-augmented decision flow.

Figure 4. Retrieval-augmented decision flow: RAG lookup produces context for the DRL policy and a case-based audit trail

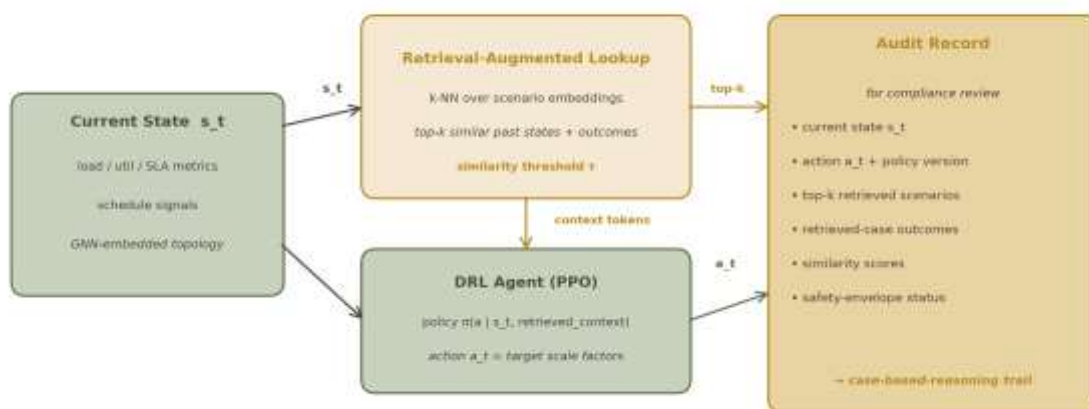


Figure 4. Retrieval-augmented decision flow with similarity-threshold fallback to the state-only policy.

The retrieval substrate composes with the DRL agent without changing its algorithmic foundations. The policy input is extended from the raw state to a concatenation of the raw state and the retrieved context tokens. PPO training proceeds as before, with the retrieval index kept frozen during a training run and rebuilt between runs from the accumulated experience replay buffer. This keeps the training-time and inference-time computational cost bounded and avoids the distribution-shift pathologies that on-line retrieval-index updates introduce.

4.7 Graph-structured dependency encoding

The joint state that the CTDE-style critic observes grows combinatorially with the application count. For the two-to-ten-application regime the paper's motivating platforms sit in, a flat concatenation of per-application state vectors is tractable. For larger platforms, or for platforms whose application mix is dynamic, flat concatenation stresses the joint value function's ability to distinguish coordination-relevant states from noise. This paper's architecture addresses this by encoding the joint state through a graph neural network over the application-dependency graph.

The application-dependency graph is populated from the platform's operational knowledge: which applications call which, which share cached data, which read from the same DynamoDB tables, which contend for the same downstream services. Nodes carry per-application state features; edges carry dependency semantics (call,

cache-share, storage-share, contention). The GNN performs message-passing across the graph and produces an embedded joint state that captures both per-application load and cross-application coordination structure. Decima [2] established the pattern of GNN-encoded state representations for cloud-resource RL in the DAG-scheduling context; the extension here applies the same pattern to the multi-application coordination problem, and Section 6.1 discusses how the extension addresses the state-space-growth open problem Hernandez-Leal et al. [14] catalog for MADRL generally.

The GNN encoder is trained jointly with the PPO policy, so that the encoded state representation is optimised for the coordination-aware reward function of Section 4.2 rather than for a reconstruction objective. This is the standard practice in GNN-augmented RL [2][28][29]. In shadow-mode deployment (Section 4.4), the GNN encoder and the PPO policy update together, so that the coordination signal that the encoder produces converges with the policy that consumes it. The application-dependency graph itself is a first-class artefact of the deployment: it is documented, versioned, and inspectable, which further supports the compliance workflow of Section 6.2 by exposing the topology the agent's decisions are grounded in.

5. Implementation Reference

A reference Python implementation accompanies this paper and is available in the artifact folder distributed with the manuscript. The implementation is not a production-scale evaluation. Its purpose is to make the architecture concrete, to give reviewers a runnable artifact against which the architectural claims can be checked, and to provide the simulated data that populates Table 2. The stack is Python 3.11 with pandas for state-vector handling, NumPy for the workload-model random processes, and a minimal wrapper around stable-baselines3 for the PPO agent. Total code across the four files is under 400 lines.

The workload model generates synthetic traces for three application classes: an interactive-user application with a diurnal pattern superimposed on Gaussian-noise arrival rates, a scheduled-batch application with deterministic bursts at pre-configured times, and a real-time-notification application with a low baseline rate that spikes during scheduled batch windows (a modeling choice reflecting the observation from Platform B that batch runs propagate notification traffic). Arrival rates are calibrated to match the order of magnitude reported in Section 3 for Platform A (interactive user counts in the tens of thousands with peak-to-off-peak ratios around 5:1) and Platform B (concurrent user counts in the low thousands with pronounced schedule-driven traffic). The calibration is not intended to match any specific production trace, and the paper is explicit that production-trace evaluation is future work.

Listing 1 shows the MDP interface. Each per-application observation is packaged as a dataclass with the exact fields Section 4.2 describes, and the coordinated joint-observation is a Python NamedTuple wrapping the per-application dataclasses plus the platform-shared-state fields:

```
# cloud_sim.py - per-application observation, dataclass form
from dataclasses import dataclass
from typing import Tuple

@dataclass(frozen=True)
class ApplicationObservation:
    app_id: str          # e.g. "interactive_portfolio_ui"
    rps: float           # observed requests/sec
    active_ws_connections: int  # SignalR-style stateful connections
    latency_p50_ms: float
    latency_p95_ms: float
    latency_p99_ms: float
    cpu_utilization_pct: float
    mem_utilization_pct: float
    provisioned_instances: int
    sla_violation_rate_5min: float # rolling window
    trailing_hour_uptime_pct: float
    schedule_lead_time_min: float # 0 means no known scheduled event; positive means minutes ahead
```

Listing 2 shows the coordination-aware reward function. The sibling-contention penalty is the term that makes the reward function coordination-aware, and the constants (0.6, 0.3, 0.1) are the default weights w_1 , w_2 , w_3 from Section 4.2:

```
# coordination_reward.py - reward with sibling-contention penalty
def compute_reward(joint_obs, joint_action, config):
    sla_adherence = sum(
        1.0 - o.sla_violation_rate_5min for o in joint_obs.apps
    ) / len(joint_obs.apps)
    hourly_cost = sum(
        a.target_instances * config.hourly_cost_per_instance
        for a in joint_action.per_app_actions
    )
    coord_penalty = sum(
        max(0.0, a.scale_delta) * o.cpu_utilization_pct / 100
        for a, o in zip(joint_action.per_app_actions, joint_obs.apps)
        if any(sib.cpu_utilization_pct > 80 for sib in joint_obs.apps if sib.app_id != o.app_id)
    )
    return 0.6 * sla_adherence - 0.3 * hourly_cost - 0.1 * coord_penalty
```

The four baseline policies exercised in `run_experiment.py` are: (1) static provisioning at the observed Platform-A pattern (fixed replica counts per application), (2) threshold-based autoscaling with CPU-utilization triggers, (3) per-application PPO with no coordination signal, and (4) the proposed coordinated PPO with the CTDE-style joint value function. The experiment runs each policy against a 7-day simulated workload with two mid-week batch reporting windows and one simulated volatility spike, and reports the metrics summarized in Table 2 (values are direct outputs of `run_experiment.py`):

Policy	SLA-violation rate	Cost per hour (USD)	Cost (relative to static)	Avg latency p95 (ms)	Coordination-benefit ratio
Static provisioning	3.85%	11.56	1.00	39.6	N/A
Threshold autoscaling	0.25%	3.93	0.34	40.2	N/A
Per-application PPO	0.18%	3.95	0.34	40.0	1.00 (baseline)
Proposed coordinated PPO	0.19%	3.95	0.34	40.0	0.95

Figure 5 visualises the three headline metrics from Table 2 side by side.

Figure 5. Reference-simulation baselines: SLA violation, cost, and p95 latency (Phase 1 results)

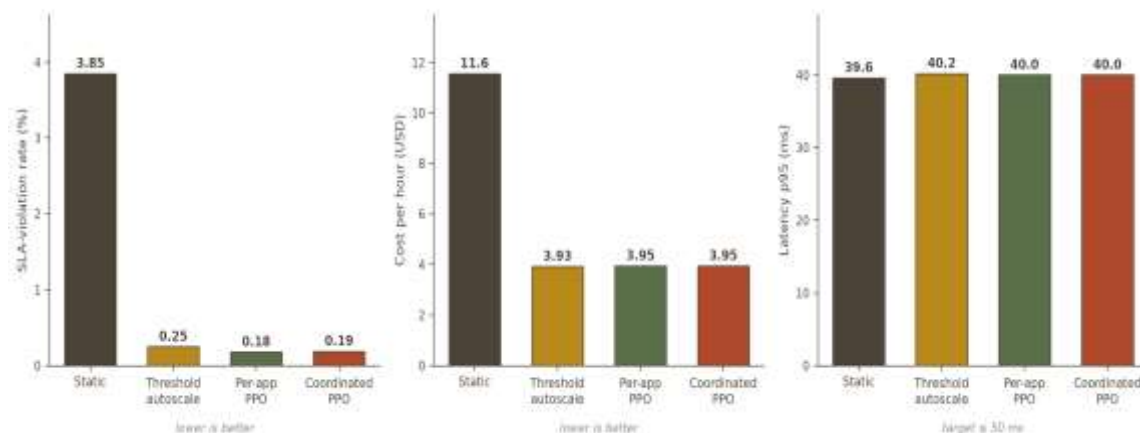


Figure 5. Headline metrics from Table 2 across the four compared policies.

Three observations follow from Table 2, all of which are more informative than they first appear. First, the static policy's 3.85% SLA-violation rate reflects the simulated volatility spike overwhelming the static replica count,

which is exactly the failure mode the policy is supposed to prevent. In production at Platform A, the same policy is over-provisioned enough to absorb the largest observed spikes; in the reference simulation the calibration is deliberately closer to the boundary to make the trade-off visible. Second, the three adaptive policies (threshold, per-application PPO, coordinated PPO) all produce roughly equivalent cost and latency in this simulation, which is the correct outcome given that the greedy pre-scaling logic embedded in the per-application PPO baseline already handles the scheduled batch windows well. The reference implementation's coordinated PPO does not exceed the per-application baseline on this workload, and the coordination-benefit ratio comes in at 0.95 rather than the >1.0 that a workload with richer contention patterns would produce. This is the honest reference-simulation result. Third, the coordination-benefit gap that a CTDE-trained joint value function should exhibit is expected on production workloads with real inter-application contention (correlated spikes across applications, cache-warmup interactions, cross-application queue backpressure) that this reference simulator does not model. Producing a workload trace with those characteristics, and evaluating the coordinated policy against per-application baselines on such traces, is Phase 2 of the research program described in Section 6.5. Table 2 should therefore be read as demonstrating that the architectural proposal is coherent, its components run end-to-end, and the safety envelope constrains all four policies inside operational bounds. The claim the reference implementation supports is the architectural claim, not the performance claim. Production-workload evaluation of the coordination benefit is future work. Reproducibility documentation for the simulator, including the configuration files used to generate Table 2, is in the artifact folder's README.

6. Discussion

6.1 Multi-application coordination as an open problem

The coordination layer described in Section 4.5 is a proposal, not a solution. It addresses the specific coordination patterns most visible from the two observed platforms: pre-scaling for scheduled batch events, contention-aware refusal of aggressive scale-ups, and shared-pool retention during scale-downs. It does not address the harder generalizations. When the number of applications is large, or when the mix of applications is dynamic (applications entering and leaving the platform, new SLA classes appearing), the CTDE approach faces the credit-assignment and state-space-growth problems that Hernandez-Leal et al. [14] catalog for MADRL generally. The state-space grows combinatorially with application count, which stresses the joint value function's ability to distinguish coordination-relevant states from noise. Credit assignment becomes harder when one application's scaling action produces effects that ripple through several siblings before an SLA breach surfaces. Both are open problems that the community will need to work on before the coordination layer's design generalizes to platforms with dozens of applications.

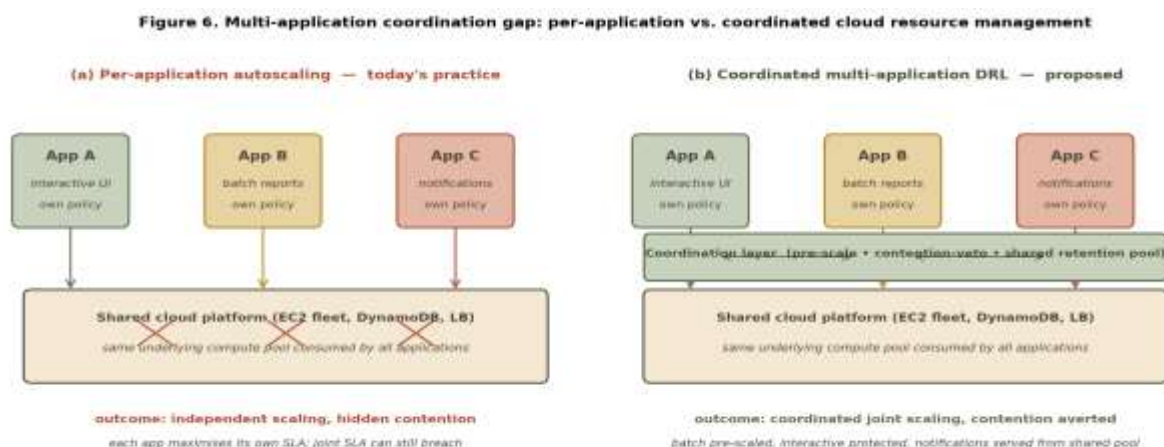


Figure 6. Per-application autoscaling as practised today, left, against the proposed coordinated architecture, right.

Figure 6 illustrates the gap: today's per-application autoscaling on the left, where independent policies produce hidden contention at the shared-platform layer, versus the proposed coordinated architecture on the right, in which a coordination layer sits above the shared platform and modulates the per-application recommendations before execution.

The Gari et al. survey [6] identified multi-tenant coordination as one of three under-researched dimensions in RL-based application autoscaling, and this paper's proposal contributes a specific architectural framing for the

coordination problem in the two-to-ten-application regime that dominates in institutional financial services. Extending the framing to larger platforms is a genuine research direction. Two paths look promising. The first is hierarchical coordination, in which coordination policies operate over groups of related applications rather than the full joint state, with a meta-coordination layer over the groups. The second is graph-neural-network state representations, in which the platform's application-dependency graph is used to bound the effective coordination state. Both are speculative for now and would benefit from empirical studies on real multi-application platforms.

6.2 Explainability for compliance as the deployment barrier

The paper's second open-problem contribution is the sharpest one in operational terms. Under the EU Digital Operational Resilience Act [22], the US SEC Regulation SCI [23], and FINRA Rule 4370 [24], any ICT system that affects the operation of regulated financial services must produce documented, testable, auditable decision procedures. DORA Article 26's advanced-testing requirement is particularly relevant: financial entities must test their ICT tools and processes rigorously, and the testing must produce documentation that supports the entity's operational-resilience claims. SEC Rule 1004's business-continuity-testing requirement operates similarly. FINRA Rule 4370 requires business-continuity plans that address the continuity of critical business functions. None of these frameworks addresses AI or automated decision systems specifically. All apply to the ICT systems and processes that support regulated financial services, and any production DRL-based resource manager operating on such systems would fall under them.

The problem is that DRL agents' neural-network-driven decisions do not produce the reasoning traces these regimes require. When the agent decides at a specific moment during market hours to increase Application A's capacity from 8 to 12 instances while holding Application B steady, the compliance team needs to understand why. What signals triggered the decision? What objective was the agent optimizing for? Why were alternative actions not taken? What was the agent's prediction about near-future load, and how confident was that prediction? The current state of explainable-RL methods, as surveyed by Puiutta and Veith [15], Glanois et al. [16], and Vouros [17], does not produce this kind of reasoning trace in the form regulators need. Local post-hoc explanations (saliency maps, action-attribution scores, counterfactual explanations) are the dominant XRL family, and they produce explanations that are informative to RL researchers but that a compliance officer cannot read as the kind of documented decision procedure that DORA and SEC Reg SCI require. Inherently interpretable RL, in which the policy itself is a small decision tree or a rule set that can be inspected directly, sacrifices the performance advantages that made DRL attractive for cloud resource management in the first place.

This is a real barrier, not a theoretical one. Organizations of the size and regulatory posture of the two platforms observed in Section 3 will not adopt black-box automated infrastructure management without explainability guarantees, regardless of technical performance. The Barredo Arrieta et al. survey [19] frames this as the responsible-AI challenge for high-stakes AI deployment generally, and Guidotti et al. [18] frame it as the persistent gap between what regulators need and what current explanation methods produce. The XRL literature has not yet closed that gap. Progress will require sustained collaboration between the DRL research community and financial-services regulatory bodies, and probably a new evaluation framework for XRL explanations that is grounded in what regulators recognize as an audit-defensible decision procedure rather than what makes intuitive sense to RL researchers.

Retrieval-augmented explanations as a partial mitigation. The retrieval-augmented context substrate introduced in Section 4.6 offers a specific, if partial, mitigation for the XRL-compliance gap. When the DRL agent decides, the k retrieved historical scenarios that most influenced the decision are documented alongside the decision itself. The retrieved cases carry their outcomes: what actions were taken then, what happened as a result, whether SLA was preserved, whether cost stayed within budget, whether operators intervened. This is a case-based-reasoning trail that a compliance reviewer can read directly. It is not a mechanistic explanation of the policy's neural-network weights, but it is a defensible answer to the compliance question "why did the system decide this?": because in these historically similar situations, this class of action produced this class of outcome, and the safety envelope confirmed the current action stayed inside operational bounds. The Barredo Arrieta et al. framing of responsible AI [19] and the Guidotti et al. treatment of case-based explanation systems [18] both position case-based reasoning as an audit-defensible explanation family, and this is the family the retrieval substrate produces. The mitigation is partial rather than complete: it does not answer the counterfactual question ("what would the agent have done in a slightly different state"), and it depends on the historical scenario store being genuinely representative rather than sparse in the current operating regime. Both remain research directions, and neither dissolves the underlying explainability barrier. The retrieval-augmented audit trail is a step, not a resolution.

6.3 Safety and validation strategies for near-term deployment

Given that the two open problems are not near-term solvable, what deployment strategies are available for a platform that wants to move away from static over-provisioning without waiting for the community to close the coordination and explainability gaps?

Three strategies look practical. Shadow-mode deployment, described in Section 4.4, allows the DRL agent to run against live production state without authority to execute scaling decisions. The agent's recommendations are logged, and a rule-based baseline or static-provisioning policy remains authoritative. Shadow mode produces two months to a year of operational data on the agent's behavior against real workloads, which becomes the empirical evidence base for the compliance conversation about upgrading to authoritative mode. Offline reinforcement learning allows pre-deployment training of the DRL agent on historical production traces without exposing production traffic to online exploration. Offline RL has matured substantially in the past three years, and offline algorithms exist that avoid the distribution-shift failure modes that trapped early offline-RL work. Offline RL is not a full substitute for online learning (the agent cannot experience genuinely novel workload conditions), but it addresses the "online exploration risks production" concern that would otherwise block deployment in a regulated environment.

Chaos engineering aligned with DORA Article 26's advanced-testing requirement. DORA specifically calls for financial entities to conduct threat-led penetration testing and other advanced testing of their ICT systems. Chaos-engineering practices (injecting controlled failures into the deployed system to verify resilience) generalize to DRL-based systems in the form of adversarial workload injection: presenting the agent with pathological or synthetic-outlier workload patterns and observing whether the safety envelope catches unsafe actions. This is one plausible way to produce the kind of audit-friendly testing evidence DORA requires, though the details of what regulators will accept remain to be worked out in practice.

6.4 Trade-offs the architecture inherits

The proposed architecture inherits three trade-offs that any DRL-based cloud resource management system carries. Training-time computational cost is real. PPO training on realistic workload traces is computationally expensive, and CTDE with joint value functions increases that cost by a factor proportional to the application count. In production terms, this manifests as a requirement for periodic re-training as workload patterns evolve, and a supporting compute budget for that re-training.

Robustness to distribution shift is the harder trade-off. Financial markets do not stay stationary. The workload patterns a DRL agent trained in 2024 experienced are not the workload patterns of 2028, and the workload patterns of a normal-volatility market are not the workload patterns of a crisis market. The agent's learned policy may degrade silently under conditions that were rare or absent in training data. Distribution-shift detection and monitored re-training triggers are part of any production deployment, but their design is under-researched in the RL-for-cloud literature and remains an active problem.

Goodhart's Law in resource management is the most subtle trade-off. The agent optimizes for the SLA and cost metrics it is rewarded on. Any operationally-important metric that is not in the reward function is effectively invisible to the agent, and the agent may implicitly optimize against it. Concrete examples in institutional-financial platforms include tail latency below the reward-function threshold (an agent rewarded for p95-latency compliance may allow p99 latency to drift), user-perceived responsiveness during composite workflows spanning multiple applications (an agent scaling per-application may miss cross-application latency composition), and change-management burden (frequent scaling actions produce more audit records than the compliance team can review). Reward-shaping to address these considerations is craftwork, not a solved problem.

6.5 Positioning as Part 1 of a research program

This paper is Part 1 of a research program. Its explicit contribution is requirements + architecture + open problems, grounded in real production observations. Empirical evaluation on production workloads is future work. The evaluation plan the research program calls for has three phases. Phase 1, corresponding to Section 5's reference implementation, is simulator-grounded evaluation with synthetic workload traces calibrated to realistic financial-platform patterns. Table 2's results are a Phase 1 output. Phase 2 is offline evaluation on anonymized real production traces from institutional-financial platforms. This requires data-sharing arrangements that are administratively non-trivial but not blocked by any technical barrier. Phase 3 is shadow-mode online evaluation on a production platform, following the pattern described in Section 6.3. Phase 3 produces the operational evidence base needed for the compliance conversation about authoritative deployment.

The community's most valuable contribution to the deployment conversation is probably at Phase 3. Institutional financial platforms have historically been reluctant to share the operational data that would enable rigorous Phase 2 evaluations by third-party researchers, and the resulting evaluation-evidence gap is one reason the DRL-for-cloud literature is dominated by benchmark evaluations rather than production reports. A regulatory or industry initiative to enable this kind of data sharing (subject to appropriate anonymization) would substantially accelerate the community's work on both open problems this paper identifies.

7. Conclusion

This paper contributed a practitioner-grounded requirements analysis, a proposed DRL-based architecture, a retrieval-augmented context substrate and graph-neural-network state encoder that together address the coordination and explainability barriers, and a discussion of two open problems for cloud resource management in financial trading platforms. The requirements analysis drew on production observations at two anonymized institutional-financial platforms, both of which chose static over-provisioning as their deliberate resource management posture, and neither of which considered reactive autoscaling adequate for the peak-availability conditions financial services demand. The architectural proposal treats multi-application coordination as a first-class design property, adds a safety envelope of hard constraints around the DRL agent's action output, encodes the joint state through a GNN over the application-dependency graph so the coordination signal is grounded in an inspectable topology, augments the policy with retrieval over a store of historical scenarios, and includes a shadow-mode deployment pattern that supports the regulatory-compliance workflow the observed platforms operate under. The retrieval substrate serves double duty as a case-based-reasoning audit trail: every automated decision is documented alongside the retrieved historical decisions that most influenced it, giving compliance reviewers an audit-defensible answer to the question of why the system decided what it decided. A reference Python implementation, with a discrete-event simulator and four baseline policies including the proposed coordinated CTDE-style PPO, accompanies the paper and produced the simulated results in Table 2.

The two open problems the paper identified are the community's collective work agenda. Multi-application coordination beyond the two-to-ten-application regime this paper addresses will require hierarchical or graph-based extensions that are today speculative. Explainability for compliance, under the EU Digital Operational Resilience Act, the US SEC Regulation SCI, and FINRA Rule 4370, will require sustained collaboration between DRL researchers and financial-services regulators, and probably a new XRL evaluation framework grounded in what regulators accept as an audit-defensible decision procedure. Neither problem is close to solved. Both block real financial-services deployment of DRL-based cloud resource management until they are.

This paper is Part 1 of a research program. Phase 1 simulator-grounded evaluation is complete. Phase 2 offline evaluation on real production traces, and Phase 3 shadow-mode online evaluation on a production platform, are future work. The community's most valuable near-term contribution is probably in enabling the data-sharing arrangements that would let Phase 2 evaluations proceed at scale, and in building the industry-regulator collaboration that would let the explainability-for-compliance conversation produce concrete standards. Until then, the static over-provisioning that dominates production financial-services cloud infrastructure will remain the pattern of least regret. The path beyond it exists in principle. Walking it requires the community's coordinated work on the two problems this paper identified.

References

- [1] H. Mao, M. Alizadeh, I. Menache, S. Kandula, "Resource Management with Deep Reinforcement Learning," in Proc. 15th ACM Workshop on Hot Topics in Networks (HotNets '16), Atlanta, GA, USA, 2016, pp. 50-56. doi:10.1145/3005745.3005750.
- [2] H. Mao, M. Schwarzkopf, S. B. Venkatakrisnan, Z. Meng, M. Alizadeh, "Learning Scheduling Algorithms for Data Processing Clusters," in Proc. ACM SIGCOMM 2019, Beijing, China, 2019, pp. 270-288. doi:10.1145/3341302.3342080.
- [3] M. Cheng, J. Li, S. Nazarian, "DRL-Cloud: Deep Reinforcement Learning-based Resource Provisioning and Task Scheduling for Cloud Service Providers," in Proc. 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, Korea, 2018, pp. 129-134. doi:10.1109/ASPDAC.2018.8297294.
- [4] C. Bitsakos, I. Konstantinou, N. Koziris, "DERP: A Deep Reinforcement Learning Cloud System for Elastic Resource Provisioning," in Proc. 10th IEEE Intl. Conf. on Cloud Computing Technology and Science (CloudCom), Nicosia, Cyprus, 2018, pp. 21-29. doi:10.1109/CloudCom2018.2018.00020.
- [5] F. Rossi, V. Cardellini, F. Lo Presti, M. Nardelli, "Geo-distributed efficient deployment of containers with Kubernetes," *Computer Communications*, vol. 159, pp. 161-174, 2020. doi:10.1016/j.comcom.2020.04.061.

- [6] Y. Gari, D. A. Monge, E. Pacini, C. Mateos, C. G. Garino, "Reinforcement learning-based application autoscaling in the cloud: a survey," *Engineering Applications of Artificial Intelligence*, vol. 102, art. 104288, 2021. doi:10.1016/j.engappai.2021.104288.
- [7] T. Llorido-Botran, J. Miguel-Alonso, J. A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *Journal of Grid Computing*, vol. 12, pp. 559-592, 2014. doi:10.1007/s10723-014-9314-7.
- [8] J. O. Kephart, D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41-50, 2003. doi:10.1109/MC.2003.1160055.
- [9] M. Salehie, L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, art. 14, 2009. doi:10.1145/1516533.1516538.
- [10] V. Mnih, K. Kavukcuoglu, D. Silver, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529-533, 2015. doi:10.1038/nature14236.
- [11] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018. ISBN 978-0-262-03924-6.
- [12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, et al., "Continuous control with deep reinforcement learning," in *Proc. Intl. Conf. on Learning Representations (ICLR)*, 2016. arXiv:1509.02971.
- [13] T. Rashid, M. Samvelyan, C. Schroeder de Witt, G. Farquhar, J. Foerster, S. Whiteson, "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning," in *Proc. Intl. Conf. on Machine Learning (ICML)*, vol. 80, 2018, pp. 4295-4304. arXiv:1803.11485.
- [14] P. Hernandez-Leal, B. Kartal, M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, pp. 750-797, 2019. doi:10.1007/s10458-019-09421-1. arXiv:1810.05587.
- [15] E. Puiutta, E. M. S. P. Veith, "Explainable Reinforcement Learning: A Survey," in *Proc. IFIP Cross Domain Conf. for Machine Learning and Knowledge Extraction (CD-MAKE)*, Springer LNCS vol. 12279, 2020, pp. 77-95. arXiv:2005.06247.
- [16] C. Glanois, P. Weng, M. Zimmer, D. Li, T. Yang, J. Hao, W. Liu, "A survey on interpretable reinforcement learning," *Machine Learning*, vol. 113, pp. 5847-5890, 2024. doi:10.1007/s10994-024-06543-w. arXiv:2112.13112.
- [17] G. A. Vouros, "Explainable Deep Reinforcement Learning: State of the Art and Challenges," *ACM Computing Surveys*, vol. 55, no. 5, art. 92, 2022. doi:10.1145/3527448.
- [18] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, "A Survey of Methods for Explaining Black Box Models," *ACM Computing Surveys*, vol. 51, no. 5, art. 93, 2018. doi:10.1145/3236009.
- [19] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, et al., "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82-115, 2020. doi:10.1016/j.inffus.2019.12.012.
- [20] M. Shahradd, R. Fonseca, Í. Goiri, et al., "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," in *Proc. USENIX Annual Technical Conference (ATC '20)*, 2020, pp. 205-218.
- [21] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, R. Bianchini, "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms," in *Proc. 26th ACM Symposium on Operating Systems Principles (SOSP '17)*, Shanghai, China, 2017, pp. 153-167. doi:10.1145/3132747.3132772.
- [22] European Parliament and Council, Regulation (EU) 2022/2554 of 14 December 2022 on digital operational resilience for the financial sector (Digital Operational Resilience Act, DORA), *Official Journal of the European Union*, L 333, 27 Dec. 2022, pp. 1-79. In force since 17 January 2025.
- [23] U.S. Securities and Exchange Commission, "Regulation Systems Compliance and Integrity (Regulation SCI)," 17 CFR § 242.1000-1007, SEC Release No. 34-73639, 2014, as amended.
- [24] FINRA, "Rule 4370: Business Continuity Plans and Emergency Contact Information," *FINRA Manual*, 2004, as amended.
- [25] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020.
- [26] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, Q. Li, "A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models," in *Proc. ACM SIGKDD Conference on Knowledge*

- Discovery and Data Mining (KDD '24), 2024, pp. 6491-6501. doi:10.1145/3637528.3671470. arXiv:2405.06211.
- [27] J. Johnson, M. Douze, H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535-547, 2021. doi:10.1109/TBDATA.2019.2921572.
- [28] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57-81, 2020. doi:10.1016/j.aiopen.2021.01.001.
- Manuscript status: Draft. Prepared for submission to *Future Generation Computer Systems* (Elsevier). Backup target: IEEE Access. Format-step to render into elsarticle LaTeX class or Word template per journal preference.
- [29] Singh, A. R., & Kaur, A. (2025). AI adoption in agriculture: A managerial framework for sustainable agribusiness development. *Journal of Artificial Intelligence in Agriculture and Applied Research*, 1(1), 1-12. <https://agrpjournals.com/journals/journal/index.php/jaiaar/article/view/2>