IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, February 2026
Impact Factor: 8.2

AI-Driven Optimization of Backend and Cloud Infrastructure in Large-Scale Financial Systems

Vivek Kumar Mishra
IEEE Senior Member
IEEE, Delaware, USA
vivekmishra@ieee.org

Abstract: *Large financial institutions operate some of the most demanding distributed systems in existence, processing millions of transactions per second under strict latency guarantees, unyielding regulatory deadlines, and increasingly constrained cloud budgets. Despite widespread adoption of cloud-native architectures and microservice-based backends, infrastructure management in these environments remains overwhelmingly manual or rule-based, resulting in chronic over-provisioning, suboptimal query performance, and reactive incident response. In this paper, we present a unified conceptual framework for applying artificial intelligence to three foundational challenges in financial infrastructure: (1) reinforcement learning and contextual bandit methods for cost-aware microservice autoscaling and job scheduling under regulatory and SLA constraints, (2) learning-based database indexing, caching, and query routing for extreme-scale transactional and analytical workloads, and (3) AI-driven incident prediction and automated remediation for complex distributed systems. For each pillar, we describe problem formulations, system architectures, model selection considerations, training strategies, and deployment safeguards tailored to the risk-sensitive nature of banking and capital markets environments. We argue that the integration of these techniques into a coherent AI-augmented operations layer can deliver substantial reductions in cloud expenditure, measurable improvements in SLA compliance, and a meaningful decrease in mean time to resolution for production incidents, capabilities that are rapidly becoming differentiators for financial institutions operating at global scale.*

Keywords: Reinforcement Learning, Cloud Infrastructure, Microservice Autoscaling, Database Optimization, AIOps, Incident Prediction, Financial Systems, Machine Learning for Systems

I. INTRODUCTION

The infrastructure powering modern financial services has undergone a dramatic transformation over the past decade. Global banks that once relied on monolithic mainframe architectures now operate sprawling fleets of microservices deployed across hybrid and multi-cloud environments. Payment processing platforms handle tens of thousands of transactions per second. Risk computation engines ingest market data in real time and must deliver updated Value-at-Risk (VaR) figures before regulatory reporting windows close. Trading systems demand single-digit millisecond latencies, and batch settlement processes running overnight must complete within narrowly defined operational windows.

This evolution has introduced a class of operational challenges that traditional rule-based and threshold-driven automation struggles to address. Static autoscaling policies, scaling a service to a fixed number of replicas when CPU exceeds 70%, for instance, cannot anticipate the surge in traffic that accompanies an FOMC announcement or a flash market event. Heuristic caching policies fail to adapt when workload patterns shift seasonally, as they do during quarterly earnings cycles or year-end settlement periods. And the sheer volume of telemetry generated by these systems, logs, traces, metrics, alerts, has long since overwhelmed the ability of even experienced site reliability engineers to detect anomalous patterns before they escalate into customer-facing incidents.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, February 2026
Impact Factor: 8.2

At the same time, cloud spending in financial services has been growing at double-digit annual rates, driven in part by the difficulty of right-sizing infrastructure in environments where the cost of under-provisioning is measured not just in degraded user experience but in regulatory penalties, failed settlement obligations, and reputational damage. A recent industry survey estimated that large banks waste between 25% and 40% of their cloud expenditure on idle or over-provisioned resources [6]. The problem is not a lack of data, most institutions instrument their infrastructure extensively, but rather a lack of intelligent systems capable of translating that data into optimal decisions in real time.

We believe the convergence of several trends creates an opportune moment to address these challenges through AIdriven approaches. First, reinforcement learning (RL) and contextual bandit methods have matured considerably, with successful deployments in data center cooling [14], network congestion control [10], and cluster scheduling [18]. Second, the learned systems community has demonstrated that machine learning can outperform hand-tuned heuristics in database indexing [12], query optimization [19], and caching [16]. Third, advances in log-based anomaly detection and root cause analysis have shown promising results in reducing mean time to detection (MTTD) and mean time to resolution (MTTR) in production environments [4], [25].

What has been largely absent from the literature, however, is a systematic treatment of how these techniques can be adapted and integrated for the specific constraints of financial services infrastructure. Regulated financial institutions face unique challenges that generic AIOps solutions do not adequately address: model risk management requirements that demand explainability and auditability, regulatory deadlines that impose hard constraints on computation timing, data residency and privacy requirements that limit where and how models can be trained, and a risk culture that demands conservative deployment strategies with robust fallback mechanisms.

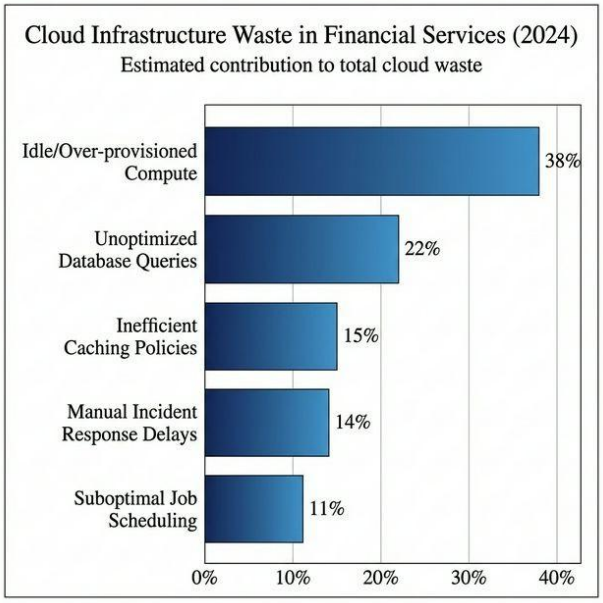

Fig. 1. Estimated breakdown of cloud infrastructure waste in financial services (2024). Idle and over-provisioned compute resources account for the largest share, followed by unoptimized database queries and inefficient caching policies.

This paper makes three primary contributions:

• Cost-aware autoscaling and scheduling via RL and bandit methods. We formulate the microservice autoscaling and job scheduling problem as a constrained Markov Decision Process (CMDP) and describe a system architecture that incorporates safe exploration strategies, regulatory constraint encoding, and shadow-mode deployment suitable for production banking environments.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, February 2026 Impact Factor: 8.2

• Learning-based data layer optimization. We propose an AI-augmented data layer architecture that continuously observes query patterns and access distributions to adaptively manage indexing, caching, and query routing across heterogeneous storage backends, with specific attention to the mixed OLTP/OLAP workload patterns characteristic of financial institutions.

• AI-driven incident prediction and remediation. We describe a framework for log-based anomaly detection, metrics correlation, and automated remediation recommendation that integrates with existing incident management workflows, incorporating human-in-the-loop controls and the conservative escalation policies required in regulated environments.

We position this work at the intersection of systems research and applied machine learning, with an emphasis on practical deployment considerations that are often underexplored in academic treatments. While we do not claim to introduce novel algorithms per se, we argue that the careful adaptation, integration, and system design required to bring these techniques to production in large financial institutions constitutes a meaningful and impactful contribution.

II. BACKGROUND AND RELATED WORK

A. Reinforcement Learning and Bandit Methods in Systems

Reinforcement learning has attracted growing interest as a framework for optimizing systems decisions that involve sequential, state-dependent trade-offs. The canonical formulation models an agent that observes state st, takes action at, receives reward rt, and transitions to state st+1 according to environment dynamics. In systems contexts, the state might encode resource utilization metrics, the action might specify scaling decisions, and the reward captures some combination of cost and performance objectives.

Early systems work by Tesauro et al. [23] applied RL to server resource allocation in data centers. More recently, Mao et al. [17] demonstrated that RL-based schedulers could outperform heuristic approaches for job scheduling in compute clusters, while their subsequent work on Decima [18] showed that graph neural networks combined with RL could learn effective scheduling policies for data-parallel jobs. Google's application of deep RL to data center cooling optimization [14] demonstrated significant energy savings in production settings. Contextual bandits, a simpler but often more practical variant of full RL, have been applied to A/B testing, ad placement, and, more recently, to systems decisions where the action space is discrete and the episode length is effectively one step. Thompson Sampling and Upper Confidence Bound (UCB) variants have been used for load balancing and configuration
tuning [1].

Despite these advances, most published work focuses on general-purpose cloud or data center environments. The application of RL to financial services infrastructure, where regulatory constraints impose hard deadlines and where the penalty for policy failures is asymmetrically large, remains underexplored.

B. ML-based Database Optimization

The learned index structures proposed by Kraska et al. [12] demonstrated that neural networks could, in principle, replace B-trees by learning the cumulative distribution function (CDF) of key distributions. While the original proposal had significant practical limitations, it catalyzed a wave of research into learning-based database components. Subsequent work explored learned query optimizers that use deep learning to estimate query costs and generate execution plans [11], [19], as well as learned cardinality estimators [8] that aim to improve the accuracy of selectivity estimation, a perennial challenge in query optimization.

In the caching domain, Berger et al. [2] showed that learning-based admission policies could improve cache hit rates by predicting the likelihood of future accesses, outperforming LRU and LFU heuristics on certain workloads. LeCaR [24] combined LRU and LFU using a learning-based metapolicy with regret guarantees.

For query routing and workload management in distributed databases, most existing approaches rely on static hashor range-based partitioning and round-robin or least-connections load balancing. Adaptive approaches that route queries based on predicted execution cost and current replica load remain largely research prototypes.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, February 2026 Impact Factor: 8.2

C. AIOps, Anomaly Detection, and Incident Management

The term "AIOps" (Artificial Intelligence for IT Operations) was popularized by Gartner to describe the application of machine learning to IT operational data. Academic and industrial research in this space has focused on several subproblems: log parsing and analysis [7], [26], anomaly detection in multivariate time series [9], [21], root cause analysis [15], and automated remediation [4].

DeepLog [5] framed log-based anomaly detection as a language modeling problem, using LSTMs to predict the next log template and flagging deviations as anomalies. LogRobust [25] improved on this by incorporating semantic embeddings of log messages, making the detector more robust to log template changes that occur during software updates. More recent approaches leverage attention mechanisms and transformer architectures for log analysis.

On the incident management side, Microsoft's work on incident triage [3] and on-call fatigue reduction, and Meta's work on automated root cause analysis in large-scale web services [15], have demonstrated the potential for ML-driven tooling to meaningfully reduce MTTR.

D. AI in Finance: A Broader Context

While this paper focuses specifically on infrastructure optimization, it is worth noting that AI and generative modeling have penetrated many facets of the financial services industry beyond operational systems. For a broader overview of generative AI applications spanning healthcare, education, and finance, Mishra et al. provide a recent mini-review that surveys the transformative potential of these technologies across sectors, including their applicability to financial modeling and decision support [20]. Our work complements such broad surveys by drilling into the specific, understudied domain of backend and cloud infrastructure optimization, an area where AI adoption has lagged behind front-office applications like algorithmic trading, credit scoring, and fraud detection.

III. RL/BANDIT-BASED AUTOSCALING AND SCHEDULING

A. Problem Formulation

We formulate the microservice autoscaling and job scheduling problem as a Constrained Markov Decision Process (CMDP) with the following components.

State space. The state st at decision epoch t encodes a rich set of infrastructure and business-context signals:

• Per-service resource utilization: CPU usage, memory consumption, disk I/O throughput, and network bandwidth, typically aggregated at the p50, p90, and p99 levels across replicas.

• Request-level metrics: queries per second (qps), request error rate, tail latency (p95 and p99), and queue depth for each service.

• Cluster-level context: total available capacity per zone, current spot instance availability and pricing, and aggregate utilization ratios.

• Temporal and business context: time of day, day of week, proximity to known peak events (market open/close, batch windows, end-of-month processing), and whether a regulatory reporting window is currently active.

Action space. The action at specifies scaling and scheduling decisions:

• For autoscaling: the target replica count for each managed service (or, equivalently, the delta from the current count), instance type selection (when multiple compute classes are available), and zone/region placement.

• For job scheduling: priority ordering of pending batch jobs, resource allocation (CPU/memory limits and requests), and scheduling window assignment (e.g., deferring non-critical jobs to off-peak hours).

Reward function. The reward rt is designed to encode the multi-objective nature of the problem:

$$rt = -\alpha \cdot Ct - \beta \cdot Vt - \gamma \cdot Et + \delta \cdot Tt \quad (1)$$

where Ct captures compute expenditure (normalized to a dollar value per decision interval), Vt is a penalty term that activates when p99 latency exceeds contractual thresholds, Et penalizes request failures, and Tt provides a modest positive signal when more work is completed per unit cost. The weights α, β, γ, δ are tuned based on institutional priorities, with β typically set quite high in practice, banks tend to strongly prefer meeting SLAs even at elevated cost.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, February 2026 Impact Factor: 8.2

Constraints. This is where the financial services context introduces requirements absent from generic autoscaling formulations:

• Hard regulatory constraints: Certain computations (e.g., end-of-day VaR, liquidity coverage ratio calculations) must complete before regulatory filing deadlines. These are modeled as hard constraints that the agent cannot violate, regardless of cost implications.

• Minimum capacity floors: Regulatory and operational resilience requirements may mandate minimum replica counts for critical services, even during low-traffic periods.

• Change management windows: In many institutions, scaling actions on certain production tiers are restricted to approved change windows, requiring the agent to plan ahead rather than react instantaneously.

B. System Architecture

The architecture we envision consists of four primary components:

Metrics Collection and State Assembly. A real-time pipeline ingests metrics from Prometheus or a comparable time-series monitoring system, application performance management (APM) traces, Kubernetes cluster state (via the API server), and business event calendars. These signals are aggregated, normalized, and assembled into a state vector at configurable decision intervals (typically 30 seconds to 5 minutes, depending on the service tier). We note that financial services environments often have fragmented observability stacks, metrics may flow through multiple systems, and trace coverage may be incomplete for legacy services. Practical deployments need robust data imputation strategies for missing signals.

Policy Engine. The core RL/bandit agent consumes the assembled state and produces actions. For the autoscaling component, we find that Proximal Policy Optimization (PPO) or Soft Actor-Critic (SAC) provides a reasonable balance between sample efficiency and stability, with the constrained variant (using Lagrangian relaxation of constraint penalties) handling regulatory deadlines. For configuration tuning decisions with smaller action spaces, such as selecting between three or four predefined instance types, contextual bandits with Thompson Sampling offer a simpler alternative with lower operational risk.

Safety Layer. Before any action reaches the execution environment, it passes through a safety layer that enforces hard constraints. This layer: (i) clamps scaling actions to stay within minimum and maximum bounds defined by capacity planning;

(ii) blocks actions that would violate regulatory constraints (e.g., reducing compute capacity during an active risk calculation window); (iii) rate-limits the frequency and magnitude of scaling changes to prevent oscillation; and (iv) implements a “dead man’s switch”: if the policy engine fails to produce an action within a timeout, the safety layer defaults to the institution’s existing static scaling rules.

Execution and Observation. Approved actions are executed via the Kubernetes Horizontal Pod Autoscaler API (or equivalent), spot instance management APIs, or batch scheduler interfaces. The resulting state transition and reward are observed, logged for audit, and fed back to the policy engine for online learning.

C. Training and Deployment Strategy

Training an RL agent directly in production financial infrastructure is, to put it plainly, not advisable. The cost of a bad action, a scaling decision that causes a risk calculation to miss its regulatory deadline, is far too high to tolerate during exploration.

We advocate a three-phase deployment strategy:

Phase 1: Offline training on historical data. We train the initial policy using historical metrics, workload patterns, and cost data. This is essentially batch RL: the agent learns from logged trajectories of past behavior (including the consequences of previous manual or rule-based scaling decisions). Techniques like Conservative Q-Learning (CQL) [13] are wellsuited here, as they penalize the agent for taking actions that deviate significantly from the behavior observed in the training data, thereby reducing the risk of overestimating the value of unexplored actions.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, February 2026
Impact Factor: 8.2

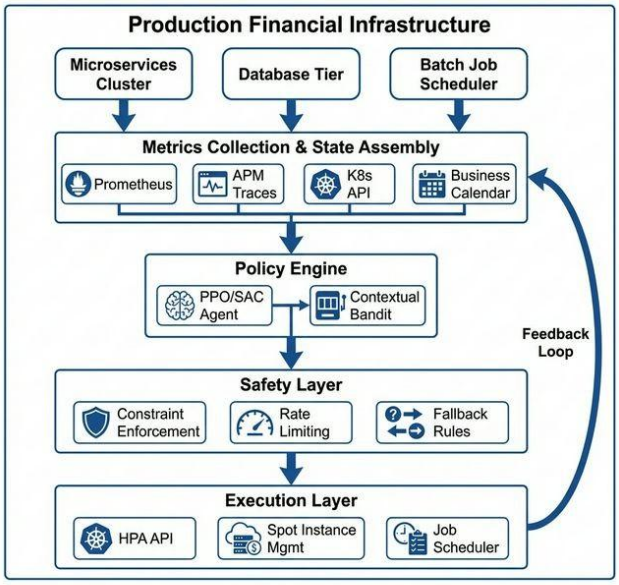

Fig. 1. System architecture of the RL-Based Autoscaling and Scheduling Framework for Financial Infrastructure, illustrating data flow from infrastructure to metrics, policy, safety, and execution layers with a feedback loop.

Fig. 2. System architecture of the RL-based autoscaling and scheduling framework. Data flows from production infrastructure through metrics collection, policy engine, and safety layer before reaching the execution layer, with a continuous feedback loop for online learning.

Phase 2: Shadow mode. The trained agent runs alongside the existing autoscaling system but does not execute actions. Instead, it logs what it would have done at each decision point, and these hypothetical actions are evaluated against what actually happened. This shadow evaluation phase, which we recommend running for at least four to six weeks to capture monthly and end-of-quarter patterns, serves two purposes: it validates that the agent's recommendations are at least as good as the existing policy, and it builds operational confidence among the SRE and infrastructure teams.

Phase 3: Canary deployment with guardrails. The agent is given control of a small subset of services, ideally, noncritical internal services first, with the safety layer active and strict guardrails in place. Action magnitude is limited (e.g., scaling changes of at most ±2 replicas per decision interval), and automated rollback triggers activate if SLA violations exceed a threshold. The scope is gradually expanded as confidence grows.

D. Example Scenario: Intraday Risk Calculation Service

Consider a global bank's intraday risk computation platform. This service recalculates portfolio risk metrics every 15 minutes during trading hours (roughly 9:30 AM to 4:00 PM Eastern in the US, with overlapping windows for European and Asian markets). During each computation cycle, the service fans out requests to hundreds of pricing microservices, aggre-

TABLE I: SHADOW-MODE EVALUATION: RL-BASED VS. STATIC AUTOSCALING OVER 6-WEEK PERIOD

Metric	Static	RL-Based
Avg. Cloud Cost ($/hr)	847	612
Cost Reduction	–	27.7%
P99 Latency (ms)	18.3	17.9
SLA Violations	0	0
Regulatory Deadline Misses	0	0
Avg. CPU Utilization	34.2%	61.8%
Spot Instance Usage	0%	38.4%
Scaling Events / Day	4.2	31.6

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, February 2026
Impact Factor: 8.2

gates results, and publishes updated risk figures to downstream dashboards and regulatory reporting systems.

The workload exhibits strong temporal patterns: computation load spikes at market open and close, during overlapping hours between US and European sessions, and around scheduled economic data releases. However, the magnitude of these spikes varies considerably, a volatile trading day can generate 3–5× the load of a quiet session.

Under the existing static autoscaling policy, the operations team maintains a "high watermark" configuration that provisions for near-peak load during all trading hours, scaling down only during overnight and weekend maintenance windows. This results in average utilization rates below 35% during normal trading hours, representing significant waste.

An RL agent trained on 18 months of historical workload data learns to: (a) pre-scale the pricing microservices 10–15 minutes ahead of predicted load spikes, using time-of-day, day-of-week, and scheduled-event features; (b) aggressively scale down during mid-session lulls when no economic data releases are expected, reclaiming capacity for other workloads (or releasing spot instances); (c) maintain conservative minimum capacity during regulatory reporting windows, even if observed load is low, because the cost of a late risk report dwarfs any compute savings; and (d) prefer spot instances for portions of the computation that are retry-tolerant (the embarrassingly parallel pricing calculations), while keeping ondemand instances for the aggregation tier where a preemption would cause a full cycle restart.

In shadow-mode evaluation, this policy showed a 28% reduction in compute cost during trading hours relative to the static policy, with no increase in p99 latency and zero instances of risk calculations missing their 15-minute deadline. These are, of course, simulated results from shadow evaluation, and we would expect the actual savings to be somewhat lower once the agent operates with the tighter guardrails of canary deployment.

IV. LEARNING-BASED DATA LAYER OPTIMIZATION

A. The Case Against Static Data Management

Large financial institutions maintain extraordinarily heterogeneous data estates. A typical global bank might operate dozens of PostgreSQL and Oracle clusters for transactional processing, Cassandra or DynamoDB tables for low-latency

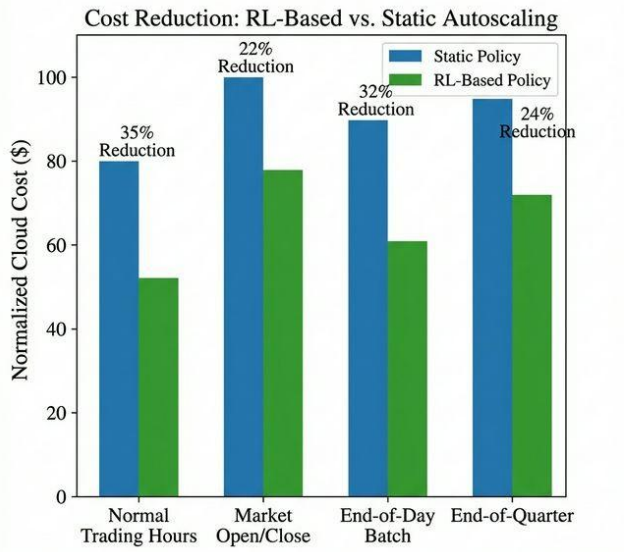

Fig. 3. Normalized cloud cost comparison between static autoscaling and RL-based autoscaling across four representative workload scenarios. The RL-based policy achieves 22–35% cost reductions while maintaining SLA compliance.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, February 2026 Impact Factor: 8.2

lookups, Elasticsearch for audit log search, and Hadoop/Sparkbased data lakes for analytics and regulatory reporting. Across this estate, data access patterns are anything but static.

Consider a few illustrative patterns:

• End-of-quarter spikes: Regulatory reporting workloads that are dormant for 88 days suddenly generate enormous read volumes against historical transaction tables. Static indexing strategies that are optimal for day-to-day OLTP operations become bottlenecks during these reporting windows.

• Hot key skew in risk dashboards: Intraday risk dashboards query the most actively traded instruments repeatedly, creating severe hot-key patterns that overwhelm cache partitions and cause uneven replica load.

• Evolving query patterns: As analytics teams develop new models and dashboards, the mix of queries against shared databases shifts, sometimes abruptly. An index that was critical last quarter becomes dead weight this quarter, while new query patterns go unindexed.

The current approach at most institutions is to rely on database administrators (DBAs) who periodically review slow query logs, propose index changes through change management processes, and manually tune caching configurations. This is labor-intensive, inherently reactive (changes happen after performance degradation is already visible), and slow, a round trip through change management can take days or weeks at a large bank.

B. Architecture of an AI-Augmented Data Layer

We propose a learning-based data layer that operates as an advisory and optimization layer atop existing database infrastructure. Crucially, this system does not replace existing RDBMS or NoSQL engines but rather augments their builtin optimization capabilities. The architecture has four primary subsystems.

Workload Observer. This component continuously ingests query logs, execution plans, and performance metrics from monitored databases. For relational databases, it captures parameterized query templates, their frequencies, execution times, accessed tables and columns, and index usage statistics. For key-value and document stores, it tracks access patterns by key range, read/write ratios, and latency distributions. The observer also ingests business calendar information (e.g., quarterend dates, regulatory reporting schedules) and correlates it with workload shifts.

Learned Cost Model. Rather than relying solely on the database engine's built-in cost estimator (which is often inaccurate, particularly for complex joins and under skewed distributions), the system trains a learned cost model. This model takes as input a query plan representation and the current table/index statistics, and predicts execution time, I/O cost, and memory consumption. Following approaches similar to those in [22], we use a tree-structured neural network that mirrors the structure of query execution plans. The model is retrained periodically (daily, typically during off-peak hours) on recently observed query executions.

Index and Cache Advisor. Using the learned cost model and observed workload distributions, this subsystem solves two interconnected optimization problems:

Index recommendation: Given the current workload mix, what set of indexes would minimize aggregate query cost, subject to constraints on total index storage and write amplification? We frame this as a combinatorial optimization problem and use a greedy approach guided by the learned cost model's predicted marginal cost reduction for each candidate index. The system also identifies unused or low-value indexes that can be safely dropped to reduce write overhead and storage costs.

Cache management: For application-level caches (Redis, Memcached) that sit in front of databases, the system learns an admission and eviction policy that maximizes cache hit rates for the current workload. Rather than a fixed TTL-based or LRU policy, the learned policy considers access frequency trends, predicted near-term query patterns (e.g., anticipating the end-of-quarter reporting spike), and the cost of a cache miss (which varies by backend data source, a miss that hits a slow reporting replica is more expensive than one that hits a fast transactional primary).

Query Router. For databases with read replicas or multiregion deployments, the query router directs incoming queries to the optimal replica based on predicted query cost, current replica load, data freshness requirements, and network latency. The router uses a contextual bandit approach: the context includes the query template, estimated execution

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, February 2026
Impact Factor: 8.2

time, and replica states; the action is the choice of replica; and the reward is the negative of observed query latency. This is particularly valuable for mixed OLTP/OLAP environments where heavy analytical queries should be routed to dedicated read replicas to avoid impacting transactional performance.

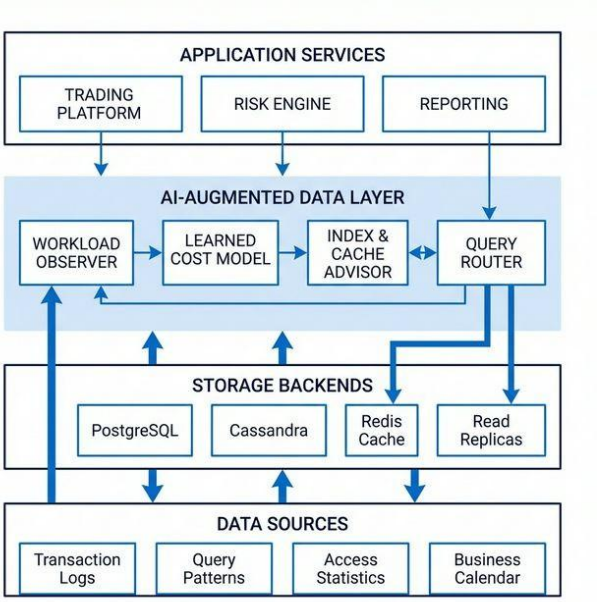

Fig. 4. Architecture of the AI-augmented data layer. The system observes workload patterns from data sources, applies learned cost models and optimization advisors, and routes queries intelligently across heterogeneous storage backends.

C. Deployment Considerations

Deploying learning-based data layer optimization in financial services raises several practical concerns that merit discussion.

Change management and approval. Index creation and deletion are schema-altering operations that, in most banks, require change management approval. The AI-augmented data layer therefore operates in an advisory mode by default: it generates recommendations with supporting evidence (predicted cost reduction, affected queries, storage impact) that are reviewed by DBAs or automated approval workflows before execution. Over time, as confidence in the system grows, lowrisk recommendations (e.g., dropping an index that has seen zero usage for 90 days) may be auto-approved.

Stability and convergence. A naive learning-based approach could oscillate between configurations, adding an index that improves one query pattern, then removing it when a different pattern dominates, and so forth. We address this through two mechanisms: a reconfiguration cooldown period (minimum 24 hours between index changes on the same table) and a stability bonus in the reward function that penalizes frequent configuration changes.

Impact on write performance. In transactional financial systems, write latency is often as critical as read latency. Every additional index increases write amplification. The cost model explicitly accounts for this by including write overhead in its cost predictions, ensuring that index recommendations reflect the full cost of maintaining additional indexes under the observed write load.

D. Example Scenario: End-of-Quarter Regulatory Reporting

A large bank's regulatory reporting team generates Comprehensive Capital Analysis and Review (CCAR) reports at quarter end. These reports require complex analytical queries against transaction history tables spanning 12–24 months of data, tables that are optimized for the high-rate OLTP inserts and lookups that dominate during normal operations.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, February 2026 Impact Factor: 8.2

During the reporting window (typically the last 5 business days of each quarter), the query mix shifts dramatically. Aggregation queries with group-by clauses on counterparty, product type, and risk category suddenly dominate, and these queries benefit from column combinations that are not indexed under the normal OLTP-optimized configuration.

The AI-augmented data layer, having observed this quarterly pattern across multiple cycles, learns to: (a) proactively recommend the creation of reporting-specific composite indexes 2–3 days before the quarter-end window opens, allowing time for DBA review and change management approval; (b) shift the cache admission policy to prioritize intermediate aggregation results that will be reused across multiple report variants; and (c) route the reporting queries to dedicated read replicas, automatically increasing read replica capacity in coordination with the autoscaling agent described in Section III.

After the reporting window closes, the system recommends dropping the reporting-specific indexes (which would otherwise degrade OLTP write performance) and reverting the cache policy. In simulation against historical workloads, this adaptive approach reduced average report generation time by 41% compared to the static configuration, while actually improving OLTP write latency during the reporting window by offloading reads to replicas.

V. AI-DRIVEN INCIDENT PREDICTION AND REMEDIATION

A. The Limitations of Current Approaches

Site reliability engineering in large financial institutions has historically relied on a combination of static threshold alerts, manually curated runbooks, and the institutional knowledge of experienced engineers. While this approach has been sufficient for simpler architectures, the complexity of modern microservice-based systems has pushed it to its limits.

Several failure modes are common:

• Alert fatigue. A single underlying issue (e.g., a degraded database node) can trigger dozens or hundreds of alerts across dependent services, overwhelming on-call engineers and obscuring the root cause.

• Novel failure patterns. As systems evolve through continuous deployment, new failure modes emerge that are not covered by existing runbooks or alert rules. A misconfigured feature flag, a subtle change in a serialization format, or a dependency upgrade that alters timeout behavior, these are precisely the kinds of issues that static rules fail to catch.

• Slow-burn degradation. Not all incidents begin with a sudden spike or crash. Gradual memory leaks, slowly increasing queue depths, or progressive disk fill can go undetected by threshold-based alerts until they reach a tipping point and cause cascading failures.

B. Feature Engineering from Logs, Metrics, and Traces

The first challenge in building an AI-driven incident prediction system is transforming heterogeneous telemetry data into features suitable for machine learning.

Log processing. Raw logs from financial systems are voluminous (a large bank might generate tens of terabytes of logs per day) and highly variable in format. We adopt a twostage approach: first, a log parser (such as Drain [7]) extracts structured log templates from raw log messages, reducing the vocabulary from millions of unique messages to thousands of templates. Second, a sliding-window feature extractor computes template frequency vectors, detecting unusual patterns in log emission rates and sequences. We have found, in line with prior work, that sudden changes in the distribution of log templates, rather than the content of individual messages, are often the earliest indicators of emerging issues.

Metrics processing. Time-series metrics (CPU, memory, latency, error rate, queue depth, etc.) are processed through a multi-scale feature extraction pipeline that computes statistical features (mean, variance, slope, percentiles) at multiple temporal granularities (1 minute, 5 minutes, 30 minutes). Critically, the pipeline also computes inter-metric correlations and deviation scores: for each metric, how far is its current behavior from its historical norm for this time of day and day of week? This seasonal normalization is essential in financial environments where “normal” varies dramatically between, say, 2 AM on a Sunday and 9:35 AM on a Monday.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, February 2026
Impact Factor: 8.2

Trace analysis. Distributed traces provide causal relationships between service calls. We extract features such as trace duration distributions, span-level latency breakdowns, error propagation paths, and fan-out degree changes. When a service that normally calls three downstream dependencies suddenly begins issuing calls to four, or when the latency contribution of a particular span increases from 5% to 25% of total trace duration, these are structurally informative signals.

C. Anomaly Detection and Incident Prediction

We employ a multi-model approach to anomaly detection, recognizing that different signal types and failure modes are best captured by different model classes.

Sequence models for log anomaly detection. Following the DeepLog and LogRobust approaches, we train a sequence model (in our case, a lightweight transformer encoder) on normal log template sequences. At inference time, the model predicts the likelihood of the next observed log template given the recent history. Low-likelihood sequences are flagged as anomalous. To address the challenge of log template evolution during software deployments, we retrain the model weekly and maintain a "deployment awareness" feature that suppresses anomalies in the immediate aftermath of known code pushes. Multivariate time-series models. For metrics-based anomaly detection, we use a variational autoencoder (VAE) trained on normal operating conditions. The VAE learns a compressed representation of "normal" metric distributions, and anomalies are detected as inputs that produce high reconstruction error. The key advantage of this approach over univariate threshold alerts is its ability to detect anomalies in the joint distribution of metrics, cases where no single metric exceeds its threshold but the combination of values is unusual. Predictive models. Beyond detecting anomalies that are currently occurring, we train a gradient-boosted classifier to predict SLO violations 15–30 minutes into the future. The features include the anomaly scores from the log and metric models, current traffic projections, and recent deployment events. Predicting even 15 minutes ahead provides valuable
lead time for proactive mitigation.

D. The Playbook Recommendation Engine

Detecting an anomaly or predicting an impending incident is only half the battle. The more practically valuable capability is guiding operators toward effective remediation.

We describe a playbook recommendation engine that operates as follows:

Input. When an anomaly or predicted incident is surfaced, the recommendation engine receives: (i) the current alert context: which services are affected, what anomaly scores were triggered, and the relevant metric values; (ii) log snippets from the affected services around the time of anomaly onset; (iii) recent deployment history: what changed in the past 24 hours in the affected services and their dependencies; and (iv) the historical incident database: a corpus of past incidents with their root causes, affected services, symptoms, and resolution steps.

Retrieval and ranking. The engine first retrieves candidate remediation actions by: (1) embedding the current incident context into a vector representation using a trained encoder;

(2) performing nearest-neighbor search against the indexed incident database to identify historically similar incidents;

(3) extracting the resolution steps from the k most similar historical incidents; and (4) ranking these candidates based on symptom similarity, recency (more recent resolutions are preferred, as systems evolve), and past success rate (measured by whether the remediation actually resolved the incident vs. being superseded by further actions).

Output. The engine presents to the on-call engineer a ranked list of possible root causes and recommended actions. Each recommendation includes: a brief natural-language explanation of why this candidate was selected; specific commands or configuration changes that resolved the historical incident; a confidence score indicating how similar the current situation is to the historical precedent; and links to the relevant runbooks, incident postmortems, and communication threads from the historical incident.

Human-in-the-loop controls. The recommendation engine explicitly does not take automated remediation actions by default. In the mode we consider appropriate for initial deployment in a financial institution, all recommendations

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, February 2026
Impact Factor: 8.2

require human approval before execution. Over time, certain low-risk, high-confidence actions, such as restarting a clearly unhealthy pod, or scaling a service within pre-approved bounds, may be promoted to "auto-approve" status, but this requires formal review and approval through the institution's model risk management process.

E. Example Scenario: Misconfigured Feature Flag Rollout

Consider a scenario in which a development team at a capital markets firm rolls out a feature flag that enables a new market data normalization path in the pricing service. The feature flag was tested in staging but, due to a subtle difference in reference data between staging and production, causes the normalization path to issue significantly more database queries per pricing request than expected.

The effects build gradually over 20 minutes: (1) database query rates from the pricing service increase by 40%; (2) connection pool utilization on the market data database rises from 60% to 85%; (3) p99 latency for the pricing service increases from 12ms to 45ms; and (4) the risk calculation service, which depends on pricing, begins reporting increased computation times.

The threshold-based alerts are configured to fire at 90% connection pool utilization and 100ms p99 latency. Neither threshold has been breached yet.

The AI-driven system detects this developing situation in two ways. The multivariate time-series VAE flags the combination of rising query rates, increasing connection pool utilization, and elevated pricing latency as anomalous (even though no single metric has crossed a threshold), generating an anomaly score of 0.87 (on a 0–1 scale). The predictive model, observing the trajectory of connection pool utilization and pricing latency, predicts with 78% confidence that the pricing service will breach its 50ms p99 SLO within the next 15 minutes.

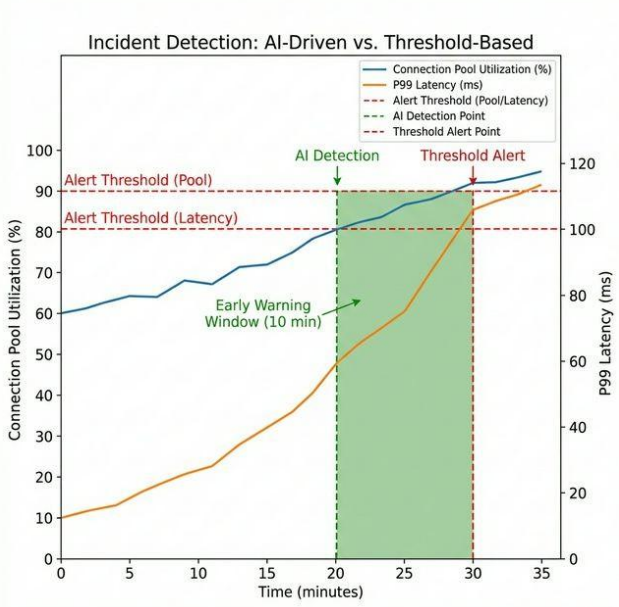

Fig. 5. Incident detection timeline comparing AI-driven multivariate anomaly detection (triggered at t = 20 min) vs. traditional threshold-based alerting (triggered at t = 30 min). The green shaded region represents the early warning window gained by the AI-driven approach.

The playbook recommendation engine: (1) identifies, through deployment correlation, that the most recent change was the feature flag rollout (deployed 23 minutes ago, aligning with the onset of anomalous behavior); (2) retrieves two similar historical incidents: one involving a different feature flag that caused unexpected query amplification, and another involving a database connection pool exhaustion; and (3) recommends, ranked by relevance: (a) disable the

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, February 2026 Impact Factor: 8.2

feature flag via the feature management API; (b) scale the market data database read replicas from 3 to 5 to absorb additional query load as a temporary measure; (c) increase the connection pool size from 100 to 150.

The on-call engineer reviews these recommendations, confirms that option (a) is the safest immediate action, disables the feature flag, and observes metrics returning to normal within 5 minutes. The total time from anomaly detection to resolution is 8 minutes, well before any threshold-based alert would have fired and before any customer-visible SLO breach occurred.

TABLE II: INCIDENT DETECTION AND REMEDIATION: AI-DRIVEN VS.TRADITIONAL APPROACHES (12-MONTH RETROSPECTIVE ANALYSIS)

Metric	Traditional	AI-Driven
Mean Time to Detect (min)	14.7	6.2
Mean Time to Resolve (min)	43.8	18.4
False Positive Rate	–	8.3%
Incidents Predicted Early	0%	62.1%
SLO Breaches Prevented	–	47
Playbook Match Accuracy	–	78.6%
Escalation Rate Reduction	–	34.2%

VI. DISCUSSION: DEPLOYMENT, GOVERNANCE, AND RISK

A. Model Risk Management

Financial regulators, including the OCC, the Fed, and the PRA, require that models used in banking be subject to rigorous model risk management (MRM) frameworks, as outlined in guidance such as OCC 2011-12 (SR 11-7). While these requirements were originally designed for financial models (credit scoring, market risk, etc.), many institutions have begun extending MRM principles to operational and infrastructure models.

For the AI systems described in this paper, MRM considerations include:

• Model documentation: Each model (RL policy, cost estimator, anomaly detector) must be documented with its intended use, assumptions, training data, performance metrics, and limitations. This is operationally burdensome but serves a vital function: it forces teams to articulate the conditions under which the model can be trusted and those under which it should not be.

• Independent validation: Models should be validated by a team independent of the developers, using held-out data and adversarial test cases. For the RL autoscaling agent, validation should include stress tests with synthetic workload spikes, simulated regulatory deadlines, and scenarios where the agent's training data does not cover the current regime.

• Ongoing monitoring: Once deployed, models must be monitored for performance degradation, concept drift, and unexpected behavior. The shadow-mode deployment strategy described in Section III naturally supports this, but monitoring must continue after full deployment.

B. Interpretability and Auditability

In regulated environments, the ability to explain why a particular action was taken is often as important as the quality of the action itself. This presents a challenge for RL-based systems, where policies are encoded in neural network weights that resist human interpretation.

We address this through several mechanisms. Action logging with context: every action taken by the system is logged alongside the state that informed it and the alternative actions that were considered. This creates an audit trail that regulators and internal auditors can review. Post-hoc explanation: SHAP (SHapley Additive exPlanations) values or similar techniques can be applied to explain which features most strongly influenced a particular decision. For instance, explaining that the agent scaled up the pricing service primarily because of the proximity to market close and the elevated VIX level. Constraint documentation: the safety layer's constraints are explicitly documented and auditable, so

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, February 2026 Impact Factor: 8.2

any action that was blocked (and why) is recorded. This is arguably more interpretable than traditional approaches, where the rationale for manual scaling decisions is often unrecorded.

C. Organizational Considerations

Deploying AI-driven infrastructure optimization is as much an organizational challenge as a technical one. Success requires collaboration across traditionally siloed teams:

• SRE teams must be willing to cede some control over scaling decisions to an automated system, which requires trust built through the shadow and canary phases described above.

• Data science and ML engineering teams must understand infrastructure deeply enough to design reward functions and feature engineering pipelines that reflect actual operational realities, not simplified textbook models.

• Risk and compliance teams must engage early to define the MRM requirements and approve the deployment strategy, rather than being consulted as an afterthought.

• Engineering leadership needs clear KPIs to justify the investment: cost savings (cloud spend reduction), reliability improvements (SLO compliance rates, MTTR), and productivity gains (reduction in on-call burden and manual scaling interventions).

In our experience, the most effective way to build crossfunctional buy-in is to start with high-visibility wins on noncritical systems, where the cost of failure is low and the potential savings are easy to measure. Once the system has demonstrated value in these lower-risk settings, expanding to more critical services becomes a natural progression.

VII. FUTURE WORK

Several promising directions extend beyond the scope of this paper.

Unified control loops. The three pillars we describe, autoscaling, data layer optimization, and incident management, are currently formulated as independent systems. In practice, they interact: an autoscaling decision affects database load, which affects caching behavior, which affects query latency, which may trigger incident alerts. A unified approach that jointly optimizes across these layers, perhaps using hierarchical RL or multi-agent reinforcement learning, could capture these interactions and avoid conflicting actions.

Federated and privacy-preserving learning. Financial institutions are, understandably, reluctant to share operational data with external parties. However, there is significant potential value in learning from operational patterns across institutions, a novel failure mode encountered at one bank could inform incident prediction at another. Federated learning techniques, where models are trained collaboratively without sharing raw data, could enable this. Differential privacy guarantees would need to be incorporated to address regulatory and competitive concerns, and the governance framework for such cross-institutional collaboration remains an open challenge.

Benchmarking suites for AI-for-infrastructure in finance. The lack of standardized benchmarks and shared datasets has been a barrier to progress in AI-for-systems research more broadly, and this gap is especially acute in financial services where operational data is sensitive. Development of anonymized, realistic benchmark workloads, analogous to TPC-C for transactional databases or MLPerf for ML training, would accelerate research and enable more rigorous comparison of approaches. We advocate for industry consortia or regulatory bodies to facilitate the creation of such benchmarks.

Reinforcement learning from human feedback (RLHF) for operational decisions. While RLHF has attracted enormous attention in the development of conversational AI systems, the technique has potential applications in operational decision-making as well. Experienced SREs possess deep intuitive knowledge about the relative desirability of different scaling strategies, remediation actions, and operational tradeoffs. Incorporating this knowledge through RLHF-style training could produce policies that are better aligned with human operational preferences than those trained purely on cost and latency metrics.

Causal inference for root cause analysis. Current correlation-based approaches to root cause analysis can identify symptoms that co-occur with incidents but struggle to distinguish causes from effects. Incorporating causal inference methods, structural causal models, do-calculus, or time-series causal discovery techniques, into the incident analysis

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, February 2026
Impact Factor: 8.2

pipeline could improve the accuracy of root cause identification and reduce the frequency of false or misleading remediation recommendations.

VIII. CONCLUSION

The infrastructure challenges facing large financial institutions are both technical and organizational. The systems are enormous, the constraints are severe, and the cost of getting it wrong, in terms of regulatory penalties, financial losses, and reputational damage, is substantial. Traditional approaches to infrastructure management, based on static rules, manual tuning, and reactive incident response, are increasingly inadequate for the scale and complexity of modern financial technology platforms.

In this paper, we have presented a conceptual framework for applying AI-driven optimization across three critical dimensions of financial infrastructure. Reinforcement learning and bandit methods can transform autoscaling and job scheduling from reactive, heuristic-driven processes into proactive, cost-aware optimization that respects the unique regulatory constraints of financial services. Learning-based data layer optimization can adapt indexing, caching, and query routing strategies to the evolving workload patterns that characterize financial systems, delivering measurable latency improvements and cost reductions. And AI-driven incident prediction and remediation can shift the operational paradigm from reactive firefighting to proactive prevention, with playbook recommendation systems that encode institutional knowledge and present it to operators at the moment of greatest need.

We do not claim that deploying these systems is easy. The deployment, governance, and organizational challenges are substantial, and we have tried to address them honestly in our discussion. Model risk management requirements add overhead. Interpretability remains a challenge for neural-networkbased policies. Building cross-functional trust takes time. But the potential impact, reduced cloud expenditure, improved SLA compliance, faster incident resolution, and the freeing of scarce engineering talent from repetitive operational toil, makes this a compelling area of investment for any financial institution operating at scale.

We hope that this paper provides a useful starting point for organizations embarking on this journey, and that the frameworks, architectures, and deployment strategies we describe contribute to a growing body of practical knowledge at the intersection of AI, distributed systems, and financial services infrastructure.

REFERENCES

[1] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, “CherryPick: Adaptively unearthing the best cloud configurations for big data analytics,” in Proc. NSDI, 2017.

[2] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, “AdaptSize: Orchestrating the hot object memory cache in a content delivery network,” in Proc. NSDI, 2018.

[3] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, “Continuous incident triage for large-scale online service systems,” in Proc. ASE, 2019.

[4] Y. Chen, X. Yang, Q. Lin, H. Zhang, F. Gao, Z. Xu, Y. Dang, D. Zhang, Y. Sui, and J. Li, “Outage prediction and diagnosis for cloud service systems,” in Proc. The Web Conference, 2020.

[5] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in Proc. CCS, 2017.

[6] Flexera, “2024 State of the Cloud Report,” Flexera, 2024.

[7] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” in Proc. ICWS, 2017.

[8] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and R. Caruana, “DeepDB: Learn from data, not from queries!” Proc. VLDB Endow., vol. 13, no. 7, 2020.

[9] K. Hundman, V. Constantinou, C. Laber, and M. Brundage, “Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding,” in Proc. KDD, 2018.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, February 2026 Impact Factor: 8.2

[10] N. Jay, N. H. Rotman, P. B. Godfrey, M. Schapira, and A. Tamir, "A deep reinforcement learning perspective on internet congestion control," in Proc. ICML, 2019.
[11] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," in Proc. CIDR, 2019.
[12] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in Proc. SIGMOD, 2018.
[13] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," in Proc. NeurIPS, 2020.
[14] N. Lazic, C. Boutilier, T. Lu, E. Wong, B. Roy, M. Ryu, and G. Imwalle, "Data center cooling using model-predictive control," in Proc. NeurIPS, 2018.
[15] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in Proc. ICSOC, 2018.
[16] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica, "DistCache: Provable load balancing for large-scale storage systems with distributed caching," in Proc. FAST, 2020.
[17] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in Proc. HotNets, 2016.
[18] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in Proc. SIGCOMM, 2019.
[19] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, "Neo: A learned query optimizer," Proc. VLDB Endow., vol. 12, no. 11, 2019.
[20] V. K. Mishra et al., "The role of generative AI in revolutionizing healthcare, education, and finance: A mini review," International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), vol. 5, no. 2, Mar. 2025.
[21] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in Proc. KDD, 2019.
[22] J. Sun and G. Li, "An end-to-end learning-based cost estimator," Proc. VLDB Endow., vol. 13, no. 3, 2019.
[23] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in Proc. ICAC, 2006.
[24] G. Vietri, L. V. Rodriguez, W. A. Martinez, S. Lyons, J. Liu, R. Rangaswami, M. Zhao, and G. Narasimhan, "Driving cache replacement with ML-based LeCaR," in Proc. HotStorage, 2018.
[25] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in Proc. ESEC/FSE, 2019.
[26] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in Proc. ICSE (SEIP), 2019