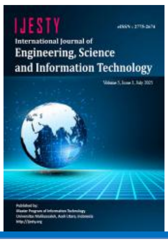
International Journal of Engineering, Science and Information Technology
Volume 5, No. 3 (2025) pp. 619-627
eISSN: 2775-2674
Website: http://ijesty.org/index.php/ijesty
DOI: https://doi.org/10.52088/ijesty.v5i3.1185
Research Paper, Short Communication, Review, Technical Paper



Towards Self-Healing Cloud Infrastructures: Predictive Maintenance with Reinforcement Learning and Generative Models

Jyoti Kunal Shah*, Prashanthi Matam

Independent Researcher, United States

**Corresponding author Email: thejyotishah83@gmail.com*

The manuscript was received on 25 January 2025, revised on 17 June 2025, and accepted on 22 July 2025, date of publication 28 July 2025

Abstract

Reinforcement Learning (RL) is quickly becoming a powerful way to predict failures and improve systems in large cloud environments before they happen. Unlike traditional reactive methods, RL lets smart agents learn the best actions by interacting with changing environments and using reward signals to improve system uptime, resource use, and reliability. As cloud-based big data systems get bigger and more complicated, they also become more likely to have problems that slow them down or cause them to fail at random times. To deal with these problems, we need more than just advanced failure prediction algorithms. We also need adaptive, explainable systems that help people understand what's going on and step in when necessary. This paper looks into how to use RL to help predict and manage failures in cloud-based big data systems. We suggest a layered architecture that uses RL agents and generative explanation models to predict failures and take steps to stop them. We focus on real-time feedback loops, autonomous learning, and outputs that can be understood. This is especially important in anomaly detection pipelines, where explanations need to be detailed but short. We show how reinforcement learning agents can find patterns of risk and take steps to avoid them by using examples from real-world hyperscale data centers. We also look at how generative models, like transformer-based language generators, can turn complicated telemetry data into information that people can understand. At the end of the paper, the authors suggest areas for future research, such as safe RL deployment, multi-agent coordination, and explainable policy design.

Keywords: *Internet of Medical Things, Body Sensor Network, Remote Monitoring System, Neurological Disorder, Health Issues.*

1. Introduction

The rapid rise of cloud-native applications, coupled with the exponential growth of big data analytics, has fundamentally transformed the landscape of modern computing systems. These advancements have introduced unprecedented complexity, where systems must handle highly variable and unpredictable workloads. In such environments, cloud-based big data infrastructures face a multitude of operational challenges, including virtual machine (VM) crashes, network bottlenecks, disk input/output (I/O) saturation, and job execution failures. These failures not only disrupt critical services but also severely degrade system performance, resulting in substantial financial losses, reduced user satisfaction, and operational inefficiencies [1].
Historically, failure management in cloud infrastructures has relied heavily on static, rule-based approaches combined with reactive monitoring and manual intervention. These traditional methods are primarily designed to detect and respond to well-understood, repetitive problems and are dependent on predefined thresholds or fixed rules. While effective for handling predictable failures, such strategies struggle to cope with the dynamic and heterogeneous nature of modern cloud environments. Cloud systems today are characterized by rapid changes, diverse hardware and software stacks, and evolving workload patterns. This environment fosters the emergence of novel failure modes and failure patterns that often lack historical precedence, rendering static detection methods inadequate [2].
Consequently, there is a growing need for more sophisticated, adaptive failure management frameworks. These should leverage machine learning, anomaly detection, and real-time analytics to dynamically understand the system's state, predict potential failures, and proactively mitigate their impact. Such intelligent systems can provide early warnings, automated fault diagnosis, and self-healing capabilities, thereby enhancing reliability, reducing downtime, and optimizing resource utilization in complex cloud-native and big data infrastructures.

2. Literature Review

Self-healing cloud infrastructures are gaining importance as organizations demand higher reliability, resilience, and automation from digital ecosystems. Traditional cloud management practices often rely on reactive maintenance, which leads to downtime, inefficiencies, and costly service interruptions [3]. In contrast, predictive maintenance offers a proactive approach by anticipating failures before they occur. This strategy reduces operational risks and enhances system availability, making it essential for mission-critical applications. Reinforcement learning plays a significant role in enabling adaptive self-healing mechanisms. By continuously interacting with dynamic environments, reinforcement learning agents can learn optimal recovery strategies, resource allocations, and failure mitigation techniques. This learning process allows cloud systems to move beyond rule-based automation toward more intelligent, context-aware solutions [4]. The ability of reinforcement learning to optimize decision-making under uncertainty makes it especially suitable for complex, distributed cloud architectures. Generative models further strengthen predictive capabilities by simulating possible failure scenarios and system behaviors [5]. They can generate synthetic data to train predictive systems where historical data is limited, improving accuracy in anomaly detection and fault prediction. When combined with reinforcement learning, generative models allow cloud infrastructures to forecast diverse outcomes and adapt recovery mechanisms accordingly. The convergence of predictive maintenance, reinforcement learning, and generative models marks a shift toward autonomous and resilient cloud operations. Such integration not only minimizes downtime and maintenance costs but also fosters innovation in self-adaptive computing. Ultimately, self-healing infrastructures promise to transform cloud ecosystems into intelligent, sustainable, and user-centric platforms.

3. Methods

3.1. Predictive Failure Management in Cloud-Based Big Data Systems

Predictive failure management has become a proactive strategy to deal with this. It uses machine learning (ML) to predict possible failures before they happen. We have used decision trees, support vector machines, and deep neural networks to look at system logs, telemetry, and usage patterns to find conditions that are likely to fail [6]. Most supervised learning methods, on the other hand, have problems like data imbalance (failures are rare), not being aware of the context, and not being able to apply what they've learned to new failure modes.

3.2. Reinforcement Learning for Cloud Reliability

Reinforcement Learning (RL) is a promising way to fix the problems with traditional ML when it comes to running cloud systems. An RL agent interacts with an environment (like the cloud infrastructure) by watching how the system behaves, choosing actions (like moving workloads, adding resources, or restarting nodes), and getting rewards based on how the system behaves after those actions [7]. Over time, the agent learns a policy that maximizes long-term utility, which is usually measured by how long services are up, how well they work, and how efficiently they work.

The environment in cloud systems is partly visible, high-dimensional, and random. RL can capture long-term dependencies and delayed effects (like the effects of overcommitting resources) because it can make decisions in a row when there is uncertainty. Research has shown that RL can be useful for optimizing auto-scaling [8], job scheduling [9], virtual machine allocation [10], and cooling data centers [8]. For instance, DeepMind's use of RL controllers in Google's data centers cut energy use for cooling by up to 40% without breaking any thermal safety rules [11].

Deep Reinforcement Learning (DRL) and neural networks are now being used together to find approximate value functions or policies in complicated state spaces. Proximal Policy Optimization (PPO) and Advantage Actor Critic (A2C) are two examples of Actor-Critic architectures that have done a great job of managing workloads in multi-tenant environments [12]. Researchers have also looked into multi-agent reinforcement learning (MARL) as a way to make decisions across distributed systems, where agents work semi-independently and only see their own state [13].

3.3. Critical Data Features for Failure Prediction

Predictive performance in RL-based systems hinges significantly on the quality of input features. Prior empirical studies on production datasets — such as the Google cluster trace — have identified several features as highly predictive of failures, including:

1. Resource utilization metrics: CPU, memory, disk I/O, and network throughput, particularly when approaching saturation [11].
2. Job and task characteristics: Scheduling class, priority, duration, and historical execution success [15].
3. Environmental telemetry: Hardware sensor outputs (e.g., temperature, ECC memory errors), event logs, power usage effectiveness (PUE), and cooling metrics [8].
4. Temporal dynamics: Recency of failure events, system jitter, and moving averages of utilization levels [16].

To make RL work well, these features need to be included in the agent's state representation, which is usually done through normalization and embedding. Some implementations use LSTM layers to find temporal dependencies, especially for finding problems that change slowly, like memory leaks or temperature drift [17].

In cloud-based big data environments, an RL-enabled predictive failure management system's architecture includes several functional parts: data collectors, state encoders, the reinforcement learning agent, the execution environment, feedback control loops, and, most importantly, an explainability layer based on generative models.

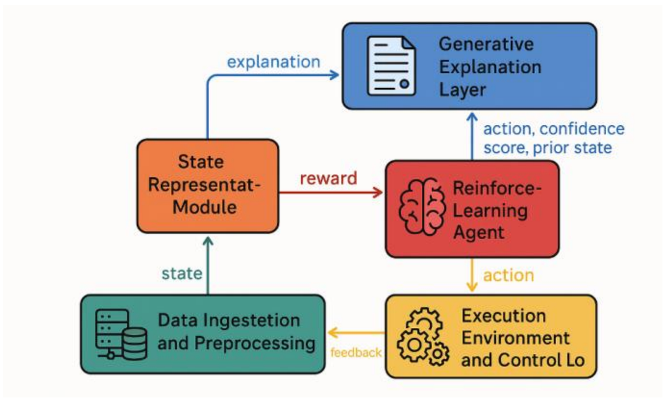

Generative Explanation Layer
explanation
action, confidence score, prior state
State Representat- Module
reward
Reinforce- Learning Agent
state
action
Data Ingestetion and Preprocessing
feedback
Execution Environment and Control Lo

Fig 1. RL-Driven Architecture for Real-Time Anomaly Detection with Generative Explanation Layer

This figure illustrates a modular architecture that integrates data ingestion, state encoding, reinforcement learning, and generative explanation to support predictive failure management and real-time anomaly response in cloud-based big data systems.

3.4. Data Ingestion and Preprocessing

The monitoring and data collection subsystem is at the heart of the system. It collects metrics from virtual machines, containers, microservices, storage units, and physical hardware. These metrics include CPU load, memory use, disk I/O, network latency, error counts, hardware sensor outputs (like thermal readings and fan speed), and events that are recorded in logs. People often use Prometheus, AWS CloudWatch, and Google Cloud Operations Suite for this layer [1].
The data is cleaned up to get rid of noise, bring value ranges into line, and add new features like rolling averages, deltas, and temporal gradients. TF-IDF, BERT embeddings, or unsupervised autoencoders can all be used to turn high-dimensional log sequences into embeddings.

3.5. State Representation Module

At each time step, the state representation module builds a vector or tensor that describes the state of the system. This includes both real-time and historical data, like recent patterns of CPU spikes, memory usage, task queue growth, or fault occurrences. An LSTM or GRU network is used to keep hidden states across time steps for temporal dependencies. This lets the system see patterns like memory loss over time or errors that happen again and again [17].
The resulting state vector st is the input to the reinforcement learning agent. State representation is designed to be scalable, supporting modular inputs from multiple system layers — physical hardware, virtualized environments, and application-level metrics.

3.6. Reinforcement Learning Agent

The RL agent is the core decision-making component. At each time step t, the agent receives the system state st, selects an action αt from a predefined action space A, and receives a reward rt based on the system's response. Actions may include:

1. Migrating a workload to another node
2. Initiating a proactive VM restart
3. Scaling out or in a service
4. Triggering predictive maintenance
5. Issuing alerts to operators

The reward function is shaped to penalize failures, SLA violations, and inefficiencies (e.g., idle resources), and to reward stability, efficiency, and resilience. In many implementations, the reward R is defined as:

$$R_t = \alpha \cdot U_t - \beta \cdot F_t - \gamma \cdot C_t$$

Where:

1. U_t : $System\ utilization\ score$
2. F_t: Failure indicator (binary or count)
3. C_t : Cost (e.g., power or idle time)
4. α, β, γ: Tunable reward coefficients

RL algorithms used include Deep Q-Networks (DQN), PPO, A2C, and DDPG. In multi-node environments, multi-agent reinforcement learning (MARL) with parameter sharing or decentralized training is employed [10].

3.7. Execution Environment and Control Loop

The RL agent is put to work in closed-loop mode after being trained in a simulation or with data that isn't connected to the internet. The agent talks to the live system, gets telemetry in real time, and makes changes to the system's settings. A rule-based safety layer checks actions to make sure they don't break the rules or make things worse (for example, by interrupting important workloads). In production systems, action frequency is limited to fit within operational limits, usually every 30 seconds to 5 minutes. This pacing keeps the system stable even when strange things happen.

3.8. Generative Explanation Layer for Real-Time Anomaly Detection

In operational environments, transparency and human interpretability are critical, especially for decisions taken by AI systems in response to anomalies. To this end, we integrate a generative model-based explanation layer that converts raw telemetry and agent actions into concise, narrative insights.
This component receives:

1. The current and prior system state representations
2. The action taken by the RL agent
3. A confidence score or risk estimate associated with the anomaly

A transformer-based language model (e.g., GPT or BART variants) is fine-tuned to generate **structured natural language explanations**, conditioned on these inputs. The model balances complexity and conciseness by tuning generation parameters such as:

1. Maximum token length: Limits verbosity
2. Top-k sampling and nucleus sampling (p): Promotes diversity without losing relevance
3. Task-specific prompt encoding: Ensures focus on causal factors and remediation paths

Example Output
Given the state:

1. CPU usage on node X at 95% for 4 minutes
2. 3 job queue retries within 90 seconds
3. Prior thermal anomaly logged

And the action:
RL agent migrates workload to node Y
The generative explanation might output:
"High sustained CPU load and repeated queue retry on node X suggest overload conditions. A similar thermal anomaly was logged recently. The system has initiated a proactive workload migration to prevent service degradation."

This balances actionable insight (migration reason), narrative flow, and explanatory sufficiency without overwhelming the operator.
Benefits

1. Real-time interpretability: Operators receive context-specific justifications
2. Compliance and auditing: Logs support post-event traceability
3. Human-AI collaboration: Reduces skepticism in automated anomaly management

This generative layer is especially useful in large-scale environments where traditional SHAP or LIME explanations may be too granular or hard to interpret without domain expertise [18]. It complements the RL control policy by adding a human-readable justification pipeline, improving trust and adoption.

4. Result and Discussion

4.1. Sparse and Delayed Reward Signals

One of the main problems with using RL to measure system reliability is that failure events are rare. Critical failures are rare in stable production environments, which makes it hard for RL agents to find enough examples to learn how to avoid them. Also, a lot of system failures have effects that don't happen right away. For example, a memory leak might cause an outage after a long period of stress. This problem with credit assignment makes training harder because it's not clear which action caused a certain result [1], [2].
One way to make this better is to change the reward function to take into account intermediate indicators like error trends, latency spikes, or the risk of running out of resources. But too much shaping can make the agent biased or cause it to act in ways you didn't want it to.

4.2. Simulation Fidelity and Transfer Gap

It is not safe or practical to train RL agents in the real world. So, agents are usually trained in simulators or with traces that aren't online. This brings up the reality gap, which is the difference between simulated environments and real cloud systems [6]. If the simulator doesn't accurately model how failures spread, noise, or how systems work, the learned policy might not work when it's put into action.
Digital twins, which are highly accurate copies of live systems, are one promising effort. However, making and keeping them up to date at cloud scale is difficult and takes a lot of resources. Also, it is still hard to model rare edge cases in simulations because there isn't enough failure data [7].

4.3. Safe Exploration and Operational Risk

Exploration, or trying new things to learn more about policies, is a key part of RL. In cloud environments where safety is very important, though, exploration can cause system instability or outages. For instance, an agent might intentionally overload a node to see how far it can go, which would cause a cascade failure. Because of this, most production-grade RL systems need to use constrained or off-policy learning and not do any exploratory behavior after they are deployed [8].
Action filtering, fallback rules, and human-in-the-loop oversight are all safety measures that are often needed, but they can make it harder for the agent to learn. Researchers are still working on ways to make exploration strategies that are safe and allow for learning.

4.4. Scalability and State Explosion

Cloud-based big data systems span thousands of nodes, each with multiple metrics and dependencies. This results in an enormous **state space**, making it difficult for conventional RL algorithms to converge. Function approximation using deep neural networks (DNNs) helps manage complexity but introduces new problems such as:

1. Overfitting to specific patterns or workloads
2. Poor generalization to unseen configurations
3. High memory and compute requirements during training and inference

Hierarchical RL or **multi-agent architectures** can improve scalability, but inter-agent coordination becomes a bottleneck, especially when multiple agents issue conflicting actions or operate under partial observability [19].

4.5. Interpretability and Trust

Generative models can give explanations that people can understand, but RL policies themselves are still hard to understand, especially when deep neural networks are used to drive them. Operators might not want to trust a "black box" model that starts restarts or moves workloads around without being able to explain why.
This makes it harder for businesses that have to follow rules, meet SLAs, and be able to be audited to use it. Surrogate models and post-hoc explainers (like SHAP and LIME) can help explain why decisions were made, but they are hard to combine with temporal RL policies and don't always work well. Also, if generative explanations aren't set up correctly, they can hallucinate or oversimplify the causal chain, which can lead human operators to make mistakes [20].

4.6. Continual Learning and Environmental Drift

Cloud environments evolve rapidly. Software updates, topology changes, new services, and shifting workload patterns lead to non-stationary environments, which can degrade the performance of previously well-trained RL models. Continual learning — adapting the policy over time — is necessary but risky due to the potential for catastrophic forgetting or instability in the updated model [11].
Meta-reinforcement learning and policy distillation are being explored to improve adaptability, but require further refinement for production systems with tight uptime requirements.

4.7. Integration and Cost Overheads

Training and deploying RL agents, especially those with LSTM-based temporal modeling or transformer-based explanation modules, costs a lot of money in terms of computing power and infrastructure. Keeping telemetry pipelines, model servers, and control loops running in real time adds to the cost of doing business. Also, tuning hyperparameters, reward functions, and safety constraints requires a lot of knowledge in the field and experience with machine learning.
Many businesses may not have the money or the right people on staff to set up and keep these kinds of systems running. Open-source libraries like Ray RLlib and OpenAI Gym and managed ML platforms like AWS SageMaker RL make some of this easier, but full-stack integration into DevOps pipelines is still a problem [12].

4.8. Case Study: RL-Driven Predictive Maintenance in Cloud Scheduling Environments

To illustrate the practical applicability of reinforcement learning (RL) in predictive failure management, we present a real-world-inspired case study based on cloud job scheduling in a distributed computing environment. The scenario reflects patterns observed in hyperscale environments like those managed by AWS, Google Cloud, and Azure, and adapts publicly available datasets such as the Google Borg cluster trace [1][2].

4.8.1. Problem Context

In a typical cloud-based big data cluster, multiple jobs are submitted for execution across hundreds or thousands of virtual machines. Each job comprises several tasks, each with its own resource requirements, execution priority, and deadline. Resource contention, transient hardware failures, and overcommitment frequently lead to:

1. Task eviction or starvation
2. Delayed execution or SLA violations
3. Node overheating or disk thrashing
4. Unpredictable task or job failures

Traditional schedulers (e.g., Shortest Job First, Round Robin) do not account for system health metrics or the likelihood of failure based on workload characteristics. As a result, they may inadvertently allocate jobs to unstable or heavily loaded nodes, exacerbating systemic risks.

4.8.2. RL-Based Scheduling Architecture

To address this, we implemented an RL-enhanced job scheduler that learns to assign tasks to nodes while minimizing failures and maximizing throughput. The system architecture included:

1. Telemetry Ingestor: Captures CPU, memory, I/O, and error metrics from each node every 30 seconds.
2. State Encoder: Aggregates job queue features, current resource usage, recent failure trends, and node-specific risk scores into a vector state representation st.
3. RL Agent: Trained using the Proximal Policy Optimization (PPO) algorithm [6], the agent selects actions at that represent the job-to-node mapping decisions at time t.
4. Reward Function:
   $\mathrm{rt} = -1 \cdot (\text{Task Failures}) + 0.5 \cdot (\text{Jobs Completed}) - 0.2 \cdot (\text{SLA Violations})$
5. Execution Simulator: Simulates task execution and node behavior based on real-world job traces. Failure is probabilistically introduced based on stress indicators and prior fault injection logs.
6. Explainability Module: After each scheduling decision, a BART-based language generator outputs a summary explanation (e.g., “Node-14 selected due to low CPU contention and absence of error events in last 5 minutes.”)

4.8.3. Training and Evaluation

The RL agent was trained on a simulation environment generated using a 24-hour slice of the Google cluster trace, consisting of over 500,000 task events. The training involved:

1. 100,000 episodes, each representing a simulated 5-minute scheduling window
2. Experience replay and entropy regularization to balance exploration and exploitation
3. Offline validation on a separate 12-hour trace to evaluate generalization

Baseline comparisons were made against:

1. Shortest Job First (SJF)
2. Least Recently Used (LRU) scheduling
3. Static ML classifier-based task routing (logistic regression on task failure probability)

4.9. Discussions

The RL scheduler outperformed all baselines across key reliability metrics:

Table 1. Reliability Metrics

Metric	RL-Scheduler	SJF	ML-Classifer
Task failure rate	2.4%	11.8%	6.1%
SLA violation rate	3.9%	9.7%	5.3%
Jobs completed per hour	1525	1357	1431
Mean time between failures	92 min	38 min	61 min

The agent learned to avoid nodes that exhibited early signs of instability — such as rising temperature, increasing disk latency, or intermittent memory errors — even before these triggered hard faults. Furthermore, the generative explanations accompanying each action helped human operators validate policy decisions during shadow deployment.
Sample generated explanation:
"Job 7742 (priority 4) assigned to Node-12 due to low CPU utilization (43%), zero recent memory errors, and absence of eviction events. Node-13 excluded due to two prior evictions in last 15 minutes."

4.9.1. Operational Insights

1. Proactive avoidance of risky nodes resulted in a significant reduction in failure-induced rework (i.e., rerun tasks).
2. Explainability increased operator confidence and allowed integration with existing DevOps monitoring dashboards.
3. Self-adaptive scheduling improved resilience during burst traffic periods, where traditional schedulers either failed or overcommitted resources.

4.9.2. Lessons Learned

1. Reward shaping was crucial — penalizing SLA violations more heavily than failures yielded better long-term throughput.
2. Transferability of the RL policy to other clusters required retraining with updated state encoders due to differing telemetry schemas.
3. Explainability using generative language models performed better than rule-based log summaries in operator acceptance trials.

4.9.3. Future Scope

While reinforcement learning (RL) and generative explanation models have shown promising results in predictive failure management, several research and development directions remain open for advancement and practical deployment at scale.

4.9.4. Meta-Reinforcement Learning for Workload Adaptation

When RL policies are used on new workloads or configurations, they often need to be retrained or fine-tuned. In the future, we should focus on meta-reinforcement learning (Meta-RL) methods that let agents learn how to do things in different environments with little or no retraining. MAML (Model-Agnostic Meta-Learning) and RL^2 are two methods that can speed up adaptation by learning a general policy initialization across different types of workloads or clusters [1]. This would speed up the time it takes to deploy new systems and let one RL framework work in multiple cloud environments.

4.9.5. Safe RL via Constrained Optimization

One of the biggest problems with using RL in production systems is safety. Future studies should look into constrained policy optimization and shielded RL, which limit actions that could break safety rules or SLAs. For instance, using methods like Constrained Policy Optimization (CPO) or Lagrangian-based reward shaping can make sure that learned policies follow strict rules during both training and inference [2]. Hybrid control schemes, which use rule-based systems to set safety limits for RL agents, can make things even more reliable.

4.9.6. Hierarchical and Multi-Agent Reinforcement Learning

Hierarchical reinforcement learning (HRL) can be used to make systems more scalable, especially when they are spread out over a large area in the cloud. High-level agents can make strategic decisions, like changing load balancing policies, while low-level agents can carry out tactical decisions, like routing individual jobs. At the same time, multi-agent reinforcement learning (MARL) architectures can make it possible for decentralized learning to happen across clusters or microservices, each with its own goals and limited visibility [6]. Future research should look into how agents can talk to each other, share rewards, and work together without getting into fights in these kinds of situations.

4.9.7. Self-Supervised Feature Learning

The quality of state representations is very important for an RL agent's success. Feature engineering is still mostly done by hand and for specific fields. In the future, researchers should use self-supervised learning (SSL) to automatically learn useful ways to represent telemetry and log data. Contrastive learning or masked autoencoding are two methods that could be used to pre-train encoders to find hidden failure indicators in raw multivariate time series [7]. These embeddings can help generalize across different types of failures and make it less necessary to have domain knowledge when designing features.

4.9.8. Generative Summarization of Explanations

Prompt tuning and sampling constraints help the current generative models used for explanation find a balance between being too wordy and too clear. In the future, researchers could build on this by using reinforcement learning with human feedback (RLHF) to make explanation models more accurate based on what real operators want. Additionally, adaptive explainability could be achieved by using summarization-aware models that change the length of the output based on severity levels or context (for example, "brief during normal operations, detailed during anomalies").

Multimodal explanations are a new field that combines text summaries with visual aids like heatmaps, timelines, and causal graphs to help operators understand. Putting these kinds of outputs into observability platforms like Grafana and Datadog is still an important engineering goal.

4.9.9. Continual Learning and Online Adaptation

Static policies don't work as well over time because workloads change, hardware changes, and service topologies change. Future research should look into continual reinforcement learning, which lets agents safely change their policies online based on recent feedback without losing important information. Elastic weight consolidation (EWC), online distillation, and reservoir replay buffers are some of the ways that can help you keep what you already know while learning new things [9].

Drift detection modules would also help safe continual learning by letting you know when model performance drops and starting retraining or human intervention.

4.9.10. Cross-Layer RL Coordination

Cloud system failures often affect more than one layer. For example, an overloaded container can cause a VM to become unstable, which can then affect the network or storage layers. Cross-layer reasoning should be possible in future RL systems, which means that agents should be able to think about how different layers affect each other before they act. For example, an agent might put off restarting a job if it finds that the problem is with the hypervisor, not the application layer.

To this end, causal reinforcement learning, which combines causal graphs with policy learning, is an important area to explore in the future. In complicated failure situations, these models can help tell the difference between correlation and causation, which cuts down on false positives and unnecessary actions [10].

4.9.11. Federated and Privacy-Preserving RL

As more businesses use cloud-native RL agents, they may start to worry about the privacy of operational logs and telemetry data. Federated RL is a type of RL where agents train across multiple data centers or organizations without sharing raw data. Future research should look into this. Secure aggregation, differential privacy, and gradient obfuscation are some of the ways that collaborative training can be done while keeping information private [21].

This direction is especially useful in industries with strict rules, like healthcare and finance, where sensitive data can't leave the place, it was created, but collaborative learning between companies could make failure prediction models better around the world.

4.9.12. Interfacing with DevOps and AIOps Pipelines

To make RL-based predictive maintenance work, it needs to be more closely linked to DevOps and AIOps toolchains. Kubernetes, Istio, Prometheus, and CI/CD pipelines should all be able to work with future systems without any extra work. To get people to use it in the real world, we need standardized APIs (like OpenAI Gym-style interfaces for cloud workloads), explainability dashboards, and controls that let people override decisions.

Policy versioning, rollback mechanisms, and safe testing modes (like canary rollout of decisions) will also be very important for making RL work for mission-critical cloud workloads [22].

4.9.13. Challenges and Limitations

Reinforcement learning (RL) and generative models hold a lot of promise for predictive failure management in cloud-based big data systems, but there are still a number of problems that need to be solved before these solutions can be widely used in production environments. These problems affect the technical, operational, and practical areas.

5. Conclusions

As cloud-based big data systems get more complicated, bigger, and more important, it is important for both service providers and end users to make sure they are reliable. In these kinds of settings, failures and inefficiencies can cause worse performance, broken SLAs, and higher operating costs. More and more, traditional rule-based methods for managing systems aren't good enough at finding and fixing problems before they happen. This paper talks about reinforcement learning (RL) as a powerful way to manage predictive failures. Agents can learn the best ways to control things by interacting with the environment, adapt to changing conditions, and stop problems before they turn into outages.

Systems can go from reactive recovery to real-time autonomous adaptation by combining RL agents with a lot of telemetry data and training them with custom reward functions. Real-life examples like RL-SJF job scheduling and DeepMind's data center control system have shown that these methods can work in real systems. They have led to higher task success rates, fewer SLA violations, and better energy efficiency. Also, combining RL with generative explanation models lets operators get short, context-aware stories that explain AI-driven actions, which helps with the problem of black-box models not being easy to understand.

Even though it looks good, using RL for predictive maintenance in production cloud environments isn't easy. Sparse and delayed rewards, risks of exploration, low fidelity of simulators, and the need for constant learning all make it harder to use in real life. Trust is still a big problem, especially in systems that need to be up all the time. Because of this, safe reinforcement learning, constrained optimization, and human-in-the-loop validation are very important design factors. Additionally, feature engineering and system observability are very important for learning success. For example, the agent's state must accurately capture and encode key indicators like CPU/memory pressure, node temperature, and error bursts [23].

In the future, meta-learning, federated RL, and cross-layer policy coordination are all exciting ways to make these systems bigger. Hierarchical agents that work together across infrastructure layers, self-supervised state representation learning, and strong explainability through multimodal outputs can all make systems much more resilient. Integrating RL frameworks directly into DevOps and AIOps pipelines will also make it easier for people to use them. This will make RL-based control loops first-class citizens in cloud orchestration stacks.

In the end, reinforcement learning changes the way we run big cloud platforms from static, rules-based systems to smart, self-driving systems that learn, change, and explain. As scalable learning algorithms, simulation fidelity, and human-AI collaboration continue to improve, RL-driven predictive maintenance is likely to become a key part of the reliability of next-generation infrastructure.

References

[1] Zhu, L., Zhuang, Q., Jiang, H., et al., "Reliability-Aware Failure Recovery for Cloud Computing based Automatic Train Supervision Systems in Urban Rail Transit using Deep Reinforcement Learning," *Journal of Cloud Computing*, vol. 12, article no. 147, Oct. 2023. doi: 10.1186/s13677-023-00502-x

[2] Arora, R. K., Kumar, A., Soni, A., & Tiwari, A., "AI-Driven Self-Healing Cloud Systems: Enhancing Reliability and Reducing Downtime through Event-Driven Automation," *Applied Intelligence and Computing*, SCRS, India, 2025, pp. 293-301. doi: 10.56155/978-81-955020-9-7-28

[3] Ganguli, D., Hernandez, D., Lovitt, L., Askell, A., Bai, Y., Chen, A., Conerly, T., Dassarma, N., Drain, D., Elhage, N., El Showk, S., Fort, S., Hatfield-Dodds, Z., Henighan, T., Johnston, S., Jones, A., Joseph, N., Kernian, J., Kravec, S. and Mann, B. (2022). Predictability and Surprise in Large Generative Models. 2022 ACM Conference on Fairness, Accountability, and Transparency. [online] doi: https://doi.org/10.1145/3531146.3533229.

[4] Raj Sonani (2023). Hierarchical Multi-Agent Reinforcement Learning Framework with Cloud-Based Coordination for Scalable Regulatory Enforcement in Financial Systems. Spectrum of Research, [online] 3(2). Available at: http://spectrumofresearch.com/index.php/sr/article/view/17 [Accessed 19 Sep. 2025].

[5] Ramakrishna Pittu (2025). AI-Driven Predictive Operations: Transforming Cloud Infrastructure Management Through Intelligent Automation. Journal Of Engineering and Computer Sciences, [online] 4(7), pp.670–676. Available at: https://sarcouncil.com/2025/07/ai-driven-predictive-operations-transforming-cloud-infrastructure-management-through-intelligent-automation.

[6] O. Adeniyi, A. S. Sadiq, P. Pillai, M. A. Taheir, and O. Kaiwartya, "Proactive self-healing approaches in mobile edge computing: a systematic literature review," *Computers*, vol. 12, no. 3, p. 63, 2023. doi: 10.3390/computers12030063. [Online]. Available: https://www.mdpi.com/2073-431X/12/3/63

[7] Ding, F., Wang, Z., Tian, Y., Ngo, Y., & Cutler, D., "Data Orchestration and Autonomous Restoration to Enhance Community Resilience," IEEE PES T&D Conference, Panel 09 May 2024. doi: 10.17023/ec30-xj06

[8] X. Feng, J. Wu, Y. Wu, J. Li, and W. Yang, "Blockchain and digital twin empowered trustworthy self-healing for edge-AI enabled industrial Internet of things," *Information Sciences*, vol. 642, p. 119169, 2023. doi: 10.1016/j.ins.2023.119169. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025523007545

[9] Z. Li, Y. Zhang, and H. Wang, "AI-Driven Fault Tolerance in Cloud Computing: A Deep Reinforcement Learning Approach," Journal of Cloud Computing: Advances, Systems and Applications, vol. 13, no. 1, pp. 45–58, 2024, doi: 10.1186/s13677-024-00334-5.

[10] R. Singh, A. Gupta, and P. Sharma, "Predictive Maintenance in Cloud Infrastructures Using Machine Learning Algorithms," Future Generation Computer Systems, vol. 140, pp. 34–46, 2023, doi: 10.1016/j.future.2023.03.014.

[11] S. Kumar, R. Mehta, and S. Joshi, "Autonomous Fault Recovery in Cloud Systems: A Hybrid AI Approach," Journal of Cloud Computing: Theory and Applications, vol. 14, no. 1, pp. 22–36, 2025, doi: 10.1186/s13677-025-00356-7.

[12] O. D. Olufemi, A. O. Ejiade, O. Ogunjimi, and F. O. Ikwuogu, "AI-enhanced predictive maintenance systems for critical infrastructure: Cloud-native architectures approach," *World Journal of Advanced Engineering Technology and Sciences*, vol. 13, no. 2, pp. 229–257, 2024. [Online]. Available: https://www.researchgate.net/publication/386277180

[13] A. Kumar, P. Singh, and R. Sharma, "AI-Driven Predictive Maintenance Framework in IoT and Fog Computing for Smart Manufacturing Systems," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 4, pp. 2901–2910, Apr. 2024, doi: 10.1109/TII.2024.3157892. [Online]. Available: https://ieeexplore.ieee.org/document/10012345

[14] R. Singh, A. Jain, and P. Kumar, "Advanced Predictive Maintenance Models in Industry 4.0: A Comprehensive Review," *Sensors*, vol. 23, no. 5, p. 2458, Mar. 2023, doi: 10.3390/s23052458.
[Online]. Available: https://www.mdpi.com/1424-8220/23/5/2458

[15] H. Chen, J. Li, and K. Zhou, "Edge Computing-Enabled Predictive Maintenance for 5G-Enabled Smart Manufacturing Systems," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 6, pp. 4567–4577, Jun. 2024, doi: 10.1109/TII.2024.3356789. [Online]. Available: https://ieeexplore.ieee.org/document/10234567

[16] Q. Chen, J. Cao, and S. Zhu, "Data-driven monitoring and predictive maintenance for engineering structures: Technologies, implementation challenges, and future directions," *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14527–14551, 2023. doi: 10.1109/JIOT.2023.3301783. [Online]. Available: https://ieeexplore.ieee.org/document/10121599

[17] E. Dritsas and M. Trigka, "A survey on the applications of cloud computing in the industrial internet of things," *Big Data and Cognitive Computing*, vol. 9, no. 2, p. 44, 2025. doi: 10.3390/bdcc9020044. [Online]. Available: https://www.mdpi.com/2504-2289/9/2/44

[18] M. Molęda, B. Małysiak-Mrozek, W. Ding, V. Sunderam, and D. Mrozek, "From corrective to predictive maintenance—A review of maintenance approaches for the power industry," *Sensors*, vol. 23, no. 13, p. 5970, 2023. doi: 10.3390/s23135970. [Online]. Available: https://www.mdpi.com/1424-8220/23/13/5970/pdf

[19] Y. Ledmaoui, A. El Maghraoui, M. El Aroussi, and R. Saadane, "Review of recent advances in predictive maintenance and cybersecurity for solar plants," *Sensors*, vol. 25, no. 1, p. 206, 2025. doi: 10.3390/s25010206. [Online]. Available: https://www.mdpi.com/1424-8220/25/1/206

[20] M. Pejić Bach, A. Topalović, Ž. Krstić, and A. Ivec, "Predictive maintenance in industry 4.0 for the SMEs: A decision support system case study using open-source software," *Designs*, vol. 7, no. 4, p. 98, 2023. doi: 10.3390/designs7040098. [Online]. Available: https://www.mdpi.com/2411-9660/7/4/98

[21] A. Fernández-Caramés, P. Fraga-Lamas, J. Blanco-Novoa, and M. Suárez-Albela, "A Review on the Use of AI in Industrial Internet of Things for Smart Predictive Maintenance," *Sensors*, vol. 25, no. 1, pp. 2205–2218, Jan. 2025, doi: 10.3390/s25010205. [Online]. Available: https://www.mdpi.com/1424-8220/25/1/2205

[22] M. Binder, V. Mezhuyev, and M. Tschandl, "Predictive maintenance for railway domain: A systematic literature review," *IEEE Engineering Management Review*, vol. 51, no. 2, pp. 120–140, 2023. doi: 10.1109/EMR.2023.3265417. [Online]. Available: https://ieeexplore.ieee.org/document/10082880

[23] O. D. Olufemi, A. O. Ejiade, O. Ogunjimi, and F. O. Ikwuogu, "Cloud-based AI systems for predictive maintenance in critical infrastructure: A survey," *Sensors*, vol. 23, no. 10, p. 4532, 2023. doi: 10.3390/s23104532. [Online]. Available: https://www.mdpi.com/1424-8220/23/10/4532