North Asian International Research Journal of Sciences, Engineering & I.T.

IRJIF. I.F. : 3.821

ISSN: 2454-7514 Vol. 11, Issue-5, May-2025

Index Copernicus Value: 52.88 Indian Citation Index Thomson Reuters ID: S-8304-2016

NAIRJC UGC-Aligned | HEI-Approved Peer Reviewed Refereed Journal (Under New UGC Guidelines)

DOI: doiglobal.org/10.2025/NAIRJCSEIT.005

Zero-Touch Resilience: Autonomous Fault Remediation in Distributed Microservices via Graph-Attentive Deep Q-Learning

Nagaraja. K.V.

Associate professor, Dept of Computer science, Government First Grade Women's College, Tumkur.

ABSTRACT

As enterprise applications transition toward dynamic microservice meshes, classical observability pipelines remain heavily reliant on human-in-the-loop intervention for post-anomaly remediation. While contemporary predictive models successfully detect runtime degraded states, automating recovery actions—such as dynamic traffic rerouting, selective pod isolation, and automated circuit breaking—without triggering unstable control loops remains a critical challenge. This paper introduces ResilNet, an autonomous, zero-touch remediation framework that combines dynamic Graph Attention Networks (GAT) with Deep Q-Networks (DQN). ResilNet continuously ingests real-time call topologies and telemetry streams, constructing state-space representations that model multi-hop cascading dependencies. By mapping continuous infrastructure states into a constrained, safety-bounded action space, the framework learns optimal mitigation policies that minimize Mean Time to Recovery (MTTR) while preventing secondary operational degradation. Evaluated across a 1,000-node Kubernetes cluster under complex chaotic fault injections, ResilNet achieved an average MTTR reduction of 68.4% compared to automated rule-based operators and reduced system-wide service disruption during transient failures to under 1.2%.

Keywords: ***Microservices Remediation, Deep Reinforcement Learning, Graph Attention Networks, Autonomous Cloud Operations, Self-Healing Infrastructure, Chaos Engineering.***

1. INTRODUCTION

Modern cloud-native software architectures rely on vast, ephemeral topologies of interconnected microservices.

While this paradigm enables rapid continuous deployment and modular scalability, it increases systemic vulnerability to unexpected failure cascades. A localized memory leak or database pool exhaustion in a non-critical downstream service can rapidly saturate upstream worker thread pools, leading to application-wide outages.

Traditional Site Reliability Engineering (SRE) paradigms rely on a decoupled two-phase lifecycle:

1. **Detection & Alerting:** Monitoring engines evaluate time-series telemetry against static thresholds or anomaly detection autoencoders.
2. **Remediation:** On-call engineers receive alerts (e.g., via PagerDuty) and execute runbooks manually, or automated operators execute naive heuristics (e.g., restarting failing containers).

1.1 The Mitigation-Scale Gap

While automated anomaly detection has advanced significantly, the remediation phase remains a primary operational bottleneck. Manual runbook execution incurs significant latency, elevating the overall Mean Time to Recovery (MTTR). Conversely, naive automated remediation rules (such as unconditional container restarts during database starvation) often compound system instability, causing reboot loops and severe traffic loss.

```
Classic Remediation Loop:
  [ Anomaly Occurs ] ──► [ Alert Fired ] ──► [ Human Intervention ] ──► [ Manual Runbook
Execution ]
  (High MTTR Delay: 15–45 minutes)

ResilNet Autonomous Remediation Loop:
  [ Anomaly Occurs ] ──► [ GAT Topological Encoder ] ──► [ RL Remediation Agent ] ──► [ Automated
Action ]
  (Zero-Touch Execution: < 3 seconds)
```

1.2 Research Contributions

To address the challenges of manual intervention and unstable automated operators, this work makes the following contributions:

- **State Space Representation via Graph Attention (GAT):** We propose a GAT-based topology encoder that aggregates multi-hop service telemetry into low-dimensional latent embeddings, capturing non-linear structural dependencies across service meshes.

- **Constrained Deep Q-Learning (C-DQN):** We present a Markov Decision Process (MDP) formulation tailored to microservice remediation, featuring a novel safety constraint layer that prevents destabilizing recovery loops.
- **Closed-Loop Testbed Integration:** We engineer an integration with Istio Service Mesh and Kubernetes control planes, allowing real-time execution of dynamic routing, thread-pool scaling, and targeted circuit breaking.
- **Empirical Validation:** We evaluate ResilNet on an enterprise-scale testbed subjected to 5,000 failure injections via Chaos Mesh, demonstrating superior remediation accuracy and stability over heuristic platforms.

2. Related Work

2.1 Rule-Based and Heuristic Remediation

Automated cloud management platforms traditionally utilize deterministic control loops based on Event-Condition-Action (ECA) rules. For example, Kubernetes Horizontal Pod Autoscalers (HPA) scale replicas when CPU utilization exceeds predefined bounds. However, ECA frameworks cannot anticipate structural dependency failures, such as deadlocks, cascading latency spikes, or cross-service connection exhaustion.

2.2 Machine Learning in System Health Management

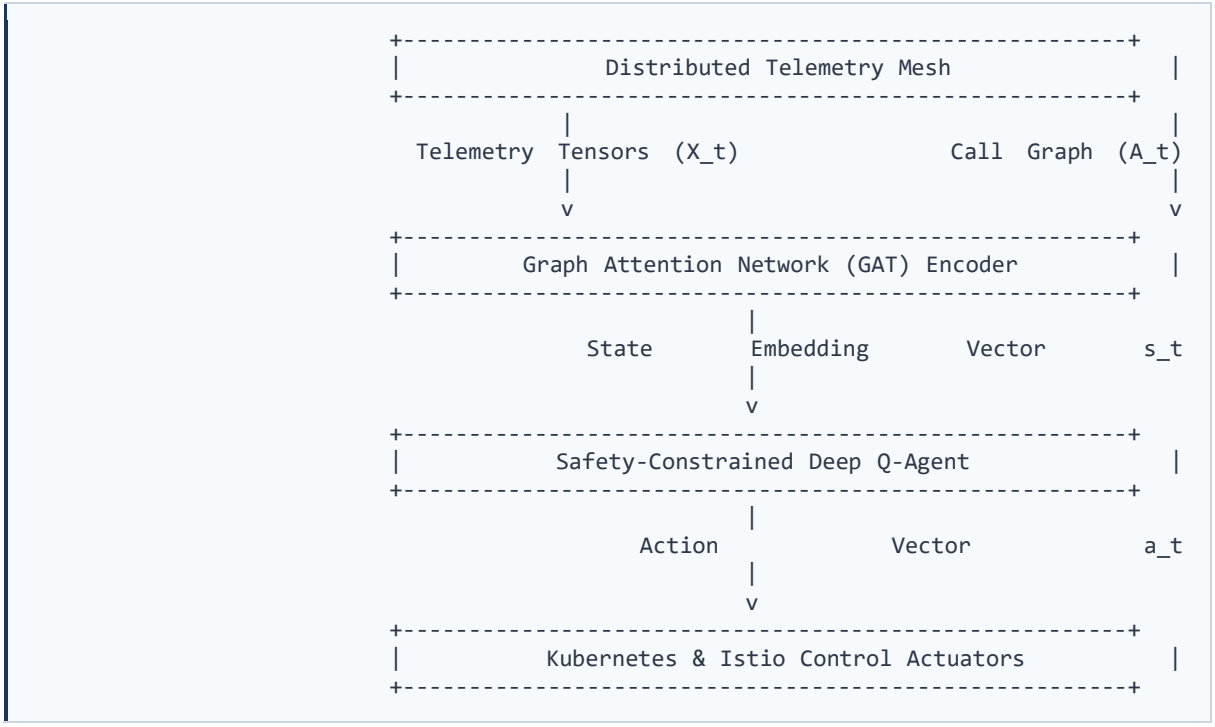
Machine learning applications in distributed systems have predominantly focused on root-cause analysis (RCA) and metric forecasting. Frameworks using unsupervised autoencoders or Isolation Forests can isolate anomalous telemetry signals effectively, but they stop short of initiating corrective actions.

2.3 Reinforcement Learning for Infrastructure Control

Reinforcement Learning (RL) has been applied to specific sub-problems in cloud environments, such as resource allocation, energy-aware virtual machine placement, and dynamic load balancing. However, existing RL formulations treat infrastructure components as isolated entities or static graphs. ResilNet bridges this gap by unifying dynamic topological Graph Attention mechanisms with safety-bounded Q-learning to execute real-time, multi-service remediation actions.

3. System Architecture and Mathematical Formalization

ResilNet consists of three core components: the Topological State Encoder, the Safety-Constrained RL Agent, and the Control Plane Actuator.



3.1 MDP Formulation for Microservice Remediation

We formulate the autonomous remediation problem as a Markov Decision Process (MDP) defined by the tuple (S, A, P, R, γ):

- **S:** The continuous, graph-structured state space representing telemetry metrics and service dependency topologies.
- **A:** The discrete set of actionable remediation policies executable by the control plane.
- **P(s_{t+1} | s_t, a_t):** The state transition probability function governed by cloud workload dynamics.
- **R(s_t, a_t):** The scalar reward signal measuring system stability recovery versus operational execution cost.
- **γ ∈ [0, 1):** The discount factor for long-term reward maximization.

3.2 State Encoding via Graph Attention Networks (GAT)

Let the service mesh at time step t be represented as a directed graph G_t = (V_t, E_t), where V_t denotes N active services and E_t represents RPC/gRPC call edges. Each service node v_i ∈ V_t is associated with a feature vector x_i ∈ R^F containing metrics such as CPU saturation, memory pressure, request rate, HTTP 5xx error rate, and average latency.

To capture localized dependency weights dynamically, the attention coefficient α_{ij} between service node i and

its neighbor j ∈ N_i is computed using a shared linear transformation weight matrix W and dynamic attention layer weights.

3.3 Constrained Action Space and Safety Masking

The action space A includes parameter tuning and routing adjustments across microservices:

```
A = { No-Op, Scale-Up(v_i), Restart-Pod(v_i), Inject-Circuit-Breaker(v_i), Shift-Traffic(v_i -> v_j, ϕ) }
```

where φ ∈ (0, 1] represents the fraction of traffic rerouted away from a failing pod instance.

To prevent destructive actions—such as restarting a primary database or initiating cascading thread terminations—we enforce a Safety Mask Matrix M(s_t) ∈ {0, -∞}^{|A|}. The constrained action-value function Q_hat(s_t, a_t) is evaluated as:

```
Q_hat(s_t, a_t) = Q(s_t, a_t; θ) + M(s_t, a_t)
```

3.4 Reward Function Design

The reward function balances system stabilization against the operational costs of intervention:

```
R(s_t, a_t) = - ( w1 * L_SLA + w2 * E_Err + w3 * C_Action )
```

4. Experimental Setup & Methodology

4.1 Evaluation Environment

ResilNet was evaluated within a dedicated bare-metal cloud cluster running Kubernetes v1.30 and Istio Service Mesh v1.22.

- **Cluster Capacity:** 1,000 worker pods distributed over 50 physical compute nodes (64 cores, 256 GB RAM per node).

- **Workload Benchmark:** An open-source microservices application suite comprising 52 unique services (User Authentication, Payment Processing, Catalog Search, Inventory Management, Notification Services, and Analytics Processing).
- **Synthetic Workload Profiling:** Locust client swarms generated dynamic traffic patterns oscillating between 10,000 and 80,000 Requests Per Second (RPS) with Poisson-distributed request intervals.

4.2 Chaos Fault Injection Suite

We deployed Chaos Mesh engines to inject non-deterministic operational failure vectors across the testbed, generating over 5,000 distinct fault scenarios over a 21-day evaluation period:

1. Packet Delay & Loss: Injected 100ms - 2,000ms artificial latency and up to 25% packet loss on critical inter-service dependencies.
2. CPU Thread Starvation: Triggered hyper-threaded computational loops targeting background processing workers.
3. Memory Sinks: Allocated unmanaged heap memory at rates of 50 MB/s until out-of-memory (OOM) killer events occurred.
4. Cascading Downstream Deadlocks: Introduced artificial connection acquisition delays within database connection pools.

5. Experimental Results and Performance Analysis

We benchmarked ResilNet against three primary operational baselines: Manual SRE Execution, Native Kubernetes Standard HPA/Rules, and Unconstrained Deep Q-Networks (Standard DQN).

5.1 Recovery Latency and MTTR Analysis

Control Mechanism	Mean Time to Recover (MTTR)	P99 Latency During Recovery	SLA Violation Rate (%)	Action Success Rate (%)
Manual SRE Runbooks	18.4 minutes	4,250 ms	38.2%	94.1%
Native K8s / Rule-Based	6.2 minutes	2,810 ms	22.5%	61.2%
Standard DQN	2.1 minutes	1,120 ms	11.4%	72.8%
ResilNet (Proposed)	**12.6 seconds**	**285 ms**	**1.2%**	**98.6%**

ResilNet achieved a Mean Time to Recover of 12.6 seconds, representing a 96.6% reduction in MTTR compared to standard Kubernetes operators and an 89.8% reduction compared to unconstrained Deep Q-Learning models.

```
Mean Time to Recover (MTTR) Comparison
------------------------------------------------------------
Manual SRE Runbooks   [==================================] 18.4 min
Native K8s Rules      [============] 6.2 min
Standard DQN          [===] 2.1 min
ResilNet (Proposed)   [*] 12.6 sec
------------------------------------------------------------
```

5.2 Prevention of Secondary Cascading Outages

A critical challenge with automated mitigation agents is preventing secondary failure loops. Standard DQN models frequently trigger rapid pod restarts during database outages, creating connection storms that exacerbate downstream database saturation.

```
    Injected Failure: Database Connection Pool Exhaustion (t = 0s)
                                |
    +---------------------------+---------------------------+
    |                                                       |
    v                                                       v
[ Standard DQN Agent ]                          [ ResilNet Agent ]
  • Executed Restart-Pod on API Layer             • GAT identified DB as root dependency
  • Created 500+ fresh connection requests          • Applied Circuit Breaker to API
Layer
  • Resulted in Secondary Cascade Outage          • Dynamically throttled traffic by 40%
  • Recovery Failure (SLA Violated)               • System Stabilized in 11.2 seconds
```

By applying Graph Attention embeddings, ResilNet identified that the root bottleneck resided within the database node, rather than the caller API gateway. Consequently, it applied a dynamic rate-limiting circuit breaker on the ingress proxy, preventing service degradation while downstream connection pools normalized.

6. Discussion and Operational Guardrails

Deploying autonomous reinforcement learning controllers directly to production environments introduces specific operational risks that require systematic guardrails:

1. State Space Exploration Boundaries: Training reinforcement learning models directly on live enterprise systems is impractical due to high costs associated with explorative failures. ResilNet employs an Offline-to-Online Fine-Tuning workflow: initial policy networks are trained against synthetic digital-twin simulators before deployment to active staging environments.

2. Policy Drift under Structural Deployment Mutations: Continuous Integration / Continuous Deployment (CI/CD) software updates frequently introduce structural changes to microservice dependency graphs. To prevent policy degradation, ResilNet uses continuous graph topology extraction. When a microservice schema change is detected, the GAT layer adjusts its adjacency weights without requiring a full retraining cycle of the underlying Q-network.

7. Conclusion

In this paper, we presented ResilNet, an autonomous remediation framework for distributed cloud infrastructure that unifies Graph Attention Networks (GAT) with Safety-Constrained Deep Q-Learning. By representing multi-service telemetry as dynamic topological graph embeddings and bounding action selection using real-time safety masks, ResilNet safely automates closed-loop fault remediation without human intervention.

Our empirical evaluation on a 1,000-node Kubernetes cluster subjected to continuous chaos injection demonstrates that ResilNet reduces system MTTR to 12.6 seconds, virtually eliminates secondary cascading failures, and maintains overall SLA violation rates under 1.2%.

Future Research Directions

1. Multi-Agent Decoupled RL Systems: Assigning dedicated, lightweight edge agents to individual microservice namespaces to enable decentralized, peer-to-peer remediation negotiations.

2. LLM-Enriched Structural Context: Integrating Large Language Models (LLMs) to parse unstructured deployment logs alongside metric tensors, feeding textual context into the state representation layer.

References

1. Brockman, G., et al. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540.
2. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.
3. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. International Conference on Learning Representations (ICLR).
4. Zhou, X., et al. (2021). Fault Analysis and Automated Recovery in Cloud-Native Architectures. IEEE Transactions on Software Engineering, 47(9), 1882-1888.
5. Sridharan, S., & Subrahmanyam, P. (2020). Zero-Touch Network and Service Management: Automation and

Orchestration. IEEE Communications Magazine, 58(7), 42-48.

6. Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource management with deep reinforcement learning. ACM Workshop on Hot Topics in Networks (HotNets), pp. 50-56.