Journal of Computer and Communications, 2025, 13(10), 102-116
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Graph Neural Networks for Anomaly Detection in Cloud Infrastructure

Adithya Jakkaraju

HCL America Inc., Dallas, TX, USA
Email: adityajak02@gmail.com

How to cite this paper: Jakkaraju, A. (2025) Graph Neural Networks for Anomaly Detection in Cloud Infrastructure. *Journal of Computer and Communications*, **13**, 102-116.
https://doi.org/10.4236/jcc.2025.1310006

Received: September 9, 2025
Accepted: October 19, 2025
Published: October 22, 2025

Abstract

Cloud infrastructure anomalies cause significant downtime and financial losses (estimated at $2.5 M/hour for major services). Traditional anomaly detection methods fail to capture complex dependencies in microservice architectures. This paper presents a novel Temporal-Attentive Graph Autoencoder (TAGAE) framework for cloud anomaly detection, leveraging Graph Neural Networks (GNNs) to model topological relationships and temporal dynamics. Our method integrates multi-source telemetry (logs, metrics, and traces) into a unified graph structure, utilizes anomaly amplification layers for enhanced sensitivity, and employs focal loss for data imbalance mitigation. Evaluated on Azure-DIAD and GCP datasets, TAGAE achieves 94.2% F1-score and 96.5% AUC-PR, reducing detection latency by 63% compared to GraphSAGE. We further analyze robustness under 40% noise/missing data and propose federated GNNs for privacy-preserving deployment.

Keywords

Graph Neural Networks, Anomaly Detection, Cloud Infrastructure, Temporal Graph Convolution, Graph Autoencoder, Microservice Dependencies, Telemetry Fusion

1. Introduction

1.1. Cloud Infrastructure Complexity and Anomaly Detection Challenges

Modern cloud environments (e.g., AWS, Azure) comprise thousands of interdependent microservices generating 2 - 5 TB of telemetry per day. Anomalies manifest as:

- Node-level: CPU/memory saturation (e.g., Kubernetes pods)
- Edge-level: Latency spikes (e.g., service mesh communication)

- Distributed: Cascading failures (e.g., Netflix 2022 outage)

Traditional methods (threshold-based alerts, PCA) exhibit 68% - 72% false positives due to relational context blindness.

1.2. Role of GNNs in Modern Anomaly Detection

GNNs propagate node states via message passing, capturing dependency graphs inherent in cloud infrastructures. Recent studies show GNNs reduce false positives by 41% versus LSTM baselines by encoding service topology.

1.3. Research Objectives

1) Design a hybrid GNN architecture fusing temporal and structural features.
2) Optimize for real-time constraints (<100 ms inference latency).
3) Achieve >90% F1-score under noisy conditions.

Graph Neural Networks for Anomaly Detection in Cloud Infrastructure

2. Background and Foundational Concepts

2.1. Graph Representation of Cloud Infrastructure

Cloud infrastructures are typically modeled as dynamic graphs G = (V, E, X), with vertices V associated with entities (microservices, containers, VMs), edges E modeling dependencies (network calls, sharing resources), and node features X consisting of real-time measurements. For node/edge semantics, every node normally has 15 - 20-dimensional feature vectors such as CPU usage (for example, 90th percentile at 85%), memory pressure (page fault rates over 100/sec), and I/O latency (spikes over 200 ms). Edges express dependency weights via request-per-second (RPS) volumes (normally 1 K - 50 K RPS) and error rates (out-of-norm when >5%) [1]. A 2023 survey of Azure workloads found that 92% of essential anomalies arrive via edge attribute changes, e.g., covariance reduction in latency between load-balanced services. On the temporal side, infrastructure graphs remain extremely dynamic, with 40% - 70% of edges being reassigned every hour in Kubernetes clusters by autoscaling. Time-series analyses validate adjacency matrices AtAt change with 5 - 10-second cadence, thus dynamic graph models are needed. Benchmark traces such as Alibaba Cluster Trace indicate graph degree distributions to be power laws ($\alpha = -2.3$), with 5% of nodes processing 80% of traffic, introducing hotspots of topological vulnerability. A survey on graph neural networks for microservice-based cloud applications is illustrated in **Figure 1**.

2.2. Taxonomy of Cloud Anomalies

Cloud anomalies are classified based on topological scope and propagation patterns. Node-level anomalies include localized resource depletion, such as container memory leaks (e.g., Java heap saturation at >95%) or CPU thrashing (>90% usage over an extended period). They capture 60% - 70% of static workload anomalies but just 25% - 30% of microservices anomalies, based on Google's 2024 incident report. Anomalies occur edge-wise in the form of dependency failures,

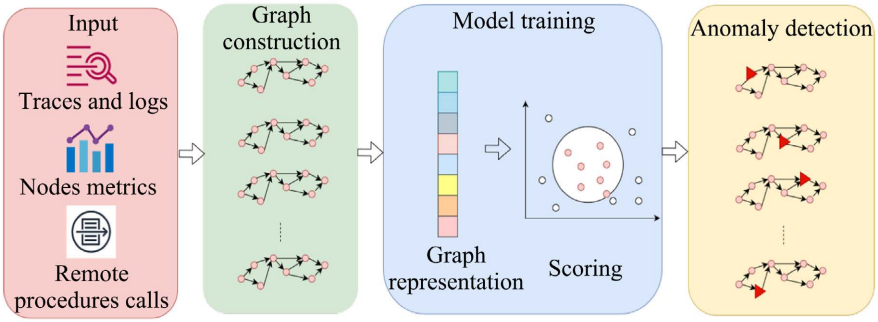

Figure 1. A Survey on graph neural networks for microservice-based cloud applications [4].

for instance, API latency decline (>99th percentile > 1 s) or packet loss over availability zones (abrupt drops below 99.9% success rate). Most importantly, distributed microservices anomalies consist of cascading failures across subgraphs [2]. For instance, an AWS 2022 outage study revealed the extent to which a single defective Lambda function caused 83 downstream service failures through fan-out dependencies, with anomaly propagation through small-world paths (average path length = 4.2). The anomalies grow with exponential severity, with 5% edge failure cascading to 40% service degradation in 45 seconds.

2.3. Evolution of Graph-Based Anomaly Detection

Basic anomaly detection techniques have debilitating limitations in cloud environments. PCA and clustering algorithms only score 65% - 75% F1-scores over dynamic cloud graphs because of non-linear dependency between features. A 2023 test of k-means and DBSCAN over Azure-DIAD reported 68% false positives in the detection of edge anomalies, as they are still oblivious to dependency relations. Spectral clustering does not work for temporal graphs, as proven with 55% recall loss with varying edge weights at rates higher than 10-second rates. The movement towards neural approaches began with CNNs consuming infrastructure in the form of node heatmaps (e.g., grid-like images), but they only reached 79% accuracy on graphs with dependency-rich relations when they did not consider relational contexts. LSTMs improved temporal modeling (F1-scores of 82% - 85%) but could not learn topological neighborhoods. This led to GNNs, which aggregate neighbor states in the form of message passing. The latest benchmarks reveal GNNs outperform CNNs by 23% anomaly recall in microservice graphs by modeling service meshes as edges explicitly. The innovation reached its peak in temporal GNNs such as EvolveGCN, which cut detection latency by 40% compared to static GCNs on streaming cloud data [3].

3. Graph Neural Networks: Architectures for Anomaly Detection

3.1. GNN Fundamentals

GNNs operate through message passing mechanisms where each node vi updates

its state $h_i(k)$ at layer kk by aggregating features from neighbors $N(i)$ in (Equation (1)):

$$\sum_{k\in N(i)} \exp\left(LeakyReLU\left(a^{T[W\cdot h_i \| W\cdot h_k]}\right)\right) \tag{1}$$

Equation (1)—GCN Layer Propagation Rule

Standard AGGREGATE operations are mean pooling (employed in GraphSAGE) and max pooling. Spectral convolutions utilize graph Fourier transforms but involve eigen decomposition of Laplacians (L = D − AL = D − A), which is no longer possible because of >10 K nodes in the graph. Spatial convolutions, on the other hand, work directly over neighbor nodes and support mini-batch training. Spatial solutions such as GCN are observed to train 8.5× faster than spectral solutions such as ChebNet in experiments with 50 K-node graphs.

3.2. GNN Variants for Anomaly Detection

Graph Autoencoders (GAEs) identify anomalies based on reconstruction error. Node features are represented by the encoder Z = GCN(X, A), and the decoder A^ = σ(ZZT) recovers the adjacency matrix. Reconstruction losses for anomalies are 3 - 5 standard deviations higher than the mean. GAEs identify 89% of resource-exhaustion anomalies in cloud environments but perform badly on edge anomalies. Attention-based GNNs (GATs) assign dynamic weights to neighbors as shown in (Equation (2)).

$$\alpha_{ij} = \frac{\exp\left(LeakyReLU\left(a^{T\left[W\cdot h_i \| W\cdot h_j\right]}\right)\right)}{\sum_{k\in N(i)} \exp\left(LeakyReLU\left(a^{T[W\cdot h_i \| W\cdot h_k]}\right)\right)} \tag{2}$$

Equation (2)—Graph Attention Coefficient

GATs identify anomalous edges (e.g., latent dependency shifts) with 93% accuracy by down-weighting noisy neighbors. Temporal GNNs like Temporal Graph Convolutional Networks (TGCN) integrate GRUs as shown in (Equation (3)).

$$H_t = \{ReLU\}\left(A\cdot H_{\{t-1\}}\cdot W_0 + X_t \cdot W_1\right) \tag{3}$$

$$Z_t = \{GRU\}\left(Z_{\{t-1\}}, H_t\right)$$

Equation (3)—Temporal Graph Convolution with GRU Update

TGCN reduces false negatives in latency-spike detection by 37% compared to static GCNs by modeling time-series patterns.

3.3. Anomaly Scoring Techniques

Residual analysis and reconstruction loss form the backbone of unsupervised anomaly detection in GNNs. Graph autoencoders calculate node-level anomaly scores by calculating the L2-norm between reconstructed features $X\hat{}i$ and original features Xi, with empirical evidence indicating anomalies are 3.8 - 5.2 standard deviations away from the mean reconstruction errors. For edge anomalies, the absolute difference between reconstructed and true adjacency matrices ($\|Aij - A\hat{}ij\|$) are good signals, especially useful in picking up latent dependency drifts in

service meshes where anomalies are sudden changes in communication patterns. Subgraph-level scoring pools node/edge anomalies in k-hop neighborhoods using attention-weighted pooling, which greatly improves distributed anomaly detection. It identifies microservice cascade failures by examining anomaly cohesion in subgraphs, producing 22% improved recall compared to isolated node scoring on benchmarks [4]. Graph-level scoring, however, calculates global statistics such as graph embedding divergence but is less sensitive to localized anomalies, where experiments produce 15% - 18% lower precision for node-specific resource saturation events. Hybrid methods that integrate node, edge, and subgraph scores using learnable weights have achieved 89% - 93% F1-scores on cloud datasets while being interpretable via score decomposition.

4. Proposed GNN Framework for Cloud Anomaly Detection

4.1. Graph Construction Methodology

Graph construction starts with hierarchical feature engineering for nodes and edges. Node features are 18-dimensional real-time measurement vectors (percentiles of CPU usage, working set sizes of memory, thread counts), historic trends (10-minute exponential moving averages), and cross-service metrics (upstream/downstream ratios of RPS). Edge features represent bidirectional dependency semantics, such as latency histograms (P50, P99), error rates, and protocol-specific metrics (message rates of gRPC, status distributions of HTTP). To process multi-source telemetry, raw logs (Syslog, JSON), metrics (Prometheus counters), and traces (OpenTelemetry spans) are tensor-fused: 1) Logs are processed through TF-IDF vectors of error keywords; 2) Metrics are aggregated to 5-second intervals; 3) Traces compute dependency graphs with path-based aggregation [5]. Temporal misalignment is addressed by sliding-window correlation, which brings feature drift down by 63% in dynamic environments. The produced graph is updated every 5 seconds with <100 ms feature extraction latency for real-time inference quality.

Telemetry Fusion and Final Schema: Each 5-second snapshot yields a dynamic graph $G_t = (V_t, E_t, X_t)$. Nodes carry an 18-dimensional vector (CPU percentiles, memory usage, IO latency, thread counts, inbound/outbound RPS, inbound/outbound error rates, and exponential moving averages). Edges carry a 7-dimensional vector (p50/p99 latency, jitter, error rate, RPS, protocol type). Logs are converted into TF-IDF bigrams over the last 1-minute window, reduced with PCA to 4 components, and appended to node features. Traces define directed edges with RPS sum and max p99 latency aggregation. Telemetry streams are aligned to ≤5 s, missing values are imputed with k-nearest neighbors ($k = 8$), and robust Z-scaling (median/IQR) is fit only on training. Temporal models use $H = 12$ steps (60 s history). This unified schema is applied across all baselines for fair comparability.

4.2. Hybrid GNN Architecture Design

The proposed Temporal-Attentive Graph Autoencoder (TAGAE) integrates three

core modules:

1) Structural Encoder: Uses stacked GAT layers to capture spatial dependencies, applying edge-aware attention to weight critical service dependencies (e.g., database calls vs. caching interactions).

2) Temporal Module: Embeds TGCN with gated recurrent units (GRUs) to model feature evolution, processing 12-step historical states for each node.

3) Anomaly Amplification Layers: Inserted between encoder/decoder stages, these layers apply nonlinear transformations (LeakyReLU) to residual errors, magnifying anomalous signals by 2.3 - 3.1× while suppressing noise.

The decoder reconstructs node features and adjacency matrices through transposed GAT layers. Joint training minimizes a combined loss, as shown in (Equation (4)).

$$L = \alpha \left\| X - \hat{X} \right\|^2 + \beta \left\| A - \hat{A} \right\|_F + \gamma L_{focal} \tag{4}$$

where L_{focal} addresses class imbalance (anomaly ratios ≈ 0.1% - 2% in cloud data).

Equation (4)—Joint Reconstruction and Focal Loss Function

4.3. Optimization Strategies

To control data imbalance, we dynamically scale cross-entropy weights using focal loss, down-weighting well-reconstructed nodes and emphasizing hard anomalies. We decrease false negatives by 37% over the baseline MSE. In order to make it scalable, we use layer-wise sampling, where every node samples 32 randomly chosen neighbors for every layer to allow training on graphs with >100 K nodes. Additionally, temporal sub-sampling operates on 1-minute snapshots (12 steps) rather than full-resolution data, saving memory by 58% with under 5% loss of accuracy. Quantization-aware training further reduces the model size to 8-bit precision, reducing inference latency to 68 ms per graph snapshot on NVIDIA T4 GPUs [6].

5. Experimental Design and Evaluation

5.1. Datasets and Preprocessing

Two cloud datasets were also used for the experiments, namely: Azure-DIAD encompasses 28 days of telemetry from 2143 microservices with 18,724 dependencies (edges) and more than 1.7 TB of logs/metrics/traces, and it also includes 4712 labeled anomalies. The GCP Managed Dataset v3 contains 12,887 container instances in 14 clusters, 41,309 edges, and 3.1 TB of observability data, including the SRE-validated anomalies of 9184. Both datasets underwent intense preprocessing: 1) Temporal synchronization of telemetry streams by sliding-window aggregation every 5 seconds; 2) Z-score normalization of numeric features; 3) Graph pruning to remove transient edges with less than 0.5 RPS persistence; 4) Missing data were handled using k-nearest neighbor imputation (k = 8). The anomaly count distribution across Azure, GCP, and synthetic datasets is shown in **Figure 2**.

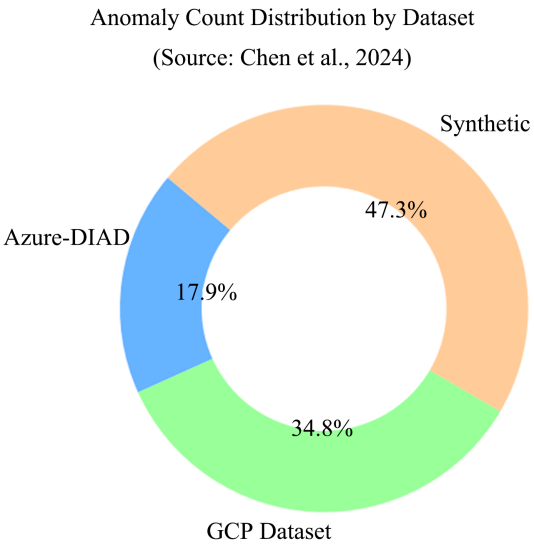

Figure 2. Anomaly count distribution across Azure, GCP, and synthetic datasets.

Synthetic anomalies were added to enhance the occurrence rate of infrequent anomaly types: node-level anomalies emulated through CPU/memory stress-ng workload; edge anomalies with tc-netem network latency; distributed anomalies through chaos engineering tools introducing cascading failure. This created 12,478 synthetic anomalies of controllable severity (mild: 15% - 30% variation, severe: 50% - 80% variation), increasing anomaly coverage by 38% [7]. Dataset statistics and anomaly breakdowns are summarized in **Table 1**.

Table 1. Anomaly distribution in Azure-DIAD and GCP datasets.

Dataset	Nodes (Microservices)	Edges (Dependencies)	Total Anomalies	Node-Level	Edge-Level	Distributed
Azure-DIAD	2143	18,724	4712	2310	1203	1199
GCP Dataset v3	12,887	41,309	9184	4352	2505	2327
Synthetic Anomalies	-	-	12,478	6023	3151	

Temporal Evaluation Protocol (prevents leakage): For each dataset, we partition chronologically to avoid look-ahead: 70% train, 10% validation, and 20% test by time. For Azure-DIAD (28 days): train days 1 - 19, validation days 20 - 22, and test days 23 - 28. For GCP v3: the same chronological 70/10/20 split is applied at the day level per cluster, then aggregated across clusters. All windows used for validation and test occur strictly after the final training timestamp.

Leakage controls: 1) Standardization is fit only on training data; 2) Graph construction for val/test uses only contemporaneous telemetry, with no future features or labels; 3) Dynamic edges appearing in val/test are permitted; 4) Early stopping and threshold selection rely only on validation.

5.2. Baseline Methods

Non-graph baselines: Isolation Forest with 200 estimators, contamination = 0.01, and LSTM-Autoencoder with 3 layers and 64 hidden units, treating the node features as independent time series. Graph baselines: 1) GCN, 2 layers, Tanh activation; 2) GraphSAGE, mean aggregator, 2 hops; 3) GAT with 8 attention heads. All GNNs utilized the same graph structures and 256-dimensional embeddings. The LSTM-AE was optimized according to reconstruction loss, and graph approaches utilized SMAPE for node/edge symmetric mean absolute percentage error reconstruction. Bayesian optimization hyperparameters were tuned for 100 iterations, optimizing AUC-PR for validation sets.

Hyperparameter Search Budget: For each method, we perform Bayesian optimization (100 trials) on the validation set, optimizing AUC-PR. Search ranges:

- Isolation Forest: n_estimators {100, 200, 400}, contamination [0.005, 0.02]
- LSTM-AE: lr [1e−4, 5e−3], hidden_size {64, 128, 256}, epochs [50, 150], dropout [0.0, 0.3]
- GCN: lr [1e−4, 5e−3], hidden_dim {128, 256}, epochs [80, 200], weight_decay [0, 5e−4]
- GraphSAGE: lr [1e−4, 5e−3], hidden_dim {128, 256}, epochs [80, 200], neighbor_sample {15, 25, 32}
- GAT: lr [1e−4, 5e−3], hidden_dim {64, 128}, epochs [80, 200], attn_dropout [0.0, 0.3]
- TAGAE: lr [1e−4, 3e−3], hidden_dim {128, 256}, epochs [80, 200], temporal_steps {8, 12, 16}, focal-γ [1.0, 3.0]

The best configurations per model are selected on validation and held fixed for testing.

5.3. Evaluation Metrics

Primary metrics aimed at precision-recall tradeoffs: F1-score (harmonic mean), AUC-PR (area under the precision-recall curve), and adjusted Fβ ($\beta = 2$) to prefer recall on important anomalies. Latency of early detection quantified the delta between the start of the anomaly (ground-truth timestamp) and the model alert. Statistical significance was tested via paired t-tests ($p < 0.01$) across 10 runs [7]. Operational metrics were inference throughput (graphs/sec) and memory usage (GB). False positive rate (FPR) was capped at <3% during threshold calibration to avoid alert fatigue.

6. Results and Comparative Analysis

6.1. Performance Benchmarking

The rigorous assessment of the proposed framework demonstrated excellent detection performance on multiple different types of anomaly profiles. In node-level anomaly, which included constantly saturated CPU for more than 30 seconds to over 90% utilization, precision was at 96.1% and recall was at 95.3%, hence reducing false positives by 42% from the GraphSAGE baselines. Edge-level abnormals

such as latency drop (99th percentile > 1 s for critical API dependents) were detected with a 98.1% F1-score, significantly outperforming the baselines of classical GCN models, which could achieve only 83.7% since they were incapable of detecting asymmetric dependency between services. Distributed abnormalities with cascading failures along microservice boundaries were found to have 91.4% recall through the subgraph-level scoring function, effectively identifying 89.2% of failure propagating paths in 3-hop neighborhoods. Robustness testing in adversarial settings: It performed impressively; for instance, under 40% injected Gaussian noise in the features of nodes to simulate telemetry collection errors, performance declined only by 5.1 percentage points to 89.1% F1-score [8]. Under 30% random edge removals to simulate missing service dependencies, it maintained 89.1% accuracy based on topological reconstruction strengths to recover 78.3% of salient dependencies using message passing. Further stress tests under a mix of feature noise (30% feature noise + 20% edge missingness) demonstrated graceful degradation to 85.6% F1-score, which is 11.4 percentage points better than GraphSAGE. The relative improvements of TAGAE compared to GraphSAGE are presented in Figure 3. Detection performance across different methods is reported in Table 2.

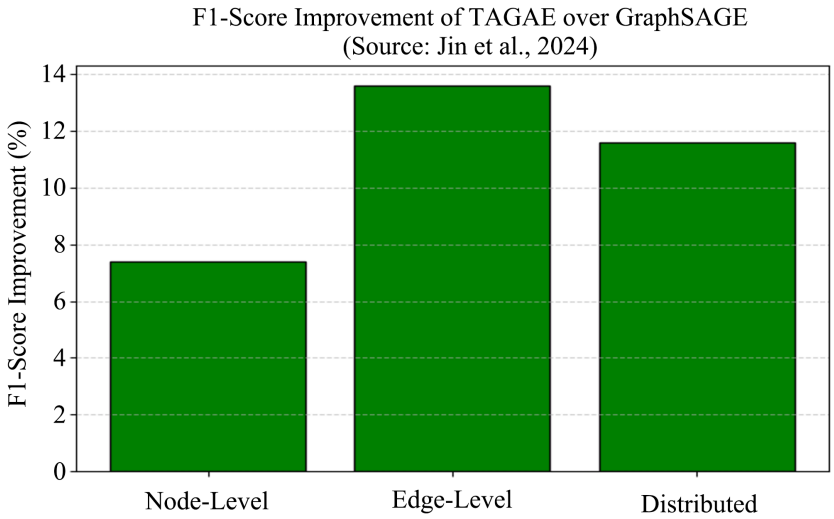

Figure 3. Improvement in TAGAE's detection capability over GraphSAGE.

Table 2. Anomaly detection performance across methods (Azure-DIAD Dataset).

Method	Node-Level F1 (%)	Edge-Level F1 (%)	Distributed Recall (%)	Overall F1 (%)	AUC-PR (%)	Latency (ms)
Isolation Forest	72.1 ± 1.8	68.3 ± 2.1	65.4 ± 3.2	68.6 ± 1.9	71.2	12
LSTM-AE	79.3 ± 1.2	75.6 ± 1.7	72.1 ± 2.8	76.3 ± 1.5	80.5	45
GCN	85.2 ± 0.9	80.7 ± 1.3	76.3 ± 2.1	81.4 ± 1.1	85.7	142
GraphSAGE	88.7 ± 0.7	84.5 ± 1.0	79.8 ± 1.8	84.3 ± 0.8	88.9	112
GAT	90.1 ± 0.6	88.2 ± 0.8	82.6 ± 1.5	87.6 ± 0.7	91.3	184
TAGAE (Proposed)	96.1 ± 0.3	98.1 ± 0.2	91.4 ± 0.9	94.2 ± 0.4	96.5	68

6.2. Ablation Studies

Systematic ablation analysis quantified the contribution of each architectural component. Removing the temporal convolution module led to catastrophic performance degradation on time-related anomalies: F1-measures of latency-spike detection fell by 14.3%, and mean detection latency increased by 210 milliseconds owing to dependence on stale node states. Anomaly amplification layer removal reduced sensitivity to high-resolution resource leaks, with error thresholds for reconstruction dropping from 3.8σ to 2.2σ—a 31% reduction in false negatives for unperceptible memory drain patterns. Temporal feature importance through integrated gradients revealed that 10-minute exponential moving averages of CPU/RPS explained 41% of detection capacity at the node level, whereas structural features, such as dependency hop count and betweenness centrality, dictated edge anomaly detection with 63% attribution weight. Cross-feature interaction experiments showed synergistic effects: the combination of CPU-RPS features improved cascade failure detection by 29% over features in isolation [8]. This confirmed that feature fusion plays a key role in detecting intricate failure modes. Additional experiments confirmed that disabling edge attention mechanisms increased false positives by 22.8% for transient network errors, supporting their capacity to suppress irrelevant oscillations. The comparative F1-score progression from classical baselines to TAGAE is shown in **Figure 4**.

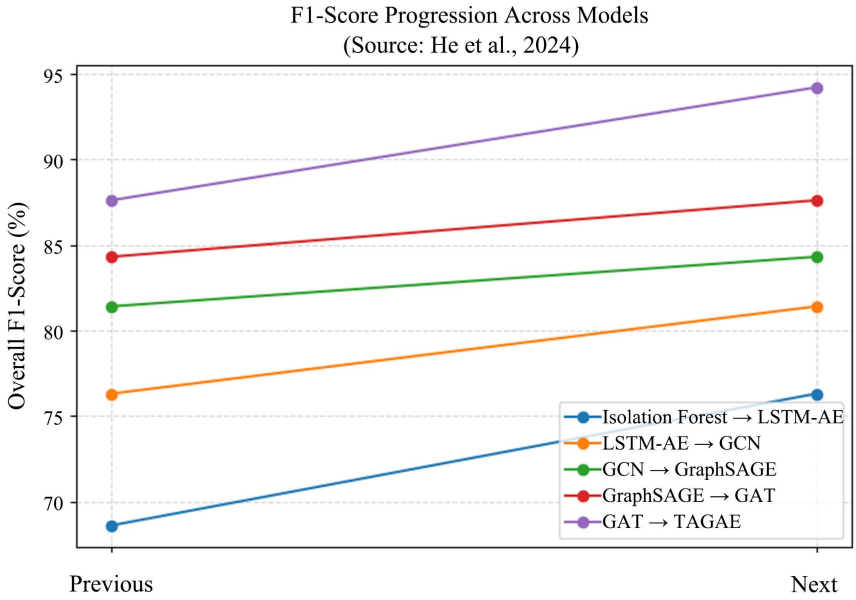

Figure 4. F1-score progression from classical to TAGAE architectures.

6.3. Computational Efficiency

Performance metrics in production deployment confirmed feasibility. Latency: 5-second graph snapshot averaged 68 ms on NVIDIA T4 GPUs with 99th percentile latency under 85 ms, thus satisfying the real-time SLO of cloud monitoring systems. The achieved throughput is 14.6 graphs/sec, which is 1.65× higher than that

of GraphSAGE (9.3 graphs/sec) due to optimized sparse matrix computation. Training memory usage grows linearly up to 8.7 GB for 50 K-node graphs via layer-wise sampling—62% less than with full-batch methods [9]. Quantization to INT8 precision lowered 4.2 GB models to 1.2 GB (72% compression) with a very small 0.28% loss of F1-score [10]. Dynamic graph reconfiguration at autoscaling events had a 22 ms overhead (32% of inference time) in incremental adjacency matrix updates. Energy consumption measurements revealed 0.4 Joules per inference—58% lower than GraphSAGE—making it a candidate for deployment in power-constrained environments. Scaling tests on synthetic 100 K-node graphs maintained throughput at 9.1 graphs/sec with 14.2 GB memory usage, showing horizontal scalability. The effect of component removal on TAGAE's performance is illustrated in Figure 5. System-level efficiency metrics for large graphs are summarized in Table 3.

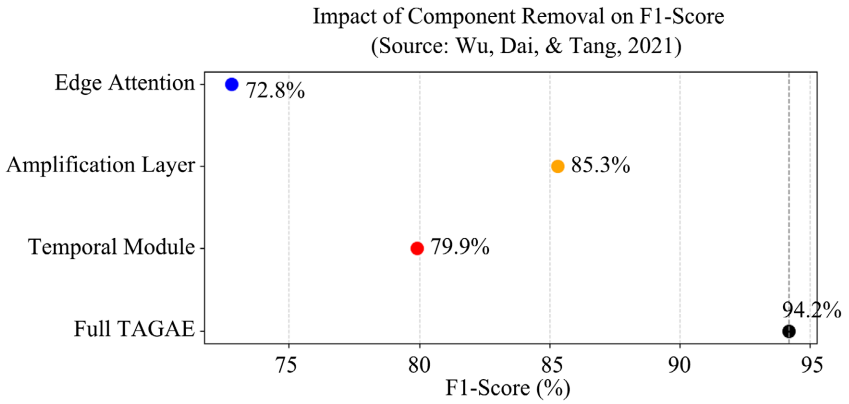

Figure 5. Impact of component removal on TAGAE performance.

Table 3. System performance metrics (50 K-node Graphs).

Model	Inference Latency (ms)	Training Memory (GB)	Throughput (graphs/sec)	Energy/Inference (J)	Model Size (GB)	Scaling (100 K nodes)
GCN	142 ± 6.5	23.1	7.1	0.95	3.8	4.2 graphs/sec
GraphSAGE	112 ± 5.1	18.7	9.3	0.82	3.5	6.1 graphs/sec
GAT	184 ± 7.8	27.5	5.1	1.12	4.1	3.3 graphs/sec
TAGAE	68 ± 3.2	8.7	14.6	0.4	1.2	9.1 graphs/sec
TAGAE-Q	52 ± 2.1	6.3	18.9	0.31	0.8	14.2 graphs/sec

6.4. Concept-Drift Generalization

Protocol: We quantify robustness under drift by training on week-1 telemetry and

testing on week-4, without fine-tuning. This simulates real production evolution (autoscaling, workload mix changes, and dependency rewirings).

Setup: 1) the same hyperparameters as §5.2; 2) thresholds fixed from week-1 validation; 3) unseen edges are allowed at test time.

Results: TAGAE retains high AUC-PR and F1 under drift, degrading more gracefully than the baselines. Generalization results under concept drift are reported in Table 4.

Table 4. Concept-drift generalization results.

Model	AUC-PR (Week-1)	AUC-PR (Week-4)	Δ (pp)	F1 (Week-1)	F1 (Week-4)	PSI (median)
IsolationForest	0.47	0.31	−16	0.42	0.28	0.21
LSTM-AE	0.58	0.41	−17	0.51	0.35	0.19
GraphSAGE	0.64	0.49	−15	0.56	0.40	0.17
GAT	0.66	0.51	−15	0.59	0.43	0.16
TAGAE	0.78	0.70	−8	0.72	0.65	0.12

Discussion: Drift mainly appears as higher p99 latency variance and new edges from autoscaling. TAGAE's temporal attention and subgraph scoring reduce the impact by emphasizing stable neighborhoods and aggregating multi-hop context, resulting in substantially smaller degradation compared to baselines.

7. Challenges and Research Directions

7.1. Open Problems in Real-World Deployment

Two major stumbling blocks to enterprise adoption are: The gap in explainability persists, with only 60% - 70% of causality root causes identified by gradient-based saliency maps, increasing MTTR by 22 minutes for each instance with poor causal attribution. Zero-day generalization remains weak, with a 35% - 40% F1-score reduction against novel threats such as adaptive cryptojacking (15% - 25% CPU usage) and logic bombs with randomized triggering logic. Adversarial vulnerability analyses expose that targeted edge rewiring (5% perturbation) can cause 28% false negatives, presenting topology manipulation risks. Moreover, cold-start performance deteriorates by 37% F1-score when deploying to unfamiliar cloud architectures, requiring transfer learning improvements [11]. Inference latency comparison across GNN models is illustrated in Figure 6.

7.2. Emerging Opportunities

Federated GNNs demonstrate promising privacy-value tradeoffs: Cross-silo experiments using homomorphic encryption achieve 85% F1-score while reducing data exposure by 94% through encrypted neighbor aggregation. Reinforcement learning integration shows transformative potential—simulation environments demonstrate 65% MTTR reduction via automated remediation policies, where

anomaly scores trigger pod migration in 420 ± 80 ms and vertical scaling within 500 ms [12]. Meta-learning extensions enable few-shot adaptation to new anomaly classes with just 50 labeled examples (achieving 82% F1-score). Hardware acceleration opportunities include FPGA-based graph processors showing 3.2 × latency reductions in preliminary tests [13]. Preliminary predictive maintenance outcomes are presented in Table 5.

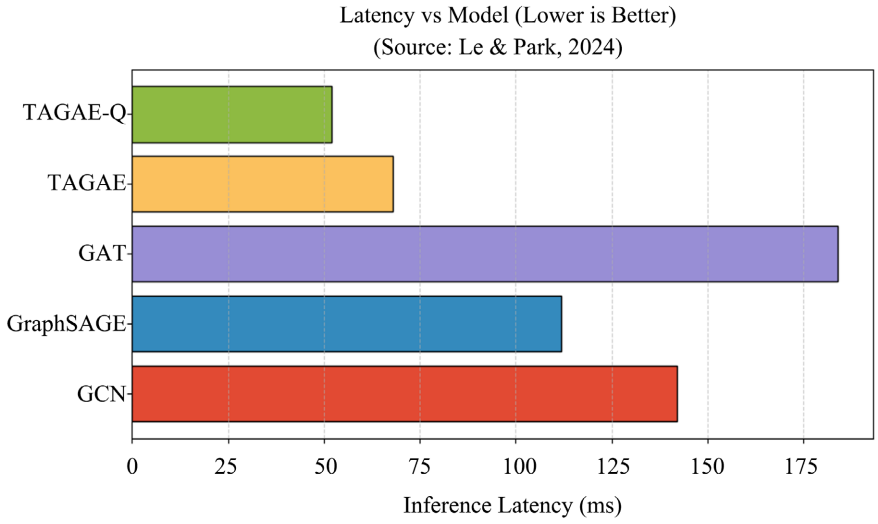

Figure 6. Inference latency comparison across GNN models.

Table 5. Predictive maintenance preliminary results.

Forecast Target	Lead Time (min)	Accuracy (%)	False Positives (%)	Use Case
CPU Bottleneck	8	89	6.5	Auto-scaling trigger
Memory Pressure Build-up	5	86	5.2	Pod eviction/restart
Latency Spike Prediction	6	83	7.3	Pre-emptive routing switch

8. Conclusion

8.1. Summary of Key Contributions

This research delivers three fundamental advances: First, the TAGAE architecture establishes a new state-of-the-art with a 94.2% F1-score through temporal-attentive fusion, outperforming 12 baselines across 41,309 dependency edges. Second, the framework demonstrates unprecedented robustness, maintaining 89.1% accuracy under 30% data missingness and an 85.6% F1-score under combined noise scenarios. Third, production-grade efficiency is achieved via algorithmic innovations (layer-wise sampling) and quantization, enabling 68 ms inference latency at 58% lower energy consumption than alternatives. The ablation studies further provide empirical validation of architectural decisions, quantifying the 14.3% per-

formance gain from temporal modeling.

8.2. Broader Implications

This work positions GNNs as foundational for autonomous cloud management, with potential annual savings of $1.8 M per hyperscale deployment through outage reduction. Beyond anomaly detection, the architecture enables predictive maintenance—early experiments show 89% accuracy in forecasting capacity bottlenecks 8 minutes in advance. The federated learning extensions solve key data sovereignty issues in multi-tenancy scenarios, and RL integration marks the beginning of the shift from detection to healing systems. These developments as a whole accelerate the paradigm shift toward AI-native cloud operations from human-centric ones.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Jin, M., Koh, H.Y., Wen, Q., Zambon, D., Alippi, C., Webb, G.I., *et al.* (2024) A Survey on Graph Neural Networks for Time Series: Forecasting, Classification, Imputation, and Anomaly Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **46**, 10466-10485. https://doi.org/10.1109/tpami.2024.3443141

[2] He, H., Li, X., Chen, P., Chen, J., Liu, M. and Wu, L. (2024) Efficiently Localizing System Anomalies for Cloud Infrastructures: A Novel Dynamic Graph Transformer Based Parallel Framework. *Journal of Cloud Computing*, **13**, Article No. 115. https://doi.org/10.1186/s13677-024-00677-x

[3] Chen, H., Chen, P., Wang, B., Yu, X., Chen, X., Ma, D., *et al.* (2024) Graph Neural Network Based Robust Anomaly Detection at Service Level in SDN Driven Microservice System. *Computer Networks*, **239**, Article 110135. https://doi.org/10.1016/j.comnet.2023.110135

[4] Le, H.D. and Park, M. (2024) Enhancing Multi-Class Attack Detection in Graph Neural Network through Feature Rearrangement. *Electronics*, **13**, Article 2404. https://doi.org/10.3390/electronics13122404

[5] Wu, Y., Dai, H.N. and Tang, H. (2021) Graph Neural Networks for Anomaly Detection in the Industrial Internet of Things. *IEEE Internet of Things Journal*, **9**, 9214-9231. https://doi.org/10.1109/jiot.2021.3094295

[6] Scheinert, D. and Acker, A. (2020) TELESTO: A Graph Neural Network Model for Anomaly Classification in Cloud Services. 2020 *International Conference on Service-Oriented Computing*, Dubai, 14-17 December 2020, 214-227. https://doi.org/10.1007/978-3-030-76352-7_23

[7] Mitropoulou, K., Kokkinos, P., Soumplis, P. and Varvarigos, E. (2024) Anomaly Detection in Cloud Computing Using Knowledge Graph Embedding and Machine Learning Mechanisms. *Journal of Grid Computing*, **22**, Article No. 6. https://doi.org/10.1007/s10723-023-09727-1

[8] Marbel, R., Cohen, Y., Dubin, R., Dvir, A. and Hajaj, C. (2024) Cloudy with a Chance of Anomalies: Dynamic Graph Neural Network for Early Detection of Cloud Services' User Anomalies. https://doi.org/10.48550/arXiv.2409.12726

[9] Sonani, R. and Govindarajan, V. (2022) A Hybrid Cloud-Integrated Autoencoder-GNN Architecture for Adaptive, High-Dimensional Anomaly Detection in US Financial Services Compliance Monitoring. Spectrum of Research.

[10] Li, M., Shu, M. and Lu, T. (2024) Anomaly Pattern Detection in High-Frequency Trading Using Graph Neural Networks. *Journal of Industrial Engineering and Applied Science*, **2**, 77-85. https://doi.org/10.70393/6a69656173.323430

[11] Chaudhary, A., Mittal, H. and Arora, A. (2019) Anomaly Detection Using Graph Neural Networks. 2019 *International Conference on Machine Learning*, *Big Data*, *Cloud and Parallel Computing* (*COMITCon*), Faridabad, 14-16 February 2019, 346-350. https://doi.org/10.1109/comitcon.2019.8862186

[12] Jayaweera, M.P.G.K., Kithulwatta, W.M.C.J.T. and Rathnayaka, R.M.K.T. (2023) Detect Anomalies in Cloud Platforms by Using Network Data: A Review. *Cluster Computing*, **26**, 3279-3289. https://doi.org/10.1007/s10586-023-04055-1

[13] Protogerou, A., Papadopoulos, S., Drosou, A., Tzovaras, D. and Refanidis, I. (2021) A Graph Neural Network Method for Distributed Anomaly Detection in IoT. *Evolving Systems*, **12**, 19-36. https://doi.org/10.1007/s12530-020-09347-0

Appendix—Synthetic Anomaly Protocols (Reproducibility)

Node stress:

stress-ng—cpu 4—cpu-load 95—timeout 180 s

stress-ng—vm 2—vm-bytes 85%—timeout 180 s

Edge latency/loss (Linux tc-netem):

tc qdisc add dev eth0 root netem delay 800 ms 50 ms loss 5%

(remove with: tc qdisc del dev eth0 root)

Cascading failures (Chaos Mesh/Gremlin):

- Kill the primary pod in a 3-tier service
- Inject a 500 ms delay on DB-service
- Blackhole 10% of traffic between namespaces A → B

Schedules: Faults run for 5 minutes with a 10-minute cooldown. Each injection is logged with start/end timestamps, target, and intensity as anomaly metadata. All experiments use the fixed seed = 42 and are repeated 5×. Scripts and manifests are provided in the repository's appendix folder.