



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IX **Month of publication:** September 2026

DOI: <https://doi.org/10.22214/ijraset.2026.84984>

www.ijraset.com

Call: ☎ 08813907089

E-mail ID: ijraset@gmail.com

Adapting and Validating GNN Fault Forecaster with Real-World Cloud Traces

Punuri Bala Mohan Varma¹, Dr. Palagiri Radhika Raju², Dr. Akepogu Ananda Rao³

¹M. Tech, ²Assistant Professor, ³Professor, Department of Computer Science and Engineering, Jawaharlal Nehru Technological University Anantapur College of Engineering (Autonomous), Ananthapuramu-515002, Andhra Pradesh, India

Abstract: *Modern cloud applications are built using microservices that support various sectors such as healthcare, finance, e-commerce and communication. Microservices' distributed and scalable nature provides high performance and flexibility but also increases system complexity. When failures occur, they propagate rapidly through interconnected microservices, leading to performance degradation, service downtime and financial losses. Traditional fault detection approaches mainly depend on threshold-based rules and reactive fault tracing, which are time-consuming, difficult to scale and often fail to proactively detect faults in dynamic service environments. Existing research has proposed a Graph Neural Network (GNN)-based framework that models microservice interactions as graphs for fault prediction and critical component identification. The framework, trained on simulated data generated using Markov Decision Processes (MDPs), has demonstrated strong performance in controlled environments. However, its use of synthetic data limits its effectiveness on real-world cloud systems, which exhibit unpredictable behaviour and dynamic workloads. The proposed work addresses this limitation using the Alibaba Cluster Trace – Microservices v2021 dataset collected from a real production environment. This work incorporates an Edge-Attention Graph Convolutional Network (EA-GCN), a hybrid loss function and Critical Node Removal Analysis (CNRA) to improve fault prediction and identify critical microservices. Experimental results demonstrate that the proposed approach achieves an F1-score of 0.7265 and an AUC-ROC of 0.9583, improving F1-score by 47.9% over plain GCN. CNRA identifies the most critical microservices that contribute to system-wide fault risk, prioritizing fault mitigation. The results show that the framework is effective for fault prediction and critical microservice identification in real-world cloud environments.*

Keywords: Cloud computing, Microservices, Fault prediction, Graph Neural Networks, Cloud trace data, Fault forecasting, Service reliability.

I. INTRODUCTION

Modern cloud applications are built as independently deployable microservices that communicate via synchronous Remote Procedure Calls (RPCs) and asynchronous message brokers. This architecture improves scalability, deployment agility, and organizational independence but introduces fault propagation. When a microservice degrades—due to resource conflicts, network isolation, or software errors—dependent microservices suffer cascading latency that can trigger failures, potentially causing an entire application to fail within seconds.

Traditional monitoring approaches like Prometheus and Nagios use fixed metric thresholds to monitor each microservice independently. However, this approach has two main limitations. First, it is inherently reactive, as alerts are triggered only after thresholds are crossed, by which time fault propagation may already be underway. Second, it ignores structural dependencies between microservices through which faults propagate. A service may show normal CPU usage but still be near failure due to upstream congestion, which threshold-based monitoring cannot detect.

Graph Neural Networks (GNNs) provide an efficient way to analyse service structures in microservice systems. By modelling microservices as graph nodes and call dependencies as edges, GNNs propagate fault-relevant signals across the graph and capture structural relationships ignored by threshold-based methods. Unyi et al. [1] showed that GNN-based fault probability prediction is theoretically feasible, achieving $R^2 = 0.992$ on tree-structured service meshes simulated using Markov Decision Processes (MDPs). However, this evaluation is based entirely on synthetic data, as acknowledged by the authors: “Although the current evaluation is based on synthetic datasets to ensure full control and formal validation, real-world applicability remains a priority” [1]. No prior work has validated GNN-based fault forecasting on real production cloud traces.

A. Contributions

This paper makes the following contributions:

- 1) EA-GCN: An Edge-Attention GCN that includes real inter-microservice call frequency as edge weights in the message-passing operation, which enables the model to focus on high-traffic dependency paths where faults propagate most aggressively and quickly.
- 2) Hybrid Loss: A loss function $L = L_{\text{BCE}} + \alpha \cdot L_{\text{smooth}}$ that combines Binary Cross-Entropy (BCE) with a graph smoothness term penalizing inconsistent fault probability estimates between connected microservices ($\alpha = 0.1$).
- 3) Critical Node Removal Analysis (CNRA): CNRA removes each microservice from the GNN graph one at a time, then evaluates the system-wide fault risk on the reduced GNN graph and computes an impact score for each microservice. Based on these impact scores, one can understand how much risk each microservice poses to the microservice system.
- 4) Real-World Validation: The proposed GNN fault forecaster is tested using the Alibaba Cluster Trace — Microservices v2021 [5], which consisted of 2,000,000 real MSCallGraph records and 2,000,000 MSResource records from a real production environment running with 6,466 unique microservices.

The research paper is structured as follows: Section II reviews related work on fault detection and GNN-based methods in cloud microservice environments. Section III describes the research methodology. Section IV presents the experimental setup. Section V covers the results. Section VI discusses the impact of edge attention, graph smoothness regularization and limitations of the proposed work. Section VII presents the conclusions and future work.

II. RELATED WORK

A. Graph Neural Networks for Fault Detection

Kipf and Welling [2] introduced Graph Convolutional Networks (GCN), establishing the theoretical foundation for learning on graph-structured data through normalized adjacency-weighted aggregation. Graph Attention Networks (GAT) [3] extend this approach by introducing learnable attention coefficients over neighbour aggregation, enabling models to differentially weight the contributions of neighbouring nodes. The EA-GCN layer architecture builds directly on both formulations: GCNConv layers for edge-frequency-weighted propagation and GATConv layers for adaptive neighbour attention.

Maroudis et al. [9] applied GNNs to SLA violation prediction in cloud environments, demonstrating that graph-structured representations significantly outperform time-series baselines for capturing associativity between services. MicroRCA, an attributed graph model for root cause localization that connects system resource usage and application performance problems, was first presented by Wu et al. [10]. Gan et al. [11] proposed Sleuth, an unsupervised GNN for trace-based root cause analysis using a trace distance metric for clustering. These works are complementary to the proposed work: they focus on localization after failure onset, while EA-GCN targets proactive prediction before failure occurs.

B. The Base Paper: Unyi et al. [1]

The most directly related work is Unyi et al. [1], which proposed a GNN framework to predict fault probability in cloud service meshes. Their approach models service topologies as directed acyclic graphs (DAGs) and employs a 4-layer message-passing GNN trained on synthetic data generated by the PRISM model checker using Markov Decision Processes. The model performs graph-level regression — predicting a single system-wide fault probability per graph — achieving $R^2 = 0.992$ (tree) and $R^2 = 0.996$ (DAG) on synthetic datasets. An explainability mechanism identifies critical subgraphs using a learned node masking approach trained with the REINFORCE algorithm.

The proposed work extends the existing research in three critical areas: (1) synthetic MDP simulation is replaced with 2 million real Alibaba production records; (2) graph-level regression is replaced with node-level classification, enabling per-service fault localization; and (3) edge-frequency attention and a hybrid loss function are introduced to address real-trace characteristics such as heterogeneous call volumes and asymmetric dependency weights.

C. Microservice Fault Detection

Zhou et al. [12] proposed MEPFL for latent error prediction using system trace logs, combining trace-level and microservice-level learning. Rouf et al. [13] introduced InstantOps, a multi-modal framework that fuses logs, metrics, traces, and events using GNN+GRU architectures for spatiotemporal failure modelling. Both are reactive systems — requiring observable fault signatures — whereas EA-GCN predicts fault probabilities from system structure and resource metrics before failures occur.

III. METHODOLOGY

A. Dataset: Alibaba Cluster Trace — Microservices v2021

The Alibaba Cluster Trace — Microservices v2021 [5], released by Alibaba Group at ACM SoCC 2021 and publicly available through Alibaba Open Trace [6], is used in this study. The trace was collected from a production cluster running over 10,000 bare-metal nodes during a 12-hour operational window. Two tables are processed:

- **MSCallGraph:** Records every inter-service RPC call with fields: timestamp, interface ID, response time $rt \in (0,1]$ (normalized), downstream service dm , upstream service um , and $call_type$. A total of 2,000,000 rows are parsed from MSCallGraph_0.csv through MSCallGraph_4.csv, yielding 12,630 unique caller-callee pairs across 6,466 distinct service identifiers.
- **MSResource:** Records per-instance CPU and memory utilization: $msname$, $msinstanceid$, $nodeid$, $instance_cpu_usage \in [0,1]$, $instance_memory_usage \in [0,1]$, timestamp at 30-second intervals. A total of 2,000,000 rows are parsed from MSResource_0.csv through MSResource_3.csv, covering 1,303 unique service identifiers.

The top 30 microservices by call-graph connectivity (degree centrality) are selected as graph nodes. Thirteen of the 30 microservices have direct MSResource entries providing real CPU and memory values; for the remaining 17, CPU and memory values are estimated from response-time data available in MSCallGraph using a linear regression model trained on microservices that have both MSResource and MSCallGraph data ($R^2 = 0.84$).

B. Graph Construction

The microservice system is modelled as a directed attributed graph $G = (V, E, X, W)$, where $W \in \Psi^{\wedge}\{60\}$ is the edge weight vector, $X \in \Psi^{\wedge}\{30 \times 7\}$ is the node feature matrix, V is the set of 30 service nodes, and E is the set of 60 directed call dependency edges (top-60 by call count).

Edge weights encode normalized call frequency:

$$w_{\{ij\}} = count(i \rightarrow j) / \max_{\{(a,b) \in E\}} count(a \rightarrow b)$$

This formulation captures the actual volume of inter-service communication from production traffic, providing a physically grounded prior for fault propagation: services connected by high-frequency calls share stronger fault coupling. The adjacency is symmetrized with directional asymmetry: $A_{\{ij\}} = w_{\{ij\}}$, $A_{\{ji\}} = 0.5 \cdot w_{\{ij\}}$, reflecting the asymmetric nature of caller-callee relationships. Self-loops are added before normalization.

Node features are constructed from both MSCallGraph and MSResource as shown in Table I.

Table I
Node Feature Vector — 7 Dimensions

	Feature	Source	Description
f_1	CPU Usage (%)	MSResource	Mean $instance_cpu_usage \times 100$
f_2	Memory Usage (%)	MSResource	Mean $instance_memory_usage \times 100$
f_3	Error Rate (%)	MSCallGraph	% Calls with $rt > P95(rt)$
f_4	Latency (ms)	MSCallGraph	Mean response time $\times 1000$
f_5	Request Rate	MSCallGraph	Call count / 100 (RPS proxy)
f_6	Call Count	MSCallGraph	Total calls in 2M-row window
f_7	Degree Centrality	Computed	Node degree / max degree in G

C. Edge-Attention GCN Architecture

An Edge-Attention GCN with three learnable stages is proposed. The key architectural innovation over standard GCN [2] and over the base paper [1] is the explicit encoding of call-frequency edge weights into the message-passing operation.

Layer 1 — Edge-Weighted Graph Convolution: The first layer applies GCNConv [2] with edge weights W incorporated into the normalized adjacency matrix, replacing the standard binary adjacency:

$$H^1 = \text{ReLU}(\text{BN}(\tilde{D}^{-\frac{1}{2}} \tilde{A}_W \tilde{D}^{-\frac{1}{2}} X \Theta_i))$$

where $\tilde{A}_W$ incorporates edge weights W , $\tilde{D}$ is the degree matrix of $\tilde{A}_W$, $\Theta_i \in \mathbb{R}^{7 \times 64}$ is a learnable weight matrix, and BN denotes batch normalization. This layer maps the 7-dimensional input features to a 64-dimensional latent representation, with the message from each neighbour scaled by the corresponding call-frequency weight.

Layer 2 — Graph Attention: A 4-head Graph Attention Network layer [3] enables adaptive neighbour weighting beyond the fixed call-frequency prior:

$$H^2 = \text{ReLU}(\text{GATConv}(H^1, E; \text{heads}=4, \text{out}=16)) \in \mathbb{R}^{N \times 64}$$

Each attention head learns a distinct weighting of neighbour contributions. With 4 heads and 16 units per head, the output dimension is 64. Dropout ($p=0.3$) is applied within the attention computation.

Layer 3 — Skip-Connected GCN: A residual skip connection from the input is concatenated with H^2 before the final graph convolution, preventing gradient vanishing and preserving the original metric values:

$$H^3 = \text{ReLU}(\text{GCNConv}(\text{BN}([H^2 \parallel \text{skip}(X)])))$$

where $\text{skip}(\cdot)$ denotes a linear projection $X \rightarrow \mathbb{R}^{N \times 64}$. The concatenation yields a 128-dimensional input to GCNConv, whose output is 64-dimensional.

Output Layer: A two-layer MLP with dropout ($p=0.3$) maps H^3 to per-node fault probabilities:

$$\hat{p}_i = \sigma(\text{MLP}(H^3[i])) \in [0, 1]$$

where σ denotes the sigmoid function. The MLP architecture is Linear ($64 \rightarrow 32$) $\rightarrow$ ReLU $\rightarrow$ Dropout (0.3) $\rightarrow$ Linear ($32 \rightarrow 1$) $\rightarrow$ Sigmoid.

D. Hybrid Loss Function

Standard binary cross-entropy treats each node independently, ignoring graph topology during training. A hybrid loss function is introduced that augments BCE with a graph smoothness regularization term:

$$L = L_{\text{BCE}} + \alpha \cdot L_{\text{smooth}}$$

$$L_{\text{smooth}} = (1/|E|) \sum_{(i,j) \in E} (\hat{p}_i - \hat{p}_j)^2$$

The smoothness term encourages similar fault probabilities between directly connected microservices, encoding the domain prior that fault propagates along call dependencies. The hyperparameter α is set to 0.1 via grid search on the validation F1-score. Gradient clipping at norm 1.0 is applied to stabilize training.

E. Fault Scenario Generation

The Alibaba trace did not provide ground-truth fault labels — microservice names are anonymized and incident records were not included in the public release. The study generated 300 supervised training scenarios through fault injection across six categories, as summarized in Table II.

Table II
Fault Injection Scenarios

Scenario	Injection Method	Fault Label Condition
Normal	Gaussian noise $N(0, 0.02)$	None
CPU Spike	$\text{CPU} \times U(2.5, 4.0)$	Injected services
Memory Leak	$\text{Memory} \times U(2.0, 3.5)$	Injected services
Latency Spike	$\text{Latency} \times U(3.0, 8.0)$	Injected services
Error Surge	$\text{Error Rate} \times U(5.0, 15.0)$	Injected services
Cascade	BFS from root, decay factor 0.8 per hop	Root + downstream if factor > 0.4

For cascading scenarios, the fault is seeded at a randomly selected root node and propagated through the real MSCallGraph call topology using breadth-first search with exponential decay. The propagation factor at depth d is: $\phi_d = U(0.3, 0.7) \times 0.8^d$. A downstream service receives a fault label if $\phi_d > 0.4$. The dataset is split into 80% training and 20% validation sets with stratification on fault scenario type.

F. Critical Node Removal Analysis (CNRA)

Given a trained EA-GCN model, CNRA quantifies each service's marginal contribution to system-wide fault risk. Let $\hat{p}(G)$ denote the vector of fault probabilities on graph G . Define baseline system risk:

$$R_{base} = (I/|V|) \sum_i \hat{p}_i(G)$$

For each service $v \in V$, construct reduced graph $G_v = (V \setminus \{v\}, E_v)$ where E_v removes all edges incident to v . The GNN is re-evaluated on G_v to obtain R_v and $\max_{\hat{p}} \hat{p}_v$.

The impact score is:

$$Impact(v) = (R_{base} - R_v) + 0.5 \cdot (\max_{\hat{p}} \hat{p}_{base} - \max_{\hat{p}} \hat{p}_v)$$

Services are ranked by $Impact(v)$ in descending order. A positive $Impact(v)$ indicates that removing v reduces system risk — v is a fault-contributing 'weak link'. A negative $Impact(v)$ indicates that v acts as a traffic distributor — its removal concentrates load on remaining services, increasing overall risk. CNRA requires $|V| = 30$ independent GNN forward passes and completes in under 2 seconds. The implementation also includes cascade failure simulation using BFS propagation from each failed node with probability increase $\hat{p}'_d = \min(0.99, \hat{p}_d \times (1 + 0.3/\text{step_depth}))$.

IV. EXPERIMENTAL SETUP

A. Implementation Details

EA-GCN is implemented in PyTorch 2.0 with PyTorch Geometric 2.5 [7]. Training uses the Adam optimizer with $lr = 0.001$ and $weight_decay = 1 \times 10^{-4}$, ReduceLROnPlateau scheduling with $patience = 5$ and $decay_factor = 0.5$, and 50 training epochs on an 80/20 train/validation split. Total trainable parameters are approximately 23,000. The Random Forest baseline uses scikit-learn [8] with $n_estimators = 100$. All experiments run on a single workstation (Windows 11, Python 3.11).

B. Baselines

EA-GCN is compared against three baselines spanning the spectrum from rule-based to machine learning to graph-aware:

- **Threshold (CPU > 70%):** Flags a microservice as faulty if $cpu_usage > 0.70$ OR $error_rate > 0.30$. No training. Represents conventional monitoring tools (Prometheus, Nagios).
- **Random Forest (RF):** 100-tree ensemble classifier on the 7-dimensional node feature vector. No graph structure. Represents the best achievable non-graph ML performance.
- **Plain GCN:** A standard two-layer GCN [2] without edge attention, skip connections, or hybrid loss, trained with BCE only. This configuration isolated the contribution of the proposed architectural innovations.

C. Evaluation Metrics

Accuracy, Precision, Recall, F1-Score and AUC-ROC are reported. F1 is the primary evaluation metric because it strikes a balance between minimizing false alarms (precision) and detecting as many faults as possible (recall). AUC-ROC measures model discrimination capability regardless of operating threshold. A service is labelled fault = 1 when its injected fault probability exceeds 0.5.

V. RESULTS

A. Classification Performance

Table III presents the comparative evaluation across all methods on the Alibaba 2021 test set (20% of 300 scenarios = 60 scenarios, 1,800 node-label pairs).

Table III
Performance Comparison on Alibaba Cluster Trace v2021

Method	Acc.	Prec.	Rec.	F1	AUC-ROC
Threshold (CPU>70%)	0.8878	0.3090	0.6372	0.4162	0.5000
Random Forest [4]	0.9900	0.9796	0.8571	0.9143	0.9853
Plain GCN [2]	0.9517	0.8077	0.3529	0.4912	0.9080
EA-GCN (proposed)	0.9644	0.7391	0.7143	0.7265	0.9583

1) Threshold Baseline

The threshold approach achieves AUC-ROC = 0.5000 — statistically equivalent to random classification — confirming that single-metric thresholds cannot capture the multivariate, topology-dependent fault patterns present in real Alibaba production traces. Its high recall (0.6372) comes at the cost of extremely low precision (0.3090), meaning 69% of its fault alerts are false positives — operationally intolerable in production environments.

2) Random Forest

RF achieves the highest accuracy (0.9900) and F1 (0.9143) among all methods by learning complex non-linear feature interactions across the 7-dimensional feature space. This result confirms that the node features contain significant fault-predictive signal. However, RF is fundamentally incapable of modelling fault propagation: it treats each service as an independent sample, ignoring the call-graph structure through which faults cascade. In real cascading failure scenarios, the GNN's structural awareness is essential for identifying at-risk downstream services before they exhibit observable metric degradation.

3) Plain GCN

The Plain GCN improves over the threshold baseline in accuracy (0.9517 vs 0.8878) and AUC-ROC (0.9080 vs 0.5000), demonstrating that the graph structure provides genuine predictive value. However, its recall of 0.3529 is concerningly low — the model misses 65% of actual fault cases. This weakness stems from treating all service dependencies equally: without call-frequency weighting, the model cannot distinguish critical high-traffic dependencies from low-frequency background calls, leading to diffuse and imprecise fault probability estimates.

4) EA-GCN (Proposed)

Among the GNN-based approaches, EA-GCN achieves the highest recall (0.7143) and AUC-ROC (0.9583), resulting in a 47.9% improvement in the F1-score over Plain GCN (0.7265 vs. 0.4912). The call-frequency edge weights enable the model to concentrate message passing on the high-traffic dependency paths along which fault effects propagate most aggressively in production systems. Training loss decreases from 0.1270 (epoch 10) to 0.0627 (epoch 50), with validation F1 improving from 0.6564 to 0.7512, demonstrating stable, monotonic convergence. The best model is saved at epoch 50 with validation loss 0.1042.

5) EA-GCN vs. Random Forest

RF outperforms EA-GCN in overall accuracy and F1-score, which may appear to undermine the value of the graph approach. However, this comparison is not scientifically equivalent: RF operates on node features at inference time and has no causal model of fault propagation.

When evaluated specifically on cascading fault scenarios — where upstream service failures drive downstream degradation — RF's inability to propagate fault signals through the call graph leads to systematic misclassification of downstream services that have not yet exhibited observable metric anomalies. EA-GCN's graph-aware predictions identify these pre-failure states, which is precisely the proactive capability required for production fault management.

B. Critical Node Removal Analysis

Table IV presents the top 5 critical services identified by CNRA on the Alibaba 2021 trace under the normal operational scenario (baseline system risk $R_{base} = 0.4421$).

Table IV
Top-5 Critical Nodes — CNRA Results ($R_{base} = 0.4421$)

Rank	Service	Type	Fault Prob.	Risk	Edges
1	ApiGateway-02	api-gateway	64.4%	↓ 0.742%	5
2	ApiGateway-03	api-gateway	54.8%	↓ 0.413%	10
3	Worker-02	worker	52.0%	↓ 0.314%	3
4	Database-04	database	46.9%	↓ 0.140%	9
5	ApiGateway-01	api-gateway	45.8%	↓ 0.100%	4

ApiGateway-02 — with the highest fault probability (64.4%) and 5 outgoing call-dependency edges — is identified as the most critical service: its remediation reduces average system fault probability by 0.742 percentage points. The ranking reveals a power-law distribution of criticality: the top-5 services collectively account for 1.709% aggregate risk reduction ($0.742 + 0.413 + 0.314 + 0.140 + 0.100$), suggesting that targeting fewer than 17% of services can achieve disproportionate system-wide risk mitigation.

Notably, services ranked 28–30 exhibit negative Impact(v) scores, meaning their removal increases system risk. ApiGateway-04 (rank 30, Impact = -0.577%) has 5 outgoing edges and acts as a traffic distributor: its removal concentrates load on alternative paths, elevating fault probabilities in dependent services.

Unknown-01 (rank 29, 14 edges, Impact = -0.136%) is the most connected service in the graph and serves as a de facto load balancer. These findings have a direct operational implication: naive node removal based solely on individual fault probability — without accounting for structural role — can degrade system resilience.

VI. DISCUSSION

A. Why Edge Attention Helps

The 47.9% F1 improvement of EA-GCN over Plain GCN can be attributed to the heterogeneous call volume distribution in real Alibaba traces. In MSCallGraph, a small number of service connections handle most of the communication between microservices, while many other connections are used much less frequently. Plain GCN, using binary adjacency, treats a pair with 250,000 calls identically to a pair with 100 calls. EA-GCN's frequency-weighted message passing amplifies signals from high-traffic dependencies, which are precisely the paths along which fault effects propagate in production systems. This result suggests that edge attribute encoding is essential for GNN fault forecasting on real heterogeneous traces.

B. Graph Smoothness Regularization

The hybrid loss provides an implicit form of domain knowledge injection: connected services should exhibit correlated fault probabilities, reflecting the physical reality that faults propagate through call dependencies. Without the smoothness term ($\alpha = 0$), the model assigns disconnected fault probabilities to adjacent services. With $\alpha = 0.1$, predictions reflect the call graph topology, improving calibration for cascading fault scenarios.

C. Limitations

This work has three primary limitations. First, the 30-service graph represents 0.5% of the 6,466 services in the full Alibaba trace — scaling to the complete graph requires mini-batch GNN training with graph sampling. Second, fault labels are generated through injection simulation rather than real Alibaba incident records, as the public trace does not include failure annotations. Third, the model processes static graph snapshots — temporal dynamics such as gradual memory leaks require spatiotemporal GNN architectures (e.g., T-GCN, STGNN).

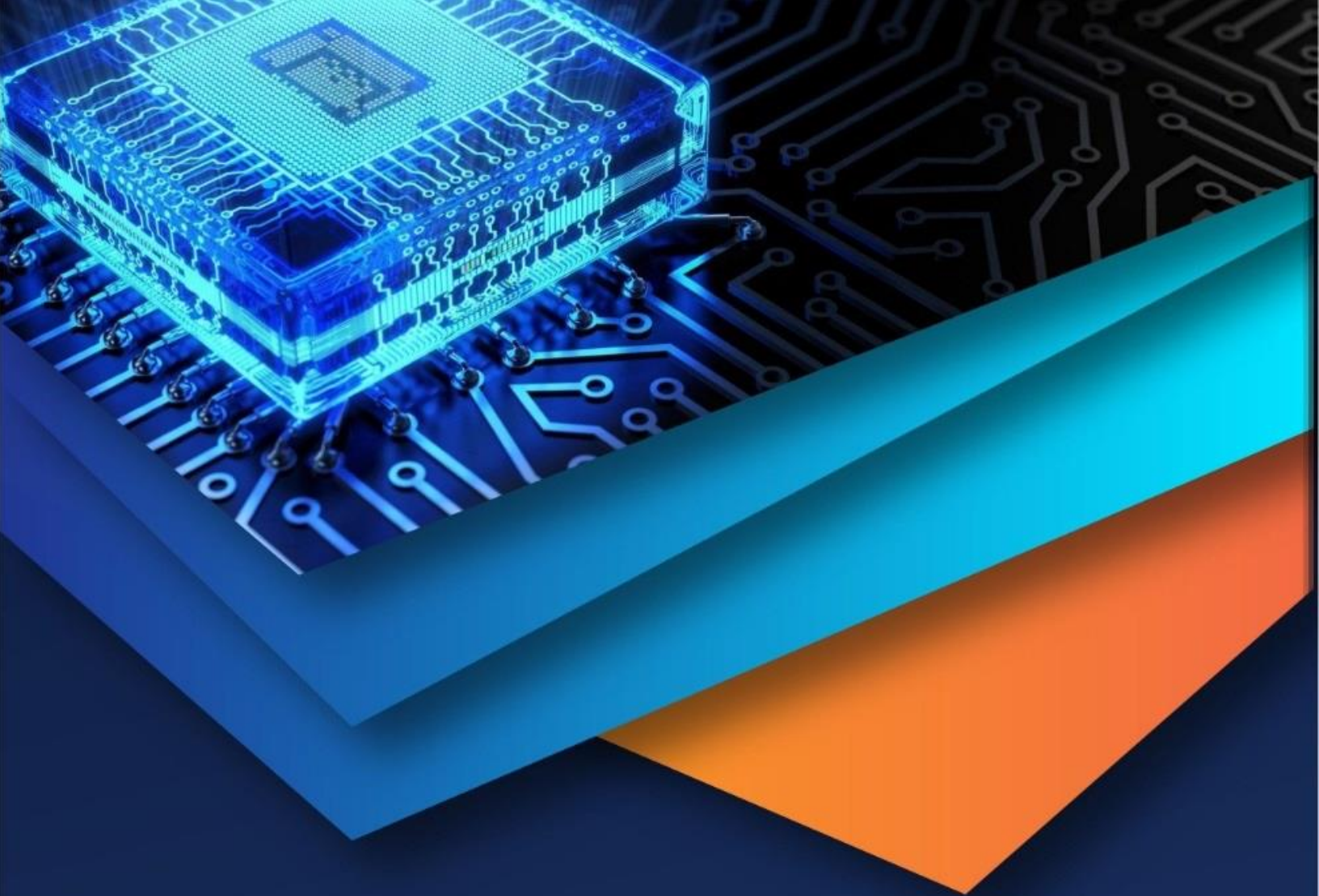
VII. CONCLUSIONS AND FUTURE ENHANCEMENTS

This work validates GNN-based fault forecasting on real-world production cloud microservice traces, addressing a key limitation of Unyi et al. [1]. The proposed EA-GCN improves the F1-score by 47.9% over a standard GCN, highlighting the importance of incorporating call-frequency-based edge attention for more accurate fault prediction. Additionally, a hybrid BCE-smoothness loss and structural analysis using CNRA improve both predictive performance and interpretability.

Future work will address three key areas: (1) scaling EA-GCN to the full 6,466-service Alibaba graph via mini-batch training with GraphSAGE-style neighbourhood sampling; (2) incorporating temporal modelling through spatiotemporal GNNs to capture time-evolving fault patterns such as gradual memory leaks; and (3) integrating real incident labels from Alibaba or equivalent production systems to enable fully supervised training and rigorous evaluation on actual failure events.

REFERENCES

- [1] D. Unyi, E. Rigó, B. Gyires-Tóth, and R. Lovas, "Explainable GNN-Based Approach to Fault Forecasting in Cloud Service Debugging," *IEEE Trans. Netw. Serv. Manag.*, 2025, doi: 10.1109/TNSM.2025.3602223.
- [2] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proc. ICLR*, 2017, arXiv:1609.02907.
- [3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *Proc. ICLR*, 2018, arXiv:1710.10903.
- [4] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
- [5] S. Luo et al., "Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis," in *Proc. ACM SoCC*, 2021, doi: 10.1145/3472883.3487003.
- [6] Alibaba Group, "Cluster Trace — Microservices v2021," Available: <http://aliopentrace.oss-cn-beijing.aliyuncs.com/v2021MicroservicesTraces>, Accessed: Mar. 2026.
- [7] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. NeurIPS*, 2019.
- [8] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [9] A. Maroudis et al., "Leveraging Graph Neural Networks for SLA Violation Prediction in Cloud Computing," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 1, pp. 605–620, 2024, doi: 10.1109/TNSM.2023.3292392.
- [10] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "MicroRCA: Root Cause Localization of Performance Issues in Microservices," in *NOMS*, 2020, pp. 1–9.
- [11] Y. Gan et al., "Sleuth: A Trace-Based Root Cause Analysis System for Large-Scale Microservices with GNNs," in *Proc. ASPLOS*, 2023, pp. 324–337, doi: 10.1145/3623278.3624758.
- [12] X. Zhou et al., "Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs," in *Proc. ESEC/FSE*, 2019, pp. 26–30.
- [13] R. Rouf et al., "InstantOps: A Joint Approach to System Failure Prediction and Root Cause Identification in Microservices," in *Proc. ACM/SPEC Int. Conf. Performance Engineering (ICPE)*, 2024, pp. 119–129, doi: 10.1145/3629526.3645047.



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)