Assessing ML-based anomaly detection across individual telemetry modalities in Cloud-native B5G Networks

Stoyan Kutsarov

Supervisor(s): Sehan Samarakoon Mudiyanselage
Responsible Professor(s): Nitinder Mohan
[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 17, 2026

Name of the student: Stoyan Kutsarov
Final project course: CSE3000 Research Project
Thesis committee: Nitinder Mohan, Sehan Samarakoon Mudiyanselage, Jérémie Decouchant

Abstract

Modern cloud-native systems generate large amounts of telemetry data, including logs, metrics, and traces, which are useful for monitoring and diagnosing system behavior. However, the effectiveness of machine learning-based anomaly detection varies significantly depending on the telemetry modality and the nature of the faults. With the ever increasing demands of 5G applications and the upcoming 6G, system operators must ensure that their networks are able to rapidly respond to and stop faults, the first step of which is detecting them. This project investigates the performance of machine learning models for anomaly detection when logs, metrics, and traces are analyzed independently, with the goal of understanding their relative strengths and limitations. It analyses fifteen models' performance across five different fault classes, comprising 22 faults in total. After creating an appropriate dataset, each model was evaluated on the data collected from several runs, reaching the conclusion that one modality cannot cover all the faults, two get all but one and three modalities can cover every fault.

1 Introduction

Beyond 5G (B5G) networks are the currently-in-development successor to the 5G networks we are using. As the limits of the existing architecture are already being pushed, B5G networks will be expected to be highly scalable, fast, reliable and will be used for various services, from industrial automation and automated vehicles to services such as Augmented and Virtual Reality and remote surgery [39]. To meet these demands, initial B5G network prototypes are headed towards adopting cloud-native architectures and microservice-based designs for their deployment [40]. The microservices such a network would employ can be deployed independently of each other, and as such can be scaled and managed on a large scale through platforms like Kubernetes [31].

While this architectural change provides benefits when it comes to deploying and managing the network, it also increases complexity. Deployments of such networks can rely on dozens of various services across hundreds of pods and containers working in tandem and exchanging data. For monitoring purposes, such networks are regularly scraped for observability data - namely metrics, logs and traces [22]. Logs capture system events and various errors and warning messages, metrics provide numerical information regarding system resource usage and traces provide the execution path of the requests as they traverse from service to service.

The massive amount of data such a network is capable of generating when scraped regularly [41], paired with its complexity make it incredibly hard to conduct fault analysis manually. As a result, there is a growing interest in applying machine learning techniques to automate the analysis and enable faster fault detection [6].

The purpose of this project is to quantify and analyze the performance of existing machine learning models and approaches on a dataset consisting of healthy and faulty observability data collected from a 5G core network deployment, so as to establish how they perform in comparison to each other and if any in particular stand out when tackling the peculiarities of such systems.

The research questions this work aims to cover are as follows:

- How does anomaly detection performance differ when using logs, metrics or traces independently?
- How does performance vary across different categories of faults (e.g., resource exhaustion, component failutre, network delay)?
- Which telemetry modality provides the most useful signal for detecting specific anomaly types?
- How robust are anomaly detection models to noise and variability in each modality (measured in terms of performance degradation under noisy or reduced data)?

The remainder of this paper is structured as follows. Section 2 Describes the background of the problem and the knowledge gap, as well as what the paper will do to tackle them. Section 3 will cover the tech stack used, how the dataset was collected and the faults injected with a focus on reproducibility. Section 4 covers the results of the experiment and offers discussion on them as well as their implication. Section 5 discusses the ethical implications of the paper and what measures were taken to ensure compliance with responsible research throughout. Section 6 concludes the paper with a summary, identifies knowledge gaps that still remain and offers advice for further research on the topic.

2 Background and Relevant Work

Prior research has explored learning-based anomaly detection across individual telemetry modalities. Log-based approaches identify anomalies from sequential patterns, ranging from statistical clustering [14] to deep learning with DeepLog [12] and LogBERT [17]. LogRobust [50] uses weighted embeddings to improve resilience to unstable log formats. Log-based methods have since evolved toward graph models such as DeepTraLog [48], though its reliance on combined trace and log data places it outside the scope of this paper. Despite this progress, there are many different architectures for log analysis and encoding the raw data into a usable format remains non-trivial.

Metric-based techniques apply statistical or deep learning models to numerical time series. OmniAnomaly [38] models the distribution of normal metric sequences, USAD [4] uses two autoencoders to amplify reconstruction errors, and TranAD [43] combines attention-based sequence encoding with adversarial training. A recurring finding is that model complexity does not reliably translate to better performance: under fair evaluation, PCA matches OmniAnomaly, suggesting data quality is a stronger determinant than architecture [2].

Distributed traces record the full execution path of a request as a tree of spans, providing a causally ordered view of inter-service communication unavailable from logs or metrics alone. Approaches range from normalising flows like

TraceAnomaly [30] to graph neural networks like GAL-MAD [3], which jointly models spatial and temporal service dependencies. Trace-based models can degrade as systems evolve and new call paths emerge [9], and data collection delays from upstream service calls can hinder timely detection [44].

These approaches have largely been evaluated on generic microservice benchmarks of loosely coupled stateless services. No work directly compares all three modalities under controlled, identical fault conditions on a 5G core network, where network functions emit highly structured, domain-specific telemetry over HTTP/2 - patterns that general-purpose models were not designed for. It therefore remains unknown which modality provides the most informative signal for specific fault types, or whether any single modality achieves complete fault coverage. This gap limits the ability to design efficient monitoring strategies for B5G systems.

This paper addresses said gap of knowledge by performing a comparative study of machine learning-based anomaly detection across the three telemetry modalities. It provides data regarding their performance with regards to one another, as well as their strengths for identifying different fault types and how resistant they are to faulty data.

3 Methodology & Implementation

3.1 Dataset and Data Processing

A dataset of the kind that would be required to perform machine learning on was not found. No one dataset encompassed all of being ran on a 5G Core network, producing outputs suitable for ML (Machine Learning) models and having data related to all 3 telemetries. To that end, a dataset was assembled by the research group. The base is open5gs as it is the most feature-complete open source 5G Core implementation. It was deployed in kind (Kubernetes in Docker), which could be used to set up a multi-node cluster out of the box and had integration with Chaos Mesh, our fault injection tool. UERANSIM gNB was used as the standard pairing for open5gs for the purposes of simulating user traffic on the network. This exposes 11 5G network functions: AMF, SMF, UPF, PCF, SCP, UDM, AUSF, UDR, NEF, NSSF and NRF. They are further elaborated upon in A.

22 faults were selected as being representative of typical problems faced by a 5G network and injected into the simulated network with ChaosMesh [8]. They are described in Table 1 with the category they fall into, as well as the network function they target.

For each fault, telemetry was collected with Loki [16] for logs, Prometheus [33] for metrics and Jaeger [20] for traces. From there, this data goes to Grafana and is processed into CSVs. Per fault, data was gathered for the network running normally for 10m, the fault being active for 5m and the network recovering for 10m. After each fault, the entire cluster is restarted to prevent any lingering influence of one fault tainting data from further runs. The data is labeled as anomalous if it is collected during the run-time of the fault and as non-anomalous otherwise. Each type of modality collected is processed further from here.

It is important to note that the several runs the experiments were ran on were provided from two other Group 69 members working on the same research question and the pipeline ran on their own personal laptops, on native Ubuntu and WSL respectively.

Table 1: Representative 5G Network Faults

#	Fault Name	Type	NF
1	CPU Stress on AMF	Resource Exhaustion	AMF
2	Memory Pressure on UPF	Resource Exhaustion	UPF
3	AMF Pod Crash	Component Failure	AMF
4	Network Delay on AMF–SCP SBI Link	Network Delay	AMF
5	Network Partition AMF–SCP	Network Partition	AMF
6	Packet Loss on UPF	Network Partition	UPF
7	SMF Pod Crash	Component Failure	SMF
8	CPU Stress on SCP	Resource Exhaustion	SCP
9	Network Delay on NRF	Network Delay	NRF
10	PFCP Session Establishment Flood	Protocol Attack	UPF
11	PFCP Session Deletion Attack	Protocol Attack	UPF
12	PFCP Session Modification DROP	Protocol Attack	UPF
13	PFCP Session Modification DUPL	Protocol Attack	UPF
14	UPF Infrastructure Packet Loss	Network Partition	UPF
15	NRF Cascade Failure	Component Failure	NRF
16	CPU Stress on AUSF	Resource Exhaustion	AUSF
17	Network Delay on SCP	Network Delay	SCP
18	CPU Stress on NRF	Resource Exhaustion	NRF
19	UDM Pod Crash	Component Failure	UDM
20	MongoDB Database Crash	Component Failure	MongoDB
21	N2 Interface Partition	Network Partition	AMF
22	Memory Pressure on AMF	Resource Exhaustion	AMF

All models are trained with the same train/test split. The training set consists of all pre-phase records only (all label=0). Models receive no anomalous examples during training - they all use unsupervised learning. The test set is all during and post records. The test set contains approximately 50% anomalous windows from the during phase and 50% normal recovery windows from the post phase.

Each model's fit() method is called with the training records. Models that require sliding windows construct them from the record sequence, grouped by experiment slug to prevent context from one fault's pre-phase bleeding into another's. Models that operate on fixed-size time windows use the 30-second window granularity established by the data loaders.

At inference, each model's predict() method returns predictions, scores and label arrays. The anomaly score is always a real-valued continuous quantity rather than a binary prediction so as to allow for the computing of threshold-free metrics.

Results are computed per fault and then aggregated. For each model on each fault, the complete set of test records for that fault is passed to predict(). Three primary metrics are computed: AUROC, Average Precision and Recall at the optimal F1 threshold. Each of them presents a different avenue to explore when it comes to evaluating performance. AUROC is a very widely used primary metric in anomaly detection because it is threshold-free and measures a model's ability to discriminate between normal and anomalous instances across all points [34]. A model that guesses at random will achieve an AUROC of 0.5, regardless of class imbalance between anomalies and normal data in the set, making it easy to compare across different scenarios. One limitation is AUROC can produce overly optimistic scores in cases of imbalanced test sets, because when the number of true negatives is large, even a substantial number of false positives produces only a small change in the false positive rate, keeping the curve arti-

ficially high.

To tackle this, Average Precision (AP) is reported alongside AUROC. AP is the area under the precision-recall curve, i.e a weighted mean of precisions at each threshold. The precision-recall curve is preferable to the ROC curve when precision is more relevant than the false alarm rate, and AP is a common way to compute this quantity. Unlike AUROC, which gives equal weight to predictions on both the normal and anomalous class, Average Precision is more sensitive to predictions on the anomalous class specifically [1], making it more informative for anomaly detection. The tradeoff is that the random guessing baseline for AP is given by the fraction of anomalies (being relatively low in the dataset this paper conducts research on) in the test set and thus varies between applications, making it generally harder to compare across different datasets, which is why both metrics are reported together rather than relying on either alone.

Recall at the optimal F1 threshold is included to give a threshold-dependent view of each model's sensitivity. Since all models in this study are trained unsupervised on normal data only, no anomalous examples are available to pre-select a deployment threshold. In an unsupervised setting the threshold must be estimated from the score distribution itself. A standard approach is to select the threshold that maximises the F1 score on the test set, which identifies the best tradeoff between precision and recall for that model. Reporting recall at this threshold captures whether a model is capable of detecting the majority of faults when its threshold is set optimally. Reporting F1-score or recall alongside AUROC and AP gives a more complete picture of each metric's performance [1].

3.2 Machine Learning Models

To get a proper idea of how machine-learning models perform on the data, 5 models were used per modality and their performance was compared against each other, so as to get a broader view and prevent any incompatibility between a specific model and the collected data. These models were sourced from research papers and their implementations were guided by the approaches described in those papers, as well as from GitHub repositories of other implementations where applicable.

Data Formatting

All log models share the same preprocessing pipeline. Raw Loki CSV lines (timestamp_ns, pod, container, app, line) are cleaned and passed through a Drain parser (depth=4, similarity threshold=0.5, max_children=128), which groups similar lines into templates by replacing variable fields with wildcards. This yields 570 unique template IDs across the 11 NFs. Each line becomes a LogRecord carrying its timestamp, NF name (the app column), raw line text, and the integer template ID assigned by the parser. Sequence-based models draw sliding windows of consecutive template IDs from these records, grouped by experiment so no pre-phase context from one fault bleeds into another. This was also done as some of the models are inherently designed to run on the same continuous deployment in order to learn patterns. The best performing log models crater when this is not performed.

Prometheus scrapes one CSV per metric containing timestamp and value columns. These are aggregated across pods: rate and count metrics are summed (total system rate), while duration and latency metrics are averaged (mean response time). The 42 KPIs span six Beyla HTTP metrics (server/client request rates, durations, and error rates), five CPU metrics (container usage, throttling, node-level usage, Beyla and monitoring overhead), four memory metrics (container working set, node available, Beyla and monitoring overhead), two network metrics (transmit and receive byte rates) and 25 open5gs 5G control-plane counters covering AMF registration, authentication, session and subscriber counts, PFCP session state, and SMF/UPF session, PDU and bearer metrics. Missing scrape intervals are forward-filled within each phase. All models z-score normalise the resulting 42-dimensional vectors using the mean and standard deviation computed from the pre-phase training set (3,000 records across 22 experiments).

Jaeger spans are read from the CSV, providing columns trace_id, span_id, service, operation, start_us, duration_us, and error. Spans are filtered to each phase's time bounds using the start_us column and grouped into non-overlapping 30-second windows. For each window, a 48-dimensional feature vector is constructed: for each of the 11 NF services, four statistics are computed from spans whose service field matches that NF - span count, error rate (fraction of spans with error=1), log-mean span duration, and log-P95 span duration (44 dimensions). The log transform is applied to handle the heavy-tailed latency distributions that are present in microservice communication. Four statistics are appended: total distinct trace count, log-mean per-trace total duration, global error rate, and total span count. The pre-phase gives us 20 windows per fault experiment (440 training windows total); the test set contains 10 during-phase and 10 post-phase windows per experiment (442 test windows, 50% anomalous). All models are trained exclusively on pre-phase windows.

Log Models

Five log models were evaluated. **DeepLog** [12, 11, 32, 13] trains an LSTM to predict the next log event type given a window of prior events, flagging windows where the true next key falls outside the top-k predictions. **LogBERT** [18, 19] applies masked language modelling over log template sequences using a bidirectional transformer. **LogRobust** [50] encodes templates via word embeddings processed by a BiLSTM–attention architecture, adapted here to an unsupervised sequence autoencoder. **FeatureModel** [28, 36] applies Isolation Forest to a 35-dimensional time-bucketed feature vector including log rate, error rate, novel template fraction, and heartbeat deficit. **Logs2Graphs** [25, 26] converts each time window into a directed log-transition graph and trains a two-layer Inception DiGCN under the Deep SVDD [35] objective.

Metric Models

Five metric models were evaluated. **MetricPCA** [47] scores test timesteps by PCA reconstruction error relative to a nor-

mal subspace fitted on pre-phase data. **USAD** [4, 5] trains two competing autoencoders to amplify reconstruction errors on anomalous windows through adversarial cascading. **TranAD** [43, 42] uses a Transformer encoder and two decoders with progressive supervision, where the second decoder focuses on residuals from the first. **OmniAnomaly** [38, 37] models the distribution of normal metric sequences with a GRU-based variational autoencoder, scored by reconstruction error. **AnomalyTransformer** [46, 45] augments reconstruction error with an association discrepancy term that measures the divergence between learned and prior attention distributions, preventing score inversion on crash faults.

Trace Models

Five trace models were evaluated. **TraceRPCA** [7, 27] decomposes the pre-phase feature matrix via Robust PCA and scores test windows by reconstruction error against the recovered low-rank normal subspace. **TraceAnomaly** [30, 29] fits a Real NVP [10] normalising flow to the distribution of normal trace windows, scoring anomalies by their negative log-likelihood. **TraceDAE** [23, 24] trains a dual autoencoder jointly on service interaction graph topology and per-service node features. **GAL-MAD** [3] is an encoder–decoder autoencoder that applies GAT layers per timestep and a BiLSTM across sequences of eight consecutive 30-second windows, capturing how inter-NF interaction patterns evolve over 4-minute spans. **TraceSieve** [49, 15] first filters noisy training windows with a GAN-based noise filter, then trains a Variational Graph Autoencoder on the denoised set, scoring test windows by Monte Carlo negative log-likelihood with standard deviation clipping.

More details on each model's implementation are included in B.1

4 Experiment Results & Discussion

4.1 Results Overview

Table 2 summarises mean AUROC, Average Precision (AP), and Recall at the optimal F1 threshold across all 22 faults for each model. All metrics are threshold-free except Recall, which is computed at the threshold that maximises F1 on the test score distribution.

4.2 Log Models

Log-based detection is the strongest single modality, with DeepLog achieving the highest mean AUROC of 0.913 and LogBERT closely following at 0.859. Together, these two models cover 19 of 22 faults at AUROC $\geq$ 0.70. DeepLog leads across every fault class, reaching 0.974 on protocol attacks and 0.896 on resource exhaustion, demonstrating that PFCP flood and CPU stress faults produce clearly anomalous log-key sequences.

The three faults where logs fall short - the N2 interface partition between AMF and the gNB (0.510), an AMF pod crash (0.553), and memory pressure on the AMF (0.555) - share a common property: they either produce few novel log events or disrupt the system so quickly that the log stream

Figure 1: AUROC across all 22 faults for all 15 models
X = Model Y = Fault

Table 2: Mean performance across 22 faults per model.

Modality	Model	AUROC	AP	Recall
Logs	DeepLog	0.913	0.861	1.000
	LogBERT	0.859	0.799	0.994
	LogRobust	0.535	0.553	0.995
	FeatureModel	0.515	0.553	0.978
	Logs2Graphs	0.490	0.576	0.987
Metrics	TranAD	0.660	0.746	0.964
	AnomalyTransformer	0.604	0.744	0.939
	USAD	0.613	0.684	0.975
	OmniAnomaly	0.598	0.578	0.946
	MetricPCA	0.586	0.641	0.963
Traces	GAL-MAD	0.810	0.925	0.979
	TraceRPCA	0.649	0.681	0.952
	TraceAnomaly	0.637	0.694	0.961
	TraceDAE	0.625	0.684	0.944
	TraceSieve	0.568	0.623	0.974

contains mostly the same pre-fault templates. The N2 partition is nearly silent in logs as UERANSIM stops sending registration requests rather than emitting error messages.

LogRobust, FeatureModel, and Logs2Graphs all perform at near-random levels. LogRobust's poor performance can be attributed to the adaptation: converting a supervised binary classifier into an unsupervised autoencoder discards what makes the original model effective. FeatureModel's Isolation Forest is limited by the aggregated nature of the bucket-level features, which smooth over brief anomalous events. Logs2Graphs produces very small graphs from 30-second windows - typically 10-15 nodes with sparse edges - giving the GNN too little structural information to learn meaningful normal patterns.

4.3 Metric Models

Metric-based detection is the weakest modality, with all five models clustered between 0.516 and 0.597 AUROC. This is a direct consequence of how Prometheus metrics are collected: all KPIs are aggregated across pods at the cluster level, which means the stress placed on one specific NF during a fault is

diluted by the unchanged behaviour of all other NFs. A CPU stress fault on the AMF raises the AMF's CPU counter but this increase is averaged across all 11 NFs, substantially reducing the signal.

The fault-class breakdown reveals two visible splits. Protocol attacks (PFCP) and resource exhaustion are comparatively well-detected, with AnomalyTransformer reaching 0.871 and 0.804 respectively on these classes. PFCP floods cause measurable spikes in session counters (smf_pdu_req, pfcp_sessions) and CPU utilisation that are visible in the aggregated metrics. Network faults, by contrast, are nearly undetectable by all metric models, as network delays and partitions produce no direct change in the Prometheus KP we collect - their impact is visible in span latencies and log patterns rather than resource counters.

Among the models, TranAD's two-decoder focus mechanism and AnomalyTransformer's association discrepancy score provide marginal but consistent gains over the simpler baselines. The focus mechanism in TranAD is particularly well-suited to our dataset where faults manifest gradual divergence in a small subset of metrics rather than a uniform shift across all 42 KPIs. MetricPCA performing competitively against significantly more complex models echoes the finding of Alves et al. [2] that PCA and deep models are broadly equivalent under fair evaluation conditions. Mean AUROC alone understates metric model performance: models designed for resource monitoring detect the fault classes that produce resource-level signals, and their low aggregate score reflects the mismatch between the collection setup and several fault types rather than uniform failure.

4.4 Trace Models

GAL-MAD substantially outperforms all other trace models, reaching 0.81 AUROC and 0.925 AP. The margin over the next-best model, TraceRPCA, is large enough to suggest that its spatial and temporal modelling provide a qualitatively different signal rather than a marginal improvement. By learning how service interaction patterns and per-service latency statistics evolve across 4-minute sequences, GAL-MAD captures the propagation of fault effects through the NF dependency graph - for example, an AMF crash that cascades to elevated latency in SMF and NRF spans within seconds, a pattern invisible in a single 30-second snapshot.

The remaining four trace models are tightly clustered between 0.568 and 0.649. TraceRPCA's competitive performance despite being a linear method reflects the robustness of the inexact ALM decomposition to sparse pre-phase artefacts. TraceAnomaly, TraceDAE, and TraceSieve all hover around the same range, suggesting that without cross-service call graph structure, which is unavailable due to our Beyla instrumentation producing per-NF spans without parent-child links, the learned representations converge to similar quality regardless of architectural differences.

Traces cover 17 of 22 faults. The five uncovered faults are mainly resource exhaustion faults (CPU stress on AMF, memory pressure on UPF and AMF) and two infrastructure-level failures (MongoDB pod kill, UPF packet loss). These faults produce limited change in span counts and durations at the service level, as the NFs continue handling requests at reduced throughput without the span-observable latency spikes that characterise network or protocol faults.

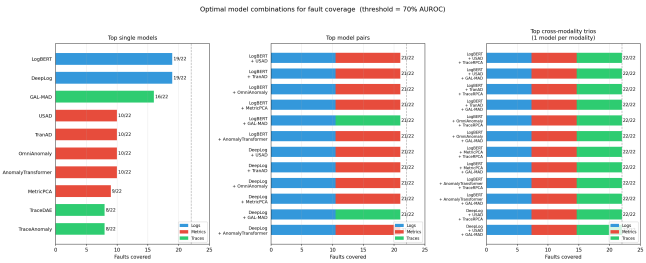

Figure 2: Fault coverage per model. The first graph shows the performance of the best individual models, the second how they do in pairs and the third in trios. Blue represents a log model, red a metrics model and green a trace model.

4.5 Cross-Modality Coverage

Table 3 shows the number of faults covered (AUROC ≥ 0.70 by at least one model) for each combination of modalities, using the best model per modality at each fault. No single modality achieves full coverage. Logs are the dominant individual modality (19/22), and adding either metrics or traces raises coverage to 21/22, with a different single fault remaining undetected in each case. Adding metrics to logs plugs the gap left by the AMF pod crash and AMF memory pressure faults - both are reflected in Prometheus KPI shifts even when logs appear near-normal. Adding traces to logs instead covers the N2 interface partition, which produces a clear reduction in span counts from the AMF-gNB interface that GAL-MAD detects.

Table 3: Fault coverage by modality combination (AUROC $\geq$0.70, $n = 22$).

Modality combination	Faults covered
Logs only	19/22
Metrics only	11/22
Traces only	17/22
Logs + Metrics	21/22
Logs + Traces	21/22
Metrics + Traces	19/22
All three	22/22

The complementarity of modalities reflects the different fault types they are sensitive to. Logs capture control-plane sequencing anomalies and error template emergence, making them effective for component failures and protocol attacks. Metrics capture resource-level deviation, making them useful for CPU and memory faults, though their coverage reflects the limitations of cluster-level aggregation rather than the ceiling of metric-based detection. Traces capture latency propagation and service interaction changes, making them sensitive to network faults and cascades.

4.6 Discussion

The most notable finding is the gap between log performance and the other two modalities. DeepLog's mean AUROC is substantially higher than the best metric model and comparable only to GAL-MAD in traces. This aligns with the 5G core's extensive structured logging: Open5GS emits detailed per-operation log lines for every NF function call, registration, session event, and failure, giving sequence-based models rich event patterns to learn. Metrics, as mentioned above, are designed for cluster-level capacity monitoring rather than per-NF failure diagnosis, and their aggregation across pods fundamentally limits fault discriminability.

The poor performance of Logs2Graphs warrants discussion. The graph-based approach is well-motivated in theory - directed transition graphs should capture structural log anomalies that pure sequence models miss - but the 30-second window granularity produces graphs that are too small and sparse to train a meaningful one-class classifier. Longer windows would increase graph expressiveness but reduce the temporal resolution needed to separate pre-fault and during-fault windows cleanly. This tension between graph quality and temporal resolution is inherent to graph-based log anomaly detection in high-frequency log streams.

The metrics results highlight a broader challenge for anomaly detection in this setting: the observability stack is designed for aggregate health monitoring, not fine-grained fault isolation.

A fault-class breakdown adds nuance to the per-modality picture. Protocol attacks are the most consistently detectable class across modalities: PFCP flood and session manipulation faults produce bursts of PDU session requests that manifest simultaneously as distinctive log-key sequences, visible session counter spikes in metrics, and UPF latency surges in traces. Network faults present the opposite pattern: metrics are nearly silent because the cluster-level aggregation discards the per-NF variance, through which network delays would otherwise be visible, and Prometheus collects no network-layer counters in this deployment, while traces are the primary signal through span-count reductions at the AMF–gNB interface that GAL-MAD detects. Component failures occupy a middle ground, in that pod crashes produce immediate log events but the post-crash recovery period can resemble normal operation to metric-based models that rely on resource-level indicators. This confirms that no modality is uniformly superior. Fault class determines which modality provides the most discriminative signal.

The trace results warrant a closer look at GAL-MAD's architectural advantage. The AUROC gap between GAL- MAD and the next-best trace model TraceRPCA is not explained by differences in window features. The critical difference is GAL-MAD's joint spatial-temporal modeling: its graph-attention layers capture which NF pairs exhibit abnormal interaction at each timestep, while the LSTM tracks how those patterns evolve across the whole sequence. This design allows GAL-MAD to detect fault propagation chain. Those patterns are invisible in any single 30-second snapshot. The tight clustering of TraceAnomaly, TraceDAE, and TraceSieve suggests a performance ceiling for single-window trace models under our dataset, which lacks parent-child span links that richer trace-based approaches rely on.

4.7 Noise Tolerance

To assess robustness when the data is contaminated by undetected anomalies, each model was re-evaluated after retraining with increasing proportions of anomalous windows injected into the training set. Noise rates of 5%, 10%, 25%, 50%, and 100% of training windows were evaluated. Table 4 reports mean AUROC across all 22 faults at each noise level. Figure 3 shows the per-fault AUROC degradation curves.

Log models show the widest range of noise sensitivity. DeepLog degrades sharply from 0.913 to 0.570 at 25% noise before recovering to 0.963 at 100%. The recovery at full noise is an artifact: when the entire training set consists of anomalous windows, the model's learned normal distribution shifts to cover the anomaly class, and the predominantly anomalous test set is then flagged accordingly, which inflates AUROC artificially. LogBERT suffers a steep initial drop to 0.540 at 5% noise and partially recovers to 0.746 at 100% noise. The three lower-performing log models (LogRobust, FeatureModel, Logs2Graphs) already operating below 0.54 AUROC at baseline change negligibly across noise levels, as their performance is bounded by the low informativeness of their feature representations rather than by the quality of the training data.

All five metric models collapse immediately to near-random AUROC (0.48–0.51) at just 5% noise and remain flat there- after. This stands in contrast to log and trace models, where degradation is gradual and modulated by noise level. The abrupt and complete collapse reflects the already-weak discriminative signal in the aggregated 42-dimensional feature vectors: the learned normal region is so diffuse that even two to three anomalous training windows per fault are sufficient to corrupt the normality boundary entirely. This fragility reinforces that metric-based anomaly detection in this deployment is unreliable even under mild training data imperfections.

For the trace models, GAL-MAD, the strongest trace model at baseline (0.810), collapses to 0.453 at 5% noise and reaches its floor of 0.391 by 50% noise. TraceRPCA, by contrast, marginally improves from 0.649 to 0.701 at 10% noise before declining gradually to 0.593 at 100% noise. This robustness is architecturally grounded: RPCA's low-rank decomposition explicitly models the training matrix as a superposition of a low-rank normal component and a sparse perturbation component. Isolated anomalous training windows are absorbed into the sparse component, leaving the learned normal sub- space intact rather than distorting it. TraceSieve similarly shows resilience, with AUROC remaining near baseline across the full range (0.568 to 0.603 at 100% noise). TraceAnomaly and TraceDAE exhibit moderate decline.

Taken together, the noise tolerance reveal a consistent tradeoff between peak performance and noise resilience across all three modalities. In every modality, the highest-performing model at baseline (DeepLog, TranAD, GAL-MAD) is also among the most sensitive to training contamination. High-capacity models learn tighter normal representations, which are correspondingly more vulnerable when the training distribution is corrupted. Simpler or structurally reg-

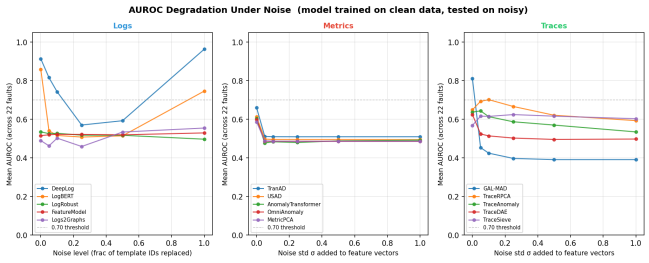

Figure 3: AUROC per model under noise

ularized models (TraceRPCA, FeatureModel) show greater resilience at the cost of lower peak AUROC. For deployment in production B5G networks, where training windows cannot always be guaranteed anomaly-free, this tradeoff has direct practical implications: the choice between a top-tier model or a worse one that can tolerate more corruption depends on whether the operational context permits clean training data collection or must be able to handle imperfect labeling.

4.8 Feature Priority

To identify which feature groups carry the most important signal, a feature dropout analysis was performed for metrics and traces, dropping one named feature group at a time and re-evaluating all models. Figures 4 and 5 show the results. For metrics, five groups were tested: CPU counters, memory counters, network counters, HTTP/SBI rate metrics, and 5G control-plane metrics. Dropping CPU counters produces the largest degradation across all five metric models, confirming that CPU utilisation metrics carry the most vital fault signal - consistent with resource exhaustion being the most common fault class in our benchmark. Dropping memory metrics has a negligible or even slightly positive effect, suggesting that memory KPIs introduce noise rather than signal in the aggregated feature vectors, likely because memory consumption varies more during normal operation than CPU does under the injected faults. Dropping 5G control-plane and HTTP/SBI metrics similarly causes minimal degradation.

For traces, four groups were tested: per-NF span count, per- NF error rate, per-NF latency (mean and P95), and global statistics. Dropping latency features produces the largest drop across all models, establishing per-NF latency as the primary carrier of fault information in the trace feature vectors. Dropping global statistics causes a counter-intuitive improvement in GAL-MAD: the graph-attention mechanism already captures inter-NF relationships from the per-service features, and the aggregate global stats appear to add noise that dilutes the signal. Simpler models such as TraceAnomaly and TraceDAE lose performance when global statistics are removed, indicating they rely on these summary statistics more heavily. The dropout analysis is not performed for log models: The best performing log-based anomaly detectors operate on sequences of template IDs, so there is no natural grouping to drop that preserves the sequential structure the models depend on.

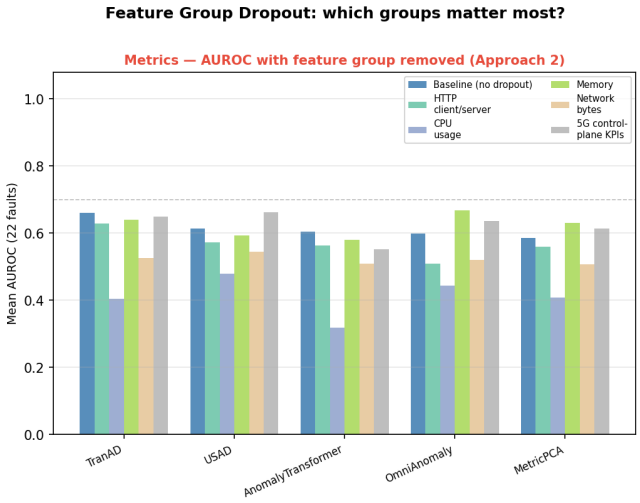

Figure 4: Metric AUROC with feature groups removed

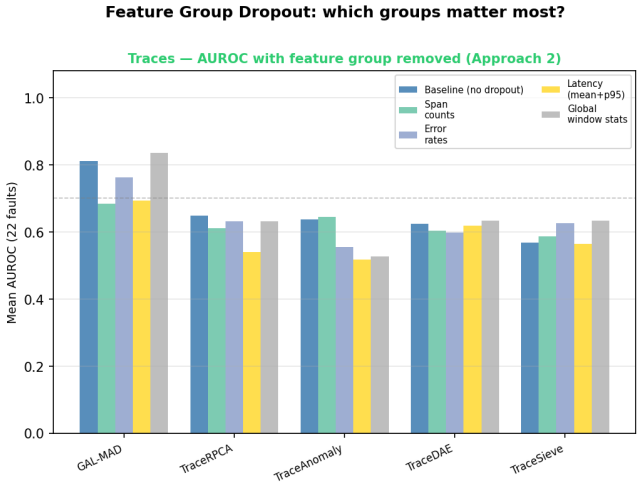

Figure 5: Traces AUROC with feature gorups removed

Table 4: Mean AUROC under training noise (proportion of anomalous training windows).

Modality	Model	0%	5%	10%	25%	50%	100%
Logs	DeepLog	0.913	0.816	0.742	0.570	0.592	0.963
	LogBERT	0.859	0.540	0.516	0.508	0.514	0.746
	LogRobust	0.535	0.525	0.527	0.517	0.517	0.497
	FeatureModel	0.515	0.520	0.522	0.521	0.519	0.529
	Logs2Graphs	0.490	0.462	0.502	0.459	0.534	0.554
Metrics	TranAD	0.660	0.511	0.510	0.509	0.509	0.509
	AnomalyTransformer	0.604	0.477	0.483	0.479	0.488	0.492
	USAD	0.613	0.497	0.495	0.495	0.495	0.494
	OmniAnomaly	0.598	0.486	0.486	0.485	0.485	0.485
	MetricPCA	0.586	0.488	0.486	0.486	0.486	0.486
Traces	GAL-MAD	0.810	0.453	0.424	0.397	0.391	0.391
	TraceRPCA	0.649	0.693	0.701	0.666	0.621	0.593
	TraceAnomaly	0.637	0.643	0.614	0.587	0.570	0.534
	TraceDAE	0.625	0.524	0.514	0.503	0.495	0.497
	TraceSieve	0.568	0.616	0.615	0.624	0.616	0.603

5 Responsible Research

The dataset used in this research was collected on a simulated 5G core network running on personal laptops, no faults were injected in a commercially available network.

Large Language Models (LLMs) were used in the creation of the testing pipeline, from providing initial setup instructions to fixing any bugs that occurred when trying to run it. For the purposes of creating the machine-learning pipeline, LLMs were also used as a coding aid when writing the models, which were also guided by various open-source implementations. In addition, they were used when creating visualizations for the results, which were subsequently verified by the author, as well as involved in polishing the language and any grammatical errors, present in this paper. All content within it otherwise was written by the author.

For reproducibility, the fault detection pipeline, to which all the members of this research group contributed, as well as the models and the analysis scripts, are available here. https://github.com/StoyanKucarov/open5gs-anomaly-detection

6 Conclusion and Future Work

6.1 Conclusion

This paper presented an evaluation of unsupervised anomaly detection across three observability modalities, namely logs, metrics, and distributed traces, applied to a deployed 5G core network running on Open5GS. Fifteen anomaly detection models were implemented and evaluated under a unified unsupervised protocol, training exclusively on pre-fault normal data and evaluating on held-out fault and recovery windows using threshold-free AUROC and Average Precision metrics.

The results show that no single modality achieves complete fault coverage. Logs are the strongest individual signal, with DeepLog reaching a mean AUROC of 0.913 across all 22 faults, driven by the rich structured event sequences emitted by Open5GS. Traces provide complementary coverage through GAL-MAD, which reaches 0.810 AUROC by jointly modelling spatial dependencies between NFs and temporal evolution across short window sequences. Metrics are the weakest modality overall, with all models clustered below 0.7 AUROC. This is a direct consequence of Prometheus aggregating KPIs at the cluster level and thereby diluting the per-NF signals that fault injection produces. Only the combination of all three modalities achieves full 22/22 fault coverage at the AUROC ≥ 0.70 threshold, with each modality contributing unique sensitivity: logs to control-plane sequencing anomalies, traces to latency propagation and service interaction changes, and metrics to resource-level deviations in CPU and session-level counters.

6.2 Future Work

Future research done on this subject should focus first and foremost on gathering a more complete dataset with more trial runs and more exhaustive data per modality. The trace models required substantial adaptation because of the Beyla issues that were mentioned previously.

The metric aggregation problem is another big limitation. Collecting per-pod rather than per-namespace Prometheus metrics would preserve the per-NF variance that is currently discarded, potentially transforming metrics from the weakest modality into a competitive one. The cross-modality coverage results where only the combination of all three modalities achieves full 22/22 coverage also motivate building a joint multi-modal model rather than taking a union of independent detectors. Finally, validating the pipeline on a larger-scale or commercially deployed 5G core would establish whether the fault signatures identified here generalise beyond a single-node kind cluster.

A Network Function Description

- Access and Mobility Management Function (AMF) - Responsible for device registration, connection establishment, authentication, and mobility tracking across cells
- Session Management Function (SMF) - Handles device sessions, allocates IP addresses, and configures traffic routing to the UPF

- User Plane Function (UPF) - Processes and routes actual user data traffic
- Policy Control Function (PCF) - Provides rules and policies for network behavior, QoS, and charging to the SMF
- Service Communication Proxy (SCP) - Acts as an intermediary for communication between network functions
- Unified Data Management (UDM) - Stores critical subscriber data, user profiles, and generates authentication credentials
- Authentication Server Function (AUSF) - Works with the UDM to authenticate devices accessing the 5G network
- Unified Data Repository (UDR) - Serves as the centralized database for subscriber and network-related data
- Network Exposure Function (NEF) - Securely exposes network data and capabilities to third-party applications and developers via APIs
- Network Slice Selection Function (NSSF) - Directs connected devices to the specific, dedicated virtual network slice required for their specific use-case
- NF Repository Function (NRF) - Acts as a central directory that allows different network functions to discover, register, and communicate with one another dynamically

B Model Details

B.1 Log Models

DeepLog [12, 11, 32, 13] trains a two-layer LSTM on normal log sequences to predict which log event type comes next, given the previous ten events. The template IDs from pre-phase CSV lines are used directly as the input vocabulary (570 types). Each training sample is a window of 10 consecutive template IDs drawn from the stream of logs. The LSTM has an embedding dimension of 32, a hidden dimension of 64, and dropout of 0.1 between layers. It is trained for 20 epochs using Adam [21] (lr=1e-3, batch size=512) with cross-entropy loss on the next-key prediction. At inference, a window is flagged as anomalous if the true next event does not appear among the model's top-9 predictions, where k=9 matches the value used in the original paper. No official implementation was released in said paper the model was implemented with inspiration from three independent open-source repositories as cited above.

LogBERT [18, 19] trains a bidirectional transformer on normal log sequences using masked language modeling. The template vocabulary is made up of pre-phase CSV lines. During training, 15% of tokens in each 10-token window are randomly replaced with a [MASK] token, and the model learns to predict them from full bidirectional context. The Transformer has d_model=64, 4 attention heads, feed-forward dimension 256 (4×d_model), and 2 encoder layers, and is trained for 20 epochs with Adam (lr=1e-3, batch size=512). When inferring, each position in a test window is masked in turn and the anomaly score is the fraction of positions whose actual token is not among the top-9 predictions. The implementation was referenced with the official repository, scaled down from the paper's d_model=128 and 4 layers to d_model=64 and 2 layers to match our smaller vocabulary (570 templates versus the paper's 30,000 HDFS log keys) and training set size.

LogRobust [50] represents each log template as the average of its word embeddings and processes sequences through a bidirectional LSTM. The word vocabulary is built from the non-wildcard tokens of each template string, encoded as learned embeddings of dimension 32 rather than the paper's GloVe (Global Vectors), which have no coverage of domain-specific 5G NF log tokens. Each window of 10 template IDs is encoded by averaging the word embeddings per template, passing the sequence through the LSTM (hidden size 64 total, 32 per direction, 2 layers) with a scalar attention layer, and projecting the context to a summary vector. The original LogRobust is a supervised classifier. It is included to assess whether its log template representation approach transfers to an unsupervised setting, or whether the discriminative capacity of the classification head is what is behind its effectiveness. It was adapted to the unsupervised setting by removing the classification head and training as a sequence autoencoder. The model predicts the mean word embedding of the full input window, and the cosine distance between the predicted summary and the actual mean embedding is the anomaly score. Training uses cosine embedding loss for 20 epochs with Adam (lr=1e-3, batch size=256). The implementation follows the paper as closely as possible.

FeatureModel applies Isolation Forest [28, 36] to a 35-dimensional feature vector computed per 30-second time bucket. Buckets are formed by assigning each log record to a bucket index using floor(timestamp_ns / 10^9 / 30). Five global features are computed: total_log_rate (lines per second, capturing volume bursts and drops); error_rate (fraction of lines matching ERROR/WARNING keywords); error_count (raw count of error lines); novel_rate (fraction of template IDs in the bucket unseen in the pre-phase, indicating previously unobserved failure patterns); and heartbeat_deficit (fraction of expected periodic-template occurrences absent from the bucket). Heartbeat templates are identified during training by computing the coefficient of variation (CV = std/mean) of intra-experiment inter-arrival times for each template; templates with CV ≤ 0.5 and rate ≥ 0.05 events/s are tagged as periodic. In 5G core, NFs send periodic registration-refresh messages to the NRF. The deficit score captures their disappearance, which should in theory be a strong signal for crash and partition faults that produce no novel log templates. The remaining 30 features are per-component log rate and per-component error rate for each of the 15 log-producing components in the cluster. The Isolation Forest is fitted on pre-phase buckets with n_estimators=100, contamination=0.05, and random_state=42, and the anomaly score is the negated sklearn decision function. This feature set was derived from the cited paper.

Logs2Graphs [25, 26] converts each 30-second window

of log lines into a directed graph using the same timestamp-bounded buckets as the FeatureModel. Each unique template ID observed in the window becomes a node; a directed edge i to j is added with weight equal to the number of times template j immediately follows template i in the log stream. Node attributes are 32-dimensional embeddings of template IDs, initialised randomly and trained jointly with the GNN, replacing the paper's pre-trained GloVe embeddings. The adjacency is row-normalised per direction before message passing. A two-layer Inception DiGCN applies both 1-hop and 2-hop directed message passing combined per layer, capturing direct transitions and two-step paths simultaneously. Normal graphs are trained under the Deep SVDD [35] objective. At inference, the anomaly score is the squared Euclidean distance from the graph's embedding to the centre. The implementation is inspired by the official repository, with learned embeddings as the only structural adaptation.

B.2 Metric Models

MetricPCA [47] reduces the 42-dimensional Prometheus KPI vectors to a lower-dimensional normal subspace using PCA fitted on the pre-phase records. The top 10 principal components are retained, capturing the dominant variance structure of normal operation. Test timesteps are z-score normalised using pre-phase statistics, projected into the subspace, and scored by their squared reconstruction error, with high error indicating a deviation from the normal correlation structure.

USAD [4, 5] trains two autoencoders on a flattened sliding window of 5 consecutive 5-second timesteps (25 seconds of context, 210-dimensional input). Each autoencoder has a single hidden-layer encoder and decoder with a latent dimension of one quarter of the input size (52 dimensions). At epoch n, AE1 minimises $(1/n)\cdot\|AE1(x)x\|^2 + (11/n)\cdot\|AE2(AE1(x))x\|^2$, learning to reconstruct accurately while progressively trying to fool AE2; AE2 minimises reconstruction of the real input while maximising error on AE1 outputs. The anomaly score is $0.5\cdot\|AE1(x)x\|^2 + 0.5\cdot\|AE2(AE1(x))x\|^2$, amplifying errors on anomalous windows through the cascade. The model is trained for 50 epochs with Adam. The implementation follows the official repository.

TranAD [43, 42] uses a transformer encoder and two transformer decoders over a window of 10 consecutive steps (50 seconds). The internal dimension is d_model=84 (2×N_features, as recommended by the paper), with 4 attention heads, 3 transformer layers, and a feed-forward dimension of 4×d_model. The first decoder reconstructs the input window normally; the second receives the first's reconstruction errors as additional context, trying to work on the parts of the input already poorly reconstructed. The training loss at epoch n blends both decoders: $(1/n)\cdot MSE(x1,x) + (11/n)\cdot MSE(x2,x)$, progressively shifting supervision toward the second decoder. The anomaly score is the mean MSE of the second decoder's output across the window. The implementation follows the repository, with two changes: the training loss was fixed to a single blended term, and the learning rate scheduler decay was changed from gamma=0.5 to gamma=0.9 to prevent the learning rate from collapsing too early on our smaller training set. The model is trained for 50 epochs.

OmniAnomaly [38, 37] uses a GRU-based variational autoencoder that learns the distribution of normal metric sequences. The encoder is a 2-layer GRU with hidden size 64 that processes a sliding window of 20 consecutive steps (100 seconds of context). The final hidden state is then projected to a latent mean and log-variance via separate linear layers, giving a latent dimension of 8. The decoder broadcasts the new sample across the window through a 2-layer GRU and projects back to the 42-dimensional feature space. Training uses the ELBO loss (MSE reconstruction + b·KL with b=0.01) for 50 epochs with Adam. The implementation follows the repository, with the GRU hidden size reduced from the paper's 500 to 64 and the latent dimension from 100 to 8 to match our training set size (3,000 steps versus the paper's SWaT/WADI datasets).

AnomalyTransformer [46, 45] computes two attention distributions at each layer over a window of 20 consecutive steps (100 seconds): a series association (learned self-attention) and a prior association (a per-head Gaussian kernel over time positions with learnable bandwidth sigma). The Transformer has d_model=64, 4 attention heads, 3 layers, and feed-forward dimension 256. During normal operation, the two distributions differ significantly because meaningful cross-metric correlations pull attention away from local positions; during anomalies, the patterns converge as the anomalous period dominates attention. The anomaly score is reconstruction_error / (discrepancy + epsilon), where discrepancy is the mean symmetric KL divergence between series and prior associations averaged across all layers and heads. This combined score avoids the failure mode where a crash fault (metrics flatlining) reduces pure reconstruction error by making all steps equally predictable. The model is trained for 50 epochs with Adam. The implementation follows the repository.

B.3 Trace Models

TraceRPCA decomposes the 440×48 pre-phase feature matrix into a low-rank component L (normal behaviour) and a sparse component S (pre-phase artefacts) using Robust PCA [7], solved via the Inexact Augmented Lagrangian Method [27] with regularisation =1/sqrt(max(m,n)), convergence tolerance 1e-6, and a maximum of 500 iterations. The top 10 singular vectors of L are retained as the normal subspace. Test windows are z-score normalised using pre-phase statistics and scored by their squared reconstruction error when projected into and out of that subspace. Unlike plain PCA, the RPCA decomposition tolerates sparse corruptions in training. Transient NRF heartbeat gaps or brief Beyla instrumentation artefacts in the pre-phase do not distort the estimated normal subspace.

TraceAnomaly [30, 29] fits a normalising flow to the distribution of normal trace windows. The architecture is

Real NVP [10]: 6 coupling layers alternating even and odd dimension masks, each with a scale network and a translation network, applied to the z-score normalised 48-dimensional window feature vectors. The anomaly score is the negative log-likelihood -log p(x) under the learned density, evaluated via the change-of-variables formula accumulated across all coupling layers. The model is trained for 80 epochs with Adam (lr=1e-3) on the 440 pre-phase windows. The implementation follows the repository. The original code constructs features from call-path latency profiles derived from parent-child span links, which are unavailable from our Beyla instrumentation. The entire feature extraction module was therefore replaced with the flat 48-dimensional window feature vector described above, while the Real NVP architecture and training procedure were kept as is.

TraceDAE [23, 24] uses two autoencoders trained jointly on per-window service interaction graphs. The structure autoencoder encodes the 11-node service topology using two single-head GAT layers and reconstructs the adjacency matrix via inner-product decoder. The attribute autoencoder is an MLP encoder-decoder operating on the flattened 44-dimensional per-service feature matrix. Since Beyla spans have no parent-child links as mentioned previously, the per-window adjacency matrix is derived from consecutive service pairs within each trace_id ordered by start_us: if service i's span immediately precedes service j's span in any trace within the window, A[i,j]=1. The joint loss is alpha·L_struct + (1alpha)·L_attr with alpha=0.1, using weighted Frobenius norms that penalise missing non-zero edges and attributes more heavily. The anomaly score is the z-score of the total joint loss relative to the training distribution. The implementation follows the repository, with two adaptations: the attribute autoencoder was changed from an LSTM to an MLP because there is no real temporal ordering within a single 30-second window of per-service aggregate statistics, and the adjacency construction was replaced with the consecutive-span ordering described above.

GAL-MAD [3] is an encoder-decoder autoencoder that jointly models spatial NF dependencies and their temporal evolution over sequences of W=8 consecutive 30-second windows (4 mins of context per sample). Per-timestep node features are the 11×4 matrix of per-service trace statistics. The static 3GPP NF reference topology (14 bidirectional interface edges plus self-loops, symmetrically normalised as $D^{1/2}AD^{1/2}$) serves as the fixed adjacency. The encoder applies two single-head GAT layers (d_gat=16) independently to each timestep, flattening the 11×16 output to a 176-dimensional vector; a bidirectional LSTM then summarises the full 8-step sequence into a 32-dimensional latent vector z via linear projection. The decoder expands z to 8 timesteps via a unidirectional LSTM, reconstructing each timestep through two mirrored GAT decoder layers back to the 11×4 node feature space. Training minimises MSE reconstruction loss over the 440 pre-phase sequences; the anomaly score is the mean MSE across all timesteps and nodes. No public implementation was released in the paper. The model was implemented based on the architecture description in the paper, using single-head GAT layers.

TraceSieve [49, 15] works in two stages. First, a GAN-based noise filter is applied to the 440 pre-phase windows before VGAE training. The GAN has a shared bottleneck encoder feeding two separate linear decoders; windows whose adversarial reconstruction loss exceeds a threshold are excluded as artefacts. Second, a Variational Graph Autoencoder learns the distribution of normal service interaction graphs from the denoised set. The two-layer GCN encoder maps the 11×4 node feature matrix and static 3GPP adjacency to their respective nodes. The decoder reconstructs node features via MLP and the adjacency via inner product, trained with standard ELBO loss. The anomaly score is the negative log-likelihood under the VGAE estimated by Monte Carlo sampling, with standard deviation clipping to address the entropy-gap issue where anomalous windows can otherwise produce spuriously high likelihoods. The implementation follows the repository, with three adaptations: window-level 30-second graphs instead of per-trace graphs (required because of the Beyla issue), the static 3GPP topology in place of a data-derived invocation graph and Elastic Weight Consolidation omitted, as it is a continual-learning mechanism and is not needed for our fixed training set.

References

[1] Maxime Alvarez et al. *A Revealing Large-Scale Evaluation of Unsupervised Anomaly Detection Algorithms*. Apr. 2022. DOI: 10.48550/arXiv.2204.09825.

[2] Bruna Alves et al. "Revisiting OmniAnomaly for Anomaly Detection: Performance Metrics and Comparison with PCA-Based Models". In: *arXiv preprint arXiv:2603.18985* (2026).

[3] Chamodya Attanayake et al. "GAL-MAD: Towards Explainable Anomaly Detection in Microservice Applications Using Graph Attention Networks". In: *arXiv preprint arXiv:2504.00058* (2025). No public code repository available at time of writing.

[4] Julien Audibert et al. "USAD: UnSupervised Anomaly Detection on Multivariate Time Series". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. ACM, 2020, pp. 3395–3404. DOI: 10 . 1145 / 3394486.3403392.

[5] Julien Audibert et al. *USAD: UnSupervised Anomaly Detection on Multivariate Time Series*. https://github.com/manigalati/usad. GitHub repository. 2020.

[6] Gustavo Bruno et al. "Anomaly Detection in Cloud-native B5G Systems using Observability and Machine Learning COTS Solutions". In: *Journal of Internet Services and Applications* 14 (Dec. 2023), pp. 189–199. DOI: 10.5753/jisa.2023.3551.

[7] Emmanuel J. Candès et al. "Robust Principal Component Analysis?" In: *Journal of the ACM* 58.3 (2011), 11:1–11:37. DOI: 10.1145/1970392.1970395.

[8] Chaos Mesh Authors. *Chaos Mesh: A Powerful Chaos Engineering Platform for Kubernetes*. https://github.com/chaos-mesh/chaos-mesh. GitHub repository; used for fault injection. 2024.

[9] Jian Chen et al. "TraceGra: A trace-based anomaly detection for microservice using graph deep learning". In: *Computer Communications* 204 (2023), pp. 109–117. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2023.03.028. URL: https://www.sciencedirect.com/science/article/pii/S0140366423001135.

[10] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density Estimation using Real-valued Non-Volume Preserving (Real NVP) Transformations". In: *International Conference on Learning Representations*. ICLR '17. 2017.

[11] Min Du. *DeepLog: Log Anomaly Detection via Deep Learning*. https://github.com/wuyifan18/DeepLog. GitHub repository (community implementation). 2017.

[12] Min Du et al. "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. ACM, 2017, pp. 1285–1298. DOI: 10.1145/3133956.3134015.

[13] Thijs van Ede. *DeepLog: PyTorch Implementation of DeepLog for the DeepCASE Project*. https://github.com/Thijsvanede/DeepLog. GitHub repository; part of the IEEE S&P DeepCASE research project. 2021.

[14] Chris Egersdoerfer, Di Zhang, and Dong Dai. "ClusterLog: Clustering Logs for Effective Log-based Anomaly Detection". In: *2022 IEEE/ACM 12th Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS)*. IEEE. 2022, pp. 1–7.

[15] GazeTheAbyss. *ISSRE23-TraceSieve: VGAE Model for Trace Anomaly Detection*. https://github.com/GazeTheAbyss/ISSRE23-TraceSieve. GitHub repository. 2023.

[16] Grafana Labs. *Loki: Like Prometheus, but for Logs*. https://github.com/grafana/loki. GitHub repository; log aggregation. 2024.

[17] Haixuan Guo, Shuhan Yuan, and Xintao Wu. "LogBERT: Log Anomaly Detection via BERT". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. 2021, pp. 1–8. DOI: 10.1109/IJCNN52387.2021.9534113.

[18] Haixuan Guo, Shuhan Yuan, and Xintao Wu. "LogBERT: Log Anomaly Detection via BERT". In: *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 33–41. DOI: 10.1109/BigData52589.2021.9671753.

[19] Haixuan Guo, Shuhan Yuan, and Xintao Wu. *LogBERT: Log Anomaly Detection via BERT*. https://github.com/HelenGuohx/logbert. GitHub repository. 2021.

[20] Jaeger Authors. *Jaeger: Open Source, End-to-End Distributed Tracing*. https://github.com/jaegertracing/jaeger. GitHub repository; distributed trace collection. 2024.

[21] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*. 2015. URL: https://arxiv.org/abs/1412.6980.

[22] Joanna Kosińska et al. "Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art". In: *IEEE Access* 11 (2023), pp. 73036–73052. DOI: 10.1109/ACCESS.2023.3281860.

[23] Junjun Li et al. "TraceDAE: Trace-Based Anomaly Detection in Microservice Systems via Dual Autoencoder". In: *IEEE Transactions on Network and Service Management* 22.5 (2025), pp. 4884–4897. DOI: 10.1109/TNSM.2025.3583213.

[24] Junjun Li et al. *TraceDAE: Trace-Based Anomaly Detection via Dual Autoencoder*. https://github.com/hellolijj/traceDAE. GitHub repository. 2025.

[25] Zhong Li, Jiayang Shi, and Matthijs van Leeuwen. "Graph Neural Networks based Log Anomaly Detection and Explanation". In: *arXiv preprint arXiv:2307.00527* (2024). Short version accepted at ICSE 2024 (poster track).

[26] Zhong Li, Jiayang Shi, and Matthijs van Leeuwen. *Logs2Graph: Graph Neural Networks for Log Anomaly Detection*. https://github.com/ZhongLIFR/Logs2Graph. GitHub repository. 2024.

[27] Zhouchen Lin, Minming Chen, and Yi Ma. *The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices*. Tech. rep. arXiv preprint arXiv:1009.5055. University of Illinois at Urbana-Champaign, 2010.

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. ICDM '08. IEEE, 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[29] Ping Liu et al. *TraceAnomaly: Unsupervised Microservice Trace Anomaly Detection*. https://github.com/NetManAIOps/TraceAnomaly. GitHub repository. 2020.

[30] Ping Liu et al. "Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks". In: *2020 IEEE 31st International Symposium on Software Reliability Engineering*. ISSRE '20. IEEE, 2020, pp. 48–58. DOI: 10.1109/ISSRE5003.2020.00014.

[31] João Bourbon Moreira et al. "Next generation of microservices for the 5G Service-Based Architecture". In: *Int. J. Netw. Manag.* 30.6 (Nov. 2020). DOI: 10.1002/nem.2132. URL: https://doi.org/10.1002/nem.2132.

[32] nailo2c. *DeepLog: PyTorch Implementation of DeepLog Anomaly Detection*. https://github.com/nailo2c/deeplog. GitHub repository; community implementation of Du et al. [12]. 2017.

[33] Prometheus Authors. *Prometheus: Monitoring System and Time Series Database*. https://github.com/prometheus/prometheus. GitHub repository; metric collection at 5-second intervals. 2024.

[34] Lukas Ruff et al. "A Unifying Review of Deep and Shallow Anomaly Detection". In: *Proceedings of the IEEE* PP (Feb. 2021), pp. 1–40. DOI: 10.1109/JPROC.2021.3052449.

[35] Lukas Ruff et al. "Deep One-Class Classification". In: *Proceedings of the 35th International Conference on Machine Learning*. ICML '18. 2018, pp. 4393–4402.

[36] Monika Steidl et al. "How Industry Tackles Anomalies during Runtime: Approaches and Key Monitoring Parameters". In: *2024 50th Euromicro Conference on Software Engineering and Advanced Applications*. SEAA '24. IEEE, 2024. DOI: 10.1109/SEAA64295.2024.00062.

[37] Ya Su et al. *OmniAnomaly: Robust Anomaly Detection for Multivariate Time Series*. https://github.com/NetManAIOps/OmniAnomaly. GitHub repository. 2019.

[38] Ya Su et al. "Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. ACM, 2019, pp. 2828–2837. DOI: 10.1145/3292500.3330672.

[39] Nagarjuna Telagam et al. "Beyond 5G: Exploring key enabling technologies, use cases, and future prospects of 6 G communication". In: *Nano Communication Networks* 43 (2025), p. 100560. ISSN: 1878-7789. DOI: https://doi.org/10.1016/j.nancom.2024.100560. URL: https://www.sciencedirect.com/science/article/pii/S1878778924000668.

[40] Quang Tung Thai, Myung-Eun Kim, and Namseok Ko. "On design and implementation of a cloud-native B5G mobile core network". In: *2023 IEEE 12th International Conference on Cloud Networking (CloudNet)*. 2023, pp. 477–482. DOI: 10.1109/CloudNet59005.2023.10490029.

[41] Daniel Tovarňák, Matúš Raček, and Petr Velan. "Cloud Native Data Platform for Network Telemetry and Analytics". In: *2021 17th International Conference on Network and Service Management (CNSM)*. 2021, pp. 394–396. DOI: 10.23919/CNSM52442.2021.9615568.

[42] Shikhar Tuli, Giuliano Casale, and Nicholas R. Jennings. *TranAD: Deep Transformer Networks for Anomaly Detection*. https://github.com/imperial-qore/TranAD. GitHub repository. 2022.

[43] Shikhar Tuli, Giuliano Casale, and Nicholas R. Jennings. "TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data". In: *Proceedings of the VLDB Endowment* 15.6 (2022), pp. 1201–1214. DOI: 10.14778/3514061.3514067.

[44] Peipeng Wang et al. "Unsupervised microservice system anomaly detection via contrastive multi-modal representation clustering". In: *Information Processing Management* 62.3 (2025), p. 104013. ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2024.104013. URL: https://www.sciencedirect.com/science/article/pii/S0306457324003728.

[45] Jiehui Xu et al. *Anomaly Transformer: Time Series Anomaly Detection*. https://github.com/thuml/Anomaly-Transformer. GitHub repository. 2022.

[46] Jiehui Xu et al. "Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy". In: *International Conference on Learning Representations*. ICLR '22. 2022.

[47] Wei Xu et al. "Detecting Large-Scale System Problems by Mining Console Logs". In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP '09. ACM, 2009, pp. 117–132. DOI: 10.1145/1629575.1629587.

[48] Chenxi Zhang et al. "DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning". In: *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 2022, pp. 623–634. DOI: 10.1145/3510003.3510180.

[49] Chenxi Zhang et al. "Efficient and Robust Trace Anomaly Detection for Large-Scale Microservice Systems". In: *2023 IEEE 34th International Symposium on Software Reliability Engineering*. ISSRE '23. No public code repository available. IEEE, 2023, pp. 69–79. DOI: 10.1109/ISSRE59848.2023.00023.

[50] Min Zhang et al. "Robust Log-Based Anomaly Detection on Unstable Log Data". In: *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE '19. ACM, 2019, pp. 807–817. DOI: 10.1145/3338906.3338931.