FALCON: A Framework for Fault Prediction in Open RAN Using Multi-Level Telemetry

Yaswanth Kumar LS*, Somya Jain*, Bheemarjuna Reddy Tamma*, and Koteswararao Kondepu†
* Indian Institute of Technology Hyderabad, India
† Indian Institute of Technology Dharwad, India
E-mail: {cs24resch11007, cs23mtech12011, tbr}@iith.ac.in, k.kondepu@iitdh.ac.in

***Abstract*—Open Radio Access Network (O-RAN) has brought in deployment flexibility and intelligent RAN control for mobile operators through its disaggregated and modular architecture using open interfaces. However, this disaggregation introduces complexities in system integration and network management, as components are often sourced from different vendors. In addition, the operators who are relying on open source and virtualized components — which are deployed on commodity hardware — require additional resilient solutions as O-RAN deployments suffer from the risk of failures at multiple levels including infrastructure, platform, and RAN levels. To address these challenges, this paper proposes FALCON, a fault prediction framework for O-RAN, which leverages infrastructure-, platform-, and RAN-level telemetry to predict faults in virtualized O-RAN deployments. By aggregating and analyzing metrics from various components at different levels using AI/ML models, the FALCON framework enables proactive fault management, providing operators with actionable insights to implement timely preventive measures. The FALCON framework, using a Random Forest classifier, outperforms two other classifiers on the predicted telemetry, achieving an average accuracy and F1-score of more than 98%.**

***Index Terms*—Open RAN, Network Telemetry, Resilient Networks, Fault Prediction**

I. Introduction

As mobile networks become increasingly complex and dynamic, the need for intelligent and adaptive network management has become crucial. Traditional Radio Access Network (RAN) solutions, designed primarily for earlier generations of cellular networks, struggle to meet the 5G use cases requirements due to their closed and monolithic nature. Vendor lock-in and proprietary hardware hinder flexibility, making it difficult for mobile operators to introduce new features, integrate diverse technologies or respond to rapid changes in traffic patterns and service demands [1]. Moreover, traditional RAN management relies on static, pre-defined configurations, lacking real-time optimization capabilities. As traffic demands fluctuate and dynamic services are introduced, the traditional RAN architectures lead to inefficiencies and increased operational costs. Fault management in traditional RAN is reactive, and addresses only after they cause significant disruptions, which makes it challenging to ensure continuous service availability [2].

The new RAN architectures, namely Open RAN (O-RAN) architecture offers a paradigm shift in how network services are deployed and managed [1]. The O-RAN provides a significant advantage by disaggregating RAN functions, by introducing RAN Intelligent Controller (RIC) and open interfaces among network functions that foster collaboration and interoperability among multiple vendors while allowing for greater flexibility in network deployments using Commercial-Off-The-Shelf (COTS) hardware and cloud native technologies. *Openness* in O-RAN facilitates innovation, enabling seamless integration of third-party apps and empowering the operators to leverage advanced Artificial Intelligence (AI)/Machine Learning (ML) algorithms for intelligent RAN control and automation.

A typical O-RAN 5G deployment requires diverse multi-vendor components, including COTS hardware, virtualization technologies, open-source and proprietary software for O-RAN functions such as Near-RT RIC, Open Centralized Unit (O-CU), Open Distributed Unit (O-DU), third-party xApps/rApps, and Service Management and Orchestration (SMO). This diversity poses integration and management challenges. Unlike previous generations tightly integrated components, O-RAN relies on multi-vendor solutions. While offering cost savings and flexibility, the disaggregated approach lacks built-in fault tolerance as these networks could be susceptible to faults, bugs, and misconfigurations and suffer from resource contentions and performance issues as shown in Table I. The link failures often require traffic rerouting, straining resources and reducing system stability. Ensuring resilience in this heterogeneous ecosystem requires advanced fault management, predictive analytics, and robust orchestration to mitigate challenges for widespread O-RAN adoption.

To address such challenges in O-RAN deployments, a precise monitoring system is crucial. Such a system shall provide transparent telemetry data by ensuring trust and accountability without vendor bias when attributing faults. This approach empowers operators to take timely preventive measures, creating an end-to-end resilient network across infrastructure, platform, and application (RAN) levels. In this work, we collect multi-level telemetry in virtualized O-RAN deployments, including infrastructure-level metrics (host-level telemetry), platform-level metrics (container-level telemetry), and application-level metrics (RAN telemetry).

This paper proposes a novel framework titled *FALCON* which aims to achieve self-resilience in O-RAN systems using this multi-level telemetry. The *FALCON* framework continuously collects and analyzes telemetry data at all levels, forecasting Key Performance Indicators (KPIs) to predict

TABLE I: Anomalous behaviors, their causes, and potential implications in virtualized O-RAN deployments

Anomalous behavior	Potential Faults/Issues	Potential Failures
Sudden spike in server temperature, decrease in CPU clock rate	Inadequate cooling, thermal throttling due to high workload, malfunctioning of fans or cooling units	Performance degradation of O-RAN NFs, CPU throttling leading to latency spikes, system shutdown, impact on UEs
Sudden increase in memory usage	Excessive logging by xApps or O-RAN NFs, memory leaks in software components, misconfigured processes or memory allocation	Crash of O-RAN NFs or containers, Disruption in UE sessions or service drops
High CPU usage, bit-rate spikes, increased traffic	Misconfiguration of routing policies, RAN overload	User Equipment (UE) disconnectivity, crash of O-RAN NFs
Reduced traffic, timeouts, Protocol Data Unit (PDU) re-establishments	Link faults on O-RAN interfaces, network congestion	Reduced throughput, Disruption in UE sessions or service drops
High CPU load, saturation of VSwitch port	VNF resource contention, high traffic volume, inefficient load balancing across NF instances	Application service crash, end-to-end service failures

faults in advance. This proactive monitoring enables operators to implement preemptive measures, addressing issues before they escalate into larger disruptions, thus maintaining network stability and performance.

The *FALCON* framework employs dimensionality reduction techniques combined with forecasting models to handle high-dimensional O-RAN telemetry data and thereby limits the focus only to the key features. It provides accurate and efficient anomaly predictions by classifying anomalous behavior to determine the type of fault, significantly reducing the resource overhead caused by the high-dimensional data of O-RAN telemetry. This approach enhances scalability, enabling FALCON to efficiently process large volumes of high-dimensional telemetry data from complex O-RAN deployments while maintaining performance as networks grow.

The rest of the paper is as organized follows: Section II presents the related work, followed by the proposed FALCON framework in Section III. Section IV describes the experimental setup, while Section V presents the performance results. Finally, Section VI provides the conclusions and future directions.

II. Related Work

This section reviews existing works on resilience in cellular networks and cloud environments, and presents research gaps.

The authors of [2] provided a comprehensive survey on fault management techniques used in traditional network deployments and also studied the impact of virtualization in network fault management. The authors of [3] used ML models to detect Service Level Agreement (SLA) violations in NFV environments and preliminary symptoms of SLAs violations so as to help the operators to locate anomalous VMs that caused SLA violations for applying appropriate recovery techniques in a proactive manner. In [4], the authors presented different stages of fault management process by using classical ML and neural network-based deep learning.

Live migration and recovery mechanisms are proposed in the literature to address failure of RAN and mobile core components. The authors of [5], [6] proposed live-migration schemes for both containerized mobile core and CU NFs in 5G. Atlas [7] ensures Distributed Unit (DU) resilience using proactive and reactive migrations through existing cellular mechanisms like handovers and cell reselection. Strategies such as pre-copy, post-copy, and hybrid migrations [8] are used for improving resiliency of specific RAN component(s) in these works. , yet challenges persist in achieving end-to-end resiliency across infrastructure, platform, and RAN levels in virtualized O-RAN deployments.

Misconfigurations are a significant source of faults in Open RAN systems. The authors of [9] categorize misconfigurations into three main categories: integration and operation related, SDN/NFV related, and AI/ML related. This work emphasizes the role of AI/ML based detection techniques, including anomaly detection for KPI analysis and correlation analysis to identify conflicting xApps in O-RAN. Similarly, systems like SpotLight [10] uses advanced anomaly detection techniques such as *JVGAN* and *MRPI*, achieving high accuracy and efficient bandwidth usage. SpotLight further incorporates explainability tools like *KFilter* and *CausalNex* for root cause analysis. While effective for anomaly detection and localization, these solutions do not offer broader application (RAN) level and infrastructure level fault management solutions. Also these works focus mainly on detecting configuration errors or faults based on RAN or platform level KPIs collected, but not on forecasting potential faults and failures for taking proactive measures to make the system self-resilient.

In summary, existing works in the literature excel in specific areas like cloud resource management, DU/CU recovery, and O-RAN anomaly detection, but they do not offer holistic solutions for predictive fault management across infrastructure, platform, and RAN levels. To address this gap, we propose FALCON, a holistic framework for virtualized O-RAN deployments.

III. Proposed FALCON Framework

Fig. 1 shows the architecture of the proposed *FALCON* framework. The O1 interface facilitates the operation and management (e.g., fault management) of O-RAN components, while the O2 interface manages the O-Cloud infrastructure and the life cycle of O-RAN network functions. In the proposed framework, the SMO periodically collects three types of telemetry data from the O-RAN system. It collects the managed element telemetry — also referred as application-level or RAN telemetry — from O-RAN components, including O-RU, O-DU, and O-CU, via the O1 interface. The platform telemetry and infrastructure telemetry are collected from the O-Cloud over O2 interface using exporters. The gathered telemetry data are stored in separate time-series databases within the SMO.

In this work, we collect diverse telemetry data from O-RAN components and the underlying O-Cloud to estimate network

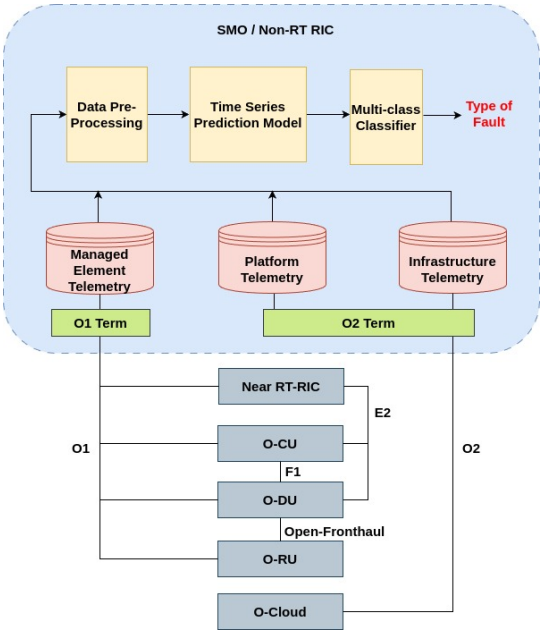

Fig. 1: Solution Architecture of FALCON Framework.

state. For example, consider a scenario where RAN telemetry shows increased packet loss (throughput degradation) and latency spikes. Analyzing only these metrics might lead to an incorrect diagnosis, such as partial link failures and network congestion faults. However, the network performance degradation may also happen due to host CPU frequency drops triggered by CPU temperature spikes. Hence, the FALCON framework considers the following multi-level telemetry data to prevent misdiagnosis.

- RAN Telemetry: It includes alarms and events for fault management, current configuration settings of the managed elements, KPIs related to network health and user-specific metrics, power metrics, etc.
- Platform Telemetry: This includes container-level CPU metrics, memory usage, network statistics, filesystem metrics, disk I/O metrics, and more which help in identifying container-specific performance issues such as bottlenecks, misconfigurations, or resource limitations, enabling efficient resource allocation and performance optimization within the containerized environment.
- Infrastructure Telemetry: This includes host-level CPU metrics, memory usage, network statistics, filesystem metrics, disk I/O metrics, and system health-related metrics which help to monitor the overall health of the host system by detecting hardware issues, excessive resource consumption, and performance degradation, ensuring the stability and reliability of the underlying O-Cloud infrastructure.

The collected telemetry data is pre-processed, where data from different levels is integrated and imputed to handle missing values. FALCON employs dimensionality reduction techniques to handle potentially large dimensionality of telemetry data while preserving its essential features and allowing scalability. In addition, dimensionality reduction helps to reduce computational cost while predicting telemetry values. Since RAN components are interlinked, the collected telemetry has both spatial and temporal dependencies. In order to preserve the spatial dependencies, telemetry's reduced features are given to forecaster model as a single input.

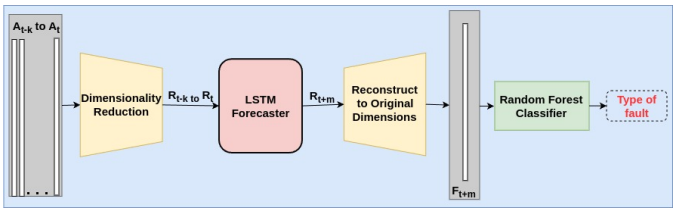

Fig. 2: Detailed ML pipeline used in FALCON.

From Fig. 2, $\mathbf{A}_{t-k}$ to $\mathbf{A}_t$ denote the preprocessed telemetry data vector spanning from time step $(t-k)^{\text{th}}$ to t^{th}, where $k > m$ and m is the future prediction step. Principal Component Analysis (PCA), a dimensionality reduction technique, is applied to this vector to obtain a set of reduced features, $\mathbf{R}_{t-k}$ to $\mathbf{R}_t$. These reduced features are provided as an input to the Long Short-Term Memory (LSTM) time-series forecasting model. The LSTM Forecaster model processes the sequential input through layers to capture the temporal dependencies and relationships within the data. Specifically, the LSTM model utilizes: (i) an LSTM layer with input size of $\mid \mathbf{R}_t \mid$, h hidden units, and some additional layers to learn the temporal and spatial patterns and (ii) a fully connected linear layer with h input features and $\mid \mathbf{R}_t \mid$ output features.

The LSTM model forecasts the future values of the reduced features at $(t+m)^{\text{th}}$ time step. A single vector $\mathbf{F}_{t+m}$ is reconstructed to original dimensions from $\mathbf{R}_{t+m}$, and subsequently fed to a multi-class Random Forest (RF) classifier. Since, RAN's anomalous behavior depends not only on CPU/Memory usage but also on the traffic generated by the UEs as well as number of UEs connected to the network, RF classifier is used as it makes decisions related to the type of fault based on lower/upper thresholds of multiple features.

IV. Experimental Setup

This section describes the experimental setup used to evaluate the performance of the proposed FALCON framework. It also includes details of the hardware and software tools used, how various faults are injected, what telemetry data is collected during the experiments, and the ML-based fault prediction pipeline.

A. Testbed Setup

We utilized a server equipped with two Intel Xeon ES-2690 v4 processors, providing a total of 54 cores, along with 64 GB of RAM and running Ubuntu 22.04 for our experimental setup. In this server, we integrate the O-RAN SC Near-RT RIC [11] together with the O-RAN components: srsCU, srsDU, and srsUE [12]. The srsCU resources restricted

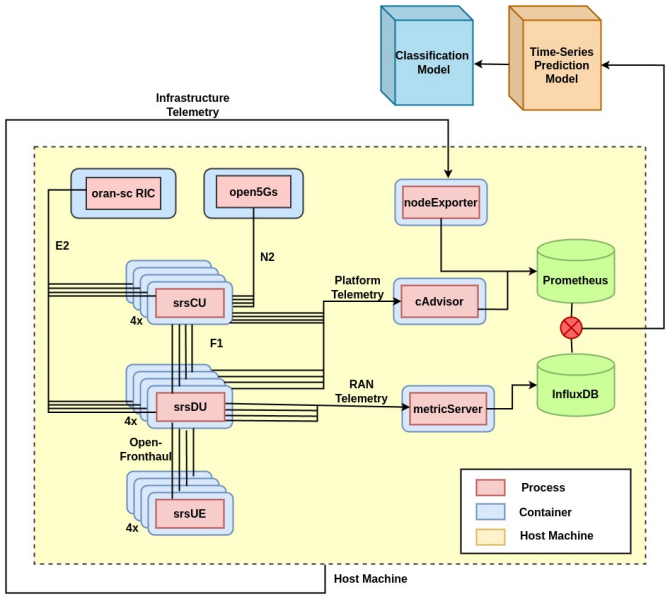

Fig. 3: Experimental Testbed.

with 3 CPU cores and 2 GB of RAM, while the srsDU had 3 CPU cores and 3 GB of RAM while other components are not resource restricted. Furthermore, Open5GS [13] was deployed as the core network. Each component is deployed as a *Docker container* to ensure modularity and scalability. In total, the testbed consists of four CUs, four DUs, and four UEs, each with a one-to-one mapping, as shown in Fig. 3 to create scenarios where only a subset of RAN components are induced with stress to study its impact on other RAN components.

Table II shows the details of telemetry collection from the containers by using node exporter [14], cAdvisor [15], and Metrics Server [12]. Prometheus [16] stores collected data from *cAdvisor* and *Node Exporter* at a frequency of one second. In contrast, InfluxDB [17] stores RAN telemetry from the Metrics Server, which receives updates from all DUs every 100 milli-sec as shown in Fig. 3.

Traffic generation: We generate user traffic by sending ping messages every 100 milli-sec to the core network, with the packet size randomly varying for every 5 seconds in such a way that the aggregate user traffic per hour follows the distribution as shown in Fig. 4 for both uplink and downlink. iPerf traffic is generated in the uplink direction from the users to fully occupy the channel with user traffic.

LSTM Forecaster Configuration: We configured the LSTM model with $\mid \mathbf{R}_t \mid = 10$, $h = 32$ which were defined in Section III to create an LSTM forecaster model with input size of 10, 32 hidden units, and two additional layers. Additionally a fully connected layer with 32 input features and 10 output features is attached to the two additional layers.

B. Fault Injection Approach

The fault injection process involves emulating different real-world faults in virtualized O-RAN deployments (refer to Table I) and collecting telemetry data at various levels for capturing anomalous system behavior due to these faults. In this work, we use tools like *stress-ng* [19] to stress-test various

TABLE II: Details of Telemetry Collection in FALCON Framework

Level	Tools Used	Type of Data Collected
RAN (DU) — Managed Element Telemetry	**Metric Server** collects metrics from DU	• Number of active UEs • Current total downlink bitrate • Maximum total downlink bitrate • Downlink bitrate • Uplink bitrate • Uplink Modulation and Coding Scheme (MCS) • Downlink MCS • Uplink Signal to Interference plus Noise Ratio (SINR) • Channel quality Indicator (CQI)
Platform (All Containers)	**cAdvisor Exporter** collects metrics from DUs, CUs	• CPU usage • Memory usage • Network statistics • Filesystem utilization • Container metrics as per [15]
Infrastructure (Host Machine)	**Node Exporter** collects from the host system	• CPU usage • Memory utilization • Disk I/O metrics • Filesystem statistics • Network metrics such as packet transmission rates, errors • Node temperature and power • Additional telemetry as per [18]

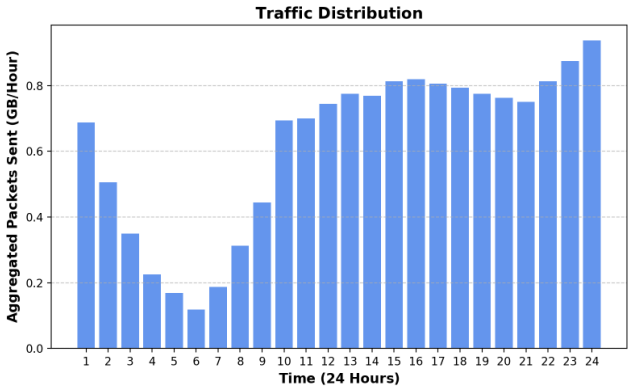

Fig. 4: Traffic distribution generated by ping messages.

instances of O-RAN components deployed in the considered testbed by forcing high CPU contentions and exhausting memory capacity. During this process, some packet loss may occur. Thus, to emulate the packet loss close to the real-time scenario, the *tc* tool is used to introduce some packet losses on the interface connecting RAN instances in our setup (refer to Fig. 3).

The various types of faults we inject are described in the following:

- **CPU stress test:** In this test, we stress the CPU cores assigned to different O-RAN components like CUs and DUs, and monitor the impact by collecting telemetry data at different levels as shown in Table II. Depending on the CPU resource demands of various components, each of these components is expected to show some negative impact of insufficient CPU resources in terms of anomalous behavior at certain stress levels, which could be captured through the telemetry data being collected at various levels in the system.
- **Memory stress test:** In this test, we stress the memory allocated to different CUs and DUs to mimic memory

Algorithm 1: Fault Injection Procedure

Require: $X = \mathcal{CU} \cup \mathcal{DU}$; // Set of CUs and DUs to be monitored in the O-RAN deployment

Sample fault duration $T_1 \sim \text{Exp}(\lambda) \mid T_1 \in [30, 90]$ minutes ; // Exp(λ) is an exponential distribution

Select fault type F using a multinomial distribution:

$$P_F(0) = 0.3,\ P_F(1) = 0.5,\ P_F(2) = 0.1,\ P_F(3) = 0.1$$

; // $F = 0$: Normal, $F = 1$: CPU Stress, $F = 2$: Memory Stress, $F = 3$: Packet Loss

for each container $C \in X$ **do**

Decide to inject stress using a Bernoulli distribution:

$$P(\text{Stress} = 1) = 0.4, P(\text{Stress} = 0) = 0.6$$

if Stress $= 1$ **then**

if $F = 1$ (CPU stress) **then**

Sample *Start stress* $\sim \mathcal{U}(0.4, 0.9)$; // CPU stress is selected between 40% and 90%

Sample *End stress* $\sim \mathcal{U}(\text{Start stress}, 1.0)$; // End stress is greater than or equal to Start stress

Execute CPU stress in container C from the selected Start stress to End Stress for T_1 duration

else if $F = 2$ (Memory stress) **then**

Sample *Start stress* $\sim \mathcal{U}(0.25, 0.35)$; // Memory stress is selected between 25% and 35%

Sample *End stress* $\sim \mathcal{U}(\text{Start stress}, 0.6)$

Execute Memory stress in container C from the selected Start stress to End Stress for T_1 duration

else if $F = 3$ (Packet loss) **then**

Sample *Start stress* $\sim \mathcal{U}(0.01, 0.03)$; // Packet loss is selected between 1% and 3%

Sample *End stress* $\sim \mathcal{U}(\text{Start stress}, 0.05)$

Execute Packet loss in container C from the selected Start stress to End Stress for T_1 duration

end if

else

No stress is injected into container C for T_1 duration

end if

end for

Repeat from Step 2 for the desired number of iterations

leaks and memory contentions. The negative impact is monitored by collecting telemetry data at different levels as shown in Table II.

- **Packet loss test:** In this test, we emulate partial link faults and congestion scenarios by introducing packet losses using the *tc* tool.

To inject these faults systematically in the considered setup, we follow the procedure given in Algorithm 1, where the selected O-RAN components are induced with stress for certain time windows.

Note that the data is collected continuously with normal and faults injection scenarios, with $65,765$ samples used for training and testing. As shown in Fig. 2, we created an ML pipeline with $k = 60$ and $m = 5$, where k represents the considered backward steps and m represents prediction step. The retraining of the considered ML pipeline depends on the topology changes, addition or deletion of features, etc.

C. Classifier Selection

We also evaluated two more classifiers (XGBoost and AdaBoost) along with RF for predicting faults based on the forecasted telemetry in the FALCON framework.

V. Performance Evaluation

The proposed *FALCON* framework was evaluated using stratified *5*-fold cross-validation to predict future telemetry and classify it as normal or fault types (CPU Stress, Memory Stress, and Packet Loss). The accuracy of *FALCON* shown in Table III is obtained using 10 PCA features from 403 initial features, which leads to $\approx 94.2\%$ reduction in features and the average RMSE value of 0.05789 is achieved for predicted features after inverse PCA (before de-normalization).

TABLE III: Averaged Performance of FALCON Across All Folds

Metric	Value
Accuracy	98.73%
F1-Score	98.71%
Macro Average Precision	96.70%
Macro Average Recall	97.00%
Macro Average F1-Score	96.56%
Weighted Average Precision	98.81%
Weighted Average Recall	98.73%
Weighted Average F1-Score	98.71%

A. Interpretation of Results

FALCON demonstrated exceptional performance across all folds, achieving an average accuracy of 98.73%. Table IV presents the average confusion matrix, which shows that the model accurately classified most of the samples for each class with minimal misclassifications. Notably, the *normal* and *CPU stress* classes were classified with very good accuracy, while a few misclassifications were observed for the *memory stress* and *packet loss* classes.

Fig. 5 shows *Accuracy*, *Precision*, *Recall*, and *F1-score* of FALCON for three different classifiers. Random Forest, which is employed by FALCON by default, outperforms XGBoost

TABLE IV: Average Confusion Matrix Across All Folds

True/Pred	Normal	CPU Stress	Memory Stress	Packet Loss
Normal	4693.2	2.0	7.8	1.2
CPU Stress	45.6	7012.4	50.4	7.4
Memory Stress	1.0	6.4	859.4	2.0
Packet Loss	8.0	3.6	31.2	421.4

and AdaBoost due to its ensemble nature, which allows it to generalize well to unseen data. XGBoost and AdaBoost struggled with outliers/noisy data and got skewed towards those anomalies as both focus on learning from mistakes made by previous iterations which leads to overfitting issues.

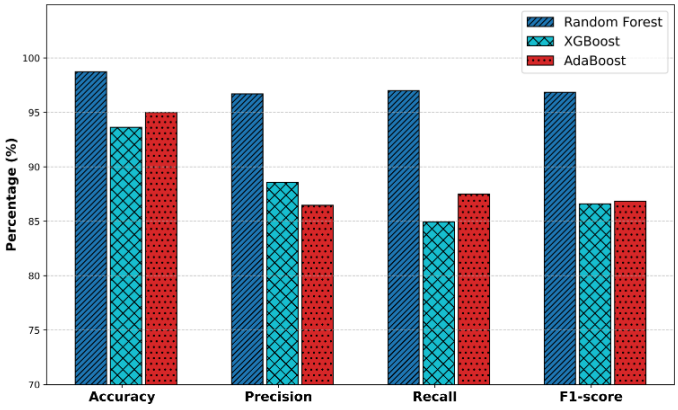

Fig. 5: Average Performance of Different Classifiers in FALCON Framework Across All Folds.

Table V provides the average classification report, highlighting FALCON's outstanding precision, recall, and F1-scores across all classes. The model achieved very good scores for the *normal* and *CPU stress* classes, with slightly lower but still good scores for *memory stress* and *packet loss* classes. Specifically, the F1-score for the CPU stress class was 99.18%, reflecting the framework's reliability even for challenging classifications.

TABLE V: Average Classification Report Across All Folds in FALCON Framework

Class	Precision	Recall	F1-Score
0 (Normal)	0.9888	0.9976	0.9931
1 (CPU Stress)	0.9982	0.9854	0.9918
2 (Memory Stress)	0.9109	0.9891	0.9471
3 (Packet Loss)	0.9702	0.9077	0.9304

VI. Conclusions and Future Work

Designing a fault-proof system that uses anomalies to predict faults in advance and takes corrective measures is challenging due to the high dimensionality of telemetry data in virtualized O-RAN deployments. FALCON, our proposed framework, addresses this challenge by using PCA for dimensionality reduction and LSTM forecaster for KPI forecasting. It successfully forecasts anomalies to predict faults up to 5 seconds (or 5 time steps) in advance, enabling proactive decision-making by the operators. FALCON demonstrated strong performance with an average accuracy of 98.73% and F1-score of 98.71%, across all folds of the stratified n-fold cross-validation. The confusion matrix and classification report show minimal misclassifications, highlighting its robustness.

Future work involves creation of a dataset on an emulated testbed consisting of a large number of O-RAN components and UEs to mimic real-world deployment scenarios with a greater number of faults and simultaneous fault cases. Subsequently, we plan to focus on root cause analysis to localize faults to be able to take preventive measures in a timely manner.

Acknowledgments

This work was partially supported by Intel India and Visvesvaraya PhD Scheme, Meity, Govt. of India. The authors would like to thank Abdulla Ovais and Michael Suguna Kumar V for their help in designing ML pipeline of the FALCON framework.

References

[1] "O-RAN: Towards an Open and Smart RAN," O-RAN Alliance, Tech. Rep., 2018.
[2] S. Cherrared, S. Imadali, E. Fabre, G. Gössler, and I. G. B. Yahia, "A Survey of Fault Management in Network Virtualization Environments: Challenges and Solutions," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, 2019.
[3] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Anomaly detection and root cause localization in virtual network functions," in *IEEE International Symposium on Software Reliability Engineering*, 2016.
[4] S. Mukherjee, O. Coudert, and C. Beard, "An open approach to autonomous ran fault management," *IEEE Wireless Communications*, vol. 30, no. 1, pp. 96–102, 2023.
[5] S. Ramanathan, K. Kondepu, and A. Fumagalli, "Resiliency in Open-Source Solutions for Disaggregated 5G Cloud Radio Access and Transport Networks," in *IEEE NFV-SDN*, 2022, pp. 124–129.
[6] S. Ramanathan, A. Bhattacharyya, K. Kondepu, and A. Fumagalli, "Enabling containerized Central Unit live migration in 5G radio access network: An experimental study," *Journal of Network and Computer Applications*, vol. 221, p. 103767, 2024.
[7] J. Xing, J. Gong, X. Foukas, A. Kalia, D. Kim, and M. Kotaru, "Enabling Resilience in Virtualized RANs with Atlas," in *Proceedings of ACM Mobicom*, 2023, pp. 1–15.
[8] A. Bhattacharyya, S. Ramanathan, A. Fumagalli, and K. Kondepu, "Towards Disaggregated Resilient 5G Radio Access Network: A Proof of Concept," in *IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE, 2023, pp. 396–401.
[9] N. M. Yungaicela-Naula, V. Sharma, and S. Scott-Hayward, "Misconfiguration in O-RAN: Analysis of the impact of AI/ML," *Computer Networks*, vol. 247, p. 110455, 2024.
[10] C. Sun, U. Pawar, M. Khoja, X. Foukas, M. K. Marina, and B. Radunovic, "SpotLight: Accurate, explainable and efficient anomaly detection for Open RAN," in *Proceedings of ACM Mobicom*, 2024, pp. 923–937.
[11] "O-RAN Software Community (SC) Near-Real-time RIC (i-release)," https://github.com/srsran/oran-sc-ric, 2025.
[12] "srsRAN Project," https://www.srsran.com/5g.
[13] "open5gs," https://github.com/open5gs/open5gs, 2025, release v2.7.2.
[14] "nodeexporter," https://hub.docker.com/r/prom/node\protect\discretionary{\char\hyphenchar\font}{}{}exporter, 2025, release v2.7.2.
[15] "cAdvisor," https://github.com/google/cadvisor, 2024.
[16] "prometheus," https://prometheus.io/, 2025, release 3.1.0.
[17] "influxdb," https://www.influxdata.com/, docker Influx Version 2.7.
[18] "Node Exporter," https://github.com/prometheus/node_exporter, 2024.
[19] "stressng," https://github.com/ColinIanKing/stress\protect\discretionary{\char\hyphenchar\font}{}{}ng, 2025, release V0.18.09.