Design and Implementation of an AI-based IDS xApp for Open RAN*

Hyeonsoo Yu, Suhyeon Lee and Hwankuk Kim†

Kookmin University, Seoul, South Korea
{suya666, gus9895, rinyfeel}@kookmin.ac.kr

Abstract

This study aims to verify the operational feasibility of a near-real-time (Near-RT) intrusion detection system (IDS) xApp in the O-RAN (Open Radio Access Network) environment. For this purpose, the NetsLab 5G ORAN IDD dataset, which contains both benign and attack traffic collected from the O-CU (network layer) and O-DU (radio telemetry layer), is utilized. The preprocessed data are fed into the AIML Framework pipeline to train LSTM (Long Short-Term Memory), CNN (Convolutional Neural Network), Transformer, and Autoencoder models. The trained models are stored in the model repository (LeoFS) within the Near-RT RIC and then loaded into the IDS xApp to perform near-real-time inference. Experimental results show that CNN is the most operation-friendly model in terms of balancing performance and resource efficiency, while Transformer demonstrated high performance but with significant resource consumption. The performance difference between the pipeline and the IDS xApp was negligible, and although latency increased quasi-linearly with the number of prediction instances, all models satisfied the Near-RT constraints (10 ms–1 s). Specifically, in the experiment using O-CU data, even in sections with more than 20,000 prediction instances, the CNN and LSTM models exhibited an average inference latency of approximately 0.2 seconds, while the Transformer showed about 0.5 seconds, all stably satisfying the Near-RT constraint of 1 second. These findings demonstrate that the IDS xApp can go beyond simple detection to support RAN control and optimization in Near-RT environments.

Keywords: Open RAN, Intrusion Detection System, AIML Framework, Near-RT RIC, xApp

1 Introduction

O-RAN (Open Radio Access Network) is a next-generation architecture for constructing RAN, based on openness and intelligence, in which functionalities are disaggregated into Central Unit (CU), Distributed Unit (DU), and Radio Unit (RU), while interoperability is ensured through standardized interfaces [14]. However, this disaggregated structure and the E2/KPM-based data flow can serve as new attack vectors [1]. In particular, communication and telemetry data generated at the O-CU (network layer) and O-DU (radio telemetry layer) are critical targets for detecting benign and malicious behaviors [5], and can be leveraged as primary analytical units for Intrusion Detection Systems (IDS) [8][3].

* Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. 49, December 16-18, 2025, Sapporo, Japan. © The copyright of this paper remains with the author(s).

† Corresponding author

To address such security threats, this study proposes a system that integrates the AIML Framework-based training and deployment pipeline with an IDS xApp operating within the Near-RT RIC. The proposed system consists of training based on O-CU and O-DU data using a public dataset (NetsLab 5G ORAN IDD) [2], model training and storage through the AIML Framework pipeline, and near-real-time inference via IDS xApp implementation within the Near-RT RIC.

The contributions of this study are as follows. First, by training LSTM, CNN, Transformer, and Autoencoder models in the AIML Framework pipeline, we quantitatively compare model performance (Accuracy, Precision, Recall, F1-score) and resource consumption (RAM/CPU energy consumption). Second, by deploying the trained models in the Near-RT RIC IDS xApp and measuring latency under varying numbers of prediction instances (9 stages at O-CU, 7 stages at O-DU), we experimentally verify that all models satisfy the Near-RT constraints (10 ms–1 s) [4].

This paper is organized into five sections. Chapter 1 presents the introduction; Chapter 2 describes the architecture of Open RAN and related studies; Chapter 3 explains the proposed IDS xApp system; Chapter 4 presents the experimental methodology and results; and finally, Chapter 5 discusses the conclusion and future research directions.

2 Background and Related Work

2.1 Open RAN Architecture

Figure 1 shows the O-RAN SC architecture defined by the O-RAN Alliance. It illustrates the separation between the radio side and the management side, as well as the standardized interfaces connecting them.

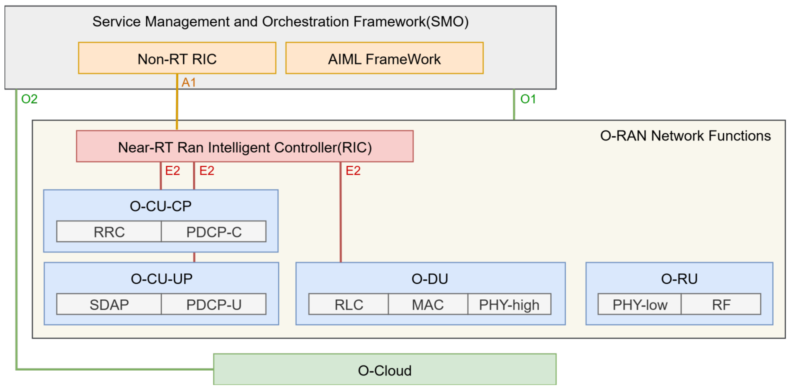

Figure 1: O-RAN Reference Architecture

1) Overview of the Radio Interface Protocol Stack

The radio interface between the base station (BS) and the user equipment (UE) consists of the PHY (Physical Layer), MAC (Medium Access Control), RLC (Radio Link Control)/PDCP (Packet Data Convergence Protocol), and RRC (Radio Resource Control) layers. In 5G, the Central Unit (CU) is separated into a control plane (CU-CP) and a user plane (CU-UP), with the SDAP layer added compared to LTE. In O-RAN, these are further extended into O-CU-CP, O-CU-UP, O-DU, and O-RU, and are interconnected with the RIC through the E2 interface [7].

2) Separation of RIC and the Three-Tier Control Loop

The RAN Intelligent Controller (RIC) is divided into three control loops according to the time scale: Non-RT, Near-RT, and Real-Time. The Non-RT RIC operates on a long-term timescale of more than one second, performing policy control and AI/ML training, and provides policy guidance to the Near-RT RIC via the A1 interface. The Near-RT RIC operates within a range of approximately 10 ms to 1 s, collecting data and performing control through xApps to optimize O-RAN resources in near real time. Finally, the Real-Time control loop operates in the ultra-low latency range of less than 10 ms, directly controlling the PHY/MAC layers at the hardware and firmware level. This three-tier control loop structure is a core design principle of O-RAN, enabling efficient management and optimization of RAN resources by separating functions according to the timescale [18].

3) Definition of O-RAN Components

O-RAN consists of five main components: O-CU (Central Unit), O-DU (Distributed Unit), O-RU (Radio Unit), SMO (Service Management and Orchestration), and RIC (RAN Intelligent Controller: Non-RT RIC and Near-RT RIC). Each component is interconnected through standardized interfaces. For example, O-CU, O-DU, and O-RU separate and connect radio access functions via interfaces such as F1, E1, and Open Fronthaul, while the Near-RT RIC controls the O-CU and O-DU through the E2 interface. The Non-RT RIC provides policies and AI/ML models to the Near-RT RIC using the A1 interface, and the SMO performs network management and orchestration through the O1 interface [17].

Component		Description
Near-RT RIC		A controller that performs RAN control and optimization in the 10 ms–1 s range via xApps.
Non-RT RIC		A controller that manages policies and AI models on timescales longer than 1 second and provides policies via the A1 interface.
O-CU	O-CU-CP	Control-plane node that hosts RRC and PDCP-C (control plane).
	O-CU-UP	User-plane node that hosts PDCP-U (user plane) and SDAP.
O-DU		Logical node that hosts the RLC/MAC/High-PHY layers.
O-RU		Logical node that hosts Low-PHY and RF processing (functionally similar to a 3GPP TRP/RRH). Examples of Low-PHY include FFT/iFFT and PRACH extraction.

Table 1: O-RAN Components

2.2 Related Work

Open RAN presents an innovative industry standard for the Radio Access Network (RAN) and offers the advantages of promoting vendor interoperability and network flexibility through open interfaces. However, due to these inherent characteristics, it faces significant security issues.

Hexuan Yu et al. argue that current O-RAN has structural vulnerabilities whereby malfunctioning or compromised components in real operational environments may violate policies without being detected, and they propose a monitoring framework for low-trust O-RAN environments to address this. The framework provides scalable and verifiable supervisory functions and achieved a total processing delay of approximately 200 ms [6].

Prudhvi Kumar Kakani et al. present the severity of adversarial attacks (C&W, BIM) targeting xApps in O-RAN environments and evaluate attack performance. The C&W attack sharply reduced the target xApp's detection accuracy from 92% to 16%, and the BIM attack likewise lowered detection accuracy to about 10% [10].

Recent studies have also been actively pursuing IDS research to solve O-RAN's security problems. Joshua Moore et al. emphasize the need for advanced security measures to maintain the integrity, confidentiality, and availability of O-RAN, and present an intrusion detection (ID) xApp using large language models (LLMs). The proposed xApp generates security recommendations by leveraging time-based traffic patterns of connected UEs, and to demonstrate its utility, compares a non-fine-tuned model with a fine-tuned model to verify task-specific accuracy. As a result, after fine-tuning the Gemma 2 model, it achieved 99% accuracy, maintained an average detection/response time of 239.66 ms, and identified an intrusion with a single KPM report at 175 seconds, demonstrating faster and more accurate performance compared to static threshold-based methods [15].

Alan Civciss et al. emphasize the need for low-latency (edge-resident) security mechanisms as the architecture evolves into a disaggregated open RAN (O-RAN), and propose a real-time IDS system that combines dApps and xApps. To resolve the issue that xApp-based IDS relies on deep packet inspection and centralized analysis—leading to high computational overhead and latency—the study proposes a lightweight dApp deployed at the Distributed Unit (DU), enabling near-real-time threat detection at the radio edge. The proposed system identifies network attacks without inspecting higher-layer packet data by utilizing radio telemetry features collected at the E2 interface. The dApp-based IDS achieves detection accuracy close to 100%, the same as the xApp, but with much shorter execution time in the millisecond range, CPU consumption of 4.53% vs. 121.78%, and about 50% of the memory consumption, thereby being significantly more lightweight [2].

Building on these recent research trends, this study proposes an IDS xApp that bridges the AIML Framework and the Near-RT RIC to address O-RAN's security challenges. In addition, it aims to conduct quantitative verification under identical model and data conditions, from training based on a public dataset to xApp inference.

3 Proposal System

Figure 2 illustrates the proposed system and experimental design. The proposed system consists of two paths: a training path and an inference path.

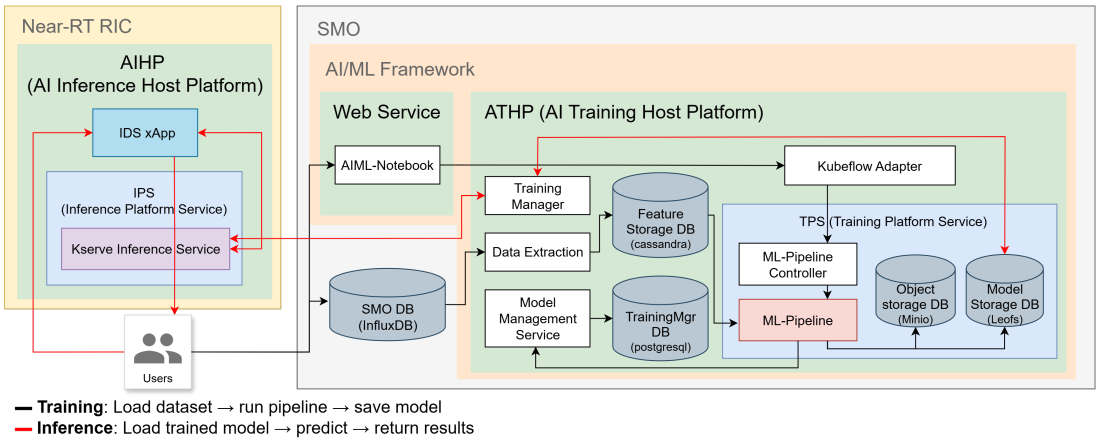

Figure 2: Near-RT RIC IDS xApp – From Training to Inference

The training path in this study is as follows. First, data stored in the SMO's InfluxDB are extracted and refined, then delivered to the Feature Storage (DB: Cassandra) of the AIML Framework. The user submits a pipeline specification in the AIML-Notebook, which is passed through the Kubeflow Adapter and ML-Pipeline Controller to the Kubeflow Pipeline, where preprocessing and training are performed sequentially. Once training is completed, the model artifacts are stored in the Model Storage (LeoFS), additional outputs such as logs and configurations are stored in the Object Storage (MinIO), and the related metadata are recorded in the TrainingMgr DB (PostgreSQL). In addition, the Model Management Service manages the version and URI of the latest model, so that they can be referenced in the subsequent inference stage.

The inference path works as follows. The KServe Inference Service on the Near-RT RIC receives reference information from the Training Manager and loads the trained model from the Model Storage (LeoFS). The IDS xApp sends prediction requests with input datasets to the KServe Inference Service, which performs model inference and returns the results to the IDS xApp. Using these prediction results, the IDS xApp can detect malicious activities occurring in the O-CU/DU. Furthermore, the system collects and monitors operational indicators such as prediction latency and resource consumption, thereby confirming that the proposed architecture operates stably under Near-RT constraints (approximately 10 ms–1 s).

4 Experiments and Evaluation

4.1 Dataset: NetsLab 5G ORAN IDD

NetsLab 5G ORAN IDD is a public dataset collected from an OAI-based 5G O-RAN testbed (O-CU, O-DU, O-RU, Near-RT RIC, UE), containing both benign and malicious traffic simultaneously. Benign traffic consists of video streaming, downloads, web browsing, system updates, and iPerf3 control traffic, while malicious traffic is generated and collected using various tools to launch DoS/DDoS, scanning, brute-force, and web attacks (e.g., SQLi, XSS). At the network layer (O-CU), 26 session- and flow-based features are provided through pcap and Zeek logs, and at the radio layer (O-DU), 25 PHY/MAC indicators are collected and provided. After preprocessing, this data is stored in the Feature Store and used for training LSTM, CNN, Transformer, and Autoencoder models. The trained models are stored in LeoFS and employed for IDS xApp inference experiments.

O-CU		O-DU	
Feature	Meaning	Feature	Meaning
uid	Unique Identifier	dlBytes	Downlink Bytes
src_ip	Source IP Address	dlMcs	Downlink Modulation and Coding Scheme
src_port	Source Port	dlBler	Downlink Block Error Rate
dst_ip	Destination IP Address	ulBytes	Uplink Bytes
dst_port	Destination Port	ulMcs	Uplink Modulation and Coding Scheme
proto	Protocol	ulBler	Uplink Block Error Rate
service	Application Service	ri	Rank Indicator
duration	Connection Duration	phr	Power Headroom Report
src_bytes	Source Bytes	pcmax	Maximum Power Capability
dst_bytes	Destination Bytes	rsrq	Reference Signal Received Quality
conn_state	Connection State	sinr	Signal to Interference plus Noise Ratio
missed_bytes	Missed Bytes	rsrp	Reference Signal Received Power
history	Packet History	rssi	Received Signal Strength Indicator
src_pkts	Source Packets	cqi	Channel Quality Indicator
src_ip_bytes	Source IP Bytes	pucchSnr	Physical Uplink Control Channel SNR
dst_pkts	Destination Packets	puschSnr	Physical Uplink Shared Channel SNR
dst_ip_bytes	Destination IP Bytes	ue_id	User Equipment Identifier
ip_proto	IP Protocol	timestamp	Data Collection Time
http_trans_depth	HTTP Transaction Depth	cellid	Cell Identifier
attack_category	Attack Category	in_sync	Synchronization Status
attack_type	Attack Type	rnti	Radio Network Temporary Identifier
files_total_bytes	Total File Bytes	pmi	Precoding Matrix Indicator
is_GET_mthd	HTTP GET Method	traffic_type	Traffic Type
http_status_error	HTTP Status Error	attack_category	Attack Category
is_file_transferred	File Transferred	attack_ subcategory	Attack Subcategory
traffic_type	Traffic Type		

Table 2: Feature list by data type in the NetsLab 5G ORAN IDD dataset

4.2 Experimental method

In this study, preprocessing was performed on CU (network layer) and DU (radio telemetry layer) data following the same procedure. First, to unify the model inputs into numerical form, features with dtype=object were excluded. In the case of CU, 9 out of 26 features—{uid, src_ip, dst_ip, proto, service, conn_state, history, attack_category, attack_type}—were excluded, leaving 17 refined features. For DU, 4 out of 25 features—{rnti, pmi, attack_category, attack_subcategory}—were excluded, reducing the set to 21 features. Subsequently, Min–Max normalization was applied to all features in order to mitigate training instability caused by scale differences among features and to prevent the dominance of specific variables [11]. The formula is as follows, where $x_i{'}$ is the normalized value, x_i is the original value, and x_{min}, x_{max} are the minimum and maximum values of the dataset, respectively [12].

$$x_i{'} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

To reduce the impact of class imbalance, the ratio of benign to attack traffic was adjusted to 1:1 (5:5), and the preprocessed data were split into training/validation/test sets at a ratio of 6:2:2. To ensure reproducibility, shuffle=True and random_state=7 were fixed. Finally, to secure input format compatibility with sequence models (LSTM, 1D CNN, Transformer) and to unify the pipeline interface, the input dimension was reshaped from (N, F) to (N, 1, F).

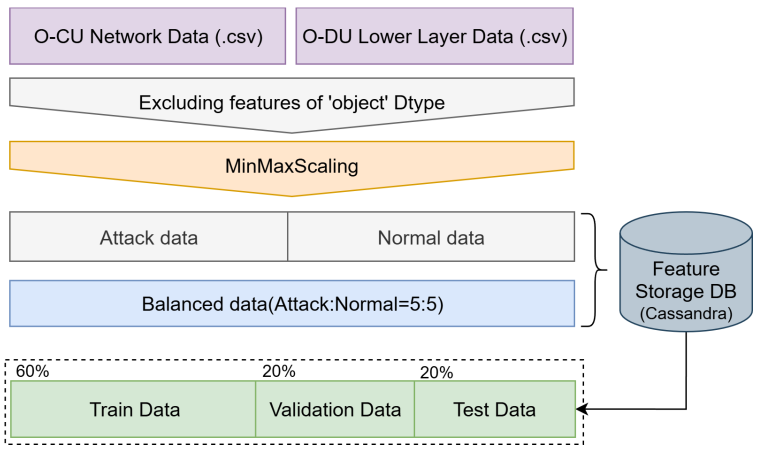
O-CU Network Data (.csv)
O-DU Lower Layer Data (.csv)
Excluding features of 'object' Dtype
MinMaxScaling
Attack data
Normal data
Balanced data(Attack:Normal=5:5)
Feature Storage DB (Cassandra)
60%
20%
20%
Train Data
Validation Data
Test Data

Figure 3: Schematic diagram of the data preprocessing process for NetsLab 5G ORAN IDD

Four models—LSTM, CNN, Transformer, and Autoencoder—were trained within the same AIML Framework pipeline. The trained models were stored in the repository (LeoFS) in the form of model.zip and subsequently loaded by the IDS xApp in the Near-RT RIC to perform predictions. The experiments were divided into two stages:

1. Resource Consumption Evaluation:

   Measuring the RAM and CPU energy consumption of each model in the pipeline environment.

   Measurements were directly obtained using the CodeCarbon library to calculate RAM/CPU energy consumption.

2. Performance and Inference Evaluation:

A. Comparison of prediction performance between the pipeline and the Near-RT RIC IDS xApp under the same number of prediction instances.

B. Measurement of inference latency in the xApp environment by varying the number of prediction instances.

The hardware and software stack were kept identical across all models to avoid bias in comparisons due to resource or environmental factors. In the xApp environment, prediction time was measured as the average from the 2nd to the 11th iteration, excluding the first warm-up run. In particular, measurements for varying numbers of prediction instances were conducted differently depending on the data type:

- O-CU data: 500, 1000, 1500, 5000, 10000, 15000, 20000, 25000, 30000 (9 stages in total)
- O-DU data: 500, 1000, 1500, 5000, 10000, 15000, 20000 (7 stages in total)

4.3 Evaluation Indicators

In the AIML Framework pipeline, RAM/CPU energy consumption (Wh) and prediction latency (ms) are used to evaluate operational efficiency. Energy consumption is obtained by actually retrieving the values carbon_data.ram_energy and carbon_data.cpu_energy using the CodeCarbon library. The CodeCarbon functions calculate energy consumption in Wh based on RAM/CPU resource consumption. This can be formulated as follows.

$$E_{RAM} = P_{RAM} \times T$$

$$E_{CPU} = P_{CPU} \times T$$

E_{RAM} , E_{CPU} denote RAM/CPU energy consumption (Wh), and P_{RAM} , P_{CPU} denote the average power consumption (W) of RAM/CPU. T is the execution time (h). In the CodeCarbon library, this is measured automatically and returned as $E_{total} = E_{RAM} + E_{CPU}$ [13].

Binary classification performance is computed as follows [16].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Here, TP (True Positive) refers to cases where an attack is correctly detected, TN (True Negative) refers to cases where benign traffic is correctly classified, FP (False Positive) refers to cases where benign traffic is incorrectly detected as an attack, and FN (False Negative) refers to cases where an attack is missed as benign.

The formula for inference latency is as follows [9]. In the experiments, excluding the first warm-up run, the average over runs 2–11 was used.

$$L_{avg} = \frac{1}{N}\sum_{i=2}^{N+1} L_i$$

L_{avg}denotes the average inference latency (ms), $L_i$denotes the latency (ms) of the i-th inference run, and N denotes the number of valid repetitions (= 10, i.e., runs 2–11). In this paper, the operational suitability of the proposed system is interpreted as a balance between performance metrics and prediction cost.

4.4 Experiment Results

4.5.1 RAM and CPU Consumption using the pipeline in AIML Framework

Two virtual operating systems were configured in the experimental environment. The SMO hosting the AIML Framework ran on Ubuntu 22.04 LTS (64-bit), while the Near-RT RIC operated on Ubuntu 20.04 LTS (64-bit). For reference, the system specifications were based on an AMD Ryzen 9 7900X 12-Core Processor and 128 GB of RAM.

In the pipeline environment, RAM/CPU energy consumption (Wh) was measured for LSTM, CNN, Transformer, and Autoencoder. The measurements were obtained by using the CodeCarbon function to compute ram_energy and cpu_energy, and the resource consumption characteristics were quantitatively compared by model.

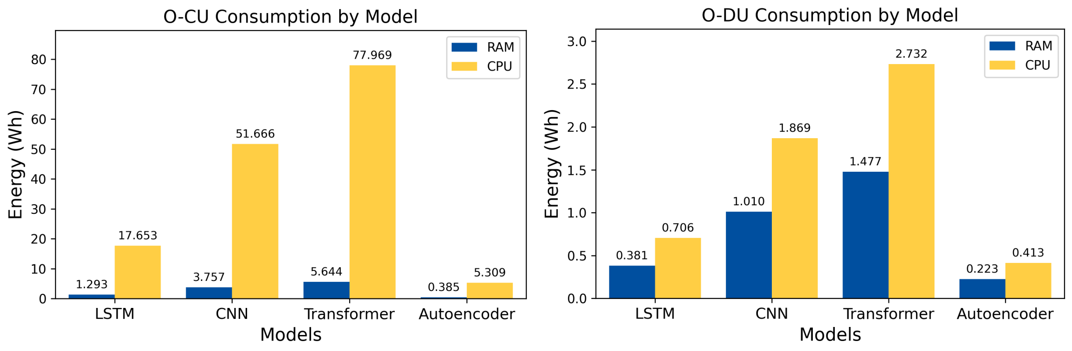

Figure 4: RAM and CPU energy figures for each model in the AIML Framework pipeline

In the O-CU environment, CPU energy consumption was highest for CNN (1.869 Wh) and Transformer (2.732 Wh), while LSTM (0.706 Wh) and Autoencoder (0.413 Wh) showed relatively low values. RAM energy likewise tended to be higher for Transformer (1.477 Wh) and CNN (1.010 Wh) than for LSTM (0.381 Wh) and Autoencoder (0.223 Wh). In other words, while Transformer and CNN deliver higher performance, they also incur substantial resource consumption, whereas Autoencoder and LSTM are more resource-efficient. A similar trend was observed in the O-DU environment. Transformer's CPU consumption was the highest at 77.969 Wh, followed by CNN at 51.666 Wh, LSTM at 17.653 Wh, and Autoencoder at 5.309 Wh. RAM consumption also increased in the order of Transformer (5.644 Wh), CNN (3.757 Wh), LSTM (1.293 Wh), and Autoencoder (0.385 Wh).

4.5.2 Performance Comparison of Pipeline and IDS xApp

The table below (Table 3) presents the pipeline training performance for each model.

Prediction Location	Data Type	Training Model	Accuracy	F1-Score	Precision	Recall
AIML Framework Pipeline	CU	LSTM	0.9968	0.9952	0.9988	0.9916
		CNN	0.9970	0.9955	0.9994	0.9916
		Transformer	0.9971	0.9956	0.9997	0.9916
		Autoencoder	0.5000	0.6666	0.5000	0.9999
	DU	LSTM	0.9946	0.9946	0.9981	0.9911
		CNN	0.9590	0.9606	0.9242	1.0000
		Transformer	0.9962	0.9962	0.9986	0.9940
		Autoencoder	0.5002	0.6667	0.5001	0.9995
Near-RT RIC IDS xApp	CU	LSTM	0.9561	0.9585	0.9534	0.9560
		CNN	0.9699	0.9878	0.9516	0.9693
		Transformer	0.9389	0.9338	0.9448	0.9393
		Autoencoder	0.5173	0.6739	0.5088	0.9976
	DU	LSTM	0.9970	0.9940	1.0000	0.9970
		CNN	0.9962	1.0000	0.9924	0.9961
		Transformer	0.9961	0.9997	0.9924	0.9960
		Autoencoder	0.8718	0.8576	0.8916	0.8743

Table 3: Performance of Four Models in the AIML Framework Pipeline and Near-RT RIC IDS xApp for O-CU/DU Data

In this section, we provide a narrative summary of model-wise classification performance for CU/DU data in the pipeline and xApp environments. First, in the pipeline–CU case, Transformer and CNN showed top-tier performance (Accuracy=0.9971/0.9970, F1=0.9956/0.9955), and LSTM followed by a small margin (Accuracy=0.9968, F1=0.9952), with all three models recording very high accuracy and well-balanced metrics. In contrast, the Autoencoder exhibited very high Recall but low Precision, revealing a clear tendency toward false positives and resulting in lower performance than the other models.

In the pipeline–DU setting, Transformer consistently maintained a top position (Accuracy=0.9962, F1=0.9962), and LSTM also showed high performance (Accuracy=0.9946, F1=0.9946). According to the table, CNN is reported with Accuracy=0.9590, Precision=0.9242, and Recall=1.0, indicating relatively lower accuracy and precision. As in the CU case, the Autoencoder remained at the level of Accuracy=0.5002 and F1=0.6667 due to high Recall but low Precision.

In the IDS xApp–CU environment, CNN delivered the most stable performance (Accuracy=0.9699, F1=0.9878), and LSTM recorded a similar level (Accuracy=0.9561, F1=0.9585). Transformer showed relatively lower metrics in this configuration (Accuracy=0.9389, F1=0.9338). Although the

Autoencoder achieved high Recall (=0.9976), its Precision (=0.5088) was low, resulting in a limited F1=0.6739.

In IDS xApp–DU, LSTM, CNN, and Transformer all exhibited very high performance, close to the upper bound. Interestingly, the Autoencoder showed relatively improved results in this setting—Accuracy=0.8718, F1=0.8576, Precision=0.8916, Recall=0.8743—indicating a better balance between precision and recall compared to other environments.

In summary, the model ranking and performance levels observed in the pipeline are largely reproduced in the IDS xApp environment, demonstrating strong performance preservation through the training–deployment transition. CNN and Transformer generally remain at the top, while LSTM shows particular strengths for DU-based operation where lightness and stability are required. The Autoencoder consistently provides high detection sensitivity (Recall), but low Precision necessitates managing false positives. Notably, although CNN's accuracy was reported to be relatively low only in pipeline–DU, it returned to a top-tier level in xApp–DU, confirming that in the DU segment of real operational environments, all three supervised models perform at levels close to the upper bound. Overall, performance preservation from the pipeline to the xApp is excellent. CNN is suitable as the default choice in terms of balancing performance and stability, and LSTM can be a practical alternative when aiming for lightweight DU-based operation. Transformer provides high performance, but—as presented in the following section—its costs in terms of resource consumption and latency should be considered. The Autoencoder's strength lies in high detection sensitivity, but a strategy to compensate for Precision—such as threshold tuning or post-processing—is required.

4.5.3 IDS xApp Prediction Latency by Number of Predicted Instances

Figure 5 shows a graph of inference latency as the number of prediction instances increases. For both datasets, inference latency increased quasi-linearly. The O-CU and O-DU exhibited the same trend, and the Near-RT requirement (approximately 10 ms–1 s) was satisfied across the entire range. Transformer exhibited the highest latency, whereas CNN and LSTM were comparatively faster. In the range of large numbers of prediction instances (≥20,000), the gap between models widens; therefore, for xApp operations where near-real-timeness is critical, it is reasonable to prioritize the CNN.

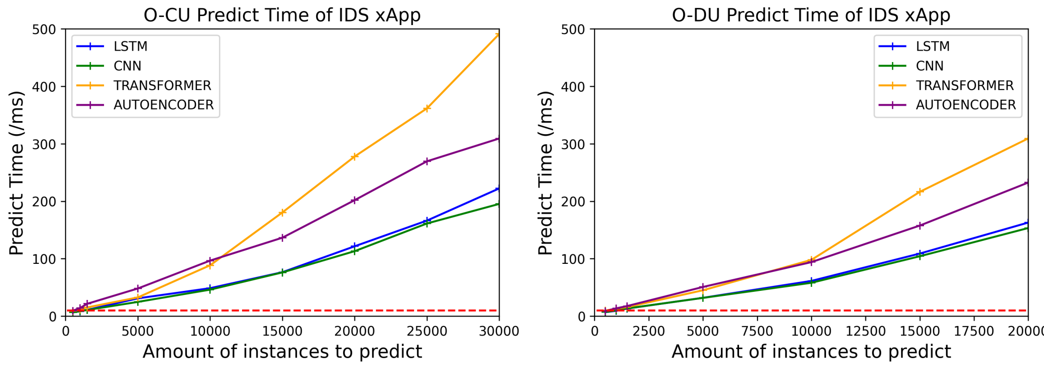

Figure 5: Graph of IDS xApp Inference Latency by Number of Prediction Instances

5 Conclusion and Future Works

This study experimentally verifies the end-to-end process from training to inference by integrating an AIML Framework–based pipeline with an IDS xApp within the Near-RT RIC in an O-RAN environment. Comparing LSTM, CNN, Transformer, and Autoencoder models on O-CU/DU data confirmed that CNN is the most operationally friendly choice in terms of balancing performance and resource efficiency, while Transformer provided high performance but exhibited increased resource consumption and inference latency. The Autoencoder showed high detection sensitivity, but its low accuracy and precision indicate the need for additional performance optimization. Specifically, the F1-scores of the CNN and Transformer models were 0.9955 and 0.9956, respectively—the highest among all models—while the LSTM achieved 0.9946 and the Autoencoder recorded 0.6667. In terms of CPU energy consumption, the Transformer consumed up to 77.969 Wh, followed by CNN (51.666 Wh), LSTM (17.653 Wh), and Autoencoder (5.309 Wh), showing a clear difference in resource efficiency across models. In addition, although inference latency increased quasi-linearly with the number of prediction instances, all models satisfied the Near-RT requirement (10 ms–1 s). In the O-CU data experiments, even with more than 20,000 prediction instances, the CNN and LSTM models achieved an average inference latency of approximately 0.2 seconds, while the Transformer showed about 0.5 seconds, all stably meeting the Near-RT limit (1 second) and demonstrating that the proposed system can be effectively applied as a real-time intrusion detection system (IDS) in practical O-RAN operating environments.

For future work, we plan to further enhance the performance of the IDS xApp by applying hybrid models, ensemble learning, and dynamic threshold adjustment to improve detection accuracy while minimizing false positives and false negatives. Beyond simple detection, we also aim to extend it into a comprehensive security xApp that incorporates response functions. Specifically, by automating response procedures such as traffic blocking and policy updates in conjunction with the Near-RT RIC control loop for detected attacks, we can establish an integrated detection-and-response framework. Furthermore, by linking with rApps in the Non-RT RIC, threat information collected by the IDS xApp can be analyzed and learned in non-real-time, and the results can be used to optimize security control via policies delivered over the A1 interface. In other words, the goal is to implement a security system in the O-RAN environment that cycles through detection, response, and policy: the xApp handles near-real-time detection and response, while the rApp is responsible for non-real-time learning and policy management.

Acknowledgement

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2024-00396797, Development of core technology for intelligent O-RAN security platform)

References

[1] Cheng-Feng Hung, Y.-R. C.-H.-M. (2024). Security Threats to xApps Access Control and E2 Interface in O-RAN. *IEEE Communications Society*, 1197-1203.

[2] Civciss, A. &. (2025, 12 5). Silent Signals, Loud Threats: Using dApps for Radio Signal Intelligence-based Intrusion Detection in 5G O-RAN. *IEEE Global Communications Conference.* Taipei, Taiwan: IEEE.

[3] Emmanuel N. Amachaghi, M. S. (2024). A Survey for Intrusion Detection Systems in Open RAN. *IEEE Access*, 88146-88173.

[4] Gabriel Matheus Almeida, G. Z. (2023). RIC-O: Efficient Placement of a Disaggregated and Distributed RAN Intelligent Controller With Dynamic Clustering of Radio Nodes. *IEEE Journal on Selected Areas in Communications*, 446-459.

[5] Heejae Park, T.-H. N. (2024). An Investigation on Open-RAN Specifications: Use Cases, Security Threats, Requirements, Discussions. *CMES - Computer Modeling in Engineering and Sciences*, 13-41.

[6] Hexuan Yu, M. M. (2025). *Closing the Visibility Gap: A Monitoring Framework for Verifiable Open RAN Operations.* arXiv.

[7] Hiba Hojeij, M. S. (2024). On Flexible Placement of O-CU and O-DU Functionalities in Open-RAN Architecture. *IEEE Transactions on Network and Service Management*, 660-674.

[8] Joshua Groen, S. D. (2025). Implementing and Evaluating Security in O-RAN: Interfaces, Intelligence, and Platforms. *IEEE Network*, 227-234.

[9] Junran Wang, M. R. (2024). Optimization of Energy Consumption in Delay-Tolerant Networks. *Networking and Internet Architecture*.

[10] Kakani, P. a.-P. (2025, 01). Mitigating ML-Driven Adversarial Attacks on xApps Using Dynamic Defense Mechanisms. *IEEE Open Journal of the Communications Society*, 1-1.

[11] Kelsy Cabello-Solorzano, I. O.-B. (2023). The Impact of Data Normalization on the Accuracy of Machine Learning Algorithms: A Comparative Analysis. *18th International Conference on Soft Computing Models in Industrial and Environmental Applications*, 344-353.

[12] Lei Xu, L. H. (2021). Mid-term prediction of electrical energy consumption for crude oil pipelines using a hybrid algorithm of support vector machine and genetic algorithm. *energy*.

[13] Mattia Merluzzi, P. D. (2021). Wireless Edge Machine Learning: Resource Allocation and Trade-Offs. *IEEE Access*, 45377-45398.

[14] Michele Polese, L. B. (2023). Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. *IEEE Communications Surveys & Tutorials*, 1376-1411.

[15] Moore, J. a. (2025). *Integrated LLM-Based Intrusion Detection with Secure Slicing xApp for Securing O-RAN-Enabled Wireless Network Deployments.*

[16] Obi, J. C. (2023). A Comparative Study of Several Classification Metrics and Their Performances on Data. *World Journal of Advanced Engineering Technology and Sciences*, 308-314.

[17] O-RANProject. (n.d.). *O-RAN Architecture Overview*. Retrieved from O-RAN Software Community: https://docs.o-ran-sc.org/en/latest/architecture/architecture.html

[18] Salvatore D’Oro, M. P. (2022). dApps: Distributed Applications for Real-Time Inference and Control in O-RAN. *IEEE Communications Magazine*.