Evaluating Lightweight Neural Models for Edge-Based Anomaly Detection: Performance and Efficiency Trade-offs

Isaac Kofi Nti

ntious1@gmail.com

School of Information Technology University of Cincinnati https://orcid.org/0000-0001-9257-4295

Lee Jo Ning

National Chung Hsing University Taichung, Taiwan

Clark Alex

School of Information Technology University of Cincinnati

Miriyala, Sai Manikanta

School of Information Technology University of Cincinnati

Murat Ozer

School of Information Technology University of Cincinnati Cincinnati, USA

Research Article

Keywords: edge-based anomaly detection, lightweight neural models, edge-optimization, resource constraints, inference efficiency

Posted Date: July 17th, 2025

DOI: https://doi.org/10.21203/rs.3.rs-7138288/v1

Additional Declarations: The authors declare no competing interests.

Evaluating Lightweight Neural Models for Edge-Based Anomaly Detection: Performance and Efficiency Trade-offs

Isaac Kofi Nti
School of Information Technology
University of Cincinnati
Cincinnati, USA
0000-0001-9257-4295

Lee Jo Ning
National Chung Hsing University
Taichung, Taiwan
joninglee18@gmail.com

Clark Alex
School of Information Technology
University of Cincinnati
Cincinnati, USA
clark5a6@mail.uc.edu

Miriyala, Sai Manikanta
School of Information Technology
University of Cincinnati
Cincinnati, USA
miriyast@mail.uc.edu

Murat Ozer
School of Information Technology
University of Cincinnati
Cincinnati, USA
ozermm@ucmail.uc.edu

***Abstract*— In edge computing scenarios where constraints on memory, latency, and energy hinder the utilization of large-scale models, lightweight neural networks are increasingly favored for anomaly detection. However, uniform benchmarks for comparing commonly utilized lightweight models under these constraints remain absent. This research addresses the gap by evaluating three prominent lightweight neural architectures, pruned convolutional neural networks (CNNs), quantized long short-term memory networks (LSTMs), and distilled transformers, across two established intrusion detection datasets: CIC-IoT-DIAD 2024 and TON_IoT (TON_IoT_Modbus and TON_IoT_Thermostat). We evaluate each model utilizing standard detection measures (accuracy, precision, recall, and F1-score) and deployment metrics (model size, inference latency, and memory consumption) under simulated edge constraints. Our findings indicate significant trade-offs between model accuracy and efficiency, with performance varying based on the dataset utilized. Certain models perform more effectively with flow-based data compared to others with IoT telemetry. No single model excelled in all evaluation criteria. This research provides future edge-optimized anomaly detection studies with a reliable, reproducible foundation and valuable insights on selecting models for real-time edge deployment. The results also guide our attention in creating our forthcoming architecture, S3LiteNet, intended to enhance performance and deployment in information-centric networks.**

Keywords*— *edge-based anomaly detection, lightweight neural models, edge-optimization, resource constraints, inference efficiency

I. INTRODUCTION

The widespread use of Internet of Things (IoT) devices, along with improvements in edge and fog computing, has changed several areas, such as Industry 4.0, digital healthcare, and home automation. Because of this, anomaly detection has become an important part of cybersecurity in our changing world. An increasing volume of security-related data is now generated at the edge—closer to data sources than centralized data centers. In this landscape, deep learning-based anomaly detection systems have demonstrated superior detection performance compared to classical methods. However, their computational demands often exceed the capacity of edge devices, making full-scale deployment infeasible in real-world resource-constrained environments [1]-[3].

To address these issues, researchers have proposed lightweight neural networks models that are compressed through techniques like pruning [4], quantization [2] and knowledge distillation [5], [6], to support inference on devices with limited memory, processing power, and energy. While these lightweight models have shown promise individually, there is a distinct lack of systematic, empirical benchmarks comparing them comprehensively in terms of both detection accuracy and deployment efficiency under simulated or real edge conditions. Most literature [6]-[8], reports model performance in isolation, uses varied evaluation setups, or fail to include crucial resource-centric metrics such as latency, memory footprint, and model size. This highlights the inherent difficulty in directly assessing the trade-offs between performance and resource consumption across diverse lightweight architectures.

This paper addresses that gap by rigorously evaluating different variants of Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and state-of-the-art distilled transformer-based lightweight neural architectures on two commonly used anomaly detection datasets: CIC-IoT-DIAD 2024 and TON_IoT (TON_IoT_Modbus and TON_IoT_Thermostat). These datasets capture diverse anomaly patterns across flow-based and telemetry-based data modalities [9], [10]. We systematically assess each model's detection performance and simulated resource usage under constrained conditions, using standard metrics and consistent preprocessing. This paper seeks to:

1. evaluate and compare anomaly detection efficiency of widely-used lightweight neural network models on standard cybersecurity datasets.
2. assess the resource efficiency of these models using simulated hardware-constrained measures, including inference latency, memory consumption, and model dimensions.
3. identify trade-offs between detection performance and deployment costs in edge computing.

4. provide evidence-based recommendations for selecting lightweight neural models suitable for edge-based intrusion detection systems.

To achieve the objectives stated, we answer the following research questions (RQ):

1. RQ1: How effectively can widely-used lightweight neural network models detect anomalies in standard datasets, as measured by accuracy, precision, recall, and F1-score?
2. RQ2: How do the inference latency, memory footprint, and model size of the chosen models compare when functioning under certain hardware limitations?
3. RQ3: What trade-offs have been observed between model accuracy and deployment efficiency across various datasets and neural network architectures?
4. RQ4: Which model(s) exhibits the ideal equilibrium of performance and efficiency for implementation in real-time, edge-based anomaly detection systems?

The following are our contributions to the field of lightweight anomaly detection for edge computing:

- **Unified Benchmarking of State-of-the-Art Lightweight Models.** We provide a controlled, experimental evaluation of 3 well-used lightweight neural network models: pruned Convolutional Neural Networks (CNNs), quantized Long Short-Term Memory (LSTM) networks, and distilled transformers, using standardized protocols across two widely adopted anomaly detection datasets: CIC IoT-DIAD 2024 and TON_IoT.
- **A quantitative look at the trade-offs between accuracy and efficiency**. This study measures and compares detection accuracy with important deployment metrics like inference latency, memory usage, and model size. This provides a clearer understanding of the inherent trade-offs when building smaller and faster models on devices with limited resources.
- **Evaluation across datasets to check for robustness**. We make it clear how performance and resources use differ between different types of data by testing all models on both flow-based and telemetry-based datasets. These outcomes contribute to a deeper understanding of the robustness and adaptability of lightweight models in heterogeneous edge computing scenarios.
- **Data-Driven Foundation for Future Model Innovation**. The insight from this comparative study establishes an empirical foundation for the design of future lightweight anomaly detection models.

We organized the remaining sections of this paper as follows: Section II presents a summary of related works. In Section III we present the study methodology. Section IV presents our results and discussions addressing the research questions. Finally, we concluded our study with recommendations for future work in Section V.

II. LITERATURE REVIEW

This section presents a summary of existing literature on anomaly detection in edge computing with light weight neural network. We categorize the studies under two main headings: supervised and unsupervised.

Supervised anomaly detection methods rely on labeled datasets, where each data point is explicitly marked as normal or anomalous. Whereas unsupervised anomaly-detection models learn only from unlabeled data and flag deviations, making it suitable for edge scenarios with scarce labels and resource-constrained devices. Rondanini et al. [3] tested different lightweight leveraging large language models (DistilBERT, TinyBERT, DistilGPT-2, TinyT5, and a 1 B-parameter LlaMA) for enhancing malware detection on IoT. After fine-tuning, TinyBERT achieved > 97 % accuracy, using a 55–60 MB storage and ≤ 5 s inference on Raspberry Pi 3, highlighting model-size trade-offs. Similarly, Huang et al. [11] proposed ConvLSTM variants-liquid time-constant (CLTC) and ConvLSTM-closed-form continuous-time (CCfC) for ECG-based anomaly detection. The study showed high moderate performance (F1-score ≈ 0.83, AUROC ≈ 0.96), within 100 kB flash and 240 kB RAM on microcontrollers. Likewise, the training cost of large edge swarms was addressed by clustering similar devices using two strategies: ICPTL (parameter transfer) and CM (shared cluster model) Fernando et al. [4]. They achieved high AUC and F1 scores while reducing training time by up to 66%. Also, Anuva et al. [9] developed LIDIT, a quantized LSTM-autoencoder for IoMT traffic. The proposed model achieved a good F1 score on CICIoMT-2024 and 15 milliseconds inference time on Raspberry Pi Zero 2 W, showing real-time detection is feasible under 300 kB memory. Chen et al. [10] introduced PPLAD, a privacy-preserving model using only similarity vectors as input. Ziegler et al. [12] adapted PatchCore for MCUs using MCUNet, quantitation, and coreset compression. The proposed model fit in <100 kB flash and 200 kB RAM, achieving AUROC up to 96%.

III. EXPERIMENTAL SETUP

This section discuss the steps, tools libraries and methods used to achieved the study's objectives.

A. Datasets and preprocessing

This study used two publicly available and frequently used cybersecurity datasets: TON_IoT [13] and CIC IoT-DIAD 2024 [14]. We chose these datasets due to their often used in research on lightweight anomaly detection, dataset diversity, and recency. The TON_IoT dataset, developed by the University of New South Wales, is designed for cybersecurity research in IoT environments. This study uses two subsets: IoT_Thermostat, which captures smart home telemetry under normal and attack conditions, and IoT_Modbus, which contains industrial control communication logs. These selections offer a balanced representation of heterogeneous edge scenarios across residential and industrial contexts. CIC IoT-DIAD 2024 is a recent tabular anomaly detection dataset focused on IoT

attacks in edge environments. Released by the Canadian Institute for Cybersecurity (CIC), it provides realistic and varied attack data across multiple IoT modalities, making it highly suitable for evaluating lightweight anomaly detection models on edge-like telemetry data. Adding them allows for a balanced and robust evaluation of model performance across traditional network traffic, IoT telemetry, and modern multi-modal attack vectors.

We preprocessed both datasets independently to remove data inconsistencies and address missing values. These columns ('Src IP', 'Dst IP', 'Flow ID', 'Timestamp') were dropped from the CIC IoT-DIAD dataset and ('date', 'time) from TON_IoT dataset due to their irrelevant for model learning. All datasets had target column 'Label' (0 = benign, 1 = attack). Each cleaned dataset was normalized to a *[0,1]* range using MinMaxScaler and split into training (70%) and test (30%) sets using stratified sampling. To address class imbalance, we applied SMOTE exclusively to the training set, ensuring the test set remained representative of real-world label distributions and avoiding data leakage.

B. Models Selected for Benchmarking

In this study, we benchmark six heterogenous lightweight neural network models for anomaly detection in resource-constrained edge environments. These included Convolutional Nural Networks (CNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) and Distilled Transformer. Three variants (baseline, optimized (with reduced complexity), and quantized + pruned) of each model were designed, implemented and evaluated on two different public edge computing datasets. The models were adopted based on the prevalence in the edge AI literature and suitability for tabular time-series data [5], [13], [15]. We briefly present how the models were implemented in this study.

1. **Pruned Convolutional Neural Network (PCNN):** In this model, L1-norm pruning (see Eq. 1) was implemented to eliminate redundant convolutional filters. We optimized a 1D CNN with global average pooling using fewer filters and batch normalization. Pruned models were then quantized with TensorFlow Lite [16].

$$L_{\text{prune}} = \sum_i |W_i|_1 \quad (1)$$

Where: W_i is the weight of the i$^{\text{th}}$ convolutional filter.

2. **Quantized Long Short-Term Memory (QLSTM)**: Post-training quantization as defined in Eq. 2 was applied to the LSTM model to reduce memory and inference latency. The LSTM architecture in this study was optimized with a smaller recurrent unit and global average pooling.

$$\hat{\omega} = round\left(\frac{\omega-\mu}{s}\right), \omega \approx s \cdot \hat{\omega} + \mu \quad (2)$$

Where: $\hat{\omega}$ is the quantized weight, ω is the original floating-point weight, s is the scale, μ is the zero-point offset.

3. **Distilled Transformer (DT)**: Inspired by TinyBERT [17], we implement a shallow transformer encoder with reduced embedding dimensions and attention heads (see Eq. 3). We embedded the input features into a dense space and processed via a multi-head attention layer, followed by pooling and a sigmoid output.

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3)$$

4. **Pruned Gated Recurrent Unit (PGRU):** Similar to the LSTM pipeline discussed above in this paper, the GRU architecture was evaluated in a baseline form and optimized version, followed by quantization and pruning.
5. **Simple Recurrent Neural Network (SimpleRNN):** We included this baseline model as a classical, low-resource recurrent benchmark. It was also evaluated in pruned and quantized forms.
6. **Hybrid CNN-GRU Model**: This composite architecture was implemented with a Conv1D layer followed by a GRU unit, enabling both spatial and temporal feature extraction. Optimizations include kernel reduction and pooled GRU output.

All experiments were carried on a Windows platform (Lenovo, 12th Gen Intel (R) Core (TM) i7-1260P (16CPUs) ~2.1GHz, 32GB RAM), simulated edge device TensorFlow single-threaded CPU. Simulation memory was limited to 2GB RAM MAX; CPU-only execution. The 2GB RAM, CPU-only environment utilized in this study resembles typical commercial edge hardware such as the Raspberry Pi 4 (2–4GB RAM), the NVIDIA Jetson Nano (4GB RAM), or the Google Coral Dev Board (1GB RAM with Edge TPU). We restrict execution to single-threaded CPU mode and limit memory to 2GB to align with the operational constraints of these commonly utilized platforms. This ensures that the performance and resource conclusions are applicable in practical IoT and edge AI contexts.

We kept models common hyperparameters consistent and untuned across all experiments to ensure a controlled and unbiased comparison of the inherent architectural differences between the models. This approach facilitates reproducibility and isolates the performance variations attributable to the models themselves, rather than individualized hyperparameter optimization. Table I presents the shared hyperparameter configurations.

We trained all models and validated them using performance and efficiency metrics as shown in Table I. Appendix A shows all model configurations details. Fig. 1 shows the end-to-end pipeline of our experimental setup.

Table I: Hyperparameters Used Across All Models

Parameter	Value
Optimizer	Adam
Learning Rate	0.001 (default Adam)
Loss Function	Binary Crossentropy
Epochs	5
Batch Size	64
Random State	42
Test Size	30%
Preprocessing	Min-Max Scaling + SMOTE
Dataset(s)	CIC-IoT-DIAD 2024 and TON_IoT (TON_IoT_Modbus and TON_IoT_Thermostat)

Table II: Performance and Efficiency Metrics

Metric Type	Metrics Used
Classification	Accuracy, Precision, Recall, F1-Score, AUROC
Efficiency	Inference Latency (sec), Model Size (MB), Memory Footprint (via psutil), CPU Usage

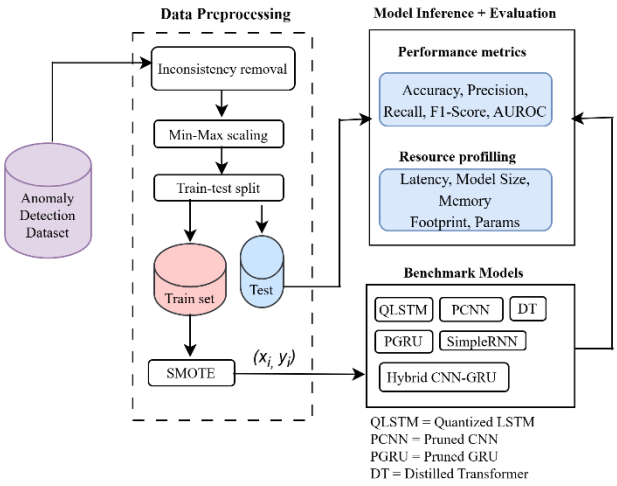

Fig. 1. End-to-end pipeline for benchmarking lightweight neural network models for anomaly detection on edge devices. The process includes data preprocessing (inconsistency removal, Min-Max scaling, stratified train-test split, and SMOTE), model training across five lightweight architectures (QLSTM, PCNN, PGRU, SimpleRNN, and Hybrid CNN-GRU), and evaluation based on both classification metrics (e.g., F1-score, AUROC) and deployment efficiency metrics (e.g., latency, memory usage, and model size)

IV. RESULTS AND DISCUSSIONS

We present the outcome of our experiments in this section.

A. *RQ1: How effectively can widely-used lightweight neural network models detect anomalies in standard datasets, as measured by accuracy, precision, recall, and F1-score?*

Fig. 2 shows how various lightweight models (CNN, GRU, LSTM, and hybrid architectures) perform on three benchmark datasets (CIC_IOT_DIAD, TON_IOT_Modbus, and TON_IoT_Thermostat). The results show that commonly used lightweight neural network models demonstrated high effectiveness in detecting anomalies in resource constraint environments across diverse standard datasets. We observed that the CNN+GRU baseline model consistently outperformed all other models in terms of accuracy, precision, recall, and F1-score. It achieved an F1-score of 0.92 on CIC_IOT_DIAD and 1.00 on the TON_IOT subsets under current evaluation settings. The findings demonstrate that integrating CNN and GRU as a lightweight framework might produce a strong performance in diverse anomaly detection contexts.

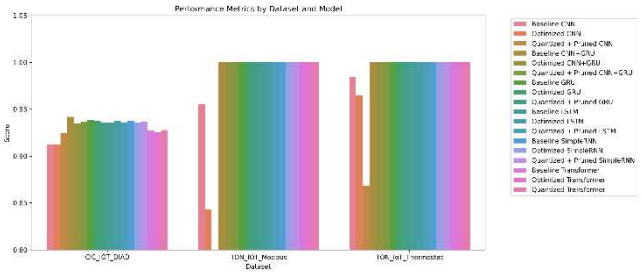

Fig. 2. Fig. 2. Comparative performance of lightweight neural models (CNN, GRU, LSTM, CNN+GRU, Transformer) across three benchmark datasets (CIC_IOT_DIAD, TON_IOT_Modbus, TON_IOT_Thermostat) using accuracy, precision, recall, and F1-score. CNN+GRU consistently achieves the highest F1-score across all datasets, indicating strong anomaly detection capability. Transformers lag slightly in recall-sensitive tasks.

Table II presents the mean and standard deviation of accuracy, F1-score, and AUC across evaluated models. As seen in Table II, the GRU and CNN & GRU architectures achieved the highest overall performance. On the other hand, the CNN models showed relatively lower AUC and recall. The distilled transformer offered competitive accuracy but showed limitations in recall-sensitive tasks.

Table II: Performance Comparison of Deep Learning Algorithms for Anomaly Detection

DL Algorithm	Accuracy (Mean ± Std)	F1-Score (Mean ± Std)	AUC (Mean ± Std)	Notes
GRU	0.908 ± 0.002	0.945 ± 0.001	0.946 ± 0.001	Highest AUC and nearly perfect precision
LSTM	0.907 ± 0.001	0.944 ± 0.001	0.945 ± 0.001	Consistently high across all metrics
CNN & GRU	0.909 ± 0.005	0.946 ± 0.003	0.942 ± 0.002	Strong hybrid performance
Distilled Transformer	0.894 ± 0.002	0.936 ± 0.001	0.910 ± 0.002	Competitive, with lower recall
CNN	0.879 ± 0.010	0.927 ± 0.007	0.871 ± 0.034	Slightly weaker, especially in AUC

B. *RQ2: How do the inference latency, memory footprint, and model size of the chosen models compare when functioning under certain hardware limitations?*

We compared models inference latency, model size, and memory usage across baseline lightweight architectures to evaluate deployment feasibility under hardware constraints, As shown in Table III, the CNN baseline achieved the lowest latency (0.0408s) and smallest size (0.0012 MB), making it optimal for fast inference and storage-constrained deployments. However, its runtime memory usage (926 MB) was higher than CNN+GRU (698 MB), which may limit use on RAM-constrained devices. The CNN+GRU baseline model offered a balanced trade-off, attaining competitive efficiency (latency = 0.1763s, memory = 698.7 MB). However, the LSTM and GRU baseline models used more memory (up to 1091 MB), which could limits their application on edge devices. Our findings highlight the importance of selecting architectures that align with both detection objectives and available resources.

Table III: Average Inference Latency, Model Size, and Memory Usage Across Lightweight Architectures

Model	Latency (s)	Size (MB)	Memory (MB)
Baseline CNN	0.0408	0.0012	926.10
Baseline CNN+GRU	0.1763	0.0248	698.72
Baseline GRU	0.0415	0.0720	873.70
Baseline LSTM	0.0478	0.0950	1091.38
Baseline SimpleRNN	0.0523	0.0239	942.18

*Size (MB) – model size on disk; Latency (s) – total inference time per model; Memory (MB) – actual runtime memory usage

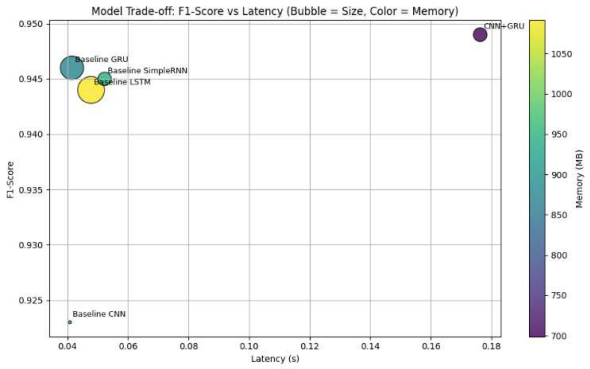

Fig. 3. Trade-off visualization of model performance (F1-score) versus deployment efficiency (latency, memory, and size). CNN+GRU achieves superior detection accuracy but incurs higher latency and memory use. In contrast, baseline CNN offers the best efficiency (lowest latency and size) but lower detection performance. GRU and LSTM models sit in the middle ground with high performance and moderate efficiency.

C. *RQ3: What trade-offs have been observed between model accuracy and deployment efficiency across various datasets and neural network architectures?*

Table IV shows a comparison summary of models' accuracy and deployment efficiency (latency, memory, and size). We observed a clear trade-off between detection performance and deployment efficiency. As summarized in Table IV:

- The CNN+GRU baseline consistently achieved the highest accuracy and F1-scores across all datasets (e.g., 94.9% F1 on CIC_IOT_DIAD, and 100% on both TON_IOT_Modbus and Thermostat). However, it sustained higher inference latency (0.1763s) and moderate memory usage (~699 MB), making it less optimal for highly constrained environments.
- In contrast, the CNN baseline recorded the best efficiency profile in terms of model size (0.0012 MB) and inference speed (0.0408s). However, its memory footprint (926 MB) was higher than some alternatives, including CNN+GRU.
- The GRU and LSTM models obtained intermediate performance. Even though they achieved strong precision and recall, they might pose deployment challenges on edge devices due to their larger sizes (up to 0.095 MB) and higher memory demands (up to 1091 MB).

These findings show that no single model works well in all areas. For example, the CNN+GRU outperformed all models in anomaly detection but requires more computational power. In contrast, the CNN baseline is a highly efficient model that could be suitable for real-time or low-resource scenarios, with moderate detection performance.

Table IV: Trade-off Summary of Lightweight Models for Edge-Based Anomaly Detection

Model	Accuracy	Latency	Memory	Size	Overall Trade-off
CNN+GRU	HT	H	M	M	Best for Accuracy
Baseline CNN	M	L	M	L	Best for Efficiency
GRU / LSTM	H	M	H	H	Balanced, Poor Efficiency

*HT = Highest; M = Moderate; L = Lowest; H = High

D. *RQ4: Which model(s) exhibits the ideal equilibrium of performance and efficiency for implementation in real-time, edge-based anomaly detection systems?*

Table V presents the trade-off matrix for selecting lightweight neural network models based on performance and deployment efficiency. We observed that among all evaluated models, the CNN+GRU baseline always demonstrated the highest anomalies detection performance. It obtained F1-scores reaching 0.949 on CIC_IOT_DIAD and 1.000 on both TON_IOT_Modbus and TON_IOT_Thermostat datasets. While it has an inference latency of 0.1763 seconds and uses about 699 MB of memory, its efficiency is still adequate for current edge devices with modest hardware (1–2 GB RAM). The CNN baseline exhibited the most favorable characteristics for stringent real-time and ultra-low-resource demands. It exhibited the lowest latency (0.0408s) and the smallest model size (0.0012 MB), albeit with a slight compromise in detection accuracy.

The model that provides the optimal overall equilibrium is:

- The CNN+GRU baseline model is the optimal selection for achieving balanced real-time performance on moderately capable edge devices.
- The CNN baseline model is ideal for scenarios requiring high speed and minimal memory, where detection compromises are permissible.

Table V: Trade-off Matrix for Selecting Lightweight Neural Network Models Based on Performance and Deployment Efficiency

Model	Detection Performance (F1)	Latency (s)	Memory (MB)	Size (MB)	Recommended Use Case
Baseline CNN+GRU	0.949	0.176	698.7	0.025	Balanced performance on capable edge devices
Baseline CNN	0.923	0.041	926.1	0.001	Storage- and latency-constrained environments (with adequate RAM)
Baseline GRU	0.946	0.042	873.7	0.072	High performance, less memory efficiency
Baseline LSTM	0.944	0.045	1091.4	0.095	Not ideal for edge (high memory)
Baseline SimpleRNN	0.945	0.052	942.2	0.024	Slightly better than

					LSTM for edge

V. Conclusion and Future Work

This work analyzed multiple lightweight neural models for edge-based anomaly detection, assessing their effectiveness in recognizing anomalies and their computing complexity. We responded to the study's research inquiries as follows:

- **RQ1**: CNN+GRU baseline model consistently achieved the highest accuracy and F1-score across datasets, confirming its effectiveness in detecting anomalies in diverse edge computing contexts.
- **RQ2**: Regarding deployment efficiency, the CNN baseline model exhibited the lowest inference latency and smallest model size, making it highly suitable for edge deployment.
- **RQ3**: A clear trade-off was observed between detection performance and resource efficiency. The CNN+GRU baseline model achieved superior accuracy; nevertheless, its moderate latency and memory consumption makes it most suitable for devices with medium capabilities at the edge. Conversely, the CNN baseline model offered considerable efficiency in latency and model size, although with a reduction in recall and AUC, and a higher memory usage that may affect edge deployment feasibility.
- **RQ4**: The CNN+GRU baseline model was seen as the most balanced model, attaining an ideal equilibrium between accuracy and efficiency. CNN is recommended for environments where latency and storage are critical, but available memory must also be considered.

These findings emphasize the necessity of synchronizing model architectural selections with practical limitations in edge-based anomaly detection systems. In the future, we will present S3LiteNet—a bespoke lightweight neural module engineered to adaptively optimize sparsity, scalability, and speed for real-time anomaly detection at the network edge. S3LiteNet will integrate structured pruning, quantization, and knowledge distillation based on the findings from this benchmark, featuring adjustable modules designed for edge hardware specifications. Subsequent research will authenticate its efficacy using cross-domain datasets and simulated edge device limitations, aiding in the advancement of reliable and resource-efficient cybersecurity AI systems.

ACKNOWLEDGMENT

This research was partially supported by the University of Cincinnati's International Office, Global Research Experience Program.

References

[1] Z. Alwaisi *et al*, "Securing constrained IoT systems: A lightweight machine learning approach for anomaly detection and prevention," *Internet of Things,* vol. 28, 2024. . DOI: 10.1016/j.iot.2024.101398.
[2] S. Vashisth and A. Goyal, "A Comparative Analysis for Designing Security Mechanism for Resource-Constrained Internet of Things Devices," *Advancements in Communication and Systems,* pp. 21, 2024. . DOI: 10.56155/978-81-955020-7-3-3.
[3] C. Rondanini *et al*, "Malware Detection at the Edge with Lightweight LLMs: A Performance Evaluation," 2025. .
[4] D. Fernando *et al*, "Efficient Training Approaches for Performance Anomaly Detection Models in Edge Computing Environments," *ACM Trans. Auton. Adapt. Syst.,* vol. 20, *(2),* pp. 1, 2025. . DOI: 10.1145/3725736.
[5] S. M. Raza *et al*, "Survey on the application with lightweight deep learning models for edge devices," Institute of Electrical and Electronics Engineers (IEEE), 2025. DOI: 10.36227/techrxiv.174439272.21224126/v1.
[6] J. Forough *et al*, "Efficient Anomaly Detection for Edge Clouds: Mitigating Data and Resource Constraints," *IEEE Access,* vol. 12, pp. 171897, 2024. . DOI: 10.1109/access.2024.3492815.
[7] N. I. Liu *et al*, "Lightweight Deep Learning for Resource-Constrained Environments: A Survey," *ACM Comput. Surv.,* vol. 56, *(10),* pp. 1, 2024. . DOI: 10.1145/3657282.
[8] F. Modu, R. Prasad and F. Aliyu, "Lightweight CNN for Resource-Constrained BCD System Using Knowledge Distillation," *IEEE Access,* vol. 13, pp. 57504, 2025. . DOI: 10.1109/access.2025.3556517.
[9] S. Tajmim, "LIDIT: Low-Latency Intrusion Detection in IoMT Devices using TinyML," 2025. .
[10] L. Chen *et al*, "Privacy-Preserving Lightweight Time-Series Anomaly Detection for Resource-Limited Industrial IoT Edge Devices," *IEEE Trans. Ind. Inf.,* vol. 21, *(6),* pp. 4435, 2025. . DOI: 10.1109/tii.2025.3538127.
[11] Z. Huang *et al*, "Efficient Edge-AI Models for Robust ECG Abnormality Detection on Resource-Constrained Hardware," *J. of Cardiovasc. Trans. Res.,* vol. 17, *(4),* pp. 879, 2024. . DOI: 10.1007/s12265-024-10504-y.
[12] T. Ziegler, "Applications of AI on Resource-Constrained Hardware with a focus on Anomaly Detection," 2023. .
[13] N. Moustafa, "A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets," *Sustainable Cities and Society,* vol. 72, pp. 102994, 2021. Available: https://www.sciencedirect.com/science/article/pii/S2210670721002808. DOI: 10.1016/j.scs.2021.102994.
[14] M. Rabbani *et al*, "Device Identification and Anomaly Detection in IoT Environments," *IEEE Internet of Things Journal,* vol. 12, *(10),* pp. 13625–13643, 2025. . DOI: 10.1109/JIOT.2024.3522863.
[15] D. Ngo, H. Park and B. Kang, "Edge Intelligence: A Review of Deep Neural Network Inference in Resource-Limited Environments," *Electronics,* vol. 14, *(12),* 2025. . DOI: 10.3390/electronics14122495.
[16] M. Abadi *et al*, "{TensorFlow}: A system for {large-scale} machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16),* 2016, .
[17] X. Jiao *et al*, "Tinybert: Distilling bert for natural language understanding," *arXiv Preprint arXiv:1909.10351,* 2019. .

Appendix A: Model Configuration Summary

Model	Architecture Summary	Optimization Applied
Baseline CNN	Reshape → Conv1D(64, kernel=3, padding='same', activation='relu') → GlobalAveragePooling1D → Dense(1, sigmoid). *Applies feature extraction with standard convolution and ReLU activation. No pooling layers or batch normalization applied.*	None
Optimized CNN	Reshape → Conv1D(32, kernel=3, padding='same') → BatchNorm → ReLU → GlobalAveragePooling1D → Dense(1, sigmoid). *Uses fewer filters, batch normalization, and ReLU for stabilized training with reduced complexity.*	None
Quantized + Pruned CNN	Same as optimized CNN. Pruned using polynomial sparsity schedule (0.0 → 0.5). Quantized with TensorFlow Lite post-training quantization using Optimize.DEFAULT. No fallback ops or TensorList support required.	Pruning + Quantization
Baseline LSTM	Reshape → LSTM(64) → Dense(1, sigmoid) *Input reshaped to (1, N). Outputs only final timestep.*	None

Optimized LSTM	Reshape → LSTM(32, return_sequences=True) → GlobalAvgPool1D → Dense(1, sigmoid) Reduced complexity. Uses sequence pooling for dimensionality reduction.	None
Quantized + Pruned LSTM	Same as optimized, with pruning (polynomial decay) and TFLite quantization using SELECT_TF_OPS and TensorList fallback enabled. Post-training quantization was evaluated using TFLite Interpreter with per-sample invocation.	Pruning + Quantization
Baseline GRU	Reshape → GRU(64) → Dense(1, sigmoid) *Input is reshaped to (1, N). Only the final output of the GRU layer is used.*	None
Optimized GRU	Reshape → GRU(32, return_sequences=True) → GlobalAvgPool1D → Dense(1, sigmoid) *Smaller GRU unit with full sequence output, pooled via GlobalAveragePooling1D for dimension reduction.*	None
Quantized + Pruned GRU	Same as optimized, with structured pruning using PolynomialDecay and post-training quantization via TensorFlow Lite. TensorList ops lowering was disabled and SELECT_TF_OPS was enabled to ensure GRU compatibility.	Pruning + Quantization
Baseline RNN	Reshape → SimpleRNN(64) → Dense(1, sigmoid) *Reshaped inputs to (1, N). The model uses only the final RNN output.*	None
Optimized RNN	Reshape → SimpleRNN(32, return_sequences=True) → GlobalAvgPool1D → Dense(1, sigmoid) *Reduced RNN complexity and used sequence pooling to lower the output dimensionality.*	None
Quantized + Pruned RNN	Same as optimized, with structured pruning via polynomial decay and TensorFlow Lite quantization. SELECT_TF_OPS was enabled for compatibility and TensorList ops lowering was disabled.	Pruning + Quantization
Baseline CNN+GRU	Reshape → Conv1D(32, kernel=3) → GRU(32) → Dense(1, sigmoid) Combines spatial and temporal modeling. Input is reshaped to (N, 1).	None
Optimized CNN+GRU	Reshape → Conv1D(16, kernel=3) → BatchNorm → ReLU → GRU(16, return_sequences=True) → GlobalAvgPool1D → Dense(1, sigmoid) Smaller convolution and GRU layers. Temporal pooling applied after GRU.	None
Quantized + Pruned CNN+GRU	Same as optimized, with structured pruning and TensorFlow Lite quantization (fallback ops enabled, TensorList lowering disabled).	Pruning + Quantization
Baseline Transformer	Dense(64) → Reshape(1, 64) → MHA(2 heads) → GlobalAvgPool1D → Dense(1, sigmoid) No positional encoding used. Transformer operates on a reshaped tabular input with one attention head per feature dimension.	None
Optimized Transformer	Same as baseline, but with reduced embedding dim (32). Quantized using post-training quantization.	Reduced embed dim
Quantized Transformer	Same as optimized transformer, evaluated using TFLite with post-training quantization. TensorList ops disabled.	Post-training Quantization