An Anomaly-based Intrusion Detection System for IoT Environments using Autoencoder Neural Networks and TinyML

Hiba Kandil
Mohammed V University in Rabat
Rabat, Morocco
https://orcid.org/0009-0008-1165-2596

Wiam Bouimejane
Mohammed V University in Rabat
Rabat, Morocco

Mohammed Mouhcine
Mohammed V University in Rabat
Rabat, Morocco

Hafssa Benaboud
Mohammed V University in Rabat
Rabat, Morocco
https://orcid.org/0009-0008-1165-2596

ABSTRACT

The scalable nature of IoT systems leads to continually evolving security challenges, threats, and device vulnerability to cyberattacks. The traditional Intrusion Detection Systems (IDS) struggle with the resource-limited nature of IoT devices. However, Machine Learning (ML) techniques have appeared as a promising solution for IDS, offering several benefits. In this paper, we introduce an unsupervised Deep Learning model combined with TinyML principles for efficient deployment of Intrusion Detection Systems on IoT networks. The model is trained exclusively on normal network traffic and detects anomalies through reconstruction error. To enable deployment on constrained devices, the model is quantized and converted to Lite format, resulting in a lightweight version suitable for TinyML environments. Evaluation was conducted using the IoT-23 dataset and NS-3-based traffic simulation. The proposed system enables real-time, on-device threat detection while operating within the strict memory, latency, and energy constraints typical of embedded IoT environments.

General Terms

Computer Science, Security, Internet of Things

Keywords

Intrusion Detection System, Autoencoder Neural Network, IoT Security, Deep Learning, Unsupervised Learning, TinyML.

1. INTRODUCTION

With the fast growth of the Internet of Things (IoT), many devices with insufficient resources are getting linked into our daily lives. While there are benefits to this, it also brings up new security problems [1, 2]. IoT devices tend to be easy targets since they lack processing power, memory, and battery life, making it difficult to use standard security measures.

Intrusion Detection Systems (IDS) that are designed to detect anomalous behavior, nowadays, can catch, analyze, and react to cyber threats instead of only focusing on known attack patterns. Thanks to new advancements in Tiny Machine Learning (TinyML) [3], we can now run small ML-based IDS models on microcontrollers. We can analyze data in real time on the device, allowing IDS to detect both known and novel intrusion attacks, while enhancing responsiveness and reducing reliance on human involvement.

In this paper, we create and set up an unsupervised anomaly detection system that uses autoencoder neural networks, designed to work on TinyML devices. The model learns from normal network traffic using the IoT-23 dataset [4] and finds odd patterns indicative of intrusions. While compacting and rebuilding the input data, the model recognizes anomalies with high reconstruction errors. After training, we shrink the model and optimize it to make it smaller and easier for real-time processing in an IoT environment, thanks to TinyML techniques. We deploy the proposed system in NS-3 traffic-simulated setups to validate this system contribution for providing a lightweight, adaptive Tiny-IDS solution.

This paper is organized as follows. The literature review delivered in Section 2. Section 3 provides the detailed model conception methodology, categorized into several parts. We give the evaluation of the enhanced model in section 4. Section 5 outlines the finding results, and section 6 concludes the paper.

2. RELATED WORK

A considerable amount of scientific contributions have been conducted to create intrusion detection models using machine learning algorithms. However, an anomaly-based intrusion detection system for IoT environments using autoencoder neural networks and TinyML remains a relatively new research topic. In this section, we present some contributions that explore related

approaches, highlight their strengths and limitations. In [5] the authors provide a network-based approach to malware detection using machine learning techniques for IoT. Principal component analysis (PCA), Random forest (RF), gradient descent (GD), and cross-validation were elected and conducted to preprocess data, evaluate, and examine the model performance efficiency in identifying the malicious and benign network traffic. The accuracy, precision, Recall, and F1 scores metrics results reveal that RF is more fitting for the IoT-23 dataset. However, the investigation lacks real-world deployment, model performance in large-scale traffic or multi-device environments. A hybrid intrusion detection framework combining autoencoders with traditional machine learning classifiers, tackling the challenge of intrusion detection with limited labeled data is introduced in [6]. The authors provide a solution for the small-sample problem that torments many intrusion detection systems. The autoencoder serves as a feature extractor that enhances generalization, even with less traffic data. The model applied to three datasets IoT-23, CIC-IDS-2017, and KDD CUP 99, demonstrates high detection accuracy and efficiency compared to advanced detection methods, especially in the context of IoT environments where data can be sparse. Still, the hybrid architecture may involve computational overhead, potentially limiting real-time deployment in resource-constrained IoT environments. Authors of [7] design a dual classification approach that uses machine learning models (Gradient Boosting (GB), Multi-layer Perceptron (MLP), and RF) to perform both binary classification (malicious vs benign) and multi-class classification (specific types of malware). The accuracy metric results reveal the triumph of RF. Nevertheless, details about feature engineering steps, the algorithms used, or performance metrics are limited, making it difficult to evaluate the approach rigorously. Other contributions are presented in [8–11]. Each of these contributions employs ML algorithms to detect various types of malicious traffic in IoT systems. However, none of them integrate TinyML-based approaches or focus specifically on autoencoder neural networks for real-time anomaly detection in resource-constrained IoT environments.
Other model [12] has achieved higher accuracy ($\approx$ 99%) on the dataset Edge-IIoTset, however the complex architecture use (LSTM-CNN, BiGRU-LSTM-Attention, and LSTM-CNN-Attention) though powerful, have substantial computational and memory requirements. These studies are likely suited for more capable edge servers or gateways rather than the most constrained IoT devices. This highlights a fundamental trade off between model performance and deployment feasibility on resource-constrained devices.
The quantized autoencoder model we propose in this paper, trained exclusively on benign traffic from the IoT-23 dataset, achieves strong performance ($F1 \approx 98\%$, $Precision \approx 96\%$) while being specifically designed for microcontroller deployment. With a tiny 47 KB model size and an estimated inference time of 7–10 ms on an ARM Cortex-M4, it represents a practical TinyML solution for endpoint security.

3. METHODOLOGY

3.1 Dataset and Feature Selection

For the development of an intrusion detection system with an autoencoder-based approach, the choice of dataset is paramount. The dataset must accurately reflect the unique characteristics of IoT network traffic, including its communication patterns and potential vulnerabilities. The IoT-23 dataset, is the chosen dataset for the proposed system. This decision was guided by multiple criteria, such as real-world IoT data, comprehensive labeling, rich features, and modern relevance. In addition, the IoT-23 dataset offers a robust foundation for anomaly detection research in IoT environments, balancing data realism, attack diversity, and suitability for embedded model training. From the extensive feature set available in IoT-23, we select key features that effectively capture IoT network behavior while remaining computationally efficient for deployment in resource-constrained environments. Table 1 shows the chosen features subset.

Table 1. Selected Network Features.

Feature	**Operational Significance**
duration	Attack flows often exhibit shorter/longer durations
orig_bytes	DDoS attacks show abnormal originator payloads
proto_tcp/udp	Protocol distribution anomalies
conn_state_*	Connection establishment patterns

3.2 Data Preprocessing

Effective data preprocessing is essential to prepare the raw network traffic data for optimal performance of the autoencoder model. The pipeline involved two main stages: benign traffic extraction and feature standardization. The extraction of benign traffic is crucial for the training step of the model. An autoencoder learns to reconstruct its input in an anomaly detection context with the normal behavior patterns training. Allowing the autoencoder to build an internal representation of expected network flows. For feature standardization, the Z-score normalization is applied to the numeric features within the selected subset.
Data balancing : the data was split into: 20% for testing, 80% for training (10% of it used as validation set).

3.3 Autoencoder-Based Anomaly Detection Model

3.3.1 Initial Proposed Autoencoder Model. Our anomaly detection system is built upon an autoencoder neural network [13]. The architecture consists of two main parts: an encoder, which compresses the input into a lower-dimensional representation, and a decoder, which reconstructs the input from this compressed representation. The goal is for the reconstructed output to be as close as possible to the original input.

—Encoding layers :
  —Dense layer with 32 neurons utilizing the Rectified Linear Unit (ReLU) activation function.
  —Dense layer with 16 neurons also using the ReLU activation function.

—Decoding layers :
  —Dense layer with 32 neurons gain employing the ReLU activation function.
  —Output layer with 9 neurons (Sigmoid activation)

The model was trained using the Mean Squared Error (MSE) [14] loss function and the Adam optimizer was chosen for its efficiency in converging during training.

3.3.2 Limitations of the Initial Proposed Model. While the initial model provided a foundational understanding, its performance and adaptability were limited by several architectural and training considerations:

—Shallow Architecture: The use of only two encoding and two decoding layers might have limited the model's capacity to learn

highly complex and nuanced patterns present in diverse network traffic.

—Simple Activation Functions: Only ReLU and Sigmoid activation functions were employed. While effective, exploring other activation functions might have improved the model's ability to capture non-linear relationships.

—Fixed Training Schedule: The initial training did not incorporate advanced strategies such as early stopping or adaptive learning rate mechanisms. This could lead to overfitting or suboptimal convergence.

—Minimal Hyperparameter Tuning: Parameters such as the learning rate and batch size were not extensively optimized. Optimal hyperparameters are crucial for achieving the best possible model performance.

3.3.3 Proposed Model Enhancement and Behavior Analysis. During the initial training of the model, we observed a significant drop in both training and validation loss during the early epochs. Indicating that the model quickly learned the basic patterns in the training data but failed to improve further, likely due to reaching its capacity limit or suboptimal training settings. To address this, we made several adjustments:

(1) Added EarlyStopping: We introduced an EarlyStopping callback during training. This prevents overfitting and saves computational resources.
(2) Adaptive Learning Rate: was implemented to dynamically reduce the learning rate during training if the validation loss stopped decreasing. This helps the model to fine-tune its weights more effectively when approaching the minimum of the loss function.
(3) Model Complexity: The early plateau suggested that the model's architecture might be too simple. We considered increasing its capacity by adding additional hidden layers or increasing the number of neurons in each layer.
(4) Regularization and Data Shuffling : We ensured the proper application of regularization techniques, to prevent overfitting and improve generalization. Furthermore, we verified that data shuffling was active during training at each epoch, to improve generalization and prevent the model from learning patterns specific to the order of data.
(5) Anomaly Dataset Integration: We introduced a separate anomaly dataset into the testing pipeline. This allowed us to evaluate the model's ability to distinguish between normal and anomalous patterns more effectively and refine the threshold selection strategy based on reconstruction error. It also supported real-world use cases where the system encounters previously unseen threats.

These changes led to more stable training and allowed the model to better generalize to unseen anomalies, preparing it for deployment. The upcoming section presents the evaluation of the enhanced model.

3.4 Optimized model architecture for training

—**Encoder:**
  —Dense layer (256 neurons, ReLU activation), followed by Batch Normalization and Dropout (0.2).
  —Dense layer (128 neurons, ReLU activation), followed by Batch Normalization and Dropout (0.2).
  —Dense layer (64 neurons, ReLU activation), followed by Batch Normalization.
  —Bottleneck: Dense layer (32 neurons, ReLU activation).
—**Decoder:**
  —Dense layer (64 neurons, ReLU activation), followed by Batch Normalization.
  —Dense layer (128 neurons, ReLU activation), followed by Batch Normalization and Dropout (0.2).
  —Dense layer (256 neurons, ReLU activation), followed by Batch Normalization and Dropout (0.2).
  —Output Layer: Dense layer (input_dim neurons, Sigmoid activation).

The model was compiled using the Adam optimizer with a learning rate of 0.0001 and trained to minimize the Mean Squared Error (MSE) loss.

4. EXPERIMENTS AND PERFORMANCE EVALUATION

To thoroughly evaluate the performance of our anomaly detection model, we employ several key metrics that provide a nuanced understanding beyond simple accuracy. These metrics are particularly important in imbalanced datasets, common in anomaly detection where normal instances significantly outnumber anomalies. The curves in figure 1 highlight the model performance metrics results.

—F1-score: 0.88

—AUC-ROC: 0.82

—Precision: 0.86

—Recall: 0.91

—False positive rate: 0.12

—False negative rate: 0.08

4.1 Model Deployment with TinyML

A significant challenge faced during this work was adapting the proposed model for deployment in IoT devices. The initial design prioritized high performance, which often necessitated a deep and complex model architecture. However, this complexity inherently conflicted with the severe constraints of IoT devices. These limitations present a critical bottleneck: a complex model either requires significant simplification of its architecture, or it demands the application of advanced model compression techniques such as quantization and pruning.

4.1.1 Quantized Autoencoder Performance. The adaptation process involved a series of structural and computational optimizations aimed at reducing the model's memory footprint and computational complexity, without significantly degrading detection performance. The end goal was to produce a lightweight, fast, and power-efficient anomaly detection model suitable for integration with microcontroller-class hardware.
The process was guided by the principles of TinyML, which emphasizes the importance of minimizing resource consumption while maintaining sufficient accuracy for real-time edge intelligence. Among the various approaches explored, quantization emerged as the most viable technique for compressing the model while preserving its functionality.

4.1.2 Quantization. The optimization process for quantization comprised the following key steps:

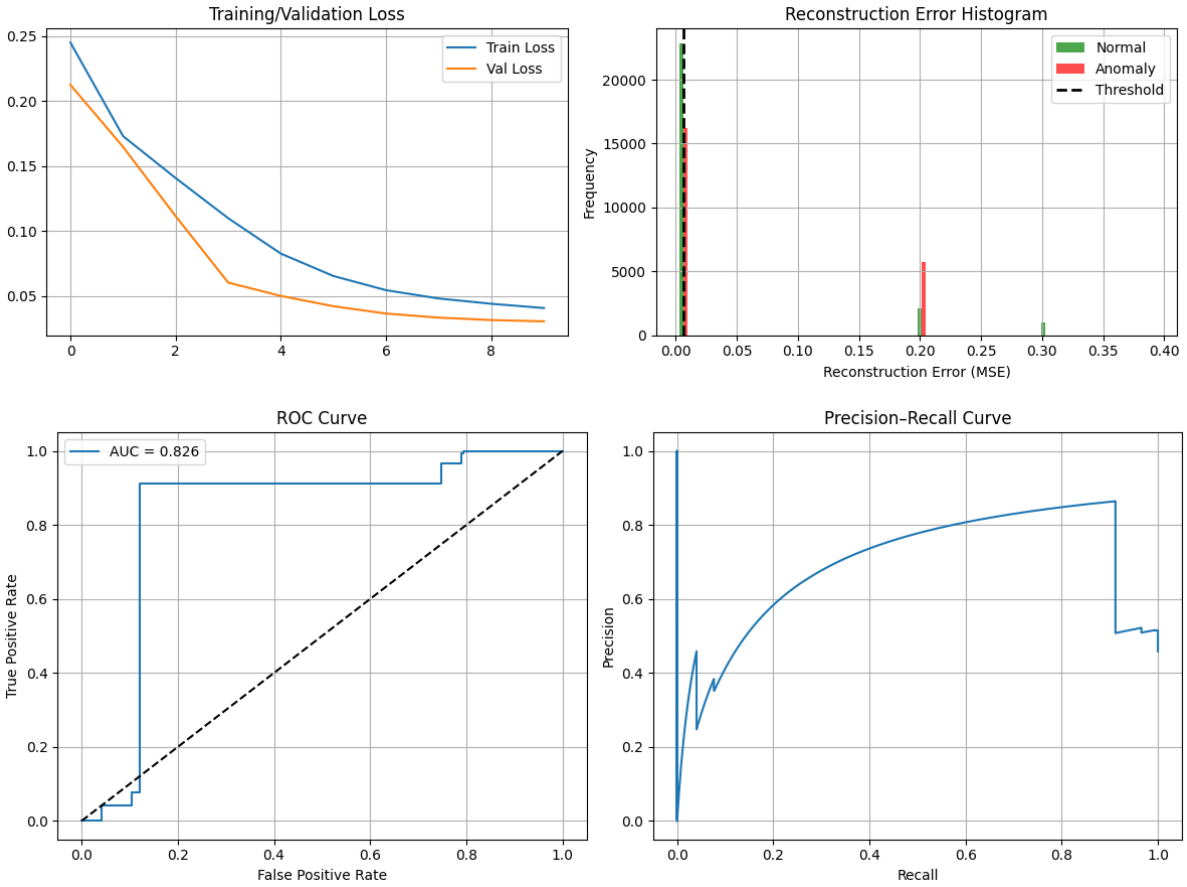

Fig. 1. Model performance metrics results.

—Reducing network complexity: The number of neurons in the hidden layers was decreased to make the model shallower and more computationally efficient. This step ensures compatibility with devices that have limited RAM and CPU power.
—Removing dropout layers: during inference Dropout introduces unnecessary overhead and is thus removed. This further simplifies the computational graph of the model.
—Applying post-training quantization: The model's weights and activations were converted from 32-bit floating-point precision to 8-bit integers, significantly reducing model size and enabling faster inference.

4.1.3 The final simplified architecture for the quantization

—**Encoding layers:**
  —Dense layer with 128 neurons utilizing the Rectified Linear Unit (ReLU) activation function.
  —Dense layer with 64 neurons also using the ReLU activation function.
  —Dense layer with 32 neurons also using the ReLU activation function.
  —Dense layer with 16 neurons using the ReLU activation function, serving as the bottleneck layer.
—**Decoding layers:**
  —Dense layer with 32 neurons employing the ReLU activation function.
  —Dense layer with 64 neurons employing the ReLU activation function.
  —Dense layer with 128 neurons employing the ReLU activation function.
  —Output layer with a number of neurons equal to the input dimension (input_dim), utilizing a ReLU activation function to reconstruct the input.

In the following sections, we detail the reconstruction error analysis of the quantized model, explain our anomaly detection logic and thresholding strategy, and discuss the trade-offs introduced by quantization. We also provide empirical estimates of the resource efficiency gains obtained through these optimizations and compare the performance of the original and quantized models.

4.2 Reconstruction Error Analysis

To assess the performance of the quantized model, we computed the MSE between the input data and the reconstructed outputs. This reconstruction error is the core mechanism for anomaly detection in autoencoders: normal traffic should have low reconstruction error, while anomalies, being deviations from learned patterns, should result in significantly higher errors. The simplified and quantized architecture successfully maintained its ability to accurately reconstruct normal patterns, while

consistently highlighting anomalies through notably higher reconstruction errors. This demonstrated that the optimization process did not compromise the model's fundamental anomaly detection capability. The figure 2 provides the Reconstruction Error of the Quantized Autoencoder with Optimized Threshold.

4.3 Anomaly Detection Logic and Thresholding

Instead of relying on an arbitrary or fixed percentile threshold for anomaly detection, we implemented a data-driven approach to optimize the threshold value. This method aimed to achieve the best balance between identifying true anomalies and minimizing false alarms.

—F1-score Optimization: We systematically calculated the F1-score across a range of possible threshold values. The F1-score is a crucial metric for imbalanced datasets, it balance the concerns of false positives and false negatives.
—Automated Threshold Selection: The threshold that maximized the F1-score was automatically selected. This data-driven approach ensured that the final threshold achieved an optimal balance between minimizing false positives and false negatives.
—Classification Rule: Based on the optimized threshold, samples were classified as follows:
  —Normal samples were those with an MSE less than or equal to the selected threshold.
  —Anomalous samples were those with an MSE greater than the selected threshold.

4.4 Quantization and Performance Insights

The overall process of quantization and model simplification yielded several key performance insights:

—The simplified architecture was able to maintain good performance in anomaly detection. This indicates that a more compact model can still learn the essential patterns for effective detection.
—8-bit quantization significantly reduced the model size. Crucially, this size reduction was achieved with minimal degradation in detection performance, demonstrating the effectiveness of post-training quantization.
—The use of integer-only operations, a direct result of quantization, enabled substantially faster inference times on edge devices that are optimized for such computations.
—The auto-selected threshold strategy proved superior to fixed percentile approaches, leading to a more robust and balanced anomaly detection system.
—Conversion to Lite format ensured broad compatibility with various target deployment environments, including Microcontroller-class devices, IoT gateways, General edge computing nodes

4.5 Estimated Resource Requirements

While actual performance may vary based on deployment hardware, we provide the following estimates based on public benchmarks for ARM Cortex-M4 devices:

—Model size: The quantized model occupies approximately 47 KB, which fits comfortably within typical flash memory budgets.
—Memory usage: Based on tensor allocation estimates, peak RAM usage is expected to remain below 25 KB.

Table 2. Simulation Traffic Parameters.

Parameter	Normal Traffic	Attack Traffic
Protocol	UDP	UDP/TCP
Packet Size	512B	64B-1024B
Data Rate	5kbps	10kbps-1Mbps
Duration	Continuous	10-40s bursts
Flow Types	Unidirectional	Bidirectional

—Inference latency: On 80 MHz Cortex-M4 microcontrollers, similar 8-bit dense networks achieve inference times between 7–10 ms.
—Energy use: Assuming a current draw of 20 mA at 3.3V during inference, energy consumption per prediction is estimated at 0.66 millijoules.

These estimates suggest that the system remains within the operational limits of typical TinyML deployment environments. Future work should involve validation on physical hardware to confirm these figures.

4.6 Simulation Framework and Detection Performance

4.6.1 Simulation Framework. To rigorously evaluate our anomaly detection system, we designed a custom simulation environment using NS-3, a Network Simulator [15]. This framework allowed us to generate realistic IoT network traffic, including both normal operational patterns and various attack scenarios, providing a controlled representative testing ground.

4.6.2 NS-3 Simulation Setup. The simulation topology consisted of:

—IoT Devices: Five IoT devices were simulated, each generating periodic sensor data. This normal traffic was characterized by the UDP protocol and a data rate of 5kbps.
—Gateway Node: A single gateway node was included to aggregate traffic from the IoT devices. This represents a common point of collection and potential vulnerability in an IoT network.
—Attacker Node: One attacker node was configured to implement various attack types, simulating malicious activities. Specifically, it launched:
  —DDoS attacks: Implemented as UDP floods with a data rate of 1Mbps, lasting between 20 to 40 seconds.
  —Port scanning: Conducted using the TCP protocol, targeting 100 ports over durations of 30 to 35 seconds.

The point-to-point links used 10Mbps bandwidth with 2ms delay, replicating typical IoT network constraints. We implemented IPv4/IPv6 dual-stack support through FlowMonitor's protocol-agnostic tracing.

4.6.3 Traffic Generation Parameters. Table 2 summarizes the parameters used to generate normal and attack traffic within the NS-3 simulation.

4.6.4 Flow Monitoring Implementation. The NS-3 FlowMonitor module captured comprehensive flow statistics:

$$\text{FlowRecord} = \{t_{\text{first}}, t_{\text{last}}, \Delta t, n_{\text{tx}}, n_{\text{rx}}, B_{\text{tx}}, B_{\text{rx}}, \text{proto}, \text{state}\}$$

Where:

—t_{first}, t_{last}: Timestamps of first/last packets
—Δt: Flow duration
—n_{tx}, n_{rx}: Transmitted/received packet counts
—B_{tx}, B_{rx}: Transmitted/received bytes

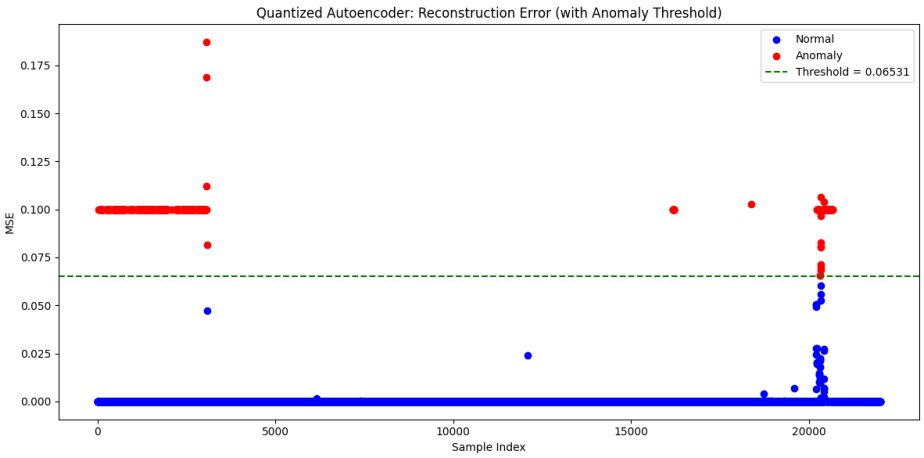

Fig. 2. Reconstruction error (quantized autoencoder) with optimized threshold.

Table 3. Threshold Performance Comparison.

Metric	Default	Balanced	High-Recall
Threshold	0.1000	0.3202	0.2241
Attack Recall	100%	11.9%	100%
FP Rate	100%	4.8%	4.8%
F1 Score	0.66	0.20	0.98

—`proto`: Protocol

—`state`: Connection state flags

Following the raw data collection, several key preprocessing steps were applied:

(1) XML-to-CSV Conversion: The raw XML output from FlowMonitor was converted to a more manageable CSV format, ensuring all flow metadata was preserved.

(2) Protocol Normalization: IPv4 and IPv6 specific fields were normalized into common features to ensure consistency across different IP versions.

(3) Temporal Feature Calculation: New features, such as packets per second and byte rates, were calculated to provide more descriptive temporal characteristics of the flows.

(4) Connection State Classification: Connection state flags were categorized into a simplified set of classes (e.g., SF for normal establishment and termination, REJ for rejected connections, S0 for connection attempts with no reply).

5. RESULTS AND ANALYSIS

5.1 Threshold Optimization

The performance of an anomaly detection system heavily depends on the chosen threshold for distinguishing normal from anomalous reconstruction errors. Through rigorous optimization, a high-recall threshold configuration was identified as achieving optimal operational performance. This configuration balances the need to detect as many attacks as possible with the acceptable rate of false positives. The threshold performance comparison is presented in Table 3 .

5.2 Discussion of Findings

The implemented system demonstrated good performance, making it suitable for real-time deployment in IoT environments. The observed false positive rate is considered acceptable for IoT security applications, where the prevention of attacks often takes precedence over extremely low alert precision. This indicates that the system is effective in identifying threats without generating an overwhelming number of benign alerts. Our current system has been validated through simulation and offline evaluation. Testing the quantized model on real IoT devices will be crucial to assess latency, power consumption, and robustness.

5.3 Limitations and Future Work

This model presents limitations that point to opportunities for future research. The model may struggle with complex, multi-stage attacks and Although initial support for IPv6 traffic is included further testing is required to ensure robust ipv6 handling. While achieving high recall, the false positive rate remains non-negligible, and the static thresholding approach may not adapt well to dynamic network environments. Moreover, the model lacks physical deployment validation on resource-constrained hardware. Future work could explore dynamic threshold adaptation, protocol-specific detection modules, and federated learning capabilities to enhance real-world applicability and privacy preservation.

6. CONCLUSION

This paper proposed and implemented a lightweight, anomaly-based Intrusion Detection System tailored for IoT environments, leveraging the capabilities of autoencoder neural networks and TinyML techniques. By training exclusively on benign network traffic, the system was designed to detect previously unseen threats through reconstruction error analysis. The integration of TinyML optimization techniques enabled the model to be deployed on resource-constrained microcontroller-class devices. Through a rigorous evaluation process involving both real-world datasets (IoT-23) and simulated traffic generated using NS-3, the system demonstrated

high detection accuracy, low inference latency, and robustness against common attack types such as DDoS and port scanning. Key optimizations ensured that performance remained acceptable even under strict memory and energy constraints typical of embedded devices. However, the system is not without limitations and has not yet been deployed or validated on physical hardware, which may introduce further deployment enhancement on constraints or edge-case scenarios to push the model's efficiency even further.

7. REFERENCES

[1] H. Kandil and H. Benaboud, "Using Machine Learning to Deal with Privacy and Confidentiality in Internet of Things: An Overview," In: Innovations in Smart Cities Applications Volume 8. SCA 2024. Lecture Notes in Networks and Systems, vol 1310. 2025. Springer, Cham. https://doi.org/10.1007/978-3-031-88653-9_73

[2] H. Kandil and H. Benaboud, "About Scalability and End-to-End Security in Internet of Things: An Overview," In Proceedings of the 7th International Conference on Networking, Intelligent Systems and Security (NISS'24). Association for Computing Machinery, New York, NY, USA, Article 6, 1–7. 2024. https://doi.org/10.1145/3659677.3659690

[3] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, and S. Han, "Tiny Machine Learning: Progress and Futures," arXiv preprint arXiv:2403.19076, Mar. 2024. [Online]. Available:https://arxiv.org/abs/2403.19076

[4] Stratosphere Laboratory, "IoT-23: A labeled dataset with malicious and benign IoT network traffic," Jan. 2020. [Online]. Available:https://www.stratosphereips.org/datasets-iot23

[5] R. Butt, N. Tariq, M. Humayun, and A. Ishaq, "Malware Detection in Network Traffic Data for Internet of Things," in 2024 26th International Multi-Topic Conference (INMIC), Dec. 2024, pp. 1–6. doi:10.1109/INMIC64792.2024.11004366.

[6] N. Wei et al., "An Autoencoder-Based Hybrid Detection Model for Intrusion Detection With Small-Sample Problem," IEEE Transactions on Network and Service Management, vol. 21, pp. 2402–2412, 2024, doi:10.1109/TNSM.2023.3334028.

[7] A. Ahli, A. Raza, K. O. Akpinar, and M. Akpinar, "Binary and Multi-Class Classification on the IoT-23 Dataset," in 2023 Advances in Science and Engineering Technology International Conferences (ASET), Feb. 2023, pp. 1–7. doi:10.1109/ASET56582.2023.10180848.

[8] N. Abdalgawad, A. Sajun, Y. Kaddoura, I. A. Zualkernan, and F. Aloul, "Generative Deep Learning to Detect Cyberattacks for the IoT-23 Dataset," IEEE Access, vol. 10, pp. 6430–6441, 2022, doi:10.1109/ACCESS.2021.3140015.

[9] C. V. Oha et al., "Analysis of IoT-23 datasets and machine learning models for malicious traffic detection," ERA. Accessed: Jul. 16, 2025. Available:https://era.library.ualberta.ca/items/28700e1c-3afc-4892-b12e-5c229df9e056.

[10] A. K. Sahu, S. Sharma, M. Tanveer, and R. Raja, "Internet of Things attack detection using hybrid Deep Learning Model," Computer Communications, vol. 176, pp. 146–154, Aug. 2021, doi:10.1016/j.comcom.2021.05.024.

[11] M. Woźniak, J. Siłka, M. Wieczorek, and M. Alrashoud, "Recurrent Neural Network Model for IoT and Networking Malware Threat Detection," IEEE Transactions on Industrial Informatics, vol. 17, no. 8, pp. 5583–5594, Aug. 2021, doi:10.1109/TII.2020.3021689.

[12] A. Gueriani, H. Kheddar and A. C. Mazari, "Adaptive Cyber-Attack Detection in IIoT Using Attention-Based LSTM-CNN Models," 2024 International Conference on Telecommunications and Intelligent Systems (ICTIS), Djelfa, Algeria, 2024, pp. 1-6, doi: 10.1109/ICTIS62692.2024.10894509.

[13] F. Chollet, "An Introduction to Autoencoders," in Deep Learning with Python, Manning Publications, 2017. https://blog.keras.io/building-autoencoders-in-keras.html

[14] A. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009. https://doi.org/10.1145/1541880.1541882

[15] NS-3 Consortium, "ns-3 network simulator," [Online]. Available:https://www.nsnam.org