XAI-Driven TinyML with Compact Models for Compromised IoT Device Detection

Bodapatla Rakesh
Indian Institute of Information Technology and Management
Gwalior
2021IMT-022

Gulla Akshaya
Indian Institute of Information Technology and Management
Gwalior
2021IMT-038

NVNL Ishwarya
Indian Institute of Information Technology and Management
Gwalior
2021IMT-070

Vennapusa Ramana Reddy
Indian Institute of Information Technology and Management
Gwalior
2021IMT-110

***Abstract*—The growing adoption of Internet of Things (IoT) devices has introduced critical challenges in ensuring their security, especially given their deployment in resource-constrained environments. Traditional Machine Learning (ML) solutions, though effective, are often too computationally intensive for direct deployment on IoT end devices. This research explores the use of Tiny Machine Learning (TinyML) techniques augmented with Explainable AI (XAI) to detect compromised IoT devices with high accuracy and low resource overhead. Leveraging a subset of the IoT23 dataset, our study focuses on model optimization through techniques like quantization and knowledge distillation, significantly reducing model size while maintaining performance. Furthermore, SHAP and LIME are employed to provide interpretability by identifying impactful features, ensuring transparency in predictions. Experimental simulations on devices like Raspberry Pi 4 and Arduino Nano 33 BLE Sense demonstrate the feasibility of deploying compact ML models in real-world IoT settings. Our findings contribute to the development of energy-efficient, interpretable, and deployment-ready ML models, enabling secure and intelligent monitoring across IoT ecosystems.**

I. Introduction

A. Internet of Things

The Internet of Things (IoT) has revolutionized the way we interact with the world around us. It encompasses a vast network of interconnected devices, ranging from everyday appliances like smart thermostats and voice assistants to industrial sensors and complex medical equipment. These devices collect and exchange data, enabling automation, remote monitoring, and enhanced functionality across various sectors.

The applications of IoT are widespread and continuously evolving. In smart homes, thermostats adjust to our preferences, refrigerators monitor food supplies, and security systems provide real-time alerts. Smart cities leverage connected sensors for traffic management, waste collection optimization, and environmental monitoring. The healthcare industry utilizes wearable devices for patient health tracking and remote monitoring, while industrial applications employ IoT sensors for predictive maintenance and improved operational efficiency.

Despite the undeniable benefits, the expanding IoT landscape presents a growing concern: security vulnerabilities. Due to their inherent resource limitations and focus on functionality, many IoT devices lack robust security protocols. This makes them prime targets for cyberattacks, potentially compromising data privacy, disrupting critical operations, and even causing physical harm. Attackers can exploit these vulnerabilities to gain unauthorized access, steal sensitive data, or disrupt device functionality. Malicious actors can also leverage compromised IoT devices to launch large-scale attacks on networks and critical infrastructure.

This research focuses on improving the security of IoT devices by making machine learning models smaller, faster, and easier to understand. We investigate how lightweight

ML models can be used to detect if an IoT device has been compromised, while also ensuring the model can run directly on edge devices with limited resources. By using techniques like Knowledge Distillation and Explainable AI, we aim to build compact models that not only perform well but also explain their predictions clearly. This helps in making real-time decisions while saving energy and increasing trust in automated systems.

B. Intrusion Detection Systems

Intrusion Detection Systems (IDS) act as a vigilant guardian within network security ecosystems. These systems function by continuously monitoring network traffic and system activity, searching for any signs of malicious intent or policy violations. Through meticulous analysis of network packets, system logs, and other data sources, IDSs can pinpoint suspicious patterns or behaviors that may signal an ongoing attack.

There are two main classifications of IDSs, distinguished by their deployment strategy:

1) Network-based Intrusion Detection Systems (NIDS): These systems function as watchful sentinels on network segments or at the network perimeter, meticulously examining network traffic flowing across those points. NIDS meticulously analyze network packets for known attack signatures, anomalies in traffic patterns, or content that transgresses security policies.

2) Host-based Intrusion Detection Systems (HIDS): As the name implies, HIDS reside on individual devices within a network, acting as vigilant guards on those specific systems. They monitor system activity for any suspicious events, unauthorized access attempts, or modifications to critical system files. HIDS are particularly adept at detecting insider threats or attacks that have infiltrated past perimeter defenses.

The advantages of IDSs are undeniable. They function as an early warning system, providing security teams with a critical head start in detecting ongoing attacks. This allows for swift action to mitigate potential damage. Additionally, IDSs offer valuable insights into network traffic patterns and potential threats, empowering security teams to prioritize their response efforts and allocate resources effectively. Furthermore, the presence of an IDS can act as a psychological deterrent, reminding attackers that their actions are under constant scrutiny.

However, IDSs also have limitations. One challenge is the potential for false positives. Overly cautious IDSs can generate alerts for harmless activity, leading to alert fatigue for security personnel and potentially delaying the identification of genuine threats. Another challenge is the ever-evolving threat landscape. Attackers are constantly developing new tactics and techniques to evade detection. Consequently, IDSs require regular updates to maintain effectiveness against the latest threats. Finally, while some advanced IDSs can take automated actions to isolate compromised systems, most rely on security personnel to investigate and respond to detected threats.

As IoT devices become more widespread, traditional IDS solutions struggle with resource limitations and lack of interpretability. This project addresses those challenges by combining Explainable AI (XAI) with TinyML to build lightweight, interpretable intrusion detection models. Using techniques like LIME, SHAP, quantization, and knowledge distillation, we create compact models that can run efficiently on edge devices like Raspberry Pi and Arduino Nano—ensuring real-time security without sacrificing transparency or performance.

C. Role of TinyML and Explainable AI in IoT Intrusion Detection

Role of TinyML and Explainable AI in IoT Intrusion Detection Tiny Machine Learning (TinyML) and Explainable Artificial Intelligence (XAI) offer a powerful solution to the limitations of traditional IDS in IoT environments. IoT devices, often constrained by limited memory, processing power, and energy, require models that are both efficient and lightweight. TinyML addresses this by enabling the deployment of compact machine learning models directly on edge devices, reducing reliance on cloud resources and ensuring real-time responses. At the same time, XAI techniques such as LIME and SHAP provide interpretability, allowing security analysts and system developers to understand why a model flags a device as compromised which is crucial for trust and accountability in security applications.
There are several ways this integrated approach enhances IDS for IoT:

1) Efficient Anomaly Detection: : TinyML enables models to detect behavioral deviations in real-time, directly on the device, without cloud latency or power drain.

2) Transparent Classification: : With XAI, even lightweight models can offer explanations for their classifications, helping to identify the features that triggered a threat label.

3) Model Optimization: : Techniques like quantization and knowledge distillation reduce model complexity and size, making them ideal for microcontroller-class devices.

Together, TinyML and XAI form a robust, interpretable, and scalable foundation for modern intrusion detection systems in IoT networks—offering both performance and transparency without compromising on resource efficiency.

II. Past Work

The Problem of Intrusion using IOT 23 Dataset was previously worked upon by Cruz et al.[1]. Their Paper used only

XGBoost to perform a task of Binary classification. This has ignored the nuances that such a Multiclass Data Represents. The Proposed model could be optimized further by taking this into account. Furthermore, their paper was limited to only one model. Multiple models would prove to be more useful in such tasks.

Another significant work is by Tekin et al.[2], who analysed energy consumption of on-device machine learning models for IoT intrusion detection. Their study examined key factors such as model inference time, training time, power consumption, and energy efficiency. However, their work has certain limitations. In practice, when the model size is several MB's, quantization methods may not prove to be effective due to the potential loss in model accuracy. To overcome these shortcomings, we propose utilizing knowledge distillation as an alternative approach, which can significantly reduce the model size while retaining its performance.

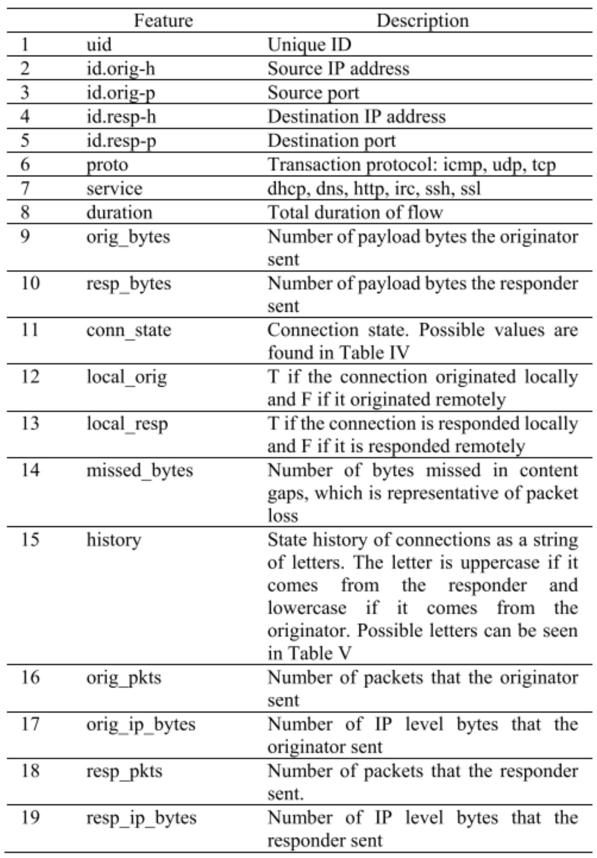
III. The Datset

The dataset chosen for this research is the famous, IoT23 Dataset.

IoT-23 introduces a novel dataset comprising network traffic data emanating from Internet of Things (IoT) devices. The dataset encompasses 20 instances of malware activity observed in IoT devices, alongside 3 instances of benign IoT device traffic. Its primary objective is to furnish researchers with a substantial dataset containing labeled instances of both IoT malware infections and benign IoT traffic, facilitating the development of machine learning algorithms. Each of the twenty-three captures, referred to as scenarios, represents distinct IoT network traffic behaviours. In malicious scenarios, specific malware samples were executed on Raspberry Pi devices, utilizing various protocols and executing diverse actions. Conversely, benign scenarios involved capturing network traffic from actual IoT devices, including a Philips HUE smart LED lamp, an Amazon Echo home intelligent personal assistant, and a Somfy smart door lock. Notably, these IoT devices are authentic hardware, not simulations, enabling the capture and analysis of real network behavior.

IV. Data Preprocessing

The features 'local_orig' and 'local_resp' were empty across all files, leading to their removal. Following established practices in pre-processing intrusion detection datasets, features containing IP addresses and port numbers were also discarded. Additionally, the 'history' feature, representing a sequence of connection values, was initially excluded. The correlation graph revealed a strong correlation between 'orig_pkts' and 'orig_ip_bytes', as well as 'resp_pkts' and 'resp_ip_bytes', resulting in the removal of the latter correlated features. Extreme minority classes containing fewer than 100 samples, such as 'C&C-FileDownload' and 'FileDownload', were eliminated. Null values within various features were eliminated. Categorical features like 'service', 'proto', and 'conn_state' underwent one-hot encoding, a common technique in previous studies, followed by normalization between 0 and 1 using min-max scaling.



	Feature	Description
1	uid	Unique ID
2	id.orig-h	Source IP address
3	id.orig-p	Source port
4	id.resp-h	Destination IP address
5	id.resp-p	Destination port
6	proto	Transaction protocol: icmp, udp, tcp
7	service	dhcp, dns, http, irc, ssh, ssl
8	duration	Total duration of flow
9	orig_bytes	Number of payload bytes the originator sent
10	resp_bytes	Number of payload bytes the responder sent
11	conn_state	Connection state. Possible values are found in Table IV
12	local_orig	T if the connection originated locally and F if it originated remotely
13	local_resp	T if the connection is responded locally and F if it is responded remotely
14	missed_bytes	Number of bytes missed in content gaps, which is representative of packet loss
15	history	State history of connections as a string of letters. The letter is uppercase if it comes from the responder and lowercase if it comes from the originator. Possible letters can be seen in Table V
16	orig_pkts	Number of packets that the originator sent
17	orig_ip_bytes	Number of IP level bytes that the originator sent
18	resp_pkts	Number of packets that the responder sent.
19	resp_ip_bytes	Number of IP level bytes that the responder sent

Fig. 1: Features and their descriptions.

V. Feature Selection using XAI

Explainable Artificial Intelligence (XAI) is all about making AI systems more transparent and understandable to humans. As AI becomes more integrated into critical areas like healthcare, finance, and law, it's no longer enough for a model to simply give accurate results—it must also be able to explain how and why it arrived at those results. XAI helps bridge the gap between complex machine learning models and human reasoning by offering insights into the decision-making process, highlighting which features were most influential or what patterns the model recognized. This not only builds trust but also allows users to detect potential biases or errors, making AI systems more reliable and accountable.

Local and Global Explanations in XAI

In Explainable Artificial Intelligence (XAI), we often talk about two types of explanations: local and global. These help us understand how an AI model makes decisions.

Local explanations focus on a single prediction. They help us understand why the model gave a certain result for one specific input. For example, if an AI model predicts that a person should not get a loan, a local explanation can show which features (like income, credit score, or debt) influenced that decision. Methods like *LIME* were used for Local Explanations and *SHAP* was used for global explanations.

Global explanations, on the other hand, give us an overall view of how the model works with the entire dataset. They help answer questions like: Which features are most important in general? What patterns does the model follow when making predictions? Global explanations are useful for understanding the model's general behavior and making sure it is fair and reliable.

We have utilised global explanations to select most relevant and important features for the models.

LIME: Local Interpretable Model-Agnostic Explanations

LIME is a technique designed to explain the predictions of any machine learning model by learning an interpretable model locally around the prediction. Instead of trying to understand the entire model (which might be complex and non-linear), LIME focuses on explaining individual predictions using a simple, interpretable model like a linear model or a decision tree.

Let $f : R^d \to R$ be the original complex model that we want to explain. Given a data instance $x \in R^d$, LIME tries to approximate f locally around x using an interpretable model g (such as a linear model).

Mathematical Formulation: To approximate the original model f around an instance x, LIME perturbs the input x to generate a set of synthetic data points $\{z_i\}_{i=1}^N$. For each perturbed sample z_i, we obtain the prediction $f(z_i)$ from the original model.

A proximity measure $\pi_x(z)$ is used to assign weights to the perturbed samples based on how close they are to the original point x. A common choice for $\pi_x(z)$ is the exponential kernel:

$$\pi_x(z) = \exp\left(-\frac{D(x,z)^2}{\sigma^2}\right)$$

Where:

- $D(x,z)$ is a distance function (e.g., Euclidean distance).
- σ controls the width of the kernel (defines how "local" the locality is).

The goal is to fit an interpretable model $g \in G$, such as a linear model, that minimizes the weighted loss over the perturbed dataset:

$$\mathcal{L}(f,g,\pi_x) = \sum_{i=1}^{N} \pi_x(z_i)\left(f(z_i) - g(z_i)\right)^2$$

This is essentially a weighted least squares regression problem, where:

- $f(z_i)$ is the prediction of the black-box model.
- $g(z_i)$ is the prediction of the interpretable surrogate model.
- $\pi_x(z_i)$ acts as the weight for each sample in the regression.

The final objective remains:

$$\arg\min_{g\in G} \mathcal{L}(f,g,\pi_x) + \Omega(g)$$

Here, $\Omega(g)$ ensures that the learned model remains interpretable, typically by limiting the number of non-zero coefficients (L1 regularization) or tree depth in case of decision trees.

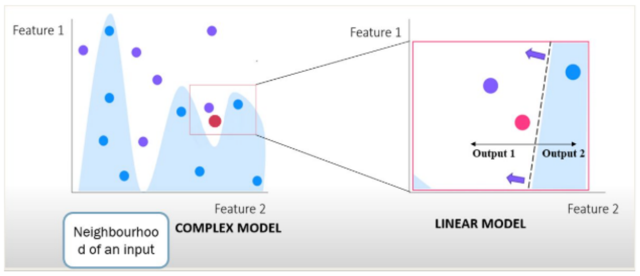


Fig. 2: Original dataset with a complex, non-linear decision boundary.

Visualization and Interpretation: In the above Figure3, we see a dataset with a complex decision boundary that cannot be easily understood or explained directly.

This figure shows the selected data point (in red) whose prediction we want to explain. A local neighborhood around this point is identified using a similarity or distance function (e.g., Euclidean distance).

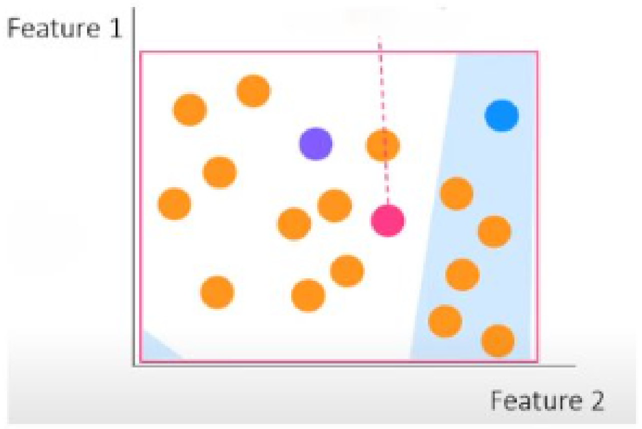


Fig. 3: Perturbed local neighborhood shown in orange

Finally, the neighborhood is perturbed by creating new samples as shown in Figure4, and the original model's predictions on these samples are collected. A simple interpretable model (e.g., linear regression) is then fit on these weighted samples to approximate the original model's behavior around the red point. This surrogate model provides feature importance scores that help explain the original prediction.

SHAP (SHapley Additive exPlanations)

SHAP is a game-theoretic approach to explain the output of any machine learning model. It connects ideas from cooperative game theory with local explanations. The method assigns an importance value (called the Shapley value) to each feature for a particular prediction.

Shapley Values from Cooperative Game Theory: Suppose we have a game where:

- The "players" are the input features.
- The "payout" is the prediction $f(x)$ from the model.
- The "value function" $v(S)$ denotes the contribution of a subset of features $S \subseteq F$.

The Shapley value for a feature i is defined as:

$$\phi_i(f, x) = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{i\}}(x) - f_S(x) \right]$$

Where:

- F is the full set of features.
- $S \subseteq F \setminus \{i\}$ is a subset of features not including i.
- $f_S(x)$ is the expected prediction when only the features in subset S are known.
- The term $f_{S \cup \{i\}}(x) - f_S(x)$ is the marginal contribution of feature i when added to subset S.

SHAP in Machine Learning: In SHAP, the prediction of a model is decomposed as:

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i$$

Where:

- ϕ_0 is the expected model prediction (i.e., the average over all data points).
- ϕ_i is the Shapley value for feature i, indicating how much i contributes to deviating from the expected prediction.

Using the shapley values, we select top 5 features which are most releavant for the model.

VI. Model optimisation using TinyML

Traditional Deep Learning models often encounter limitations such as large size, computational complexity, and high energy consumption. These factors make them less suitable for deployment on resource-constrained IoT end devices. A common approach to mitigate this issue is offloading complex computational tasks from sensor nodes to edge gateways.

However, this method may not always be efficient, as the energy required to transmit data between devices can exceed the energy used to run an ML model on the device itself.

In response to these challenges, TinyML has emerged as a specialized field focused on optimizing ML models for deployment on IoT end devices with limited resources.

A. Quantization

Quantization is a technique that reduces both the model size and computational cost by converting high-precision floating-point numbers into lower-precision formats, such as 8-bit integers. This enables faster inference and more efficient deployment, especially on resource-constrained devices like edge devices and IoT systems. The process can be compared to compressing a high-resolution image into a lower-resolution one: while some details may be lost, the result is much smaller and quicker to load.

However, this reduction in precision introduces a quantization error, which can impact model performance by creating a trade-off between size and accuracy. This error arises from the approximation of values, where certain fine-grained details are sacrificed for the sake of efficiency. Despite this trade-off, quantization remains a crucial method for optimizing machine learning models in practical, real-world applications.

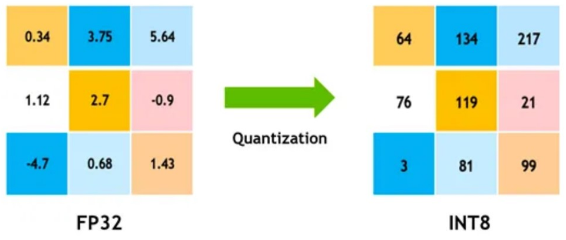


Fig. 4: Illustration of the Quantization Process.

B. Knowledge Distillation

Knowledge Distillation (KD) transfers knowledge from a large, complex teacher model to a smaller, more efficient student model. This is particularly beneficial in resource-constrained environments, where deploying large models is impractical due to limitations in memory, processing power, and inference time.

The teacher model, typically a deep and over-parameterised network, is trained on the dataset and provides soft labels (probabilities) that guide the training of the student model. The student model, being smaller and more efficient, learns both from the true labels and the soft labels from the teacher, capturing richer data distribution information.

The distillation process minimises the difference between the teacher's and the student's predictions using a combined loss function, which incorporates both the standard cross-entropy loss and the distillation loss. The distillation loss L_{KD} is defined as:

$$L_{KD} = \alpha \cdot L_{\text{CE}}(y, \hat{y}) + (1 - \alpha) \cdot T^2 \cdot L_{\text{KL}}(q_{\text{teacher}}, q_{\text{student}})$$

Where: - $L_{\text{CE}}(y, \hat{y})$ is the cross-entropy loss between the true labels y and the student's predictions $\hat{y}$, - $L_{\text{KL}}(q_{\text{teacher}}, q_{\text{student}})$ is the Kullback-Leibler divergence between the teacher's and student's probability distributions, - T is the temperature parameter that smooths the soft labels for better learning by the student, - α is a balancing factor between the true labels and the soft labels.

The teacher's output q_{teacher} and the student's output q_{student} are computed as:

$$q_{\text{teacher}} = \frac{\exp(z_{\text{teacher}}/T)}{\sum_j \exp(z^j_{\text{teacher}}/T)}$$

$$q_{\text{student}} = \frac{\exp(z_{\text{student}}/T)}{\sum_j \exp(z^j_{\text{student}}/T)}$$

By minimizing this loss, the student model mimics the teacher's behavior while being more computationally efficient. This approach significantly enhances the performance of small models, making them viable for deployment in real-world, resource-limited scenarios.

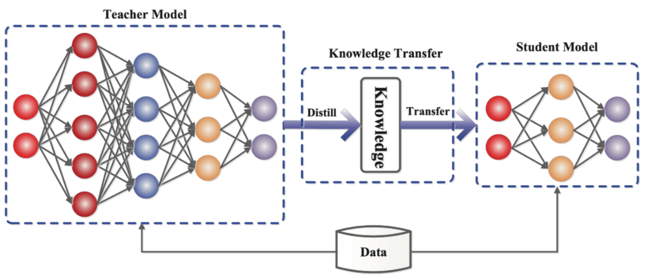


Fig. 5: Illustration of Knowledge Distillation Process.

VII. Experimental Setup

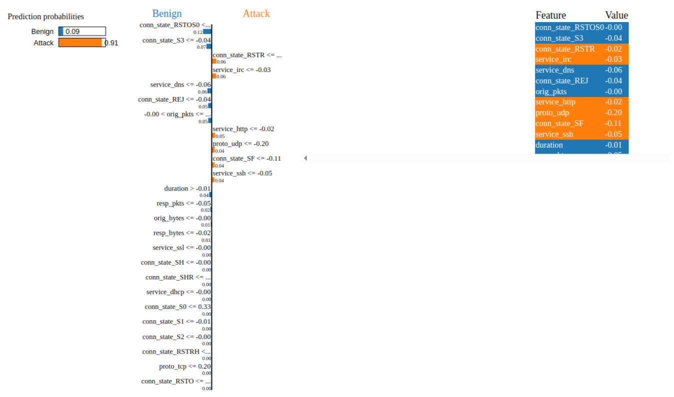
We compare the performance of our work with that of the base paper on different platforms, including cloud, edge, and IoT devices. The setup includes varying hardware configurations, as summarized in Table I.

Platform	**Base Paper [2]**	**Our Work**
Cloud	Azure Cloud (32GB RAM)	Google Colab (12GB RAM)
Edge	Dell Laptop (i7-9750H, 16GB RAM)	HP Laptop (i5, 16GB RAM)
IoT Device	Raspberry Pi 4 (8GB RAM)	Raspberry Pi 4 (8GB RAM)
IoT End Device	STM32F412 (256KB RAM)	Arduino Nano 33 (256KB RAM)

TABLE I: Experimental Comparison Setup

VIII. Results and Observations

A. XAI for feature selection



Feature	Value
conn_state_RSTOS0	-0.00
conn_state_S3	-0.04
conn_state_RSTR	-0.02
service_irc	-0.03
service_dns	-0.06
conn_state_REJ	-0.04
orig_pkts	-0.00
service_http	-0.02
proto_udp	-0.20
conn_state_SF	-0.11
service_ssh	-0.05
duration	-0.01

Fig. 6: LIME explanations

The figure 6 shows LIME Explanations. The features shown in blue colour influences Benign class and features shown in orange influences Malign class.

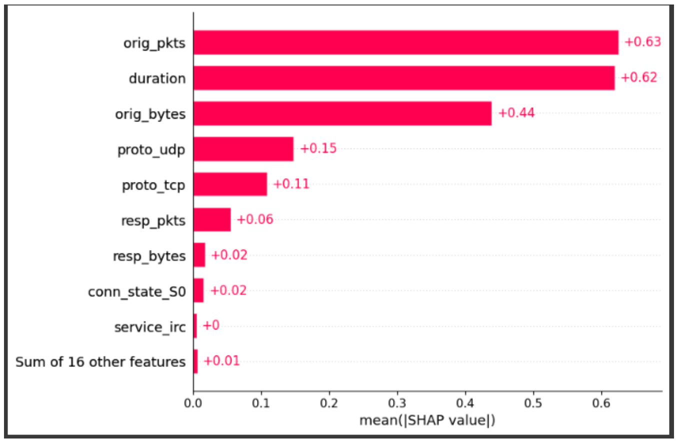


Fig. 7: SHAP explanations

The figure 7 shows the SHAP explanations. Using SHAP, we have identified orig_pkts, duration, orig_bytes, proto_udp, proto_tcp as the most important features. Figure 7 and figure 8 shows the effect of feature selection over accuracy and size respectively.

B. Feature Selection results

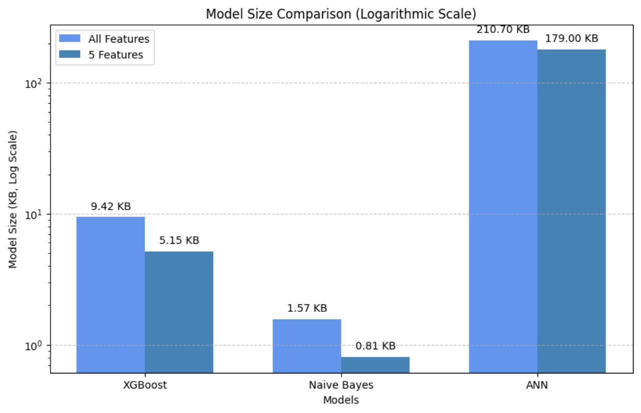


Fig. 8: Effect of feature selection over accuracy

By performing feature selection, there is 0.5%,0.8% and 1.2% accuracy loss respectively for XGB, Naive Bayes and Artificial Neural Network (ANN). On the other hand, XGB's size has been reduced to 5.15KB from 9.42KB whereas NB's size has been reduced from 0.81kB from 1.57KB. ANN's size has been reduced from 210KB to 179KB. ANN showed the least reduction in model size along with highest accuracy loss among all models.

C. Quantization Results

By performing quantization as mentioned in figure 10, we have observed 36% , 32%, 37% reduction in model size (after feature selection) with 0.8%,1.1% and 0.3% accuracy loss for

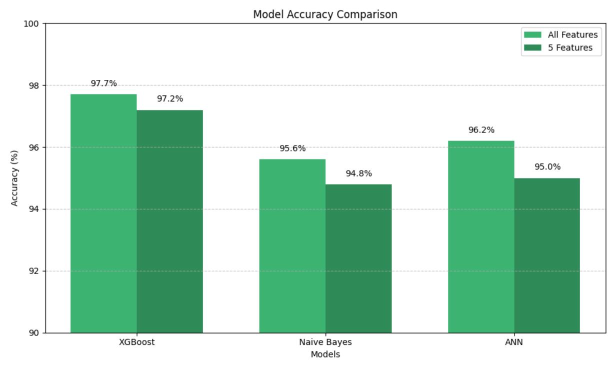


Fig. 9: Effect of feature selection over size

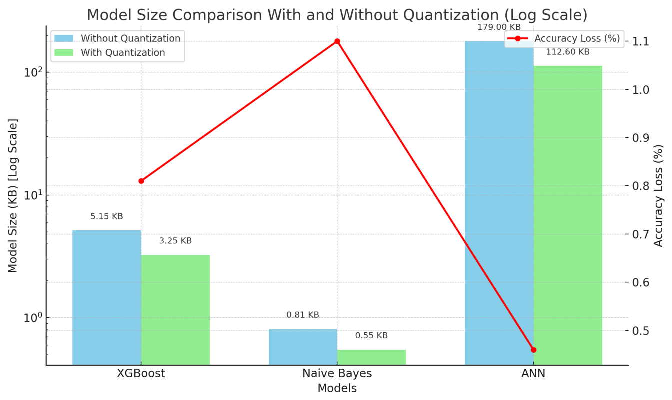


Fig. 10: Effect of Quantization along with accuracy loss

XGB,NB and ANN respectively. This is not desired in a real-life scenario where model sizes may reach upto MBs.

D. Knowledge Distillation Results

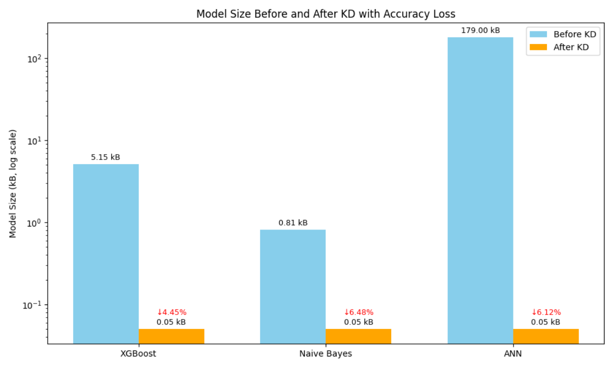
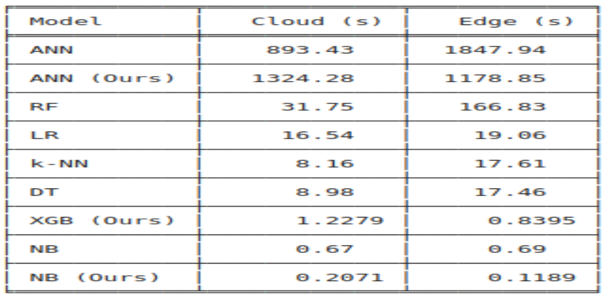


Fig. 11: Effect of Knowledge Distillation along with accuracy loss

For each of the teacher models (XGB,NB,ANN), we have performed knowledge distillation to a simpler student (Logistic Regression). Hence we can observe all the student models sizes are equal to 0.05KB as mentioned in Figure 11. But training is dependent on teacher also, hence we can observe differnet accuracy losses for different cases. XGB teacher provides the minimal accuracy loss to its student while NB provides maximal accuracy loss to its student.

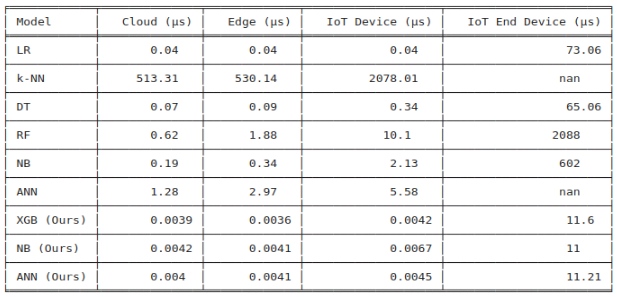
E. Comparison with State of the Art Techniques



Model	Cloud (s)	Edge (s)
ANN	893.43	1847.94
ANN (Ours)	1324.28	1178.85
RF	31.75	166.83
LR	16.54	19.06
k-NN	8.16	17.61
DT	8.98	17.46
XGB (Ours)	1.2279	0.8395
NB	0.67	0.69
NB (Ours)	0.2071	0.1189

Fig. 12: Comparison of training times with Tekin et al.

1) Training Time: Training times were only compared on Cloud and Edge. Our method achieves better training time compared to base paper [2].



Model	Cloud (µs)	Edge (µs)	IoT Device (µs)	IoT End Device (µs)
LR	0.04	0.04	0.04	73.06
k-NN	513.31	530.14	2078.01	nan
DT	0.07	0.09	0.34	65.06
RF	0.62	1.88	10.1	2088
NB	0.19	0.34	2.13	602
ANN	1.28	2.97	5.58	nan
XGB (Ours)	0.0039	0.0036	0.0042	11.6
NB (Ours)	0.0042	0.0041	0.0067	11
ANN (Ours)	0.004	0.0041	0.0045	11.21

Fig. 13: Comparison of inference times with Tekin et al.

2) Inference Time: We have used Edge Impulse to simulate the working of models on IoT device and IoT end device. Inference times for different models are mentioned in table 14. Our model surpasses [2] in terms of inference times. Our best inference time on IoT end device is 11 micro seconds compared to 76 micro seconds in [2]

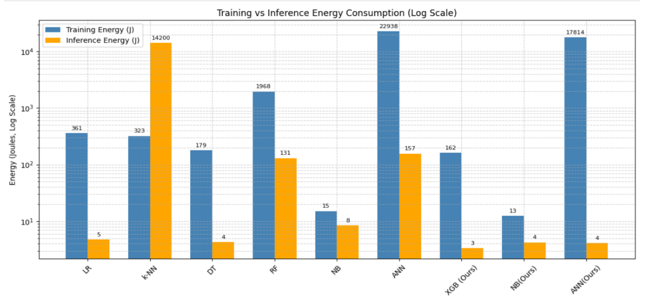


Fig. 14: Comparison of energy consumption with Tekin et al.

3) Energy consumption: To measure energy consumption, we used Intel's RAPL(Running Average Power Limit). We have made 2 check points. One before using the model for training/inference and then after using the model for training/inference. The difference in checkpoints energy consumption revealed actual energy consumption for the process. We made sure no background apps were running during training/inference to make proper measurements.

IX. Conclusion

In this work, we proposed a compact, explainable intrusion detection framework tailored for resource-constrained IoT environments by integrating Explainable Artificial Intelligence (XAI) with Tiny Machine Learning (TinyML). Through the use of LIME and SHAP, we performed informed feature selection, reducing the original 26 input features down to just 5 high-impact features. This significantly enhanced both model interpretability and efficiency.

To ensure real-time deployability on edge devices, we incorporated quantization and knowledge distillation. These methods led to a dramatic reduction in model complexity: our initial Artificial Neural Network (ANN) with 48,800 parameters was successfully distilled into a lightweight Logistic Regression model with just 6 parameters—a 99.98% reduction in size. Quantization further improved compactness by up to 1.4×, resulting in faster inference, lower memory usage, and reduced power consumption.

Despite this aggressive compression, our models maintained high accuracy and consistent performance on edge platforms such as Raspberry Pi 4 and Arduino Nano 33, simulated via the Edge Impulse platform.

These results demonstrate that XAI-driven TinyML offers a viable path toward secure, efficient, and interpretable intrusion detection in IoT ecosystems. Future work will explore dynamic adaptation to emerging threats through incremental learning and further innovations in ultra-compact model design for even more constrained environments.

X. Future Scope

The journey toward building the most efficient and reliable intrusion detection system for IoT devices doesn't stop here. While our current framework has demonstrated the power of XAI-driven feature selection and model compression through TinyML techniques, it opens the door to even more exciting advancements. As IoT ecosystems continue to grow in complexity and scale, so too must our models evolve. The following directions outline promising avenues for pushing the boundaries of adaptability, compactness, and intelligence in real-world deployments.

A. Enhanced Accuracy-Compactness Trade-off

Investigate more advanced optimization techniques to further reduce model size and complexity while minimizing the impact on accuracy. Techniques like pruning, low-rank approximation, and quantization-aware training could be explored for deeper model compression.

B. Incremental and Continual Learning

Develop incremental learning mechanisms that allow the model to adapt in real time to new types of attacks without requiring complete retraining. This is crucial for maintaining relevance in dynamic IoT threat landscapes.

C. Building a Real-Time Classification Engine:

The current setup might be limited to offline analysis. To unlock the true potential of these models, a robust backend system can be implemented. This backend can handle real-time data acquisition, seamlessly feed data to the chosen model, and deliver classification results instantaneously. This paves the way for deploying the model in real-world applications, enabling on-the-fly decision making.

D. Making the Models Portable:

Deploying these models on edge devices, which are often resource-constrained, might be hindered by their current size. Here, model compression techniques come into play. By strategically pruning unnecessary connections or quantizing weights, the model size can be significantly reduced without sacrificing much accuracy. This trade-off between size and accuracy allows for deploying these powerful models on devices with limited processing power, expanding their reach and impact.

By exploring these avenues, we can unlock a deeper understanding of the data, improve classification accuracy, and ultimately leverage these models to solve real-world problems in a more efficient and impactful way.

References

[1] M. A. A. d. Cruz, L. R. Abbade, P. Lorenz, S. B. Mafra, and J. J. P. C. Rodrigues, "Detecting compromised iot devices through xgboost," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 12, pp. 15 392–15 399, 2023.

[2] N. Tekin, A. Acara, A. Aris, A. S. Uluagac, and V. C. Gungor, "Energy consumption of on-device machine learning models for iot intrusion detection," *Cyber-Physical Systems Security Lab, Department of Electrical and Computer Engineering, Florida International University*, 2025, department of Software Engineering, Erciyes University, 38280, Kayseri, Turkey, Department of Computer Engineering, Abdullah Gul University, 38080 Kayseri, Turkey.