Accurate IoT intrusion detection system via application of AI models

Oikonomopoulos Ioannis

Thesis submitted in fulfillment of the requirements for the

Diploma of Electrical and Computer Engineering

Technical University of Crete
School of Electrical and Computer Engineering
University Campus, Akrotiri, Chania, GR-73100, Greece

Thesis Supervisor: Professor *Sotiris Ioannidis*

TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Accurate IoT intrusion detection system via application of AI models

Thesis submitted by
Oikonomopoulos Ioannis
in fulfillment of the requirements for the
Diploma of Electrical and Computer Engineering

THESIS APPROVAL

Author: Oikonomopoulos Ioannis

Committee approvals:

Sotiris Ioannidis
Professor, Thesis Supervisor, Committee Member

Georgios Chalkiadakis
Professor, Committee Member

Nikolaos Giatrakos
Associate Professor, Committee Member

Chania, February 2025

Abstract

The Internet of Things (IoT) keeps growing bigger every year. Based on IoT Analytics data [1], connected devices worldwide are expected to reach over 29 billion by 2027, spreading across transportation, healthcare, warehousing, security, and data monitoring. These devices have become so important in our daily lives and business operations that we can hardly imagine functioning without them. Yet, we've seen how device failures in the past have seriously damaged both data privacy and economic stability.

With AI becoming the star technology of the 2020s, it's revolutionizing how we approach Network Intrusion Detection Systems (NIDS). While we're getting better at training models to catch all sorts of attacks on servers and microservices, there's still a big question mark when it comes to embedded devices - what can we do when we're working with limited computational resources?

This thesis tackles the challenge of running effective NIDS on IoT devices without overwhelming their modest resources. Our approach? We're implementing classification and outlier detection algorithms in GoLang - a lightweight, low-level language that handles malicious traffic more efficiently than Python (the usual go-to for Machine Learning). While we started our experiments in Python, thanks to its user-friendly typing and rich Machine Learning libraries, we ultimately moved everything to GoLang for the final product.

We worked with a Mirai Botnet attack dataset, since it's the most common type of IoT attack out there. For our analysis, we used Light Gradient Boosting Machine (LightGBM - LGBM) for classification and One-Class Support Vector Machine (One Class SVM - OCSVM) for spotting anomalies. We compared how these methods stack up against Kitsune, a newer framework for detecting outliers. The final test? Running everything on a Raspberry Pi [2] to see how Python, GoLang, and Kitsune perform in a real embedded system setup.

Περίληψη

Το Ίντερνετ των Πραγμάτων (IoT) μεγαλώνει κάθε χρόνο. Σύμφωνα με τα δεδομένα της IoT Analytics[1], ο αριθμός των συνδεδεμένων συσκευών στο διαδίκτυο παγκοσμίως αναμένεται να ξεπεράσει τα 29 δισεκατομμύρια μέχρι το 2027, με εφαρμογές στις μεταφορές, την υγειονομική περίθαλψη, την αποθήκευση, την ασφάλεια και την παρακολούθηση δεδομένων. Η χρήση του IoT προσφέρει μεγάλα πλεονεκτήματα για τις επιχειρήσεις και την καθημερινή ζωή των ανθρώπων, ώστε αν ξαφνικά οι συσκευές αυτές έμεναν ανενεργές, η επίδραση θα ήταν αισθητή. Ωστόσο, δυσλειτουργίες αυτών των συσκευών στο παρελθόν έχουν προκαλέσει σοβαρά πλήγματα στην ιδιωτικότητα των δεδομένων και την οικονομία.

Με την Τεχνητή Νοημοσύνη να αναδεικνύεται ως η κυρίαρχη τεχνολογία της δεκαετίας του 2020, έχει μετασχηματίσει τον τομέα των συστημάτων ανίχνευσης εισβολών. Τα μοντέλα εκπαιδεύονται για να εντοπίζουν κάθε τύπο επίθεσης σε διακομιστές, μικροϋπηρεσίες κ.λπ. Αλλά τι γίνεται με τις ενσωματωμένες συσκευές που στερούνται υπολογιστικών πόρων·

Η παρούσα διπλωματική εργασία παρέχει μια ανάλυση σχετικά με την αποτελεσματική αντιμετώπιση των δικτυακών επιθέσεων στο IoT , διατηρώντας ελαφρούς υπολογιστικούς πόρους. Η κεντρική ιδέα είναι η υλοποίηση αλγορίθμων ταξινόμησης και ανίχνευσης ακραίων τιμών σε χαμηλού επιπέδου GoLang, το οποίο καθιστά την ανίχνευση και την αντιμετώπιση της κακόβουλης κίνησης πιο αποδοτική από άποψη χρόνου εκτέλεσης και υπολογιστικών πόρων σε σύγκριση με την Python , η οποία αποτελεί τη δημοφιλή γλώσσα προγραμματισμού για τη Μηχανική Μάθηση. Τα πειράματα εκτελέστηκαν σε Python λόγω της ευκολίας στη χρήση και των δημοφιλών βιβλιοθηκών μηχανικής μάθησης που παρέχει, αλλά το τελικό προϊόν χαρτογραφήθηκε σε GoLang.

Το σύνολο δεδομένων που χρησιμοποιείται είναι από επίθεση Mirai Botnet , η πιο συνηθισμένη μορφή επίθεσης στο IoT. Οι αλγόριθμοι που χρησιμοποιήθηκαν είναι το LightGBM ως μέθοδος ταξινόμησης και το OneClassSVM ως μέθοδος ανίχνευσης ανωμαλιών. Σε αυτήν την εργασία, τα αποτελέσματα αυτών των δύο αλγορίθμων συγκρίνονται με το καινοτόμο πλαίσιο ανίχνευσης αποκλίσεων σε δικτυακές επιθέσεις Kitsune . Στη συνέχεια, διεξήχθησαν πειράματα σε ένα Raspberry Pi [2] για τη μέτρηση της απόδοσης ζωντανής εκτέλεσης στα ενσωματωμένα συστήματα στις υλοποιήσεις σε Python και GoLang, αλλα και στην υλοποίηση του Kitsune .

Ευχαριστίες

Πρώτα απ' όλα, θα ήθελα να ευχαριστήσω τον επιβλέποντά μου, Καθηγητή Σωτήρη Ιωαννίδη, για την πολύτιμη καθοδήγησή του. Θέλω επίσης να εκφράσω την πιο βαθιά μου ευγνωμοσύνη στον σύμβουλό μου, Δρ. Αλέξανδρο Σεβτσοβ. Και οι δύο ήταν δίπλα μου καθ΄ όλη τη διάρκεια αυτού του περασμένου έτους, συμβάλλοντας στην εκπλήρωση αυτής της έρευνας, καθώς και στην προσωπική μου ανάπτυξη και γνώση. Επίσης θέλω να ευχαριστήσω τους καθηγητές οι οποίοι συμφώνησαν να επιτελέσουν την εξεταστική επιτροπή μου Γεώργιο Χαλκιαδάκη και Νικόλαο Γιατράκο. Θέλω να ευχαριστήσω τους φίλους μου οι οποίοι ήταν το στήριγμα μου αυτα τα χρόνια στα Χανιά. Τέλος θέλω να ευχαριστήσω τους γονείς μου και τα αδέρφια μου για την ανεξάντλητη υποστήριξη τους στο να εκπληρωθεί αυτος ο στόχος.

.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The vast field of AI is growing day by day with it becoming part of our everyday lives. This increase in need of new technologies has grown linearly along with hardware capabilities. Complex problems require complex solutions and complex solutions require a model which exploits a ton of computational force for training and a little less for delivering these solutions. Many works as depicted in the following chapter use Deep Learning in order to distinguish malicious traffic in a network, and it is for a fact that neural networks are able to capture more complex relations within the data and provide more accurate results. In this work we focus on implementing and fine-tuning ML algorithms in order to be able to distinguish the intent of network flows as extracted from the Kitsune Feature extractor[3] for detecting Mirai Botnet attack. The ML algorithms that are deployed are more computational efficient than any Neural Network Based Architecture, and more so the predictor of the final model was parsed to Golang, demonstrating even better performance time, cpu and memory wise than its Python version. As an anomaly detector this tool must be up and working every second of the IoT device lifetime, so it can't be consuming even more than half of its cpu or memory capacity.

In this thesis we compare score-wise three different NIDS approaches (Neural Network, Machine learning and Golang implementation of Machine Learning). We selected comparison of multiple implementations towards the bigger picture of the current state of state-of-the-art solutions in terms of the limited resources of the IoT device. For this purpose we selected a well known Neural Network NIDS implementation of Kitsune model [3]. We selected a particular model due to the exceptional performance over the Mirai attack dataset and very impressive performance over the Raspberry Pi device. Furthermore, we designed the comparison with traditional Python-based Machine Learning models, where more simplistic methods could improve the complexity and execution time on top of IoT devices.

Comparison of the both methods with the GoLang-based implementation provide real-case scenario application performance comparison over existing NIDS methods and their impact over the device power consumption with respect to the accurate attack detection.

According to the designed experiments and their results, we can conclude that go-based implementations of the Machine Learning classification models are capable of providing excellent NIDS detection performance while minimizing the computational load of the IoT device. Furthermore, in the comparison between Kitsune Neural Network python implementation and our GoLang-based machine learning model, our method is capable of processing network traffic in significantly shorter time making this method able to process larger volumes of traffic.

1.1 Contributions

To summarize, the main contributions of this thesis are:

- A novel approach to IoT security through the development of lightweight, efficient Network Intrusion Detection mechanisms specifically designed for resource-constrained devices. This contribution addresses the critical gap between sophisticated security needs and limited device capabilities.
- A practical demonstration of implementing Machine Learning algorithms in GoLang, moving beyond traditional Python implementations. This work provides valuable insight into the challenges and benefits of developing ML solutions in lower-level programming languages.
- Empirical evidence that simpler models can outperform complex ones in Network Intrusion Detection, particularly for resource-constrained devices. Our findings challenge the common assumption that more complex models necessarily lead to better results, especially in IoT security applications.

1.2 Outline

The layout of this thesis is as follows:

- Chapter 2 explores the background behind network intrusion for IoT.
- Chapter 3 refers related work and previous studies upon the domain of IDS in IoT environments
- Chapter 4 provides a methodological overview, summarizing and describing all the steps taken to conduct the analysis this thesis provides.

- Chapter 5 describes the implementation in depth and how each tool was utilized, while providing the results of the experiments and their evaluation.
- Chapter 6 concludes the thesis, offering future work directions and suggestions.

Chapter 2

Background

2.1 The Rise of IoT Devices

The Internet of Things (IoT) is one of the most crucial technological changes in recent years. It refers to a huge network of devices that are all connected to each other and can share and act on data. These devices range from simple gadgets, like smart thermostats and fitness trackers, to larger systems like sensors in factories or traffic management systems in cities.

What makes IoT so interesting is that these devices are small, energy-efficient, and easy to integrate into everyday life. For example, in healthcare, devices like smartwatches can monitor a person's health in real-time, helping doctors catch problems early. In agriculture, IoT helps manage water usage more efficiently. But as more devices get connected, new challenges arise, especially in terms of security and privacy. More devices mean more points for attackers to target, which makes securing them harder.

2.2 Security Challenges in IoT Networks

As IoT devices become more popular, they create more opportunities for cyber-criminals to attack. Unlike regular computers, IoT systems have many different devices that are all connected, and many of them have weak security. This makes them easy targets for attacks, especially Distributed Denial-of-Service (DDoS) attacks.

A DDoS attack happens when a large number of devices send too much data to a target system, causing it to crash and stop working. With IoT devices, these attacks often use botnets-groups of devices that have been taken over by hackers. Many IoT devices are vulnerable because they don't have strong passwords or security measures. Once these devices are taken over, they can be used to launch attacks, which could hurt not just individual users, but also essential services like hospitals or electricity grids.

2.3 Recent Trends in DDoS Attacks

Recent reports from cybersecurity companies like Netscout [4] and Getastra [5] show that DDoS attacks are happening more often and are getting stronger. In 2023, the number of DDoS attacks rose sharply, with the daily average growing from 144 at the start of the year to 611 by the middle of the year—an increase of over 350%. This shows that hackers are getting better at carrying out these attacks, and IoT devices are often the ones being used.

These attacks are also becoming more complex. Hackers are using smarter methods, like combining different types of attacks or using artificial intelligence to avoid being stopped by traditional defense systems. This means that better security is needed to fight off these increasingly advanced threats.

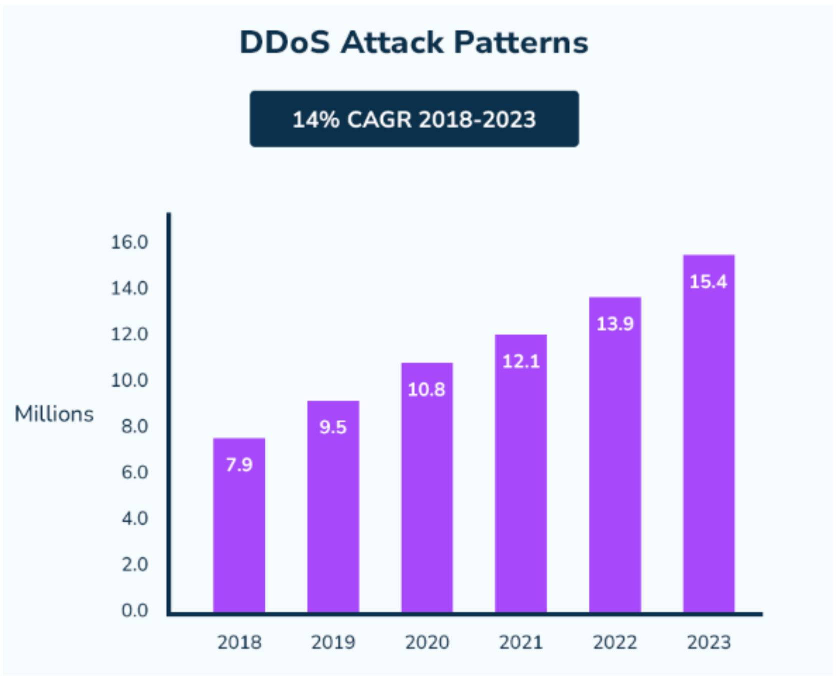


Figure 2.1: DDoS Attack Patterns in Millions, annually from 2018 to 2023.

2.4 The Three Phases of a DDoS Attack

Preparation

In this phase, the attacker acquires and configures the devices (often computers infected with malware) that they will use to carry out the attack. These devices are called "zombies" or "bots."

Trigger

The hacker sends a command to the "zombie" devices to make them start sending requests to the system or website they want to make inaccessible. This creates a huge amount of traffic, which overloads the system and makes it inaccessible to legitimate users.

Maintenance

Once the system has been overloaded, the DDoS attack continues to maintain pressure on the system, making it difficult or impossible to access online services. This can last for a long time and cause significant damage to companies that are victims of the attack.

2.5 Vulnerabilities in IoT Devices

IoT devices are especially vulnerable to cyberattacks for a few reasons:

- **Limited Processing Power:** Many IoT devices don't have enough power to run strong security features, making them easy targets for attackers.
- **Heterogeneous transmission technology:** Devices often use a variety of transmission technology. This can make it difficult to establish standard protection methods and protocols.
- **Weak Authentication:** Many devices come with default passwords that are easy to guess or don't ask users to change them.
- **Components of the device are vulnerable:** Vulnerable basic components affect millions of deployed smart devices.
- **Lack of Updates:** Manufacturers sometimes don't update devices regularly, which means any security issues aren't fixed.

These problems make it easier for attackers to take control of IoT devices. For example, in the Mirai botnet attack [6], hackers took control of over 600,000 devices to carry out large-scale DDoS attacks, leveraging the **Limited Processing Power** and the **Weak Authentication** of the devices. This attack shows just how much damage can be done if IoT devices aren't secured properly.

Another case is the Ransomware Attack on IoT Devices In 2017 [7]. It was reported that a number of smart home devices (like light bulbs, thermostats, and smart locks) were vulnerable due to **flaws in their basic components**. Attackers exploited these vulnerabilities to install ransomware, locking users out of their devices until a ransom was paid. The insecure components allowed attackers to gain control over millions of devices.

2.6 Case Study: The Mirai Botnet

The Mirai botnet is a famous example of how IoT devices can be used for cyberattacks. First discovered in 2016, Mirai used weak default passwords to hack into IoT devices, creating a huge botnet. This botnet was then used to launch powerful DDoS attacks. One of the biggest attacks was on the Dyn DNS provider, which caused major websites like Twitter, Spotify, and Netflix to go offline [8].

Mirai was successful because it was simple but effective. It searched the internet for devices with weak passwords and took control of them. Once infected, these devices could be used in attacks without the owners knowing. The Mirai botnet attack showed the world just how dangerous unsecured IoT devices could be.

2.7 Evolution of Botnets

Botnets have changed a lot over the years. Early botnets were basic, using Internet Relay Chat (IRC) to communicate. Today, however, botnets are much more advanced. Some of the ways they have evolved include:

- **Peer-to-Peer (P2P) Communication:** In modern botnets, devices talk to each other directly, making them harder to take down. The **ZeroAccess botnet** (2011)[9] was one of the first large botnets to adopt a P2P architecture. This decentralized structure made it more resilient to takedown efforts, allowing it to remain operational for extended periods despite attempts by security teams to shut it down.

- **Encrypted Channels:** Botnets now use encrypted communication, which helps them avoid detection. The **Necurs botnet** (2013–2019)[10] utilized encrypted communication to remain hidden from cybersecurity defenses. This allowed the botnet to persist for years, despite various law enforcement actions and security measures to disrupt its operations.

- **Modular Architectures:** Modern botnets can add new features as needed, which makes them more adaptable. The **Emotet botnet** (2014)[11] began as a banking Trojan but evolved into a highly modular system capable of distributing ransomware, email spam, and other types of malware. Its modularity allowed it to adapt quickly to new threats and targets.

These changes have made botnets harder to detect and stop, which is a big challenge for cybersecurity professionals.

2.8 Botnet Communication Architectures

There are two main types of botnets based on how they communicate:

- **Centralized Botnets:** This model is simple but vulnerable. These botnets have a single control server that tells all the infected devices what to do. If this server is taken down, the botnet can be stopped. The **Storm botnet** (2007)[12] used a centralized C&C model. Once the C&C server was discovered and shut down, the botnet's operations were effectively stopped.

- **Decentralized (P2P) Botnets:** Decentralized botnets are harder to detect because their traffic is spread across multiple points, often using encrypted channels and dynamic IP addresses to hide their true nature[13]. As mentioned before,the **ZeroAccess botnet** (2011)[9] used a decentralized, peer-to-peer architecture. This made it difficult for law enforcement and security teams to completely disable the botnet, as there was no central server to target.

Understanding how these botnets work is important because it helps us find ways to stop them before they can cause harm. Decentralized botnets are particularly challenging to deal with because of their resilience and stealth. Meanwhile, centralized botnets are still a threat, but they are somewhat easier to neutralize since they have a single control point. Effective detection and mitigation strategies need to address both types of communication architectures, employing techniques such as traffic analysis, anomaly detection, and legal intervention to dismantle the networks.

2.9 Emerging Technologies in IoT Security

To deal with the growing problems in IoT security, new technologies are being developed. Some of these include:

- **Machine Learning and AI:** These technologies can help detect threats more accurately and quickly by learning patterns in data.
- **Blockchain:** This technology offers secure and transparent ways for IoT devices to communicate, making it harder for attackers to tamper with the system.
- **Edge Computing:** This approach processes data closer to where it is generated, reducing delays and improving security by keeping data within the local network.
- **Quantum Cryptography:** Though still new, quantum cryptography could offer super-secure ways to encrypt data, making it almost impossible for hackers to break.

These technologies are part of the future of IoT security and may provide better protection as the number of connected devices keeps growing.

2.10 Detection and Mitigation Strategies

Traditional detection systems, such as signature-based methods, struggle to detect modern botnets due to their complexity. Newer approaches like machine learning and blockchain offer better accuracy, but they come with challenges. Machine learning can analyze traffic patterns and detect botnet activity, but it requires significant computational resources, which may not be suitable for IoT devices with limited processing power. Similarly, while blockchain provides secure communication, it is resource-intensive, which could be problematic for IoT environments. Given the constraints of IoT devices, deploying these advanced methods requires careful optimization. A hybrid approach that combines machine learning with other techniques may offer a more efficient solution for detecting botnets without overburdening IoT devices[14].

2.11 Future Challenges and Directions

As more IoT devices are connected, securing these networks will become even more important. The increasing complexity of botnets and the weaknesses in IoT devices make it clear that we need better solutions. Future research should focus on creating lightweight security systems that don't use up too many resources but can still detect threats in real-time. It will also require collaboration between device manufacturers, researchers, and governments to make sure IoT devices are secure enough to protect against evolving cyber threats [4].

Chapter 3

Related Work

The Field of intrusion detection in IoT has gathered attention due to the challenges that IoT devices pose. There have been numerous studies in the past [15, 16, 17, 18, 19] where intrusion detection systems for IoT are set up by deploying machine learning algorithms (see Table:3.2).

For example [20] where authors test various ML algorithms and different feature extraction methods for binary(attack or not) and multi class classification(type of attack), like a grid search which was evaluated by accuracy, precision (PR), recall (RC), F1-Score (F1S), MCC, and prediction Time and end up that the algorithm Decision Tree with Pearson correlation and Fisher score as feature extractor gave the best performance with an accuracy of 99.26% and runtime of 0.4s. While this approach demonstrates excellent accuracy, it relies on handcrafted feature selection, which may not generalize well across different datasets or evolving cyber threats.

In [21] authors suggest a different approach to dealing with network intrusions than classic ML algorithms. By deploying a neural network(Multi-layered perceptron) they are able to achieve an almost perfect accuracy score of 99.4% while referencing the difference an ids can pose if it is Host-Based or Network-Based. However, the computational cost of training and deploying deep learning models remains a significant challenge, particularly for resource-constrained IoT devices.

In [22] authors proposed an interesting work where the IDS appears to be a hybrid system consisting of a 2-D CNN(Convolutional Neural Network) and a Bi-LSTM(Long short-term memory) model. The first leads to discretization of parameters and recognizes the parameter which is useful & relevant, and the latter as a RNN (Recurrent Neural Network) aims to better understand patterns in the data especially if they are time-series like most system intrusion datasets are. This method results in good results in both datasets where it is tested, and is an interesting suggestion for the future of Intrusion detection mechanisms. However, it still incurs a high computational overhead, making real-time deployment challenging for IoT systems.

Another paper [23] focuses on the scalability and accuracy challenges of detecting anomalies in the large-scale data streams typical in IoT systems. This paper outlines how ML techniques are applied to process continuous data streams, detect outliers, and improve anomaly detection efficiency. The discussion covers various learning modes (supervised, unsupervised) and the trade-off between detection accuracy and time complexity. While this approach improves real-time anomaly detection, the paper highlights significant resource constraints when deploying these models on IoT networks with low-power devices.

For a broader understanding, [24] provides a comprehensive survey of ML- and DL-based network intrusion detection mechanisms, covering most of the existing techniques and offering a detailed analysis of their advantages, challenges, and future directions. This work serves as a valuable reference for researchers entering the field.

A particularly innovative approach is presented in [25], where the author, Milind Tambe, introduces Game Theory as a security mechanism. This method models attacker-defender scenarios to optimize the allocation of limited security resources, mirroring the challenge of balancing maximum intrusion detection performance with minimal computational cost—a problem directly relevant to this thesis.

In comparison with the existing related work we propose usage of Go-language which is designed to be more efficient, especially in the computationally limited devices such as IoT. Furthermore, we are taking advantage of the state-of-the-art Machine Learning implementations such as Light Gradient Boosting classifier in order to achieve maximum performance of classification tasks without unnecessarily exploding the computational cost of the model. The combination of the described methods allow us to achieve development of highly accurate models with respect to the limited hardware and power device capabilities.

Aspect	**Existing Approaches (DL-based, Classic ML, Hybrid IDS)**	**Our Approach (GoLang + Optimized ML Models)**
Accuracy	High (CNN, Bi-LSTM, MLP models achieve 99%+)	Comparable (LightGBM and XGBoost maintain high accuracy)
Computational Efficiency	High resource usage due to deep learning	Optimized for low-power IoT devices
Scalability	Limited, deep models require significant computation	Highly scalable, lightweight models
Implementation Language	Mostly Python-based (TensorFlow, Scikit-learn)	GoLang implementation for efficiency
Real-Time Performance	Slower inference times in DL-based approaches	Faster execution, lower prediction latency
Feature Engineering	CNN-based or manually selected features	Automated feature selection with Boosting models
Deployment Feasibility	Difficult for IoT due to hardware constraints	Optimized for real-world IoT deployment

Table 3.1: Comparison of Existing Approaches vs. Our Approach

Table 3.2: Comparative analysis of recent work on IoT security vulnerabilities.

References	**Contributions**	**Challenges Addressed**
Mosenia and Jha [15]	A survey on IoT security vulnerabilities and security controls.	Identification of IoT edge layer vulnerabilities.
Neshenko et al. [26]	A comprehensive survey on IoT vulnerabilities and experiment on large-scale exploitations.	Identification of vulnerabilities, attack vectors, impacts, and controls.
Rytel et al. [19]	Survey on publicly available sources of known IoT vulnerabilities.	Identifying information sources of IoT vulnerabilities.
AlLifah et al. [27]	A systematic literature survey of smart home device security vulnerabilities.	Identifying smart home device security vulnerabilities.

Continued on next page...

References	Contributions	Challenges Addressed
Yu et al. [28]	Surveyed research on methods for IoT device vulnerability analysis.	Methods for IoT device vulnerability analysis.
Xie et al. [29]	Survey on IoT firmware security vulnerabilities analysis.	Detecting vulnerabilities in IoT firmware.
Srivastava et al. [30]	Covers a multi-tiered survey covering the causes of IoT-related vulnerabilities across various layers of IoT infrastructure.	Identifying vulnerabilities across various layers of IoT infrastructure.
Anand et al. [31]	Survey covering multiple areas and tools for IoT vulnerability with a sustainability focus.	Identifying tools for monitoring and discovering IoT vulnerabilities.
Meneghello et al. [32]	A survey of security vulnerabilities in real IoT devices and IoT network protocols.	Identifying IoT devices and IoT network protocol vulnerabilities.
Costin et al. [33]	Survey of existing threats and vulnerabilities in IoT devices.	Identifying vulnerabilities in video surveillance, CCTV, and web camera systems.
Nadir et al. [34]	A systematic literature review on security and analysis techniques of IoT firmware.	Techniques of IoT firmware vulnerabilities.
Wright et al. [35]	Reviewed the literature on emulation and dynamic analysis useful for determining if firmware contains security vulnerabilities.	Identifying whether firmware contains security vulnerabilities.
Xie et al. [29]	A brief survey on methods for detecting IoT firmware vulnerabilities and classification.	Detecting IoT firmware vulnerabilities.
Frustaci et al. [36]	Analyzed critical security flaws in the communication and networking protocols used in these layers.	Identifying vulnerabilities in communication and networking protocols.
Qasem et al. [37]	A survey on the detection of software vulnerabilities in embedded devices and firmware.	Detection of vulnerabilities in embedded devices and firmware.

Continued on next page...

References	**Contributions**	**Challenges Addressed**
Feng et al. [38]	A survey focusing on the challenges in detecting vulnerabilities in IoT device firmware.	Detection of IoT device firmware vulnerabilities.
Yaqoob et al. [39]	Survey on security flaws in medical devices.	Identifying IoT medical device vulnerabilities.

Chapter 4

Methodology

The highlight of the previous section is that resource and time allocation is inversely proportional to scoring efficiency when coming to NIDS, there is a clear trade-off between the two. The motivation of this thesis is to ignore the trade-off and deploy a tool that is accurate while not draining on system resources. The methodology in order to achieve our motivation is focused around 3 axes.

1. Find ML models suitable for identifying a Network Attack, and create a pipeline to compare them with the Benchmark IDS Kitsune. In this way we mark that simpler architecture IDS can come close or even outperform complex Deep Learning works like Kitsune (see 5.2).

2. Employ advanced manual feature engineering in order to enhance model score even more (see 5.3).

3. Mimic real-world scenario and compare Kitsunes' performance against our models' performance on top of a Raspberry Pi device. This comparison concludes our methodology, as its results align with our initial objectives and performance standards. It demonstrates the effectiveness of our approach in achieving both accuracy and computational efficiency, reinforcing its suitability for real-world IoT deployment.(see 5.4).

To begin our study on anomaly detection we focused on selecting a model capable of detecting outliers in the Mirai Dataset. The idea at first was implementing an unsupervised machine learning algorithm which fits our purpose. Unsupervised Learning helps in one way, it does not demand to know what is an attack sample and what is a normal one. The first algorithm that came to mind was Isolation forest[40], whose only parameter concerning the dataset is the contamination parameter where the percentage of anomalous traffic in the dataset is estimated by the Engineer. After fine tuning the Isolation Forest model using Stratified Grid Search Cross Validation to find the parameters who produce the best F1 Score 4.3, although it didn't produce the desired results so we dropped it.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{4.1}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.2}$$

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.3}$$

$$TPR = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.4}$$

4.1 One Class SVM

Then we turned to another model. The unsupervised learning version of SVM (Support Vector Machine) One Class SVM or OCSVM[41]. This model trains only on benign data, learns the normal class, and then is able to distinguish it from anything else i.e recognize anomalies. The OCSVM aims to find a hyperplane (or nonlinear decision boundary in higher-dimensional space) that maximizes the margin between the origin (representing anomalies) and the normal data points. This hyperplane separates the majority of the data from the origin, defining the region where normal data points are expected to lie.

The intuition behind using the model was that when this tool made it to production it would be able to train for a period of time on a normal day, and then turn to prediction phase for the rest of the days it functions. One Class SVM like most ML models has a set of hyper parameters (Table: 4.1) which can be fine-tuned with the aim to maximise TPR score 4.4 this time. Fine Tuning an one class model can be tricky, because the training set consists only of the normal class(this is why the TPR fine tune and not F1), so the model is trained in order to perfectly identify this one class, not caring about the anomaly class, so there might be an overall F1 Score drop there.

Nu	sets an upper bound on the fraction of outliers and a lower bound on the fraction of support vectors
Gamma	small value leads to a smooth decision boundary and a large value allows for more complex boundaries.
Kernel	3 most common Kernels defining the Decision Boundary, Linear,Polynomial, and RBF.

Table 4.1: One Class SVM Hyper parameters

4.2 Light Gradient Boosting Machine

Next in line and in pursuit of a better score LightGBM (Light Gradient Boosting Machine) came into the picture[42]. LightGBM is a high-performance framework for gradient boosting, designed to be efficient and fast, especially with large datasets and high-dimensional data. Developed by Microsoft, it uses a unique leaf-wise tree growth strategy, which allows it to build deeper trees for more accurate predictions compared to traditional level-wise methods. LightGBM also employs histogram-based binning and Exclusive Feature Bundling (EFB) to reduce memory usage and training time, making it highly scalable. It is widely used for both classification and regression tasks, offering excellent performance with minimal tuning. After fine tuning its Hyper - parameters, LightGBM offers an outstanding performance (F1 evaluated)on the train-test split.

4.3 Model Comparison

After having experimented with the models and seen what they are capable of, the next step was comparing their performances against Kitsune[3]. In this step we tried having these 2 models (OneClassSVM and LightGBM) predict on the same testing set with Kitsune and compare their scores afterwards. Note that Kitsune is an unsupervised anomaly detection method and is being executed like we intended OneClassSVM to execute, train for a number of benign samples, then execute on all the rest predicting which are normal and which outliers. In this step, it is being made clear why the knowledge of both of classes in model training can be catalytic in the final predictor, as the classifier model LightGBM out predicted both outlier detection models (OCSVM and Kitsune Kitnet) even if the latter is build on way more complex architecture.

num_leaves	Controls the complexity of the model. A higher value allows the model to learn more intricate patterns but may lead to overfitting.
max_depth	Limits the depth of each tree. Helps prevent overfitting by controlling how deep the trees can grow.
learning_rate (lr)	Determines the step size at each iteration while optimizing. A smaller value makes training more stable but slower.
n_estimators	Defines the number of boosting iterations (trees) Higher values improve learning but may increase computation time.

Table 4.2: LightGBM Hyperparameters

Afterwards we conducted experiments on Raspberry Pi, to be able to simulate the production phase of our tool, and furthermore capture the time taken , memory and cpu consumed while predicting the same number of instances from the dataset. For measuring the predicting time built-in functions of Python and Golang were used accordingly. For the cpu and memory consumption a bash script was called inside each program which logged in a file the cpu and memory percentages of the system being used each second.

4.4 Data

Our Dataset [43] is a mirai botnet network attack dataset consisting of 115 features and 100K samples of which the first 71K are benign and the rest 29K malicious. It is time-series data captured online from a real-time mirai-Botnet Invasion.

In an attempt to better the score of OCSVM we tried applying Lasso Regression to the dataset. Lasso regression is a linear model that incorporates L1 regularisation, which adds a penalty proportional to the absolute value of the coefficients. This forces some of the coefficients to be exactly zero, effectively removing less important features from the model. This technique didn't have impact. Some features were removed but the score didn't get better or worse. So it was not applied.

Diving into further investigation and experimenting with the LightGBM classifier we counted the Cross-Validation F1 Score which gives much better insight into the classification effectiveness of the model, while it counts the score on 5 different(and shorter) train test splits. Here we detected a significant drop in score for the last split, where the transition from benign samples to malicious begins. So we applied some manual feature engineering to enhance it.

Chapter 5

Implementation & Evaluation

In this section we describe in details the implementation and evaluation steps of designed experiments, while referring also to some extra work which is the feature engineering part which gives insight into enhancing model's performance.

5.1 Model Fine Tuning

Our initial focus was on fine-tuning the machine learning models. As a first step, we needed to scale the data for use in ML classification models, particularly SVM, which is highly sensitive to feature value scales. These models heavily rely on distances between data points to find the optimal hyperplane that separates classes. When features have vastly different scales, one feature might dominate the distance calculations, potentially leading to an unbalanced or incorrect decision boundary. To address this, we utilised MinMaxScaler to normalize values between zero and one (0,1). For this part, we used a custom library that maps sklearn functionality to Go [44].

For the GoLang-based implementation of the SVM algorithm, we chose the LibSVM [45] library, which requires specific dataset formatting:

```
(Row n):  label F1:Vnx1 F2:Vnx2 F3:Vnx3 F4:Vnx4
```

where 'label' represents the target (y) value for the sample, 'F' stands for Feature, and 'V' for its corresponding value for row n. We implemented One Class SVM, where the model trains exclusively on benign traffic samples and must then distinguish between benign and malicious traffic. During fine-tuning, we needed to identify model hyperparameters that create more general decision boundaries.

For the Kitsune model, the framework utilizes Root Mean Square Error (RMSE),

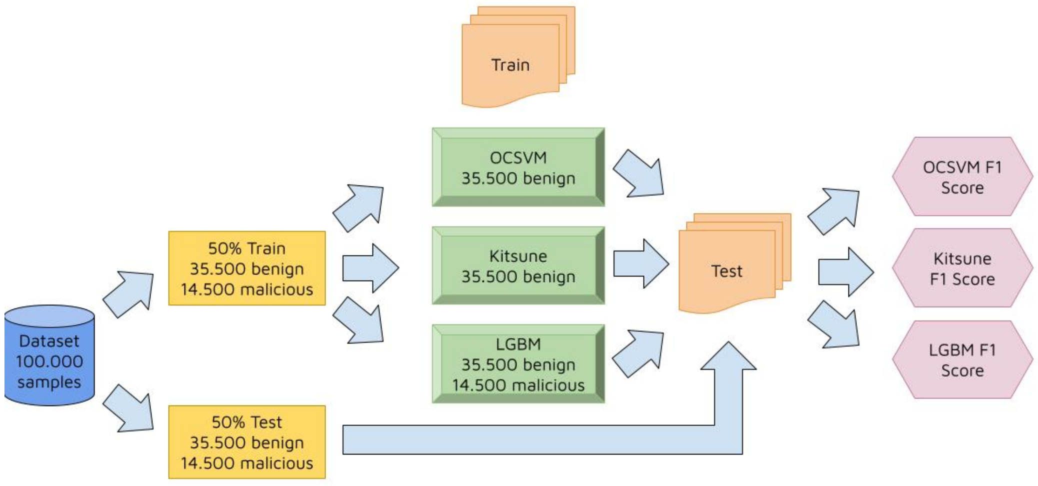


Figure 5.1: 3 model pipeline

computing it for each training sample. The framework's primary objective is to determine the threshold that optimally separates malicious and benign classes based on RMSE. Thus, fine-tuning focused on finding thresholds that accurately distinguish between malicious and benign samples.

Our third model, LightGBM, is based on a gradient boosting tree classification algorithm. This model also required hyperparameter fine-tuning to develop accurate separation between malicious and benign samples.

To ensure fair comparison between models, we maintained identical data splits during both hyperparameter fine-tuning and testing. We split our dataset in a stratified manner using a 50/50 ratio. We then employed stratified k-fold cross-validation for model fine-tuning, using identical splits across all models (excluding the benign samples of the training set of One Class SVM and Kitsune, as these models should only see benign samples). Through this process, we identified configurations that maximized performance for each model.

Metric / Model	Portion	OCSVM	Kitsune	LightGBM
TP	Train	0.99	1	1.0
TN	Train	-	-	1.0
FP	Train	-	-	0.0
FN	Train	0.01	0	0.0
F1	Test	0.9377	0.941	0.99

Table 5.1: 3-model Comparison Scores, Positive stands for the benign class

5.2 Model Comparison

During the previous step of the model hyper-parameter fine-tuning we manage to identify the configurations of the selected models that maximise the class separation abilities. In the next step, we were required to compare the performance of the selected models and their configurations on top of the same portion of unseen data.

Towards the model performance comparison we manage to train the models with their best identified configurations from the previous step. Further, the models were requested to predict the testing set data, which consist as we mentioned earlier from the 50% of the original dataset with respect to the class imbalance (Figure: 5.1). According to the experiment results, presented in the Table: 5.1 Kitsune managed to provide a high F1 Score over the testing set, showing that such one-class based implementation is capable of capturing the anomalies of the sample distribution. Furthermore, LightGBM manages to provide almost perfect separation between classes with an F1 Score equal to 0.99. The provided experiment showed that the LightGBM model is capable of accurately separating malicious and benign traffic even in comparison with more complex Neural Network models like Kitsune.

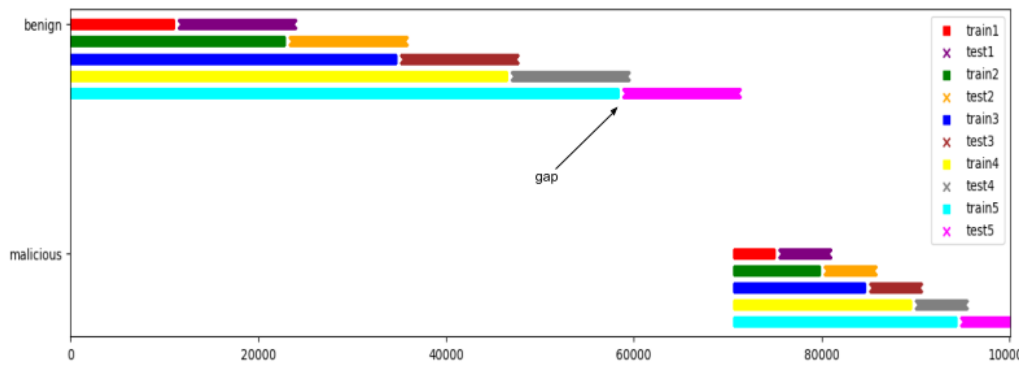


Figure 5.2: Custom Cross Validation, engineered specifically for this dataset with all malicious samples following the benign, to cross validate a classifier. With respect to the gap from rolling windows.

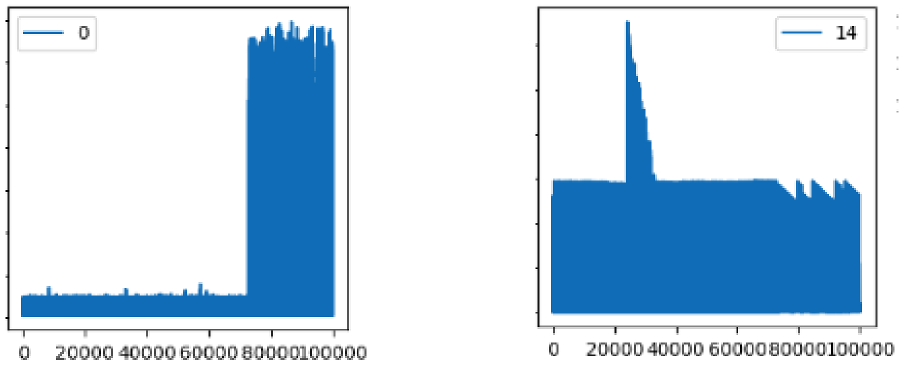
5.3 Feature Engineering

LightGBM showed us that is is the one model that competes with Kitsune, with the minor disadvantage that it is a classification model, not anomaly detection. Evaluating a model on a train-test split generally isn't an effective practice. Working with the LightGBM from now on, we employed cross-validation trying to find the mean cross validation F1 score among 5 splits.

Our dataset is time-series, but each sample can be treated as independent and does not contain information about previous samples. However, incorporating rolling features can provide additional value. The challenge we met was being able to cross-validate the classifier on both target variables, while respecting the order of the data. This motivated us to develop a custom time series cross-validation approach with respect to the max rolling window we incorporated in the features, in order to avoid data leakage(see Figure 5.2). For plotting, the seaborn library was utilized[46], specifically the kde density plot.

A first Cross Validation on the original dataset was performed with the gap parameter set to zero (Results in 5.2, Original Category).

At this point further investigation was conducted, while we tried plotting the features. With some manual inspection we separated those who can help the model predict better from those who did not (see Figure 5.3). Then ran once more the Cross Validation (Results in 5.2, After FE category).



(a) Feature 0, a useful feature

(b) Feature 14, a useless feature

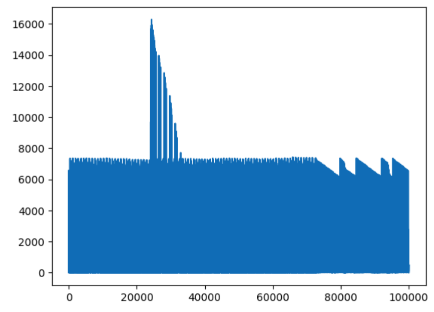
Figure 5.3: Two features of our dataset, one considered as useful and one as useless

Given the time-series nature of the dataset and its focus on transitional timing, it was essential to observe how each "useless" feature responds dynamically to value shifts. To capture these changes accurately, we applied a rolling transformation to examine the temporal evolution of each feature's mean or standard deviation, enabling a clearer view of how fluctuations impact predictive patterns. We also utilized the percentage change function on some features trying to represent the rate of change between time steps. This helps in detecting upward or downward trends over time. Applying this feature engineering in most features that did not make sense at first gave the model a clear view on what making the transition from benign to malicious samples look like as it is clearly seen in Figures 5.4 , 5.5.

We ended up with 76 features out of the 115 initial of the dataset, working with some that held information after their transformation and dropping the ones that did not. Score wise the LightGBM model on the new aggregated dataset scored an 0.96 mean weighted F1 Score among 5 splits (Table 5.2, Transformed Category) in contrast with the 0.51 over the original set of features. Demonstrating the necessity of feature and data engineering.

5.4 Resource Management

The final and most important section of this thesis presents a performance comparison among three different machine learning implementations: LightGBM in Python, LightGBM in Go, and the Kitsune framework, each tested on an embed-


(a) Feature 14, before transcformation

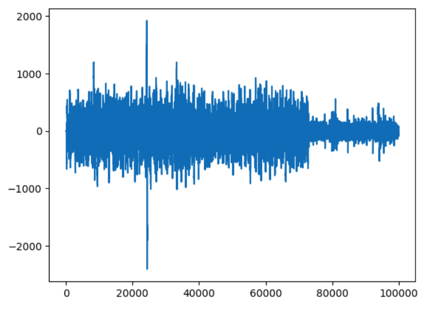
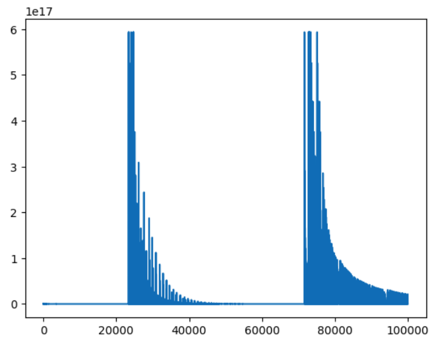

(b) Feature 14, after transformation

Figure 5.4: Feature of the dataset before and after applying the rolling transformation on standard deviation.


(a) Feature 79, before transformation

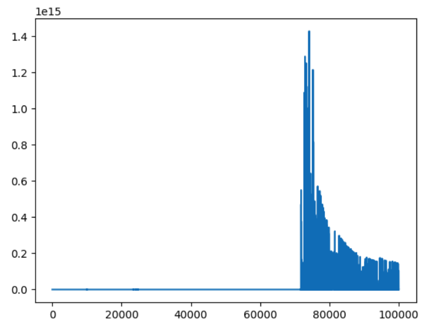

(b) Feature 79, after transformation

Figure 5.5: Feature of the dataset before and after applying percentage change.

Category	1st	2nd	3rd	4th	5th	Mean
Original	0.51	0.52	0.55	0.47	0.41	0.51
After FE	0.85	0.97	0.98	0.96	0.96	0.95
Transformed	0.88	0.97	0.99	0.97	0.97	0.96

Table 5.2: LGBM cross validation (5 folds) performance measured in F1 weighted score on original and the transformed set of features.

ded device, specifically a Raspberry Pi. This analysis aims to reveal the strengths and weaknesses of each approach, with a focus on execution speed, resource effi-

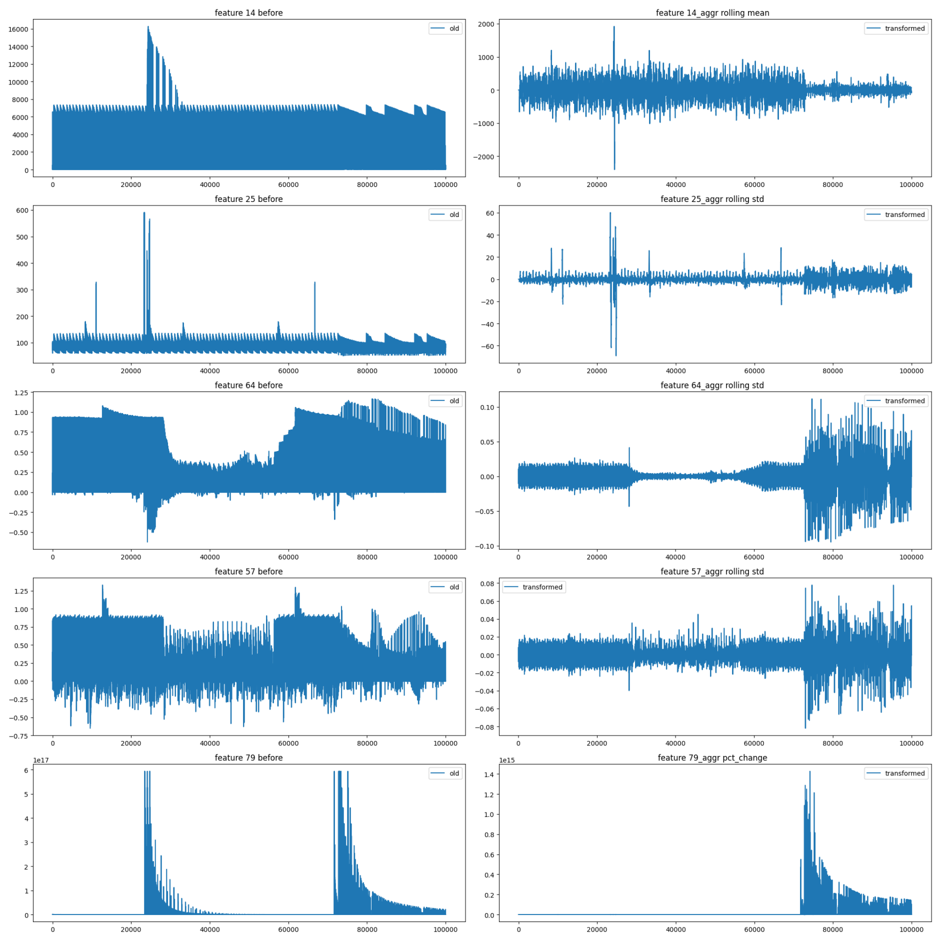


Figure 5.6: Transformed features using rolling average on standard deviation, mean and percentage change function. It is clearly seen that applying these transformations utilizing the past feature values give clear insight on the transition between benign and malicious samples.

ciency, and overall suitability for embedded applications.

5.4.1 Parsing LightGBM model in Golang

The Light Gradient Boosting Model (LGBM) was initially trained in Python since the go implementation is actually a wrapper that utilize the model only for inference. To deploy it within a Go environment, the trained model was converted to a compatible format using Python's Booster wrapper class and serialized with the json library's dump method. This approach enabled the seamless mapping of model parameters and structure from Python to Go, ensuring consistency and compatibility across programming environments. Library used for this purpose is a custom go library called leaves[47] which implements prediction code for Gradient Boosting Regression Trees models.

5.4.2 Experiments

We predicted all 100.000 samples of the dataset for all the 3 methods, measured completion time and logged memory and cpu percentages in this period of time. For measuring time in python we used the built in library time ,similarly with kitsune as it is developed with python, while in goLang the time built in library was utilised, in all the scripts we measured before and after time of execution. The resulting total time was their difference.

For memory and CPU usage measurements, we needed an outside monitor measuring the system's resources while the predict function was executed, this is why we used bash script once more. Calling the script right before the predict function and killing the process after. Taking advantage of linux commands top for cpu and free for memory we were able to log the total device cpu and memory usage percentages into a file for the entire duration of the prediction phase.

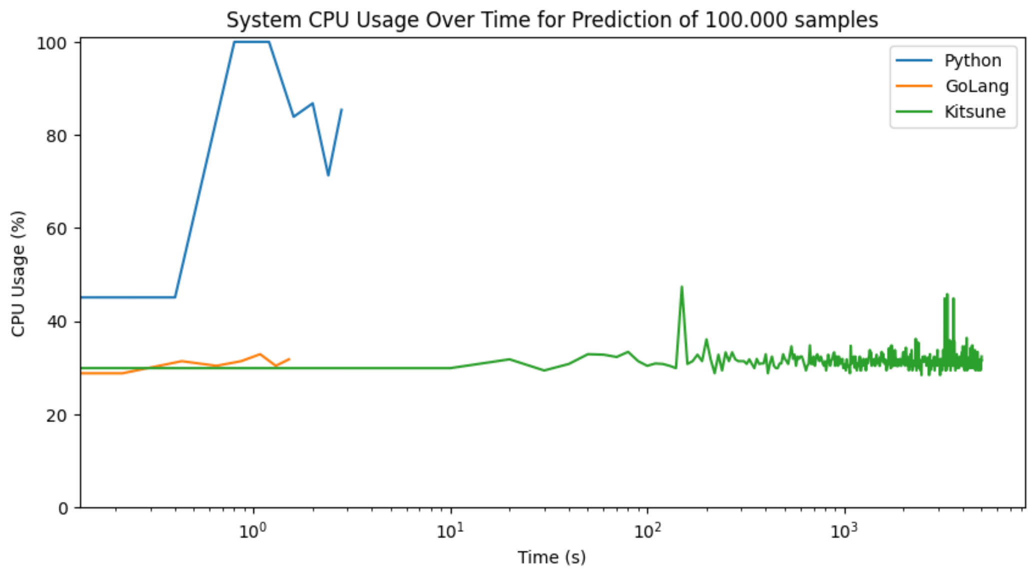


Figure 5.7: System CPU usage over time for prediction of 100.000 samples.

Evaluating the logging of cpu and memory on each method's period of time we created the Figures: 5.7 and 5.8 using Python plotting library seaborn. These

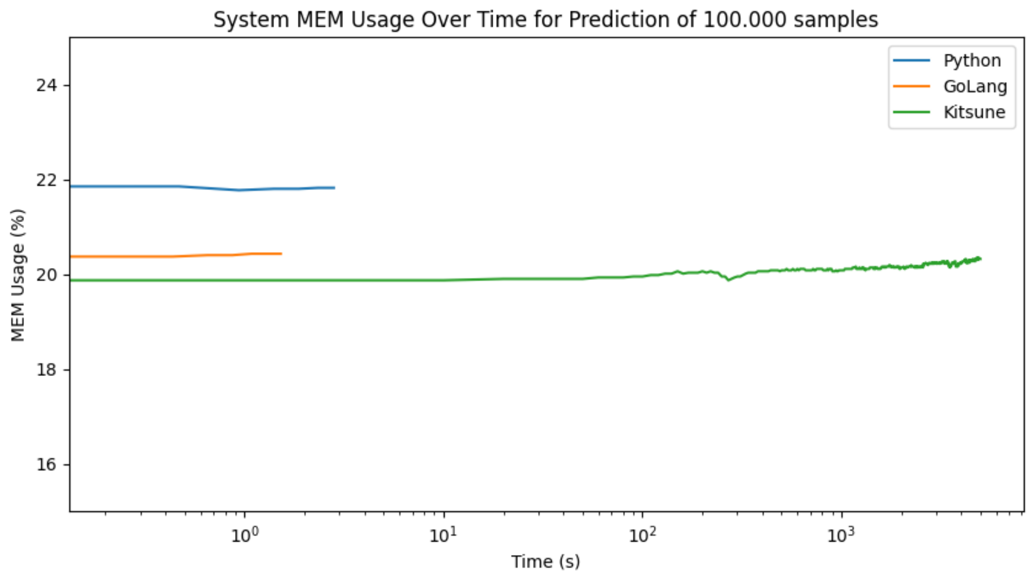


Figure 5.8: System memory usage over time for prediction of 100.000 samples.

experiments demonstrated the efficiency that a low level implementation can provide. GoLang's LightGBM implementation averages 30.8% of total cpu usage and 20.39% of total memory usage, both better than Python's (75.96% and 21.67% accordingly). Additionally, GoLang completed predictions in a shorter time (1.52 seconds compared to Python's 2.81 seconds), while as seen in figure 5.7 the Python version also caused the Raspberry Pi to reach a sustained 100% CPU usage at peak periods. On the other hand, Kitsune while exploiting more or less the same memory and cpu as GoLang, requires a blocking period of 5000 seconds (1 hour and 38 minutes) for predicting 100.000 samples on a resource constraint device.

Resource	LightGBM GO	LightGBM Python	Kitsune
CPU	30.8125%	75.96%	31.53%
RAM	20.39%	21.67%	20.17%
Time	1.52s	2.81s	5000s

Table 5.3: Average cpu and memory consumption during each method's period of execution combined with the required execution time.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This thesis explored the development of a lightweight, efficient intrusion detection system tailored for IoT environments using machine learning models, specifically focusing on LightGBM and One Class SVM models. By implementing and evaluating these models on the Mirai Botnet dataset, the study demonstrated that optimized, non complex approaches can achieve high detection accuracy with minimal computational resources, a crucial factor for IoT devices with limited processing power.

The comparison between models revealed that LightGBM consistently outperformed traditional anomaly detection methods in both accuracy and speed. Kitsune, a well-known Neural Netowrk based network intrusion detection system, showed respectable results in terms of score, but its use in an recourse constraint device revealed flaws, indicating that it cannot be used efficiently in an IoT environment. LightGBM's ability to leverage supervised learning yielded better predictive power, while when embedded in a low-level language like GoLang, its implementation notably reduced memory and CPU usage compared to its Python counterpart, making it highly suitable for real-time applications in resource constrained IoT environments.

Through careful feature engineering and model tuning, this work highlighted the significant impact of dataset transformation and cross-validation on model effectiveness, particularly in distinguishing transition phases between benign and malicious traffic.

In conclusion, this thesis provides a robust framework for developing lightweight

IDS solutions capable of handling real-world IoT intrusion detection challenges. It underscores the value of tailored machine learning approaches for IoT, opening avenues for future advancements in IoT security through resource-efficient AI applications.

6.2 Future Work

Building on the achievements of this research, future work will focus on adapting this intrusion detection framework for direct deployment on actual IoT devices. A key objective is to develop a lightweight feature extraction package in GoLang capable of processing network flow data to extract the same 115 features used by Kitsune's feature extractor. This custom feature extractor would serve as a streamlined, resource-efficient tool for IoT environments, enabling real-time data processing on devices with limited computational power.

The primary goal of this future feature extractor in GoLang is to retain the efficacy and depth of Kitsune's detailed feature extraction while significantly reducing the overhead associated with its deployment on resource-constrained IoT devices. This will involve designing optimised algorithms for feature extraction that are both computationally lean and highly effective, focusing on efficient data handling and processing routines suited for embedded environments. By achieving feature parity with Kitsune, this GoLang-based extractor would enable seamless integration with the LightGBM and One Class SVM models trained in this study, thereby enhancing the models' performance in real-world scenarios.

Furthermore, real-world testing and iterative development will be essential. Initial testing could involve deployment on devices similar to those used in this study, such as Raspberry Pi, before progressing to more constrained IoT hardware. These tests would assess the package's ability to handle live network data streams, evaluate its impact on device performance, and ensure that detection accuracy remains high under practical conditions. The ultimate vision for this future work is a fully autonomous, lightweight IDS solution embedded directly within IoT devices, providing robust security with minimal resource usage.

This development has significant implications for enhancing IoT security in fields such as healthcare, industrial automation, and smart home technologies, where real-time, on-device threat detection is critical. As IoT networks continue to expand, a streamlined, efficient IDS solution embedded within each device could greatly reduce the vulnerability of these systems to attacks, providing a more resilient foundation for IoT infrastructure worldwide. Furthermore, with the growing complexity of cyber threats, proactive and adaptive security measures will become increasingly necessary. The integration of such solutions into IoT devices could foster greater trust and safety for users and organizations alike.

Bibliography

[1] S. Sinha, "State of iot 2024: Number of connected iot devices growing 13% to 18.8 billion globally," tech. rep., IoT Analytics, 2024. `https://iot-analytics.com/number-connected-iot-devices/`.

[2] W. Gay, *Raspberry Pi Hardware Reference*. 01 2014.

[3] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection.," in *NDSS*, 2018.

[4] Netscout, "22 ddos attacks to see trends." `https://nsfocusglobal.com/22-ddos-attacks-to-see-trends-in-2023/`, 2023. Report.

[5] Getastra, "45 global ddos attack statistics." `https://www.getastra.com/blog/security-audit/ddos-attack-statistics/`, 2023. Report.

[6] NordVPN, "What is the mirai botnet, and how does it spread?." `https://nordvpn.com/blog/mirai-botnet/`, 2024. Article.

[7] M. Humayun, N. Jhanjhi, A. Alsayat, and V. Ponnusamy, "Internet of things and ransomware: Evolution, mitigation and prevention," *Egyptian Informatics Journal*, vol. 22, no. 1, pp. 105–117, 2021.

[8] S. Moss, "Major ddos attack on dyn disrupts aws, twitter, spotify and more." `https://shorturl.at/5UiHb`, 2016. Article.

[9] Wikipedia contributors, "Zeroaccess botnet — Wikipedia, the free encyclopedia," 2023. [Online; accessed 15-February-2025].

[10] Wikipedia contributors, "Necurs botnet — Wikipedia, the free encyclopedia," 2024. [Online; accessed 15-February-2025].

[11] Wikipedia contributors, "Emotet — Wikipedia, the free encyclopedia," 2024. [Online; accessed 15-February-2025].

[12] Wikipedia contributors, “Storm botnet — Wikipedia, the free encyclopedia,” 2024. [Online; accessed 15-February-2025].

[13] D. Dong, Y. Wu, L. He, G. Huang, and G. Wu, “Deep analysis of intending peer-to-peer botnet,” in *2008 Seventh International Conference on Grid and Cooperative Computing*, pp. 407–411, IEEE, 2008.

[14] B. Sudharsan, D. Sundaram, P. Patel, J. G. Breslin, and M. I. Ali, “Edge2guard: Botnet attacks detecting offline models for resource-constrained iot devices,” in *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 680–685, IEEE, 2021.

[15] A. Mosenia and N. K. Jha, “A comprehensive study of security of internet-of-things,” *IEEE Transactions on emerging topics in computing*, vol. 5, no. 4, pp. 586–602, 2016.

[16] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, “Internet of things security: A survey,” *Journal of Network and Computer Applications*, vol. 88, pp. 10–28, 2017.

[17] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. De Alvarenga, “A survey of intrusion detection in internet of things,” *Journal of Network and Computer Applications*, vol. 84, pp. 25–37, 2017.

[18] U. Farooq, N. Tariq, M. Asim, T. Baker, and A. Al-Shamma’a, “Machine learning and the internet of things security: Solutions and open challenges,” *Journal of Parallel and Distributed Computing*, vol. 162, pp. 89–104, 2022.

[19] M. Rytel, A. Felkner, and M. Janiszewski, “Towards a safer internet of things—a survey of iot vulnerability data sources,” *Sensors*, vol. 20, no. 21, p. 5969, 2020.

[20] M. Baich, T. Hamim, N. Sael, and Y. Chemlal, “Machine learning for iot based networks intrusion detection: a comparative study,” *Procedia Computer Science*, vol. 215, pp. 742–751, 2022. 4th International Conference on Innovative Data Communication Technology and Application.

[21] E. Hodo, X. J. A. Bellekens, A. W. Hamilton, P.-L. Dubouilh, E. T. Iorkyase, C. Tachtatzis, and R. C. Atkinson, “Threat analysis of iot networks using artificial neural network intrusion detection system,” *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, 2016.

[22] S. Jain, P. M. Pawar, and R. Muthalagu, "Hybrid intelligent intrusion detection system for internet of things," *Telematics and Informatics Reports*, vol. 8, p. 100030, 2022.

[23] R. Al-amri, R. K. Murugesan, M. Man, A. F. Abdulateef, M. A. Al-Sharafi, and A. A. Alkahtani, "A review of machine learning and deep learning techniques for anomaly detection in iot data," *Applied Sciences*, vol. 11, no. 12, 2021.

[24] Z. Ahmad, A. S. Khan, W. S. Cheah, J. bin Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, 2020.

[25] M. Tambe, "Security and game theory - algorithms, deployed systems, lessons learned," 2011.

[26] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[27] N. M. Allifah and I. A. Zualkernan, "Ranking security of iot-based smart home consumer devices," *Ieee Access*, vol. 10, pp. 18352–18369, 2022.

[28] M. Yu, J. Zhuge, M. Cao, Z. Shi, and L. Jiang, "A survey of security vulnerability analysis, discovery, detection, and mitigation on iot devices," *Future Internet*, vol. 12, no. 2, p. 27, 2020.

[29] W. Xie, Y. Jiang, Y. Tang, N. Ding, and Y. Gao, "Vulnerability detection in iot firmware: A survey," in *2017 IEEE 23rd International conference on parallel and distributed systems (ICPADS)*, pp. 769–772, IEEE, 2017.

[30] A. Srivastava, S. Gupta, M. Quamara, P. Chaudhary, and V. J. Aski, "Future iot-enabled threats and vulnerabilities: State of the art, challenges, and future prospects," *International Journal of Communication Systems*, vol. 33, no. 12, p. e4443, 2020.

[31] P. Anand, Y. Singh, A. Selwal, M. Alazab, S. Tanwar, and N. Kumar, "Iot vulnerability assessment for sustainable computing: threats, current solutions, and open challenges," *IEEE Access*, vol. 8, pp. 168825–168853, 2020.

[32] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "Iot: Internet of threats? a survey of practical security vulnerabilities in real iot

devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.

[33] A. Costin, "Security of cctv and video surveillance systems: Threats, vulnerabilities, attacks, and mitigations," in *Proceedings of the 6th international workshop on trustworthy embedded devices*, pp. 45–54, 2016.

[34] I. Nadir, H. Mahmood, and G. Asadullah, "A taxonomy of iot firmware security and principal firmware analysis techniques," *International Journal of Critical Infrastructure Protection*, vol. 38, p. 100552, 2022.

[35] C. Wright, W. A. Moeglein, S. Bagchi, M. Kulkarni, and A. A. Clements, "Challenges in firmware re-hosting, emulation, and analysis," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–36, 2021.

[36] M. Frustaci, P. Pace, G. Aloi, and G. Fortino, "Evaluating critical security issues of the iot world: Present and future challenges," *IEEE Internet of things journal*, vol. 5, no. 4, pp. 2483–2495, 2017.

[37] A. Qasem, P. Shirani, M. Debbabi, L. Wang, B. Lebel, and B. L. Agba, "Automatic vulnerability detection in embedded devices and firmware: Survey and layered taxonomies," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–42, 2021.

[38] X. Feng, X. Zhu, Q.-L. Han, W. Zhou, S. Wen, and Y. Xiang, "Detecting vulnerability on iot device firmware: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 1, pp. 25–41, 2022.

[39] T. Yaqoob, H. Abbas, and M. Atiquzzaman, "Security vulnerabilities, attacks, countermeasures, and regulations of networked medical devices—a review," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3723–3768, 2019.

[40] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, 2008.

[41] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[42] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, pp. 3146–3154, 2017.

[43] K. K. Dataset. `https://www.kaggle.com/datasets/ymirsky/network-attack-dataset-kitsune/data`. Kaggle Dataset.

[44] P. Masschelier, “sklearn to golang.” `https://github.com/pa-m/sklearn`, 2019. GitHub repository.

[45] Ed Walker, “libsvm-go: Full port of libsvm in the go programming language.” `https://github.com/ewalker544/libsvm-go`. GitHub repository.

[46] M. Waskom, O. Botvinnik, D. O’Kane, P. Hobson, S. Lukauskas, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Yarkoni, M. L. Williams, C. Evans, C. Fitzgerald, Brian, C. Fonnesbeck, A. Lee, and A. Qalieh, “mwaskom/seaborn: v0.8.1 (september 2017),” Sept. 2017.

[47] D. Iakhin, “leaves: A prediction-serving library for gradient boosting models in go.” `https://pkg.go.dev/github.com/dmitryikh/leaves`, 2024. Accessed: November 4, 2024.