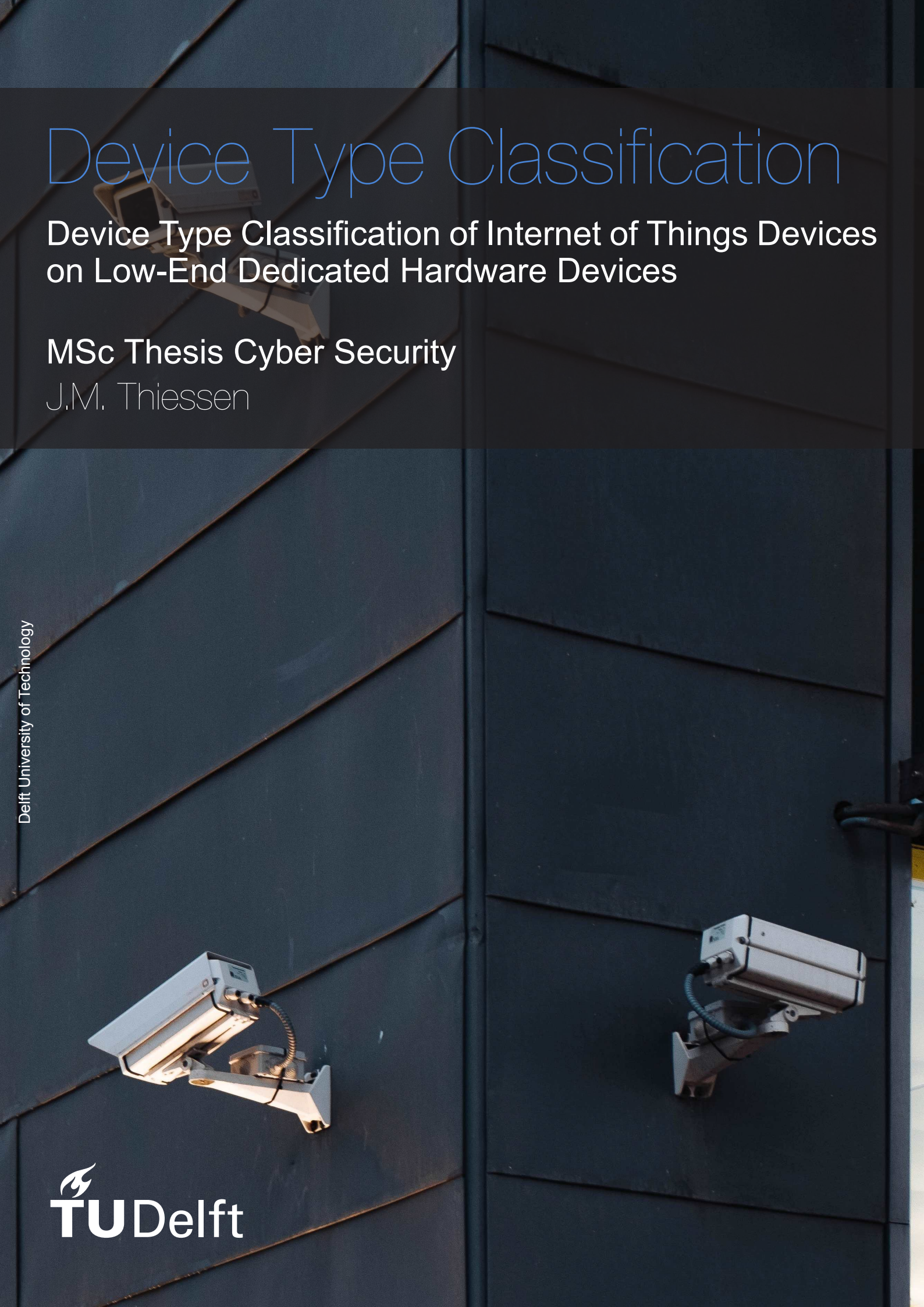


Device Type Classification

Device Type Classification of Internet of Things Devices on Low-End Dedicated Hardware Devices

MSc Thesis Cyber Security

J.M. Thiessen

Delft University of Technology

TU Delft

Device Type Classification

Device Type Classification of Internet of Things Devices on Low-End Dedicated Hardware Devices

by

J.M. Thiessen

at the Delft University of Technology,
to be defended publicly on the 12th of July at 15:00 CEST.

Student Number:	5350506	
Project Duration:	November 8, 2021 - July 12, 2022	
Thesis Committee:	Prof. George Smaragdakis	TU Delft, Advisor
	Dr. Apostolis Zarras	TU Delft, Supervisor
	Dr. Ranga Rao Venkatesha Prasad	TU Delft
Supervisor:	Dr. Elmer Lastdrager	SIDN

This research was conducted together with SIDN Labs.

Preface

After many years of studying, my journey toward a master's degree has ended. I am proud to say that I can deliver my thesis about 'Device Type Classification of Internet of Things Devices on Low-End Dedicated Hardware Devices' as my final contribution to obtaining my master's degree. Along the way, there were many obstacles, as the pandemic was very much alive. Nevertheless, it was a journey I was allowed to take with the support of a great set of people.

First, I would like to thank Dr. Elmer Lastdrager for his support during my thesis. He was always there to answer questions, proofread my work, and provide moral support. Furthermore, I would like to thank Dr. Apostolis Zarras for his advice and guidance during my thesis. Further, I would like to thank my thesis committee, Prof. George Smaragdakis, and Dr. Ranga Rao Venkatesha Prasad.

Next, I would like to thank SIDN and SIDN Labs for the fun times, the great opportunity to do my thesis there, and for including me as one of the team from the beginning. I have learned a lot from everyone there, from technical knowledge to the basics of brewing beer.

Finally, I would like to thank Armin, Aslı, Charlie, Linus, and Luke for all the support and fun (online) games we played. Furthermore, I would like to thank my friends and family for their support during my studies. I especially want to thank Angelica for her continuous support and patience during my studies.

J.M. Thiessen
Arnhem, July 2022

Abstract

The uprise of the Internet of Things (IoT) has been a hot topic for several years. While many see these devices and think they bring ease to their lives, it is far from reality. Researchers found many privacy and security problems within these devices in the last years. The popularity of these devices causes many users to bring a device into their home that could potentially infringe on privacy if not configured correctly. The first step to helping these home users secure their network starts by being able to autonomously detect the type of IoT devices connected to their network. With this information, firewall rules could enforce specific device behavior or provide the user with extra information on the type of the device and the risks it brings. It could inspire a new generation of home security devices focussing on securing IoT devices from a network perspective.

This thesis will introduce a new machine learning model to detect the type of unseen IoT device. Our work aims to classify devices into five categories, one more than the current state-of-the-art. Furthermore, we verify the model extensively on a large set of devices, with 74 measurements using Leave-One-Group-Out (LOGO) cross-validation. LOGO increases our testing set significantly in comparison with other works. Finally, LOGO will ensure that the training and test set was not handcrafted to obtain the highest possible accuracy, introducing fairness into our model design. The accuracy reached by our model is 73%, showing that Device Type Classification on unseen devices is feasible.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
CART	Classification And Regression Trees
CNN	Convolutional Neural Network
DTC	Decision Tree Classifier
HMM	Hidden Markov Model
IANA	Internet Assigned Numbers Authority
IoT	Internet of Things
ISP	Internet Service Provider
LIME	Local Interpretable Model-agnostic Explanations
LOGO	Leave-One-Group-Out
LSTM	Long Short-Term Memory
MAC	Media Access Control
RFC	Random Forest Classifier
RNN	Recurrent Neural Network
SHAP	SHapley Additive exPlanations
SVM	Support Vector Machine

List of Figures

List of Tables

1 Introduction

Ever been frustrated that your favorite website is down? While it might come as a surprise, it could be that the smart devices in your home are attacking it. In the last couple of years, the Internet of Things (IoT) concept gained popularity quickly; new devices are introduced constantly, from smart lights to egg trays. As a result, the number of connected IoT devices has increased rapidly since the introduction of, arguably, the first IoT device by Romkey in 1990 [48]. In 2021, the total number of internet-connected IoT devices has reached 14.6 billion, according to Ericsson [11]. Furthermore, predictions show that this rapid growth will continue, as Ericsson predicts there will be 30.2 billion connected IoT devices by 2027 [11].

The growing popularity has the effect that IoT devices have become a part of many people's daily lives, from controlling their lights to adjusting the temperature at home. However, they also introduced a new set of security challenges. The Open Web Application Security Project (OWASP) mapped the most common security problems in IoT [38], with the most common one having weak, guessable, or hardcoded passwords by the manufacturer. While a hacked device can impact someone personally, it also impacts corporations and other people's lives. For example, Mirai [3] is a famous botnet primarily focused on IoT devices. It used a table with default usernames and passwords to infiltrate the IoT devices and install itself. In 2016, the Mirai infected IoT devices performed a Distributed Denial of Service attack on Dyn that caused popular websites such as GitHub, Twitter, and Netflix, to go offline [40].

While the security problems of IoT devices can disrupt the daily lives of internet users, the security problems can also impact the users' privacy. For example, My Friend Cayla [19] is a smart toy for children with voice recognition functionality. The toy allows children to ask it questions, which it will reply to using the internet via a Bluetooth-connected companion application. Unfortunately, the toy had several vulnerabilities that allowed strangers to talk directly to children and even eavesdrop on someone playing with the toy through several walls [60]. The German Federal Network Agency (Bundesnetzagentur) even classified the device as a concealed surveillance device and requested parents to destroy the toy [9]. All attacks on the doll were only possible if the attackers had a Bluetooth device within 10 meters of it due to the Bluetooth connection range. If a device directly connects to the internet, it could give anyone with an internet connection access to it. For example, a vulnerable internet-connected baby monitor with a camera could potentially stream people's daily lives to anyone. Several sites try to index insecure cameras, such as Shodan[1] [55] and Insecam [17, 25]. These sites give people across the globe easy access to these vulnerable devices and their live streaming feed.

Preserving users' privacy and the security of IoT devices is a complex challenge. New IoT devices are constantly being released, behaving relatively different in terms of features and network activity. To solve the problems related to IoT devices, companies and organizations released several solutions to secure IoT devices. For example, Splunk [57] released plugins to enable monitoring of Industrial IoT devices, and Microsoft released a version of Microsoft Defender for IoT [37]. While most of the

[1] Shodan requires a member subscription to access their image service.

solutions focus on corporate network infrastructures, some projects also focus on home networks. For instance, SIDN Labs released a software package with a user-operated privacy manager and a reverse firewall to automatically block suspiciously outgoing network traffic pro-actively [30].

The downside of the existing solutions is that much expertise or interaction is required to use the software correctly. An ideal system would automatically enforce policies based on the type of devices connected to the network. For example, by default, the control panel of a smart candle should only be available via an internal network. Device or category-specific policies enable fine-grained access control for the devices in a home network, limiting access to authorized parties only and allowing vulnerable parts to be automatically blocked until patches are released and installed. In addition, automatic policies would remove most interaction requirements and allow people with less technical expertise to benefit from the system.

The first step of building an automated system relies on detecting the IoT device types within the network, also called Device Type Identification. After such types are reliably detected, one could build security policies around them to properly control what devices could and could not do. However, to allow deployment in the homes of many users, one should keep in mind the hardware already available or within the price range of a general household. An example of the hardware already available in many homes is a router provided by their Internet Service Provider (ISP). Unfortunately, many of these routers run on low-end hardware with minimal resources. Designing such a system would be the first step of returning privacy and security to the homes of many IoT users.

Researchers have proposed several automated systems for device type identification and complete systems for policy enforcement. However, to the best of our knowledge, the systems detect the IoT device's make and model [39] or behavioral clusters [33]. Moreover, it would be infeasible for the proposed systems to train a model for deployment in home networks as it would be impossible to gain network traffic from all devices present in all households. Furthermore, most home users would not have the expertise to train a model themselves. Hence, our approach will categorize IoT devices into a descriptive class. Moreover, to the best of our knowledge, the proposed classification models in this subfield focus on a small set of classes [4, 14, 35] or limited devices per class [32], and none focus on low-end hardware applications.

1.1. Research Question

In this thesis, we attempts to create a machine learning model to classify IoT devices into a set of descriptive classes. Furthermore, it attempts to do this by only using limited resources. Hence, the main research question is:

How can we perform Device Type Classification of IoT devices
on low-end dedicated hardware devices?

To answer the main research question, we define five sub-questions. The sub-questions are:

1. How do existing models perform on low-end dedicated hardware devices?
2. What are the difficulties when classifying IoT devices not present in the training set?
3. How do we categorize appliances with the same functionality from different brands?
4. How well does this new model perform on low-end dedicated hardware devices?
5. How well does this new model perform in terms of accuracy?

1.2. Main Contributions

Our work leads to three main contributions in the field. To summarize:

1. Our work re-implements two models and share our source code. The re-implementations verify whether the papers are reproducible and similar results can be produced with the information given in their work. Additionally, by sharing the source code, we allow it to be verified by other researchers.
2. We show the resource usage of existing models, verifying whether deployment on low-end hardware would be feasible.

3. Our work implements a new model for Device Type Classification into five categories, one more than the current state-of-the-art. Furthermore, we extensively tested our solution on 74 unseen devices taken from multiple datasets, which is the most extensive test performed in this field to date, to the best of our knowledge. The extensive testing of our model shows that we can reliably predict the type of IoT device on a network with features that are easy to extract. Additionally, we show that the model's resource usage is low enough to be deployed on low-end hardware.

1.3. Outline

The rest of our thesis is structured as follows. Chapter 2 will explain all the necessary background knowledge. Chapter 3 will go over the existing models in the field, exploring various machine learning methods used in the field of Device Type Classification. Chapter 4 re-implements and re-evaluates two papers published in the field, both handling unseen devices. Additionally, this chapter shows the resource usage of the models. Chapter 5 explains the approach taken for the implementation and evaluation of the model designed. Chapter 6 explores the performance of our model, including an in-depth analysis of the predictions made. Chapter 7 discusses the limitations of our research and future work. At last, in Chapter 8 we conclude our research.

2

Background

This chapter will go in-depth into the background knowledge required for this thesis. Moreover, we will explain several machine learning concepts and feature extraction algorithms used by the models discussed in this work.

Firstly, we will explain the idea behind the PageRank algorithm. Secondly, we go in-depth into how the RFC works. Furthermore, the PageRank algorithm and RFC cover the concepts used to re-implement the model proposed by Melnyk, Haleta, and Golphamid [35]. Additionally, we use the RFC as a building block for our implementation. Thirdly, we explain the basic idea of the CNN and LSTM used by our re-implementation of the model proposed by Bai et al. [5]. At last, we will go in-depth into the cross-validation technique used in our research.

2.1. PageRank Algorithm

Initially, the PageRank algorithm originated from the web page ranking system of Google. Later, it found use in several other fields where link analysis is required, such as ranking social media users [62] or rating systems [21]. For example, the work of Melnyk, Haleta, and Golphamid [35] uses PageRank to classify internal communications. The PageRank for this model gives information about the connectivity strength within the network. The idea behind it is that the connectivity strength of a single-purpose device is lower than that of a multi-purpose device, which is an essential detail in the model proposed by Melnyk, Haleta, and Golphamid [35].

Furthermore, the PageRank considers the number of outgoing and incoming links for the score, which are connections in our case. It is important to note that the connections are flows, not packets. If it were packets, then the number of links would equalize, as in many cases, as connections are often bidirectional.

Equation 2.1 [8] shows the formula of the PageRank algorithm. The formula contains a single parameter d, also known as the damping factor. The damping factor originally was defined as a probability of a person continuing to click on links randomly. Often this value is set to 0.85 [8], which is also the case for our implementation. Furthermore, the formula requires a set of devices, let's call this set $X = \{x_1, x_i, \ldots, x_N\}$, where the set contains a total of N devices. We then have three functions, PR, C, and L, where PR is the PageRank algorithm, as defined by the formula. C is the connectivity matrix. The connectivity matrix is defined as follows, let $x, y \in X$, then if and only if $y \in C[x]$, there is an outgoing link from y to x. Lastly, the function $L(x)$ returns the number of outgoing links from device $x \in X$, e.g., $L(x) = |\{y \in X | y \in C[x]\}|$.

$$PR(x) = (1 - d) + d \sum_{y \in C[x]} \frac{PR(y)}{L(y)} \tag{2.1}$$

As Equation 2.1 displays, the PageRank algorithm depends on itself. Moreover, this means that it is an

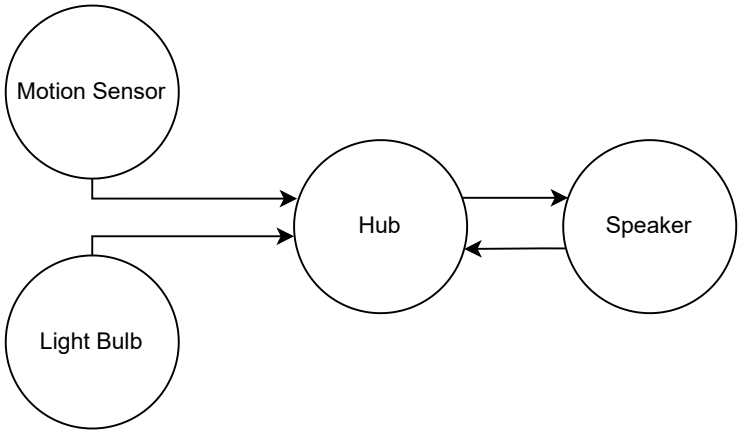


Figure 2.1: The graph of the example network.

iterative algorithm. We loop until the algorithm converges, which was after 500 iterations for our use case. Initially, all devices have a PageRank value of 1.

2.1.1. Example

To better illustrate the idea behind the PageRank algorithm, we will give a concrete example. For this purpose, we created a small graph representing a home network. The network consists of a light bulb and a motion sensor directly connected to a smart hub, which is unidirectional. Additionally, the network has a speaker with a bidirectional connection with the hub. Figure 2.1 shows the network graph of the example, and Table 2.1 shows the connectivity matrix.

From/To	Motion Sensor	Light Bulb	Hub	Speaker
Motion Sensor	0	0	1	0
Light Bulb	0	0	1	0
Hub	0	0	0	1
Speaker	0	0	1	0

Table 2.1: The connection matrix of the PageRank example.

As mentioned before, all page rank values contain the value 1 after initialization. Table 2.2 displays the outcome of the first iteration.

Device	Computation
Motion Sensor	$0.15 + 0.85 * 0 = 0.15$
Light Bulb	$0.15 + 0.85 * 0 = 0.15$
Hub	$0.15 + 0.85 * (\frac{0.15}{1} + \frac{0.15}{1} + \frac{1}{1}) = 1.255$
Speaker	$0.15 + 0.85 * \frac{1.255}{1} = 1.21675$

Table 2.2: The PageRank values of the first iteration.

Table 2.3 shows the final results of the computation, which converges after 104 iterations. As illustrated in Figure 2.1, the hub node has the most incoming connections, which directly translates to the highest PageRank. In comparison, the devices with no incoming connections have the lowest PageRank.

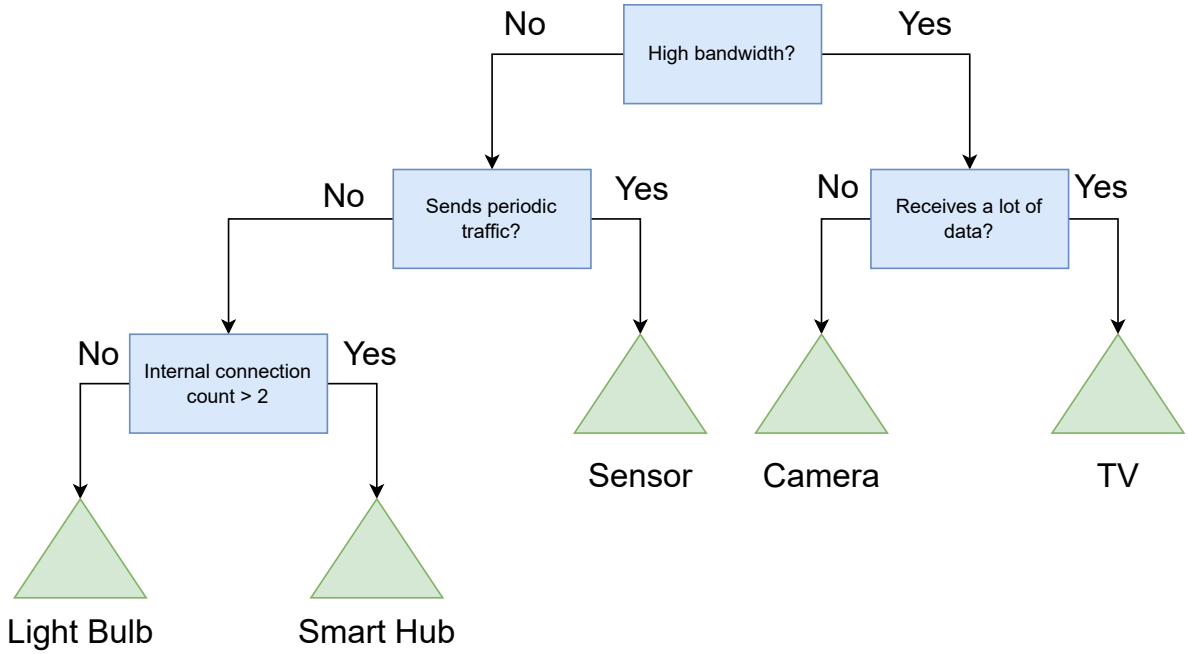


Figure 2.2: A fictional example of a DTC.

Device	Computation
Motion Sensor	0.15
Light Bulb	0.15
Hub	$0.15 + 0.85 * (\frac{0.15}{1} + \frac{0.15}{1} + \frac{1}{1}) = 1.919$
Speaker	$0.15 + 0.85 * \frac{1.255}{1} = 1.781$

Table 2.3: The PageRank values after the algorithm converges (104 iterations).

2.2. Random Forest Classifier

The RFC is an ensemble learning method, using multiple DTCs at its core, allowing it to correct the overfitting problems the DTC has on the training set [24]. In this section, we will explain in-depth how the RFC works. Firstly, we will explain the inner workings of the DTC. Next, we will explain the construction of an RFC by using the knowledge gained from the DTC section.

2.2.1. Decision Tree Classifier

The DTC works by asking questions about specific features in the data. These questions are the non-leaf nodes in the tree. Based on the outcome of the question, it will go down in the tree following the path with the answer on one of the node's edges. After traversing the tree, it will reach a leaf node. A leaf node represents the outcome of our classification task based on the answers given while traversing the tree. In our case, the leaf nodes contain the predicted class of the IoT device we are trying to classify. It is also possible that a probability is present in the leaf nodes, as some splits might not have a perfect outcome. Figure 2.2 shows a small fictional example of a DTC. The tree tries to classify what kind of IoT device the data represents. Furthermore, the squares represent the questions on the data, and the triangles represent the leaf nodes with the final answer.

Split Criteria
To build a tree, we need criteria to measure the performance of a specific split, also known as the 'criterion.' We will focus on the two implemented in the sci-kit library used in this thesis [52], the Gini impurity [45], and Entropy [45]. The Gini impurity measures the probability of misclassifying a random element of our set. Equation 2.2 [45] displays the formula of the Gini impurity, which we try to minimize. Furthermore, in the formula, P is the probability that an element has a specific class, and N is the number of classes in our classifier.

$$GINI = 1 - \sum_{i=1}^{N} P(i)^2 \tag{2.2}$$

Equation 2.3 [45] [54] shows the formula behind the Entropy criteria. In general, Entropy is a measure of information in an event. The more certain an event is, the less information it will contain [54]. The tree split is on points where we are most certain, with low Entropy, of the outcome of a class label. However, due to the log in the formula, it is computationally slower than the Gini impurity. The symbols in Equation 2.3 have the same meaning as in Equation 2.2.

$$Entropy = -\sum_{i=1}^{N} P(i) \log P(i) \tag{2.3}$$

Building a Tree
The criteria in the previous section introduced the basic theory behind building a tree. This section will explain how to construct a tree with these criteria. We will focus on the Gini impurity in all of our examples. However, the steps for the Entropy criteria are equivalent. The only difference is the formula used for evaluation.

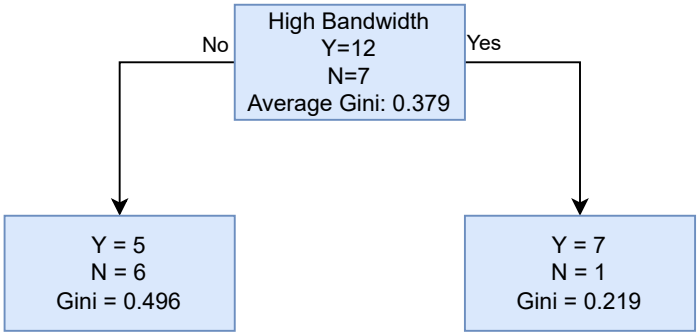


Figure 2.3: An illustrative example of the average Gini impurity of a fictional split.

The first step is to find the root node of our tree, which is the node split with the lowest average Gini impurity. The idea behind the average Gini impurity is to take the average between the nodes created by the split. The average considers the number of samples a specific child node still represents after a split. To illustrate, Figure 2.3, a small split is displayed. The split nodes have a Gini impurity of 0.496 and 0.219, respectively, with 11 and 8 elements. The Gini impurity of the split is then the average of these two nodes, resulting in the following equation:

$$\frac{5+6}{19} \cdot 0.496 + \frac{1+7}{19} \cdot 0.219 = 0.379 \tag{2.4}$$

For both branches of the new root node, we repeat the evaluation of all possible splits at that position. We only consider the elements of our dataset related to this particular path. We repeat this split creation process until we reach one of the three ending conditions:

1. The average Gini impurity of the split is higher than the original node.
2. We used all possible questions.

3. We reached a user-defined max-depth.

2.2.2. Constructing the Forest

The basis of a RFC is the DTC. The idea behind the RFC is to use multiple DTCs to remove bias [24]. However, the construction algorithm of the trees requires to be different to ensure that the trees build are not equivalent. If we use the same data, the tree's construction would be similar. Hence, the RFC algorithm changes what data every tree has available for its construction, called bootstrapping [45]. Bootstrapping takes random rows from our dataset. However, it does sampling with replacement, allowing us to have duplicate entries for a single tree.

Additionally, the algorithm of the RFC adjusts the algorithm used to split a node. The original DTC algorithm considers all available features for the split, while the RFC protocol only considers a random subset of features [45]. The bootstrapping and random subset of features generate trees differently, removing bias toward particular features or data points.

2.2.3. Majority Vote

Internally, the RFC has multiple trees, each giving a prediction of the input data. In the end, the RFC needs to settle which prediction class to take as the final one. A common way is to select the class with the majority of the predictions, also called the majority vote.

The scikit-learn [52] library used in this thesis does not use the majority vote principle but takes the probabilities into account. It chooses the class with the highest mean probability over all the trees. As explained before, the leaf nodes that contain the prediction process outcome can contain probabilities based on the impurity of the node.

2.3. Neural Networks

This section gives a brief overview of the idea behind the CNN and LSTM model as proposed by Bai et al. [5]. The model is later re-implemented. Furthermore, it aims to give a high-level overview of its idea and advantages.

2.3.1. Long-Term-Short-Term Memory

LSTM is a variant of recurrent neural networks capable of learning from data that depends on order, for example, time-series data [45]. In the case of device classification, it finds the relationship between consecutive network flows of the same device. It allows us to not only learn from a single record but also keep previous records into account. The LSTM aims to solve the vanishing gradient problem of traditional recurrent neural networks [45], which causes "short-term memory." The vanishing gradient problem causes earlier layers not to learn as efficiently, as the update gradient for the parameters gets smaller as it converges through the network. To conclude, the LSTM allows us to consider the previous network flows to create a more complex relationship, something we cannot do by simply looking at a single flow.

2.3.2. Convolutional Neural Network

A CNN can take spatial and temporal information into account. For example, it can model 2D and 3D objects without losing information. In comparison, a traditional feed-forward network expects a single dimensional vector and hence loses the information that the dimensions might carry. Instead, CNNs often use operations based on kernels and filters that slide over the input data [36]. An example of such a layer with a sliding operation is the max pooling layer, which takes the maximum value from a region within the input data. Figure 2.4 shows a simplified illustrative example of this. The input of the max pooling function is a 2D matrix, where there are four regions. The network then takes the maximum from these regions. Handling 2D data in such a way is something a traditional feed-forward network cannot achieve, as it only has a single dimension. Due to this, CNNs got popular in various fields, such as image and video recognition [36].

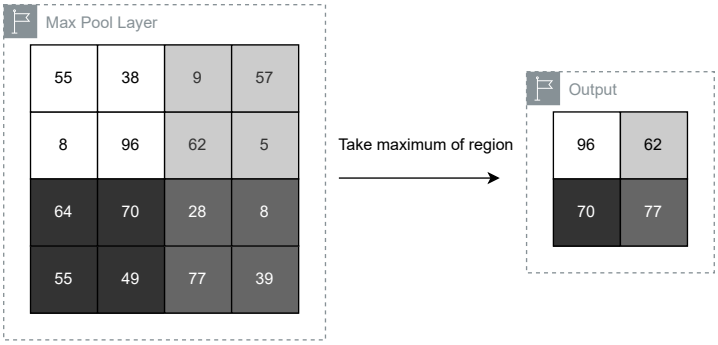


Figure 2.4: An example illustration of the max pooling layer.

2.3.3. Cascading

The idea behind the cascade is that the LSTM feeds the output directly into the CNN. The cascade combines the time-series and memory functionality of the LSTM with the spatial and temporal awareness of the CNN. Bai et al. [5] designed the model to capture the data's global and local temporal relations.

2.4. Leave-One-Group-Out Cross-Validation

In machine learning, cross-validation measures how well a model performs on different sets of unseen data. It finds its use in measuring the stability of a machine learning model. The idea behind cross-validation is to use a different chunk for testing in every step. The other chunks will be included in the training set to train the machine learning model. The accuracy of a step gets measured by running our model on the left-out chunk. We repeat this process until we have used all possible chunks for testing. In the end, we can argue about the average accuracy of our model on different sets of unseen data. It is important to note that a new model gets trained for every new data split.

While there are many variants of cross-validation, we will only focus on one in this work. Furthermore, we will use the LOGO [43] [50] cross-validation technique for our thesis. In our work, a group means an IoT device. Hence, the idea is to leave one IoT device out of the training set at every cross-validation step. We can validate our model's performance by taking the average results obtained during the cross-validation steps. Figure 2.5 shows an illustrative example of what this would look like in practice. The benefit of using LOGO is that it does not introduce a bias when splitting the training and testing set, as we test on all possible devices.

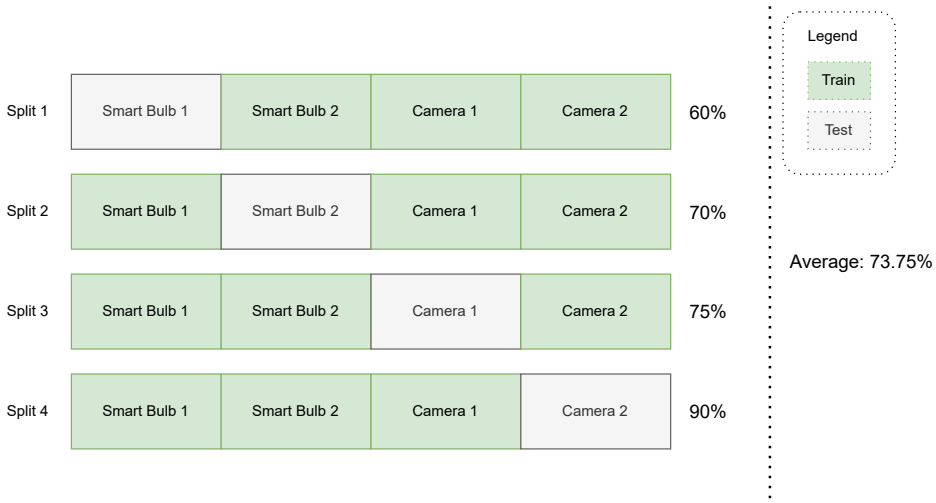


Figure 2.5: Illustration of how LOGO cross-validation splits work.

3

Literature Review

Device type identification of IoT devices has evolved significantly in the last years. Several machine learning models got proposed, all using different techniques and methodologies. This chapter aims to present a background on the research of device type identification. We split the related work into three categories:

1. Device type re-identification focuses on re-identifying the make and model of the device.
2. Device type detection classifies whether the device is an IoT device.
3. Device type classification aims to classify into a set of pre-defined categories—for example, kitchen appliances or smart speakers.

At the end of the chapter, we determine the current challenges still open in the field.

3.1. Device Type Re-identification

Device type re-identification aims to re-identify the make and model of the device. Re-identification is essential in computer networks, as relying on information broadcasted by the device is not guaranteed to be correct. For example, an attacker can easily modify the Media Access Control (MAC) address that comes with the devices [10]. Unfortunately, detecting modified MAC addresses is a challenging task, which requires complex techniques [58] or adjustments in packet frames [12]. Hence, MAC addresses are not suitable to detect which device is currently communicating. Re-identification allows us to detect the device by its behavior which is more challenging for an attacker to modify without being detected.

In 2018, Shahid et al. [53] compared six machine learning models, with the RFC having the best accuracy of 99.9%. However, the downside of the proposed solution is that the dataset created during the research only contains four devices, so the accuracy and scalability in more extensive networks are unclear.

Ammar, Noirie, and Tixeuil [1] proposed a model that trains a single classifier per class based on 28 devices. Additionally, the model can detect whether a device is unseen. Unseen devices are recognized when all classifiers indicate that they are not part of their class. Overall, the model's accuracy reaches 97% when based on the DTC using the one-vs-all method. The training set for the models contains textual features extracted from the payloads and flow features of the network traffic. The downside of the model is that the textual features will become harder to analyze as using encrypted channels is standard these days. Additionally, it is unclear what the influence of the textual features is on the model. If the influence is significant, an adversary can spoof the textual features and influence the classification outcome.

In comparison, Hamad et al. [22] proposed another one-vs-all approach based on the RFC. It does not use textual features but only bases the training on the network flow. The dataset is created by generating fingerprints of the network flows. Moreover, we can see that a similar method's accuracy without textual features has a slightly worse accuracy of only 90.3%.

Miettinen et al. [39] proposed a complete architecture for rule enforcement based on device type re-identification. The presented architecture has a set of binary RFCs at its core, one per device type. Moreover, none of the binary classifiers would match if a new device is detected. Hence, the architecture also informs the users of newly connected devices. The system allows for security rule enforcement for vulnerable IoT devices.

All the models in this section use traditional machine learning methods. The models show promising results in terms of accuracy. However, the downside of traditional machine learning methods is that complex feature engineering and data analysis is often required to get high accuracy.

3.1.1. Deep Learning

Deep learning automatically extracts features from the data fed to the neural network, simplifying the process. Hence, it tackles the problems with complex feature engineering as introduced by traditional machine learning models.

Greis et al. [20] compared two deep learning methods, a CNN and a LSTM Recurrent Neural Network (RNN). The paper shows that the automatic extraction of essential features achieves a high accuracy of at least 97% for both methods. Additionally, the paper compares itself with the IoT sentinel model, which reached an accuracy of 82%. In comparison, Yin et al. [61] proposed another deep learning model, named IoT ETEI (End-To-End IoT). The model consists of a CNN that supplies the results to a bidirectional LSTM. The model reached an accuracy of 99%, classifying two datasets of 18 and 17 devices. The model's primary focus was to introduce a method for identifying IoT devices in a scalable way. Lastly, Najari et al. [41] researched the performance differences between a Hidden Markov Model (HMM) and an LSTM RNN. The research showed that the HMM outperformed the experimental setup of the LSTM RNN. The HMM reached an accuracy of 93.1%, while the LSTM RNN only reached an accuracy of 60.7%.

While the deep learning models show high accuracy without complex feature engineering, it also has the downside of becoming a black box. Moreover, the features used for classification and their importance are unknown. While algorithms can help explain such black-box models, such as SHapley Additive exPlanations (SHAP) [31] and Local Interpretable Model-agnostic Explanations (LIME) [47], it is still the question of whether algorithms explaining algorithms increases transparency enough.

To conclude, the downside of the supervised models is that they are generally not trained to classify unseen devices, as they only identify the make and model. Furthermore, creating a model for all possible IoT devices is impossible. It would require information on all IoT devices currently being used, and the model would require constant updates.

3.1.2. Unsupervised Learning

To tackle the problem of supervised learning, researchers started to look at unsupervised learning. Anantharaman et al. [2] proposed an unsupervised learning scheme based on clustering. Furthermore, their model does not require any prior knowledge of the devices. The primary task of the model is to create a cluster per device and detect when the device would start deviating from the cluster. If the device differs from the cluster, an anomaly is detected. Overall, the model reaches an accuracy of over 95%. The downside of the approach is that it only detects anomalies; the model can apply no additional security policies to the device.

In comparison, Marchal et al. [33] introduced a system that distributes fingerprints via a cloud service named AuDI. The cloud service contains a policy database, linking specific policies for each device type it detects. These policies enforce specific security rules onto IoT devices. Furthermore, the detection task uses a clustering algorithm based on fingerprinting the periodic traffic of the IoT devices. Overall, the system achieves an accuracy of 98.2% and allows systems to learn about devices they do not have in their current network.

While unsupervised learning solves some of the problems supervised learning has, it limits specific classification tasks. For example, the supervised algorithms showed that the approaches accurately differentiate between make and model. However, unsupervised learning will not have the information about the naming of the actual device or category, making it difficult for a person to interpret. One could name the clusters by hand, requiring no overlap in said clusters. However, AuDI [33] shows

clear overlap in the clusters making this approach more difficult. For example, the algorithm clustered a smart speaker with a weather station and a hue switch.

3.1.3. Federated Learning

The downside of the aforementioned deep learning and machine learning approaches is that they only consider local data. As a result, new devices that get added to the network will be unknown and, therefore, undetected. While the AuDI model tackles some of these problems, it contains some privacy concerns. The fingerprints sent to the server allow it to learn what type of IoT devices each network has. Moreover, He et al. [23] proposed a Federated Learning comparison of a federated CNN and LSTM. The idea of the federated learning model is that the devices on the network edge extend a model locally and then share the model's parameters to a server that will aggregate the results. The server will adjust the model based on the aggregation and transfer this model back to the client, repeating the process. The goal of the model is that every client participating gains information about more than the devices locally. If a new device gets connected in the future to at least two networks, it will know about the device beforehand and successfully classify the device. Hence, solving the unseen device problem non-federated learning supervised model suffers from, as introduced in the previous chapter. Overall, the accuracy declines compared to other methods due to the federated aspects. Moreover, the model reaches 87.2% and 88.2% accuracy for the CNN and LSTM models, respectively.

3.2. Device Type Detection

Device type detection aims to detect whether a device is an IoT device. The models discussed in this chapter also can classify IoT devices. However, our focus will be on identifying whether a device is an IoT device. Meidan et al. [34] proposed ProfilIoT, a multi-stage classifier. The classification techniques used are different for each device type. Furthermore, testing and training used the same 13 devices. First, a binary classifier performs device type detection. Then, the other binary classifiers determine the IoT class. ProfilIoT has two non-IoT binary classifiers for a computer and a smartphone. The smartphone model reached an accuracy of 100%, while the computer got a near-perfect performance, with a false-positive and false-negative rate of 0.003. In comparison, Sivanathan et al. [56] proposed another multi-stage classifier using the Naive Bayes Multinomial in combination with an RFC. The average accuracy of the model is 97.32% and detects non-IoT devices with an accuracy of 99.7%.

Fan et al. [18] proposed a CNN model with two fully connected layers. One layer contains two nodes, determining whether the device under analysis is an IoT device. The other layer will determine the device type identity. The model requires only 5% of the labeled data per device from their public dataset to get an average accuracy of 99.8%. The accuracy of detecting whether a device is an IoT device is 99.5%. The model performance got evaluated on a dataset with 24 devices.

3.3. Device Type Classification

Device type classification aims to classify a device into a specific category; for example, a Samsung Smart Fridge would get the label kitchen appliance. The problem with device type re-identification is that it suffers from an explosion problem, as we require to model every device. Modeling every device would mean either a cluster or a label gets created. Therefore, the model would become large in space and computational requirements, making it unable to run on a lower-end device. Furthermore, hundreds of devices release each year which would mean constantly retraining the model. Moreover, device type classification tries to cover the middle ground between device type detection, and device type re-identification, by providing several classes that devices are classified within.

Melnyk, Haleta, and Golphamid [35] proposed a machine learning model that can differentiate between two types of devices. Namely, a single-purpose device like a lightbulb and a multi-purpose device like a smartphone. Furthermore, an RFC performs the classification task. The model achieves an accuracy of 94%, tested on a dataset with 50 devices.

In comparison, Maiti et al. [32] proposed a model to classify devices within ten categories. Furthermore, it compares an RFC, Support Vector Machine (SVM), and a Classification And Regression Trees (CART). The RFC showed the most promising results for all the tests performed, with an accuracy of 90%. The training and test data got extracted from the WiFi frames present in the link-layer of the OSI

model. While the model has excellent accuracy, it misclassifies almost all access points as cameras.

While most models use pre-defined classes based on device function, it is also possible to define classes based on network traffic behavior. For example,Cvitić et al. [14] proposed a model that classifies devices based on behavior in four categories. The model reaches an accuracy of 99.79% while using the boosting algorithm LogitBoost.

Besides focussing on the network flows as most research does, one could also look into other protocols used by IoT devices. For example, Babun et al. [4] proposed a model based on the Z-Wave and ZigBee protocols. It compared different models for each protocol. Bayes net showed the best performance for the ZigBee protocol and the RFC for the Z-Wave protocol. The average accuracy for ZigBee and Z-Wave was 91% and 93.25%, respectively.

The models discussed all focus on traditional machine learning models. Furthermore, one could also use deep learning models for device type classification. For example, Bai et al. [5] proposed an LSTM-CNN cascade model, where the input sequence first goes through an LSTM layer before being analyzed by the CNN. The model focuses on four categories and achieves an average accuracy of 74.8%.

A hybrid deep learning approach got proposed by Bao, Hamdaoui, and Wong [7]. It uses a combination of deep learning and clustering to classify and detect unknown devices. It tries to classify known devices into a specific category while giving a number to unknown devices. The goal of the model is to detect anomalies that might occur in the network, which could indicate security-related incidents. To reduce the dimensionality of the dataset, it employs an autoencoder. The average accuracy is between 81.1% and 91.2%, depending on the number of features chosen.

3.4. Current Challenges

Having evaluated the current state of device type identification of IoT devices, we can define the research gap as follows:

1. To the best of our knowledge, no model focuses on resource usage. Hence, they may be costly and challenging to deploy in a real-world scenario.
2. Many works focus on detecting the make and model of a device using supervised learning, resulting in scaling difficulties.
3. For most of the papers, the categorization of IoT devices only consisted of a limited number of categories.
4. Many works do not consider unseen devices, making it unclear how the models behave on unseen devices.

4

Reference Implementations

In this chapter, we discuss the re-implementation of two existing models. The first model is a RFC proposed by Melnyk, Haleta, and Golphamid [35]. The model's primary task is to identify whether a device is a single purpose or a multi-purpose device. Secondly, we implemented the LSTM-CNN cascade model Bai et al. [5] proposed, which has the primary task of classifying a device into four categories. Both models focus on classifying devices not present in the training set, also known as unseen devices.

It is important to note that the RFC-based model got implemented from scratch. We reached out to the authors several times without success, so we could not rely on their implementation. Moreover, our re-implementation shows similar performance as the model described in the paper. However, we made several assumptions during our re-implementation about the definitions of features. Moreover, we were also required to re-classify the devices used for training and testing, as the paper did not describe which class every device has.

Furthermore, for the LSTM-CNN cascade model, we received the model's code. The feature extraction got re-implemented from scratch, based on several assumptions led by experiments. For example, the paper did not describe the traffic direction and the features used for the traditional models. However, we could not reach a similar accuracy as described in the paper.

The main goal of the re-implementations is to get a baseline of how these types of models perform on low-end hardware. Furthermore, we will look into the resource usage of the models. It is important to note that accuracy is only a secondary concern for our re-implementations, as the original papers explore it in detail. However, while it is a secondary concern, we will explore it in detail in the next chapter for two reasons. Firstly, the accuracy indicates if the re-implementation is equivalent to that of the paper. Hence, it indirectly confirms that the resource analysis uses a model close to the original. Secondly, it shows whether reproducing the paper is possible with the provided information, which is an essential factor in scientific research.

4.1. Performance Metrics

We investigate the model's performance by exploring the results of several metrics. The results of these metrics will be a baseline for later comparison with our own designed model. These metrics are:

Average Prediction Times For this feature, we call the predict function of the models with the test set array. We then take the time it takes, divided by the number of elements in the test set.

Memory usage during predictions The memory usage will be analyzed using the memory_profiler package created by Pedregosa [42]. The analysis executes a single prediction at a time and reads them one by one from a file. The reading process ensures a fair comparison between models with different input sizes.

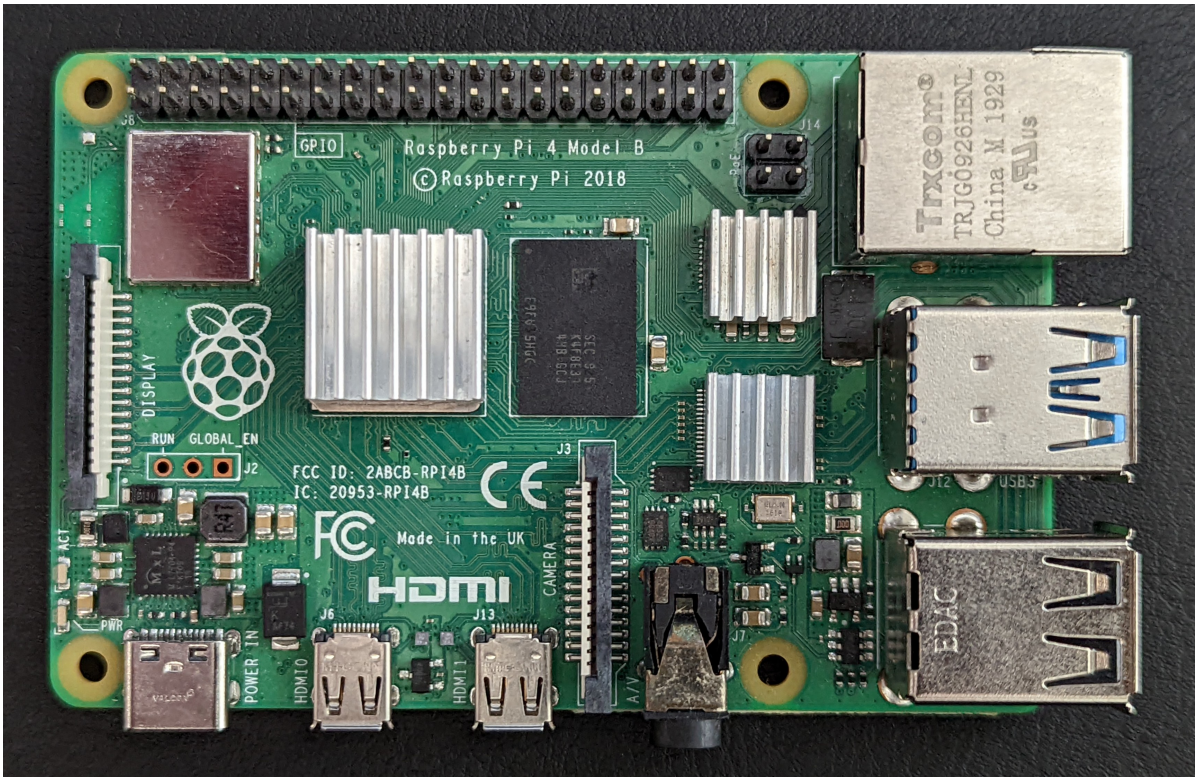


Figure 4.1: The Raspberry Pi 4 used with heatsink placement.

Disk space requirements This feature counts the bytes needed to store the model. Furthermore, the storage method used for the RFC model is joblib [28], as recommended by the scikit-learn [51] library. For the LSTM-CNN model, we use the build-in functionality of TensorFlow that the implementation of Bai et al. already utilized.

We selected these metrics because they represent the limitations of our hardware well. For example, if the model is too big, we cannot store it. The average prediction time shows if the method can classify a certain number of devices within a reasonable time. Lastly, due to the limited memory of these devices, it is essential to see how many resources a prediction takes.

We run all tests on a Raspberry Pi 4 with 4GB of ram and a BCM2711 SoC. The storage used is an Ultra MicroSDHC Class 10 UHS-I A1 manufactured by Sandisk. Moreover, the Raspberry Pi uses a few heatsinks to ensure it stays at a relatively low temperature. Figure 4.1 displays the testing setup, including the heatsink placement. Furthermore, we repeat the average prediction time experiment 50 times, and the median of these experiments is the final result. The repeated experiments ensure that external factors did not influence the result, such as the operating system taking up some CPU for kernel-related tasks.

Furthermore, we increase the test sample set for the RFC memory experiment by concatenating the file 50 times. The more extensive test set gives us a better overview of peak memory usage, as the original set is significantly smaller than the one of the LSTM-CNN set. At last, all measurements of the experiments are performed on the test dataset as described in the original paper.

4.2. Random Forest Classifier

The RFC model's primary task is to classify devices into single-purpose and multi-purpose devices. According to Melnyk, Haleta, and Golphamid [35], a single-purpose device is a resource-limited device, such as cameras and sensors. Multi-purpose devices include high-tech devices like smartphones and laptops. Our re-implementation of the 21,000 packet experiment reaches an accuracy of 97%, higher than the described accuracy of 94%. However, the accuracy of the 560,000 packet experiment did not

reach the described accuracy: we reached 94% rather than the paper's described accuracy of 97%. The accuracy difference is that a single device is classified differently. Hence, we suspect that our manual classification incorrectly predicted a single device.

The paper uses two independent datasets for the implementation of the model. The dataset by Perdisci et al. [44] provides the training data, and the dataset by Sivanathan et al. [56] provides the testing data. Furthermore, the original paper does not mention which devices are single-purpose or multi-purpose. Hence, the classification was done manually with the description provided in the paper. Appendix A provides an overview of our manual classification.

The first step of our re-implementation was creating the feature extraction. However, as mentioned before, no code or resources were provided. Hence, we re-implemented it from scratch using several scripts and wrappers around readily available tools. The tools used by our scripts are Joy by Cisco [15], which provides us with network flows, and TShark by Wireshark [13] for general analysis. Secondly, we used the scikit-learn [43] library in Python to train and test the RFC. The complete source code can be found on our GitLab[1].

4.2.1. Feature Extraction

For the feature extraction, our files must be in a specific format. Moreover, the re-implementation expects PCAPs as input, grouped by MAC address. Furthermore, the definition of grouped by is that the PCAP only contains packets where either the ethernet source or destination is equal to the device's MAC address. The original analysis requires the PCAPs to be limited to a certain number of packets, either 21,000 or 560,000. In the rest of this section, we will go more in-depth into the description of each feature used.

Number of DNS queries This feature counts the number of DNS queries done by the device. It is important to note that the DNS queries do not have to be unique. Hence, N DNS queries for sidnlabs.nl will count as N, not as 1.

Number of unique protocols The feature counts the number of uniquely used protocols. We only register one protocol per packet, which is the protocol of the highest layer identified by TShark; for example, we take the HTTP protocol for a web request and not TCP.

Device type from the User-Agent field The User-Agent field is only present for devices that perform HTTP requests over an unencrypted channel. However, a device can use different User-Agents for the HTTP requests it performs. Hence, we select the first User-Agent we encounter. Furthermore, it is also possible that the PCAP does not contain any User-Agent. In this case, we return an empty string.

Number of communications in the local network The paper does not describe what "communication" means in their setting. Hence, we assumed the definition. We defined "communication" as a bi-directional flow. Moreover, the feature counts the number of flows that contain an IP (v4 or v6) source and destination address in the private range. For both IPv4 and IPv6, we use the standard as defined by the Internet Assigned Numbers Authority (IANA) [26] [27].

Connectivity strength of the devices The connectivity strength is measured using the Google page rank algorithm, as explained in Section 2.1. The page rank calculations require a network graph, and the computation relies on the graph's edges. The graph created contains all devices that communicated with a device under analysis. Furthermore, the edges indicate a connection between two devices in the internal network.

Number of domain names with which the device has a TCP connection This feature counts the number of unique domain names to which the device made a TCP connection. Moreover, it is essential to reconstruct which domain names used which IP addresses. To get this list, we can go through the traffic of a single device and combine the DNS request and response. The requests will give us the

[1] https://gitlab.sidnlabs.nl/device-type-classification/melnik

domain names, while the responses will give us the IP addresses. After this map got created, we can go over all TCP connections and compare the source and destination addresses against the map of domain names.

Number of unknown communications Unknown communications are network flows from a device under analysis towards internal devices that are not under analysis. The construction of the feature is the same as the number of communications in the local network. However, we add an extra filter that removes all devices under analysis.

Average session time The average time a network flow is active, measured in seconds.

Average session volume This feature measures the average volume of a bi-directional flow in bytes. We add the incoming and outgoing bytes to get the volume of the bidirectional flow.

Number of communications This feature counts the number of bidirectional flows of a device. The device's IP address should either match the source or destination address of the flow.

4.2.2. Performance

As mentioned before, the performance of our implementation is comparable to that of the paper. Furthermore, it is essential to note that the RFC contains a random state, influencing the accuracy slightly. Therefore, we ran the implementation 50 times with a random chosen state from 0 to 100,000 to better understand the performance. The random implementation of NumPy was used, with a seed of 42. As a result, the RFC on 21,000 packets reached an accuracy of 97%, with an average of 91% and a low of 84%. Furthermore, for the 560,000 packet configuration, we reached an accuracy of 94%, with an average of 86% and a low of 74%. Additionally, we plotted the accuracy spread for both configurations using a box plot in Figure 4.2.

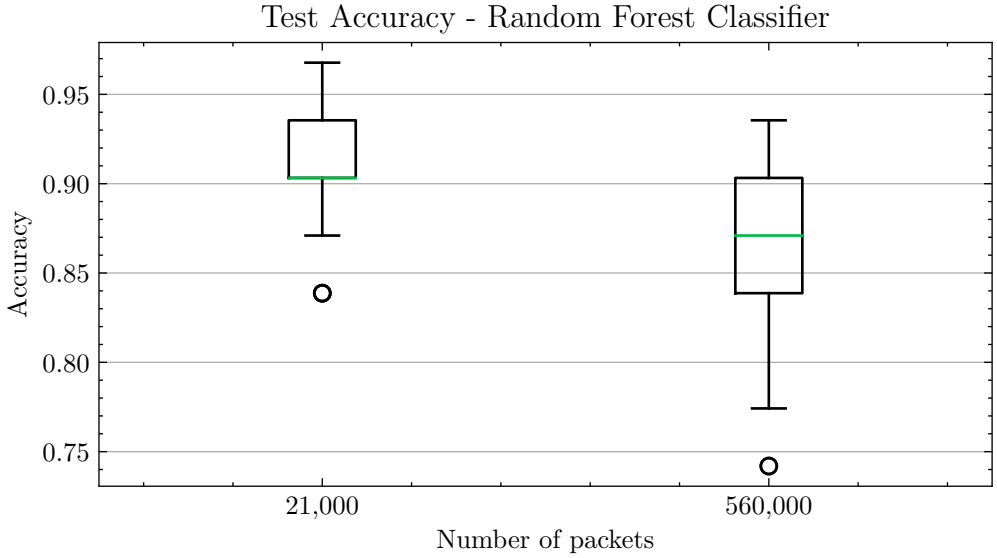


Figure 4.2: The test accuracy spread of the RFC proposed by Melnyk, Haleta, and Golphamid.

The paper briefly mentions the accuracies of other traditional machine learning models, such as K-Nearest Neighbors and a Decision Tree. However, it does not provide parameters for these models or how the accuracy is computed for the cluster algorithms. Furthermore, we could not reproduce the mentioned accuracies with the default parameters provided by the scikit-learn library. Table 4.1 and Table 4.2 summarizes the lowest, highest and average accuracy of these models. We assume that the label most common in a cluster represents the label for the clustering methods, such as DBScan and Local Outlier Factor.

Model	Low	High	Average
Random Forest Classifier	84%	97%	91%
Decision Tree	42%	65%	53%
K-Nearest Neighbors	65%	65%	65%
Local Outlier Factor	71%	71%	71%
Gaussian NB	84%	84%	84%
K-Means	68%	74%	70%
Support Vector Machine	48%	68%	56%
DBSCAN	71%	71%	71%

Table 4.1: The accuracy of the RFC proposed by Melnyk, Haleta, and Golphamid based on 21,000 packets.

Model	Low	High	Average
Random Forest Classifier	74%	94%	86%
Decision Tree	58%	77%	64%
K-Nearest Neighbors	52%	52%	52%
Local Outlier Factor	71%	71%	71%
Gaussian NB	77%	77%	77%
K-Means	55%	58%	56%
Support Vector Machine	42%	81%	66%
DBSCAN	71%	71%	71%

Table 4.2: The accuracy of the RFC proposed by Melnyk, Haleta, and Golphamid based on 560,000 packets.

Storage
The storage requirements of both the 21,000 and the 560,000 models are minimal. Furthermore, the 21,000 packet based model is 155,816 bytes, while the 560,000 packet based model is 159,706 bytes, both around 160 kilobytes. These are well within the resource requirements of low-end hardware.

Memory
The memory usage of the 21,000 and 560,000 packet models is around 95 MiB. The memory usage increases rapidly at the beginning when the model and libraries are loaded. Furthermore, after the model and libraries are loaded, the memory usage stays stable. Figure 4.3 gives a detailed overview of the memory usage of the two models over time.

Average Prediction Times
Figure 4.4 shows the average prediction times of the RFC models. Both models show very similar prediction times. However, the 560,000 packet model shows a narrower spread but a higher median than the model based on 21,000 packets. Overall, the differences between the two models are minimal.

The analysis shows that the difference between the 21,000 and 560,000 packets model is minimal. Hence, the focus will be on the model based on 21,000 packets for the rest of our research. Moreover, the model shows slightly higher accuracies in the spread and lower median times for the runtime.

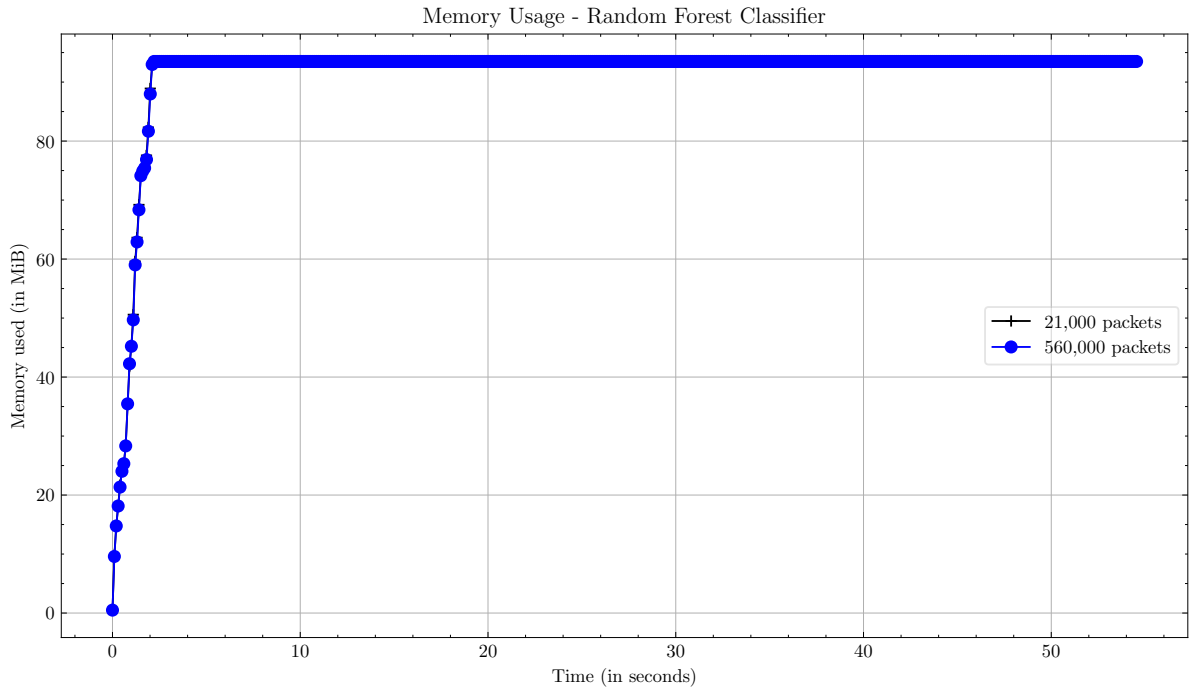


Figure 4.3: The memory usage of the two RFC models by Melnyk, Haleta, and Golphamid.

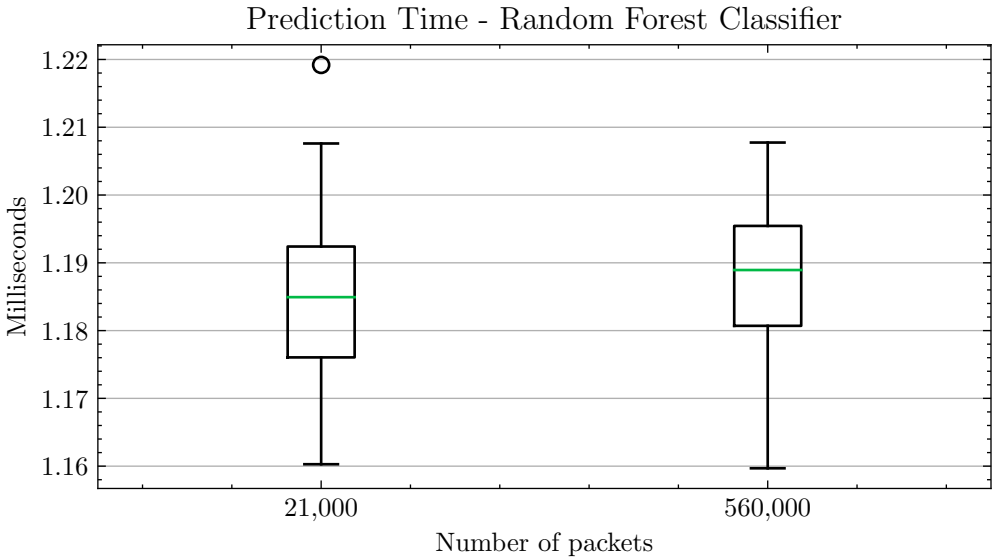


Figure 4.4: The average prediction time of the two RFC models by Melnyk, Haleta, and Golphamid.

4.3. LSTM-CNN Model

The original author of the LSTM-CNN model provided us with a base implementation of the model. Unfortunately, the author could only provide us with the code of the machine learning model. Moreover, the paper uses a single dataset to implement the model. The dataset by Sivanathan et al. [56] provides both the training and test data. It contains 60 days of network traffic, with 20 days highlighted on the main page. However, the paper only uses 19 days of data but does not describe the dates used. Hence, we decided to take the first 19 days provided on the webpage. Moreover, the number of packets in those 19 days is equivalent to the numbers presented in the paper. Therefore, the dataset range used is from the 23rd of September 2016 until the 11th of October 2016.

The paper describes that the six most discriminating features were selected. Unfortunately, it does not

provide details on the traffic direction used. We tried taking both bidirectional packets as only packets sent from the IoT devices (unidirectional). Both approaches did not yield the accuracy described in the paper. The following section will go further in-depth into the feature extraction process. At last, we will cover the performance of the model.

4.3.1. Feature Extraction

The feature extraction process requires a specific format; the re-implementation expects PCAPs grouped by MAC address, the same as the Random Forest Classifier. From the grouped PCAPs, we extract flows of a max length of 300 seconds between the first and last packet. We can use the timestamps of the packets within the PCAP to calculate the split points. After the flows got created, we can extract the features using PyShark PyShark [16]. The paper describes two types of categories for feature extraction, namely: user packets and control packets. User packets include user data and device-service communication, while control packets mainly support functional protocol packets, such as setting up the network or resolving domains [5]. Example protocols for user packets are TCP, UDP, and HTTP. In contrast, control packet protocols include NTP, DNS, ARP, and ICMP.

While extracting the features, we encountered several misclassified protocols by Wireshark. To prevent misclassification, we were required to disable multiple protocols. The disabled protocols used a port-based heuristic; for example, the heuristic of the World of Warcraft protocol only looks if the TCP port is 3724. The profile created for our analysis is shared via GitLab[2].

We extracted three features for both the user packets and the control packets: (1) the number of packets, (2) the maximum size of the packets, and (3) the average size of a packet. We extract these features for every flow. It is crucial to keep the flows sorted by time, as we need to combine six flows to create the final feature. The paper describes an overlap of three between every flow, meaning that we have a sliding window of size six with a step size of three flows. However, the paper does not explain how the features are normalized. Hence, we ran a few tests with the implementations available within the sci-kit learn library. Moreover, we decided to use Z-Score normalization, known as StandardScalar in the scikit-learn library, based on the experiment's outcome. Equation 4.1 gives the formula of the normalization, where $\bar{x}$ is the mean of the feature, and σ is the standard deviation of the feature.

$$\hat{x} = \frac{x - \bar{x}}{\sigma} \tag{4.1}$$

The next section will go in-depth into the model's performance. The accuracies described will be based on our two experiments.

4.3.2. Performance

The accuracy of our re-implementation did not match the one described in the paper. Additionally, the traditional machine learning models did not show the same accuracy as the paper. For example, the RFC model showed higher accuracy than the paper claimed. Furthermore, Table 4.3 and Table 4.4 show the accuracies of the traditional machine learning models. It is important to note that the paper does not describe which features the traditional models used for training and testing. Hence, we assume it is the same data as the LSTM-CNN model, including the sliding window, with 36 features. Again, we repeat all experiments of the traditional models 50 times and show the average, the lowest, and highest accuracy. For the models with a random state, we use the same strategy as explained in Section 4.2.2. It is important to note that the LSTM and CNN comparison models did not get evaluated, as there was no explanation of their configuration.

The LSTM-CNN model experiments are repeated 50 times, with an epoch count of 2,000. The paper does not mention any epoch count, so the variable declared in the code was used. Figure 4.5a displays the correlation between the train and the test accuracy for both the bidirectional and unidirectional flows.

Furthermore, the bidirectional traffic shows the highest accuracy on the test set. Additionally, Figure 4.5b displays the spread of the test accuracies. The figure shows that the spread of the bidirectional model is narrow with overall higher accuracy. Hence, we will use the bidirectional model for the rest of our experiments as it is closest to the paper.

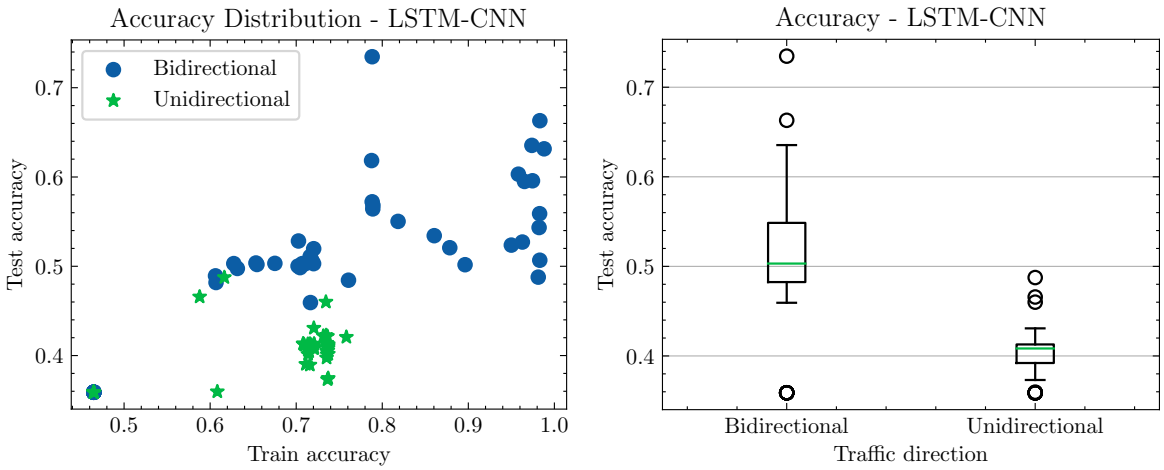
[2]https://gitlab.com/device-type-classification/wireshark-profile

Model	Low	High	Average
Support Vector Machine	37%	50%	43%
Random Forest Classifier	36%	44%	39%
K-Nearest Neighbors	39%	39%	39%
Decision Tree	28%	50%	41%
Adaboost	56%	58%	57%
Linear Discriminant Analysis	41%	41%	41%
Quadratic Discriminant Analysis	50%	50%	50%
Multi-layer Perceptron	26%	66%	43%

Table 4.3: The accuracy of the traditional models on bidirectional traffic.

Model	Low	High	Average
Support Vector Machine	17%	39%	28%
Random Forest Classifier	19%	30%	24%
K-Nearest Neighbors	27%	27%	27%
Decision Tree	16%	24%	18%
Adaboost	31%	42%	36%
Linear Discriminant Analysis	36%	36%	36%
Quadratic Discriminant Analysis	45%	45%	45%
Multi-layer Perceptron	24%	40%	32%

Table 4.4: The accuracy of the traditional models on unidirectional traffic.



(a) The test-train accuracy correlation.

(b) The test accuracy spread.

Figure 4.5: The experiment results of the LSTM-CNN model.

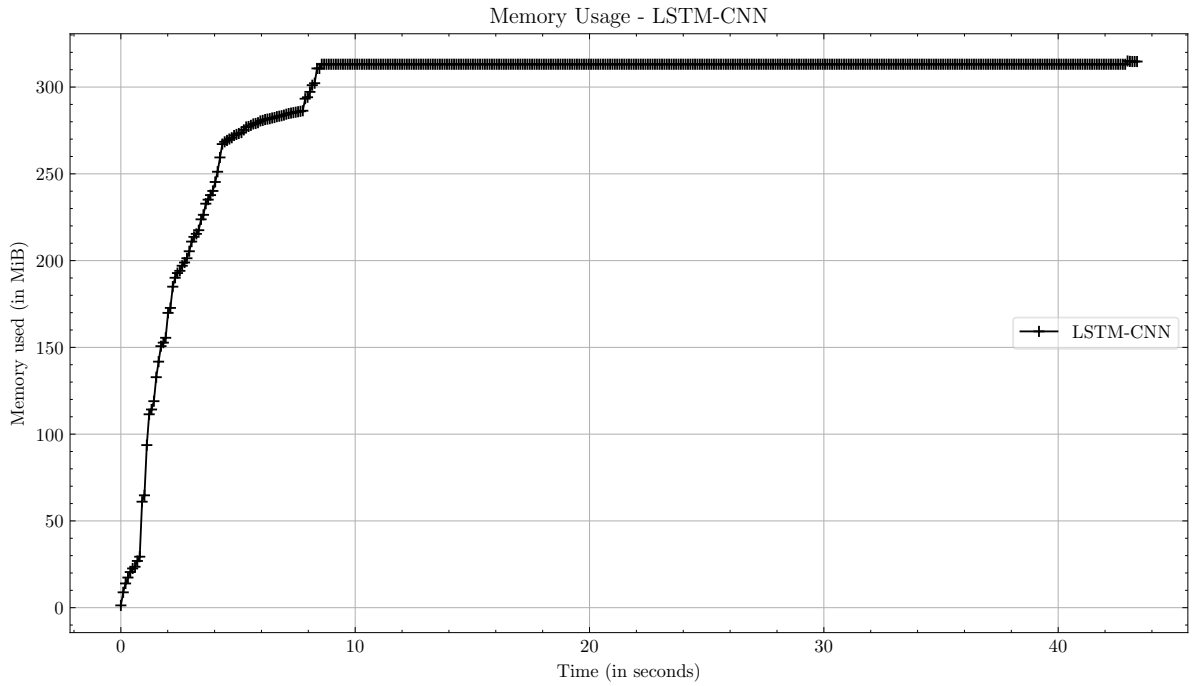


Figure 4.6: The memory usage of the LSTM-CNN model by Bai et al.

Storage
The model requires several files to be loaded, namely an index, metadata, and a data file. The model was 546,634 bytes, which is around half a megabyte.

Memory
Figure 4.6 shows the memory usage of the LSTM-CNN model. The peak usage is around 320 MiB. Overall, the memory usage is stable over the prediction's entire runtime, without outliers.

Average Prediction Times
Figure 4.7 shows that the model has low prediction times. Overall, predictions take less than 0.05 milliseconds. The median is around 0.036 milliseconds, with an overall narrow spread, as displayed by the boxplot.

4.3.3. Technical Difficulties
During the experiments run on the LSTM-CNN model, we encountered several problems with TensorFlow on the Raspberry Pi 4. Unfortunately, new TensorFlow releases do not officially support the Raspberry Pi and only release a lite version of the framework. Hence, we were required to seek unofficial community-supported packages to complete our experiments. Firstly, the unofficial community distribution, provided by Bitsy AI Labs [29], only supports Python 3.7. Hence this is the first requirement for the LSTM-CNN implementation. We tested the unofficial community distribution on newer Python versions without success. Hence, we used Python 3.7.13 for our LSTM-CNN tests.

4.4. Conclusion
The re-implementations will set the baseline for comparison for our model that we will cover in the next chapter. We see that the memory usage of the LSTM-CNN model is significantly higher than the RFC model. A possible explanation is that the LSTM-CNN model uses more complex libraries than the RFC model. However, the average prediction time decreases with higher memory usage. The prediction time is 100 times shorter, while the memory is less than five times the size. However, because we aim to deploy on low-end hardware, the system's memory usage is an essential factor, likely more important than the prediction times.

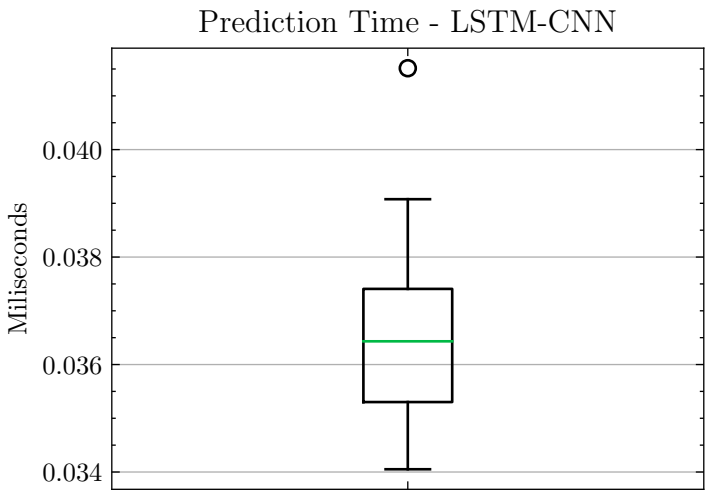


Figure 4.7: The average prediction time of the LSTM-CNN model by Bai et al.

Moreover, the prediction times of the RFC are still around a millisecond, which would be fast enough for our approach. Our system, in the end, only needs to identify a limited set of devices in a home network. Hence, millisecond prediction times are more than sufficient. However, when the device has sufficient memory that enables the deployment of the LSTM-CNN model, it would speed up prediction times significantly.

5

Design & Implementation

Our work describes the construction of a new model to classify unseen IoT devices into five categories: audio, TV, automation, hub, and camera. To achieve this, we introduce several features that describe the characteristics of these devices.

Firstly, we will discuss the challenges in detecting unseen IoT devices. Secondly, we go in-depth into how we overcome these challenges. Thirdly, we explain the datasets we use to train and test our model. Next, we go over the extracted features and how we classify a device. Lastly, we explain how we evaluate the performance of our model.

The implementation of our work is open-source and is available on GitLab[1].

5.1. The Challenges

Every device vendor can implement or buy the firmware with the features and behavior they see fit for their ideal device. Unfortunately, this means that there is no standard for the behavior of devices within the same category. The different behavior can influence the network profile of such a device. However, while the devices can have additional features and might communicate in different patterns, they are designed to perform similar primary tasks. For example, a camera should transmit more data than it receives because it streams a video or snapshot towards a device or cloud. We can design a hypothesis for general behavior per device category based on such observations. The hypotheses will set a basis for our feature selection and guide us on what is important to model. We expect that the primary task outweighs the secondary tasks introduced by the device vendor, causing similarities between the devices even when the implementations are different.

Another challenge is that the devices in our datasets do not always simulate a real-world scenario. With devices in offices and home networks, we expect that there are some behavioral patterns. For example, someone might watch TV after dinner on a workday within a home network, causing a certain traffic pattern. Out of the three datasets used, we only have a single dataset that actively automated the use of the devices to ensure some usage pattern. Unfortunately, some behavior could not easily be automated, causing some functionality to be not included in the dataset. Furthermore, the other two datasets have used the devices in an irregular and undocumented manner.

An example is the event logs of our TV did not show any sign of the TV streaming a video or movie. Instead, the event logs only show the usage of volume controls and the voice assistant. Hence, we could expect misclassifications due to devices being used similarly to that of another class. Furthermore, other datasets might not have any event logs, which makes us unable to verify whether a device was adequately simulating that of a household and an active user.

[1] https://gitlab.sidnlabs.nl/device-type-classification/rfc-model

5.2. Hypotheses

Next, we will present one or more hypotheses per device type that define the network behavior of the primary tasks of the device category. Furthermore, we needed to adjust the initial hypotheses for some devices. The adjustments were needed because some devices were not fully used, such as limited TV usage, as explained before.

5.2.1. Audio

The audio class contains devices that act like smart speakers, for example, a Google Home Mini. A smart speaker is a speaker with a built-in virtual assistant. We expect the data of a smart speaker to show peaks when a command is requested because most devices still transmit the voice query to the cloud, even if it is processed locally. Moreover, the command will trigger the transmission of an audio stream to the cloud. The cloud then interprets the speech and constructs a reply. This reply is then sent back to the device and played on the speaker. However, we expect a clear indication when a user uses voice command, as audio consumes more bandwidth than idle usage. Figure 5.1 shows an example of an audio device. We expect that the user sent a voice command to the smart speaker during the peaks, but we could not verify this.

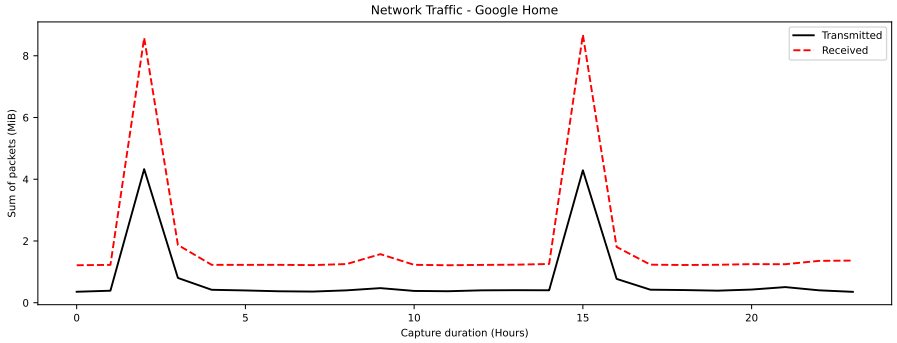


Figure 5.1: 24 hours of IP traffic from a Google Home smart speaker grouped by hour.

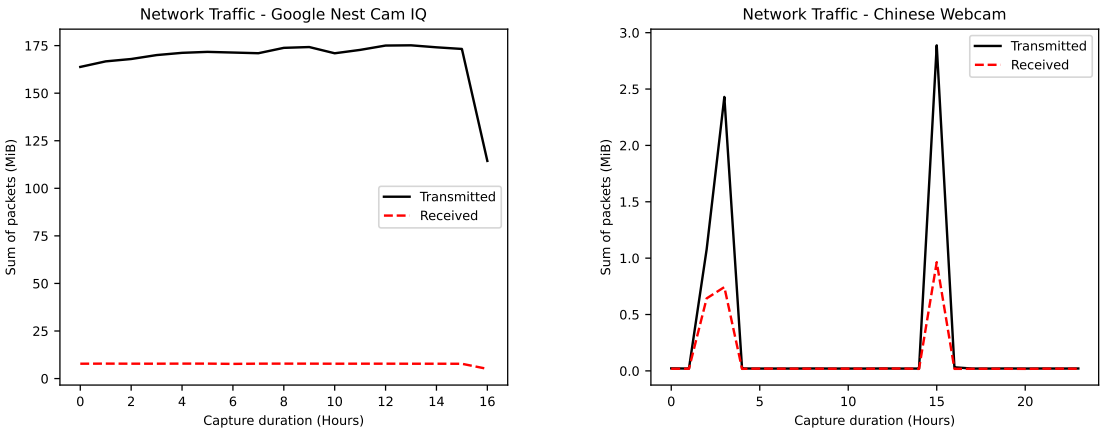
5.2.2. Camera

The camera class contains all internet-connected video doorbells and general IP cameras. We combined both devices into one category because they capture video and generally share it with their user. Hence, because their primary task is similar, we included them in the same category.

As mentioned before, we expect more outgoing traffic than income traffic. The camera most likely streams snapshots of movement or the complete video to a device or cloud. The camera may have an onboard memory chip to store video content locally, which can cause the device not to transmit as much as other cameras. However, if the user views the stream, the same principles apply. The video would be streamed from the local memory chip towards the users' device, causing snapshots or video to be transmitted again. To illustrate the idea behind the hypothesis, we show two examples. Figure 5.2a shows an example of a device we expect to have constant streaming towards the cloud due to the continuous high bandwidth. The cause of the sudden drop at the 16th hour is due to incomplete data. Figure 5.2b shows a device we suspect to have uploads during motion detection, as we see clear peaks in the traffic with an overall steady low bandwidth.

5.2.3. TV

The TV class consists of televisions and set-top boxes, like an Apple TV. These devices are popular for watching content on streaming services, such as Netflix or Amazon Prime. The video streams result in a packet stream of a large size. Furthermore, the received streams result in the TV having significantly more incoming bytes than outgoing bytes. In general, the idea is that the TV shows the opposite behavior as that of a camera. Unfortunately, we could not detect day-to-day usage of the TVs in our dataset, as the traffic was relatively low across all datasets. After inspecting the traffic and datasets where possible, we saw that only a few events happened: idle, changing power state, changing volume,



(a) Google Nest Cam IQ showing regular high traffic.

(b) Chinese webcam showing irregular high traffic.

Figure 5.2: Two examples showing the hypothesis for camera detection.

and using voice commands.

Furthermore, we expect the device to show some baseline behavior. The baseline will be related to the general TV operations and events. For example, the TV refreshes the recommendation page. We expect a constant stream of traffic without interruptions, which is relatively low, but the received bandwidth is higher than the transmitted. Figure 5.3 shows an example of the traffic of a TV. Overall, you see stable traffic with a peak in the middle.

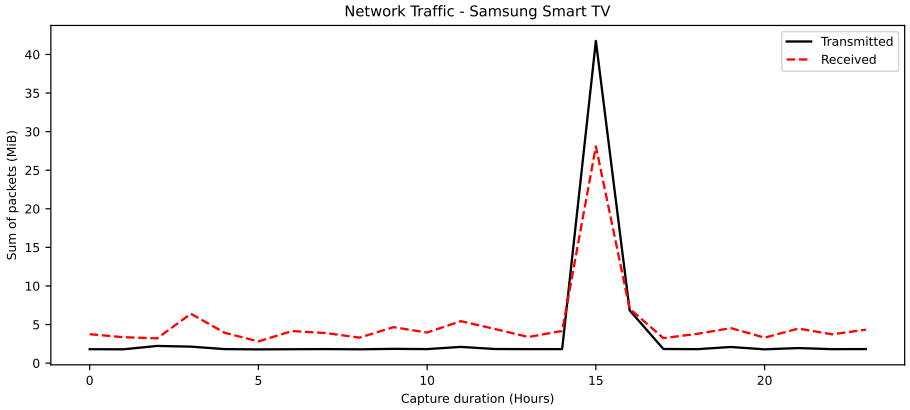


Figure 5.3: 24 hours of IP traffic from a Samsung Smart TV grouped by hour.

5.2.4. Hub

The hub is a device that controls and communicates with several devices on the internal network. The hub either pulls data from devices, or the devices push data towards the hub. Hubs are generally seen as the center point of IoT, controlling and managing data. Moreover, if the hub uses WiFi, we expect a hub to have more connections to internal IP addresses than other devices. Additionally, because the dataset contains multiple hubs and does not control all devices, we cannot only rely on this hypothesis. Figure 5.4 displays the number of internal connections for various devices over an hour. The figure shows that some devices have a similar internal connection count to a hub, especially TVs controlled via a smartphone. Additionally, we expect the hubs to have a small footprint on the network, showing irregular traffic, as the devices might react to user behavior. Figure 5.5 shows an example of such a footprint from a Hub.

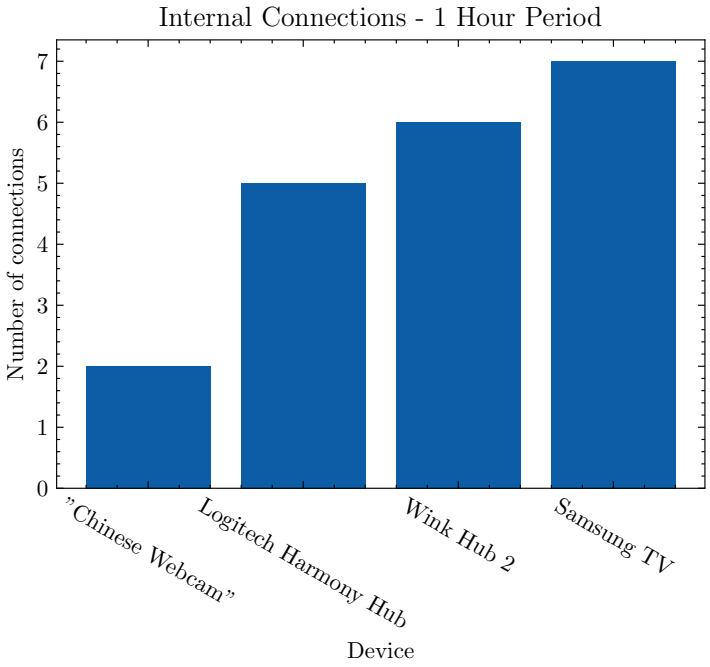


Figure 5.4: Number of internal connections over a period of an hour.

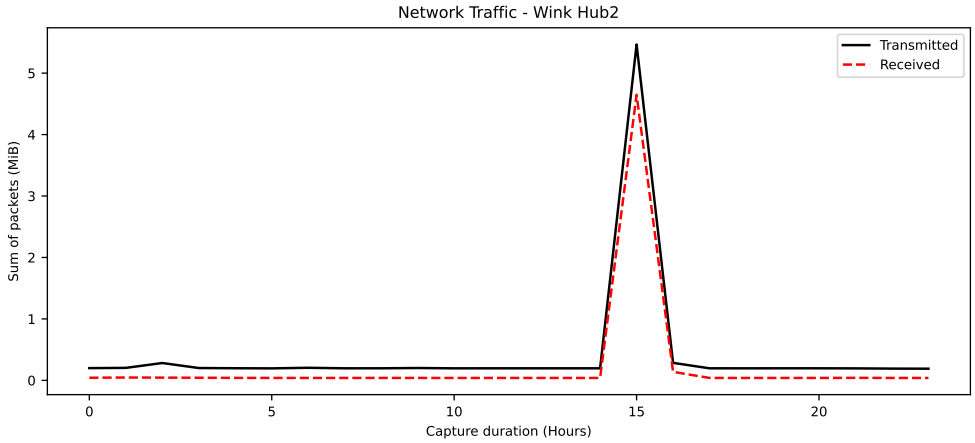


Figure 5.5: 24 hours of IP traffic from a Wink Hub 2 grouped by hour.

5.2.5. Automation

Automation is a category consisting of lights, electrical plugs, and sensors. These devices often have a regular network pattern. The device often pushes data to a hub periodically, or the hub pulls periodically. We hypothesize that these devices show a minimal network footprint, with a similar packet size captured in a reasonably periodic pattern. Figure 5.6 shows an example of a device in the automation category. It shows very minimal traffic with periodic peaks.

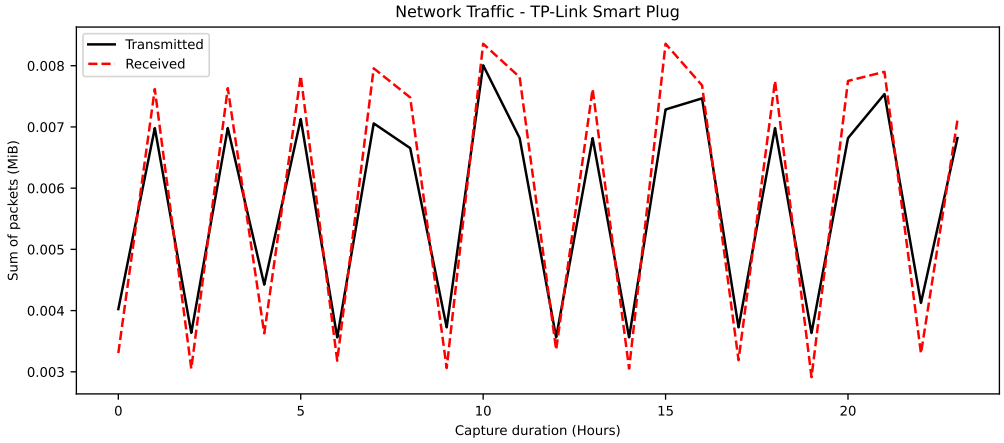


Figure 5.6: 24 hours of IP traffic from a TP-Link Smart Plug grouped by hour.

5.3. Dataset

To train and test our model, we use three datasets: (1) IoT Information Exposure (IMC '19) [2] [46], (2) UNSW IoT Traces [3] [56], and (3) Your Things [4] [44]. In total, we collected 74 devices split into the beforementioned five categories. Appendix B shows an overview of the devices used, the class, and from which dataset we included data. We pre-processed the data to have a single PCAP file per device, grouped by MAC address. Furthermore, the IMC dataset contains two sets, namely 'iot-data' and 'iot-idle.' We concatenated all PCAPs of a single device, identified by device name, location, and whether a VPN was active. The sorting was done based on timestamp, using a utility provided by Wireshark named Mergecap [59]. Finally, we used DPKT [6] to group the UNSW and Your Things dataset by checking whether the source or destination MAC address was equal to that of an IoT device. We wrote the packet to the devices' PCAP file if it was identical, leaving the timestamp and data intact.

The network captures in our dataset are of various lengths. Hence, to include similar lengths of capture data for every device, we decided to cap all PCAPs to the first 24 hours. We did this to ensure that we have equal amounts of data available for all devices, allowing us to better reason about what kind of capture duration is required to perform device type classification. Additionally, it also limit the number of events observed, as they need to happen within the first 24 hours of the capture. The downside of using only 24 hours is that we might not capture events that do not occur daily. An example of such an event is updating the firmware.

Most IoT devices are present multiple times in our dataset. Additionally, the IMC dataset observed several devices at more than a single location or even with or without a VPN. Because we wanted to avoid picking data by hand, which could introduce bias, we decided to include the first 24 hours of all device versions, which causes devices to have multiple captures of the first 24 hours. However, they are still handled as a single device to ensure they are kept unseen.

To finalize our dataset, we can extract sub-captures from the grouped PCAPs to be able to generate features. A sub-capture is a chunk of the overall capture limited to a maximum duration δ, a parameter we will explore later. A sub-capture is then formally defined as follows: Let $P = \{P_1, ..., P_N\}$ be a set

[2] https://moniotrlab.ccis.neu.edu/imc19/
[3] https://iotanalytics.unsw.edu.au/iottraces.html
[4] https://yourthings.info/data/

of packets of length N from a specific IoT device. Let $TIME$ be a function that extracts the timestamp for a packet $p \in P$. A sub-capture is then defined as a set $C = \{P_i, ..., P_j\}$ where $N \geq j \geq i$ such that $TIME(P_j) - TIME(P_i) <= \delta$ and $|C|$ is maximized. We extract as many sub-captures as possible, starting at $i = 1$, the first packet of a device. It is important to note that no sub-captures should have overlapping packets.

Next, we will investigate the features that we extract from these sub-captures.

5.4. Features

This section will go in-depth into our selected features and the parameters we will explore during our experiments. The basis for the feature selection is to pick the features that best describe the hypotheses. We have chosen to use hypotheses for our feature selection due to the scientific basis it gives us. It also makes the model more explainable, as we know why we have chosen those features. The idea for hypotheses based selection came from the early stages of our design. During this phase, we opted to include as many features as possible. However, this yielded no significant results and made the model relatively slow and hard to deploy on low-end hardware. Additionally, we focused on only including features that do not use significant computational power to extract.

The features selected for our design focus on the packet itself and the data within it. We use the packet data to extract IP layer information and the protocol. The list of features extracted is as follows:

The transmitted packet features: We extract the maximum, minimum, mean, and median size of a packet transmitted by the IoT device within the sub-capture. We included this feature because several hypotheses include transmitted packet statistics.

The received packet features: We extract the maximum, minimum, mean, and median size of a packet received by the IoT device within the sub-capture. We included this feature for the same reason as the transmitted packet statistics.

Internal Connections: The number of unique internal IP addresses the IoT device communicated with within the sub-capture. The feature is related to the hypotheses set for the Hub category. As mentioned before, we expect the Hub to have more internal connections than other categories.

Connection count: The number of unique IP addresses the IoT device communicated with within the sub-capture. We added this feature with a similar goal as the internal connections. It is mainly linked to the Hub and Automation class, as we expect these to communicate more on the internal network. Hence, we included the feature to see whether this is the case.

Protocol length: The number of unique protocols the IoT device used within the sub-capture. We use the protocol detection mechanisms from Wireshark [13]. The paper by Melnyk, Haleta, and Golphamid [35] showed that it was one of the more important features in differentiating between single and multi-purpose devices. Hence, because we have some multi-purpose categories, we also included this feature. This feature is not directly linked to a hypothesis but based on a more overall observation that combines all hypotheses.

Internal Packets: The number of packets with a source and destination address within the private range. Additionally, either the source or destination needs to match the IoT device's IP address. As mentioned before, we expect the Hub to communicate with internal devices more than others. Hence, we included this feature.

Packet count: The number of packets the IoT device sent or received within the sub-capture. The feature shows us how active a device is. We expect certain classes to only communicate sensor information, for example, the automation class.

The interpacket time features: We extract the maximum, minimum, mean, and median interpacket time. The interpacket time is the difference between the timestamp of packet P_i and P_{i-1} with $i > 1$. We extract the feature to see if the values are similar. The maximum, minimum, mean, and median should all be reasonably close for a periodic device.

The internal packet ratio: The internal packet ratio is the ratio of the internal and total packet counts within a sub-capture. Equation 5.1 shows the formula used for the computation. The feature's goal is to show us whether the device mostly communicates with external sources or internal sources. As mentioned before, we expect the Hub and Automation category to communicate more on the internal network.

$$ratio = \frac{internal\_packets}{total\_packets} \tag{5.1}$$

In total, our feature vector contains 18 features extracted per sub-capture. Features get extracted from every device in a dataset.

5.5. Evaluation

In this work, we will primarily focus on the accuracy and resource usage of the RFC. The RFC will base its predictions on a single sub-capture as introduced before. The main reason for going with an RFC is the resource usage as shown in Chapter 4 and the ability to remove bias. Additionally, the model is still explainable, and the features are manually selected. Overall, we expect improved accuracy with these properties but still have explainability.

However, we will briefly compare the results of other traditional machine learning models. The machine learning models we will show the accuracy of are:

- Decision Tree
- Adaboost
- K-Nearest Neighbors
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis
- Multi-Layer Perceptron
- Support Vector Machine

We selected these models as they match the ones proposed in the comparison of the LSTM-CNN model proposed by Bai et al. [5]. The purpose is to see whether our feature extraction methodology reaches an overall higher accuracy on simpler models. In addition, we want to see whether deploying our approach to a more or less explainable model might lead to significant performance differences. For example, a decision tree classifier follows a single path, leading to a more explainable model than an RFC with significantly more paths to evaluate. However, the RFC is still more explainable than the Multi-Layer Perceptron, which deploys multiple layers of neurons. The settings used will be the same as explained in the paper by Bai et al. [5].

For every device in our datasets, we have multiple sub-captures, and from each, we get a prediction. Furthermore, in the end, we want a single prediction that tells us which class the device has. For this, we use a majority vote on the prediction of the sub-captures. If two or more classes have the same number of predictions, we pick the first observed class.

5.6. Parameters

The RFC contains several parameters that control the behavior of the construction algorithm. The parameters we are exploring in this thesis are the influence of the random state on our accuracy and the number of estimators (trees). Furthermore, our dataset construction contains a parameter delta δ that controls the sub-capture' length. Exploring the random state is performed by repeating our experiments 50 times with a different random state, using the same procedure as described in Chapter 4. We believe

that repeating the experiments 50 times gives us good insights into the model's behavior while also keeping the running time manageable.

For the number of estimators, we use 50, 100, and 250. Furthermore, for the sub-capture length δ, we use 600, 900, 1800, 3600, 7200, 14400, and 21600 (seconds). Finally, we run all possible combinations between the number of estimators and the length of the sub-capture. To summarize, we run 21 configurations and repeat these 50 times. We believe that exploring these configurations will give us insights into the best possible sub-capture length and model size. All other parameters use the default value as defined by the scikit-learn [52] library (version 1.0.2). Appendix C summarizes the default settings.

5.6.1. Dataset Split

In this work, we use a supervised learning approach. In general terms, we require a separate training and testing set. The training dataset trains the supervised model, while the test set validates the performance of our model. A common approach is to randomly take a certain percentage of the rows for training and the rest for testing, for example, 75 percent for training and 25 percent for testing. The downside of this approach is that we would mix the data and cannot guarantee that we would have unseen devices in our set. To solve this, we would be able to take a certain percentage of our devices for testing and training, but it would result in some of the device categories having almost no devices left over for testing. Furthermore, if we would randomly select devices, it could be that the devices in our train set would not correctly explain the devices in our test set. For example, it could be that the devices we would pick for our train set do not fully explain the characteristics of the category. Additionally, it is important not to manually craft a dataset that would gain the highest accuracy, as in theory, it would be ”cheating.”

To solve both issues, we use the cross-validation technique LOGO. LOGO allows us to test our accuracy on every device in our dataset, giving us enough samples to test the stability and accuracy of our model. Furthermore, because we constantly train on all devices not being tested, we can argue that we have the best possible split for explaining the characteristics of a specific category.

5.6.2. Metrics

To evaluate our model, we will use the same metrics as in Chapter 4. However, because we use the LOGO cross-validation technique, we have for every device a different model. Furthermore, the prediction time can be performed as usual, as we measure the time the prediction takes on the model for that specific device. Unfortunately, the disk space analysis is more complex as we have 74 models. We will show the spread of the disk size in a boxplot to investigate the storage requirements. We will use two metrics for the memory consumption, namely the maximum and minimum memory usage.

Additionally, we will explore the predictions made by our model. Furthermore, we investigate the prediction strength by looking into the number of correctly predicted sub-captures per device. Finally, we investigate whether prediction accuracy might be influenced by having different device versions in the dataset. While these are different devices, they might come with very similar firmware. Additionally, it could be that some of our devices use the same firmware, as these are often readily sold.

6

Experiment Results

In this chapter, we will explore the performance of our model. First, we go over the accuracy of the RFC when using different parameters. Then, based on the results of the RFC, we determine the best possible sub-capture length. Finally, from the results of our experiments, we can determine which RFC gained the highest accuracy.

The model that reached the highest accuracy gets evaluated further, by exploring the predictions made. We do this by going in-depth into the misclassifications, and discuss devices with a high number of sub-captures predicted correctly. Next, the best-performing RFC model gets compared against several traditional machine learning models.

At last, we measure the resource usage of the best-performing RFC model.

For the rest of the chapter, we will use a naming scheme to indicate which model we are currently discussing. The format is the sub-capture length followed by the number of trees in the RFC. For example, 4h-250 is the RFC model based on a 4-hour sub-capture length and 250 estimator trees.

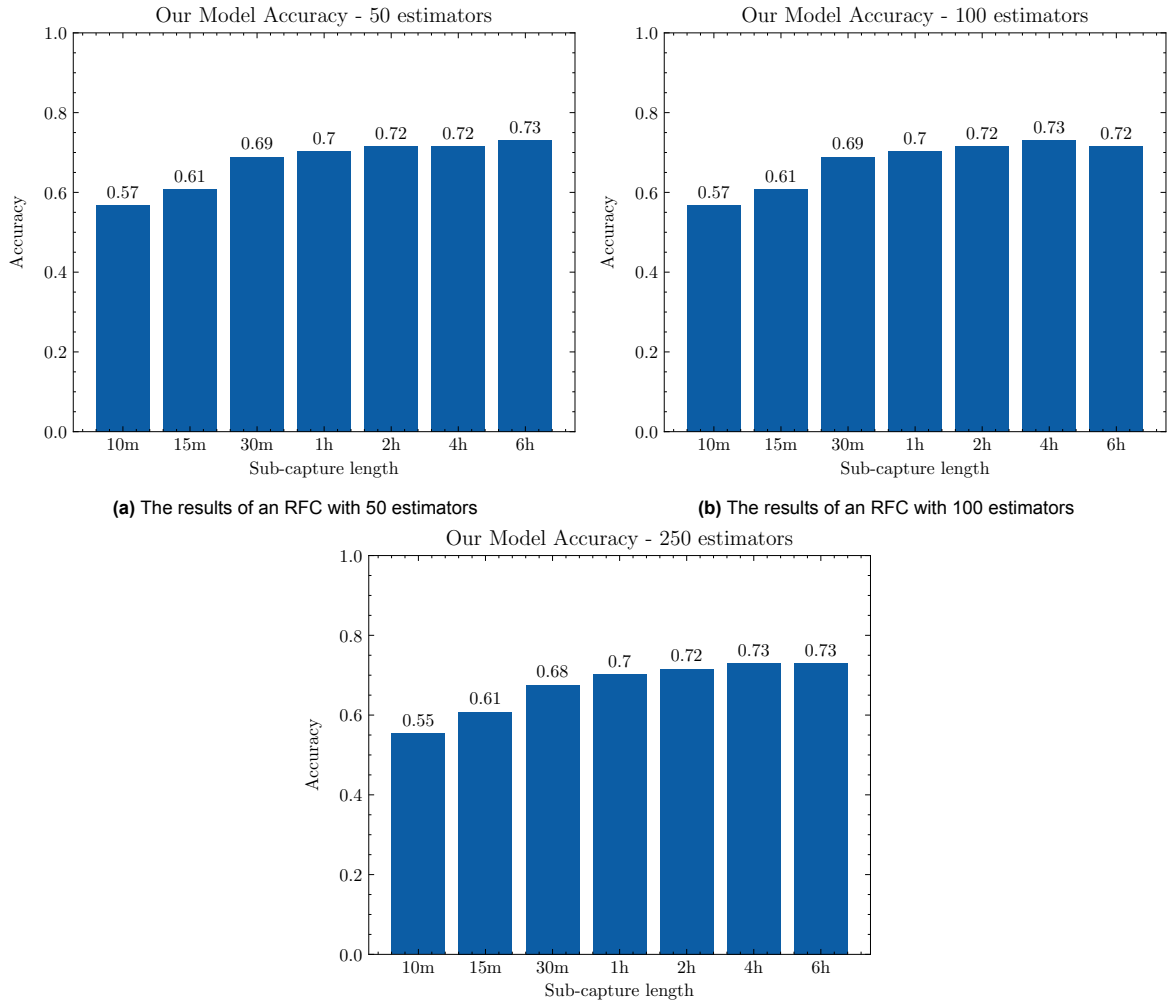
6.1. Accuracy

This section explores which parameters reach the highest accuracy for our model design. We will look at the influence of the sub-capture length and the number of estimators. We do this by looking at the highest accuracy reached in the 50 runs we performed per configuration. Figure 6.1a, Figure 6.1b, and Figure 6.1c show the results of the experiment.

From the figures, it is clear that the length of the sub capture influences the accuracy. The accuracy increases steadily up until the 2-hour mark. Afterward, it stays stable. All three estimator experiments reached the highest accuracy of 73%, meaning that 54 out of 74 devices were classified correctly using the LOGO cross-validation technique. The number of estimators does not influence the accuracy significantly, as the differences are only a few percent.

Sub-capture Length	Estimators	Mean Accuracy
6h	50	66%
4h	100	69%
4h	**250**	**70%**
6h	250	68%

Table 6.1: The mean accuracy of the RFC models with a highest accuracy of 73%.



(a) The results of an RFC with 50 estimators

(b) The results of an RFC with 100 estimators

(c) The results of an RFC with 250 estimators

Figure 6.1: The experimental results of the RFC with different number of estimators and sub-capture lengths.

For the rest of our experiments, we focus on a single configuration. Furthermore, we only take the models with the highest accuracy into account and select the model with the highest mean accuracy. Table 6.1 shows the mean accuracy of the configurations that reached 73%. We use the measurement of mean accuracy to determine the most stable model. The higher mean accuracy indicated that the random state has less influence. Hence, we will focus on the 4h-250 configuration for the rest of our analysis, as it is the model with the highest mean accuracy, and therefore likely to be the most stable model.

6.2. Accuracy Spread

The random element in the RFC model influences the prediction accuracy. Figure 6.2 shows the spread of the accuracy reached by our 4h-250 model over 50 runs. The boxplot shows that the median of our 4h-250 model is around 71%, marginally lower than the highest point reached. Additionally, the interquartile range has a total spread of about 2%. The difference between the minimum and maximum accuracy reached is around 7%. Overall, the accuracy spread is rather narrow, displaying that our 4h-250 models' accuracy is stable.

6.3. Predictions

The model has the highest accuracy of 73% but still misclassifies some devices. In this section, we will explore misclassified and correctly classified devices. We do this by looking at the number of sub-

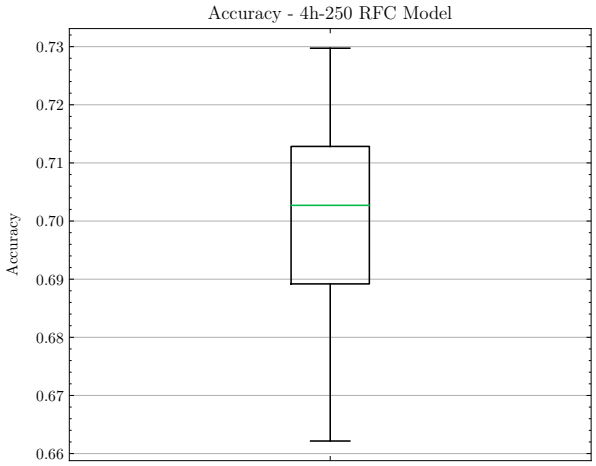


Figure 6.2: The test accuracy spread of our 4h-250 RFC model.

captures predicted correctly and the predicted categories for the misclassified sub-captures.

6.3.1. Misclassifications

The misclassifications differ depending on the random seed used. However, the number of correct predictions per class is equal over all three runs. Figure 6.3 displays the actual and predicted labels in a confusion matrix. The three results show that all TVs were classified correctly in all three runs. In addition, we can see that most of the cameras were classified correctly. The camera category is the largest in our dataset, containing almost half of the available devices.

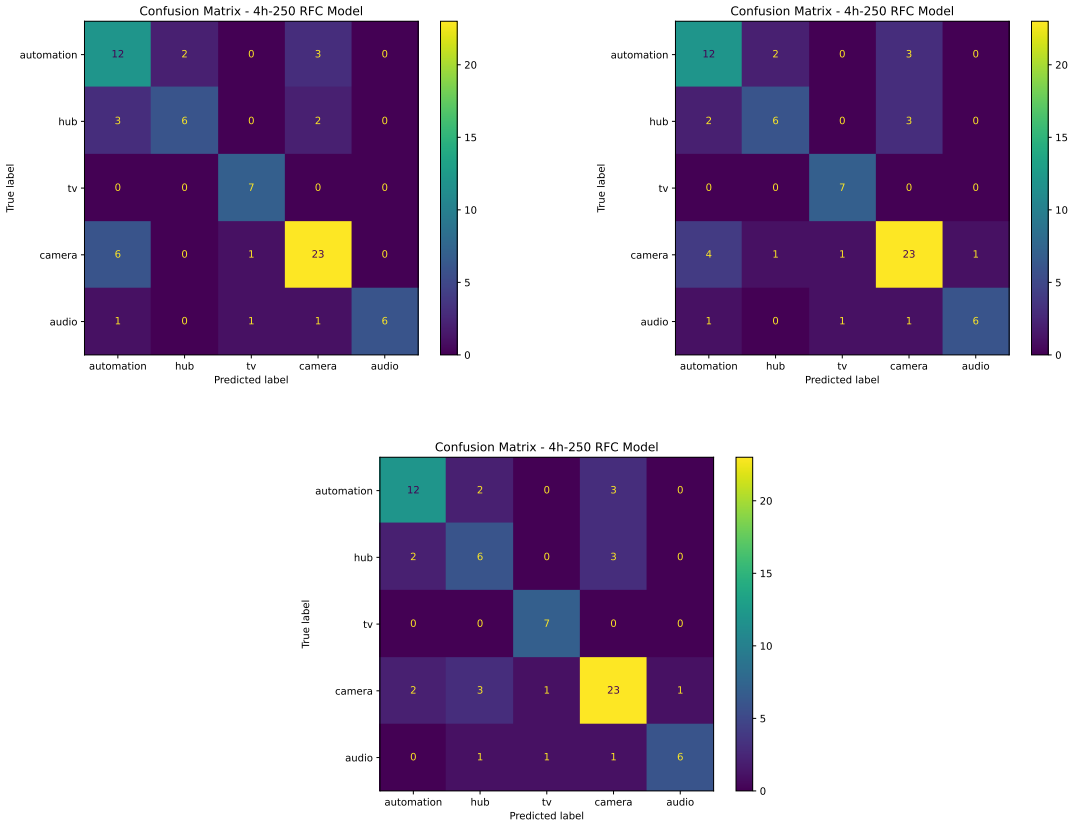


Figure 6.3: The confusion matrices with various seeds of our 4h-250 RFC model that reached 73% accuracy.

Most devices get misclassified as automation, hub, or camera. Depending on the run, the largest misclassification category changes. Additionally, the number of devices misclassified as a TV stays stable over all three runs. The hub and audio classes have the most misclassifications, with almost 50% of the devices not being predicted correctly.

Because we use a majority vote, it is essential to look at the distribution of the predictions of a device. In Table 6.2, we have highlighted the misclassified devices. The table includes the name of the device, the truth label, and the predictions made per class. The numbers on the right side of the table indicate the actual predictions made, with the number being the number of sub-captures classified under that class for that device.

Furthermore, we picked the first run of the 50 that reached an accuracy of 73% for this analysis. Additionally, we added the distribution for all devices of this run in Appendix D. The table shows us that of the 20 devices, only nine had some sub-captures predicted correctly. Furthermore, for the other 11, the misclassification is significant, as they do not have a link back to their original class.

Device	Truth	Automation	Camera	Hub	Audio	TV
Bulb1	Automation	0	7	5	0	0
Charger Camera	Camera	5	3	4	0	0
D-Link Camera	Camera	3	0	3	0	0
Insteon Camera	Camera	6	0	0	0	0
Light Bulbs LiFX Smart Bulb	Automation	0	9	3	0	0
Lightify Hub	Hub	20	2	2	0	0
Netamo Welcome	Camera	2	2	1	1	0
Philips Hub	Hub	0	23	0	0	1
Samsung SmartCam	Camera	4	0	0	2	0
TP-Link Day Night Cloud Camera	Camera	6	0	0	0	0
Triby Speaker	Audio	4	0	2	0	0
Wemo Plug	Automation	7	10	0	7	0
Xiaomi Hub	Hub	22	0	2	0	0
Xiaomi Strip	Automation	4	0	8	0	0
Wink Hub	Hub	6	0	0	0	0
Belkin Wemo Link	Automation	0	2	4	0	0
August Doorbell Camera	Camera	0	2	0	0	4
Caseta Wireless Hub	Hub	1	2	1	1	1
Harmon Kardon Invoke	Audio	0	11	0	5	2
Apple Home Pod	Audio	0	0	0	0	6

Table 6.2: The devices misclassified by the RFC model with 250 estimators and a sub-capture length of 4 hours.

6.3.2. Prediction Distribution

While the majority vote resulted in many correct classifications, it is essential to know the distribution of the sub-capture prediction. The question is whether the devices were correctly classified because most of the sub-captures had the correct class or if it came down to a slight difference. Figure 6.4 shows different boundaries of the percentages of correct guesses. To generate the graph, we again used the

model that reached the accuracy of 73% first. The idea behind the graph is to illustrate the number of devices with at least a certain percentage of correct predictions. For example, the left bar shows the number of devices with at least 40% of the sub-captures with the correct class. We then increase the counter by 10% per bar when approaching the right side of the graph.

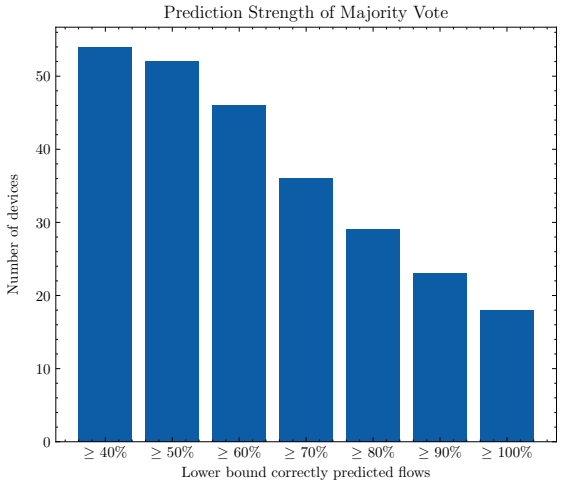


Figure 6.4: The prediction strength of the majority vote.

The graph shows that all devices have at least 40% of the sub-captures correctly predicted. In theory, because we have five classes, the minimum required is at least 20% if the share between the classes is all equal. However, up to 50%, we do not see a significant drop. There are still 52 out of 54 correctly classified devices at this level. After 50%, it quickly goes down at each step until we reach 100%. At this level, there are still 18 devices. Overall, the prediction strength indicates that our 4h-250 model is confident in its predictions per device, as we classify at least half correctly for 52 devices even when using five classes.

A significant portion of our devices have at least 90% of the sub-captures predicted correctly. In Table 6.3, we display the devices and the accuracy obtained. There are several Amazon Echo devices with high prediction accuracy. These devices likely run similar firmware, causing their network profile to be equivalent. The same applies to the Nest Cameras and the Dropcam, as Google's Nest Labs acquired DropCam in 2014. Hence, they likely use similar firmware as the original company.

For the other devices, something similar could apply. Within the IoT market, vendors are delivering readily available IoT solutions [49]. These readily available solutions get sold under different brand names with minimal adjustments. For example, the 'Chinese Webcam,' an unbranded Webcam found in the dataset, has reached an accuracy of 100%. One or more devices likely share the same firmware with this device.

Device	Class	Accuracy
Echodot	Audio	0.97
Echoplus	Audio	0.97
Echospot	Audio	1.0
Google Home	Audio	1.0
Nest Protect Smoke Alarm	Automation	1.0
Netatmo Weather Station	Automation	1.0
Withings Smart Scale	Automation	1.0
Koogeek Light Bulb	Automation	1.0
Yi Camera	Camera	0.92
Nest Camera	Camera	1.0
Nest Dropcam	Camera	1.0
Nest Camera IQ	Camera	1.0
Dropcam	Camera	1.0
Netgear Arlo Camera	Camera	1.0
Logitech Logi Circle	Camera	1.0
Canary	Camera	1.0
PiperNV	Camera	1.0
Chinese Webcam	Camera	1.0
Luohe Spycam	Camera	0.92
Wansview Cam Wired	Camera	1.0
Withings Smart Baby Monitor	Camera	1.0
Appletv	TV	0.93
Roku 4	TV	1.0

Table 6.3: The IoT devices that had a prediction accuracy of more than 90%.

6.4. Traditional Models

This section discusses the performance reached by other traditional machine learning models. Table 6.4 shows the results of the 50 runs performed. Besides the number of neighbors for the K-Nearest-Neighbors model, all model parameters use the default as defined by the scikit-learn library (version 1.0.2). The number of neighbors was set to 10, as described in the paper by Bai et al. [5].

Model	High	Low	Average
Our 4h-250 Model (RFC)	73%	66%	70%
Decision Tree Classifier	59%	45%	54%
Adaboost	45%	45%	45%
K-Nearest Neighbors	59%	59%	59%
Linear Discriminant Analysis	36%	36%	36%
Quadratic Discriminant Analysis	46%	46%	46%
Multi-Layer Perceptron	58%	36%	46%
Support Vector Machine	38%	38%	38%

Table 6.4: The prediction accuracy of other traditional models on our feature set.

Table 6.4 shows that our 4h-250 model reaches the highest accuracy of the traditional models tested. The choice of the model influences the accuracy. The best performing traditional models are the K-Nearest-Neighbors and the Multi-Layer Perceptron. In short, the K-Nearest-Neighbors perform a majority vote on the classes of k of its nearby neighboring data points. The Multi-Layer-Perceptron is a feed-forward neural network with several layers of neurons.

The RFC reaches the highest accuracy of the models that we evaluated. The RFC consists of multiple decision trees to remove bias. The single decision tree reaches an accuracy of 59%, 14% lower than the RFC. Overall, it could indicate that some features introduced bias and that the RFC was able to (partially) remove this.

6.5. Resource Usage

We will explore the same metrics as introduced in Chapter 4, with slight adjustments because of the LOGO methodology used.

6.5.1. Storage

In Figure 6.5, we display the storage size required. All models are between 6 and 7 MiB, making them easily storable on low-end hardware. Next, we will look into the prediction times of our 4h-250 model.

6.5.2. Prediction Times

In Figure 6.6a, we plotted the spread of all predictions made during the classification. As mentioned before, we measured 74 models, one per device. The graph shows several outliers that are above 100ms per prediction. The outliers are from a single model, namely the 'Nest Dropcam.' We filtered out these times to get a better overview of the spread.

Figure 6.6b displays the times without the 'Nest Dropcam' model. The graph shows that the predictions are all below 40ms. The median is around 9ms, while the interquartile spread is around 10ms.

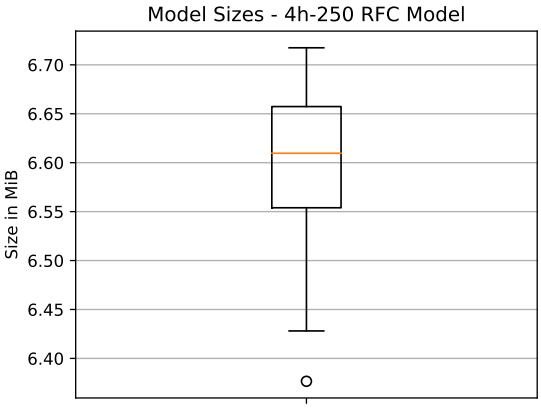
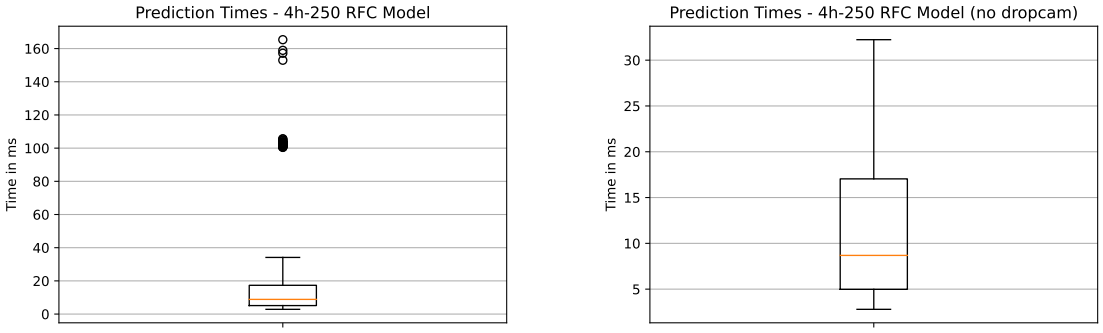


Figure 6.5: The 4h-250 RFC model sizes created during LOGO cross-validation (lower is better).



(a) The results of all devices.

(b) The results without 'Nest Dropcam'.

Figure 6.6: The prediction times of the RFC models created during LOGO cross-validation.

6.5.3. Memory

As mentioned before, we will use the RFC models with the minimum and maximum memory usage. Figure 6.7 shows both these runs. As displayed, the runs are not significantly different in memory usage. Furthermore, both runs climb to 100 MiB and stay around this value for the rest of the evaluation. The sudden increase in memory is caused by loading the model and the data.

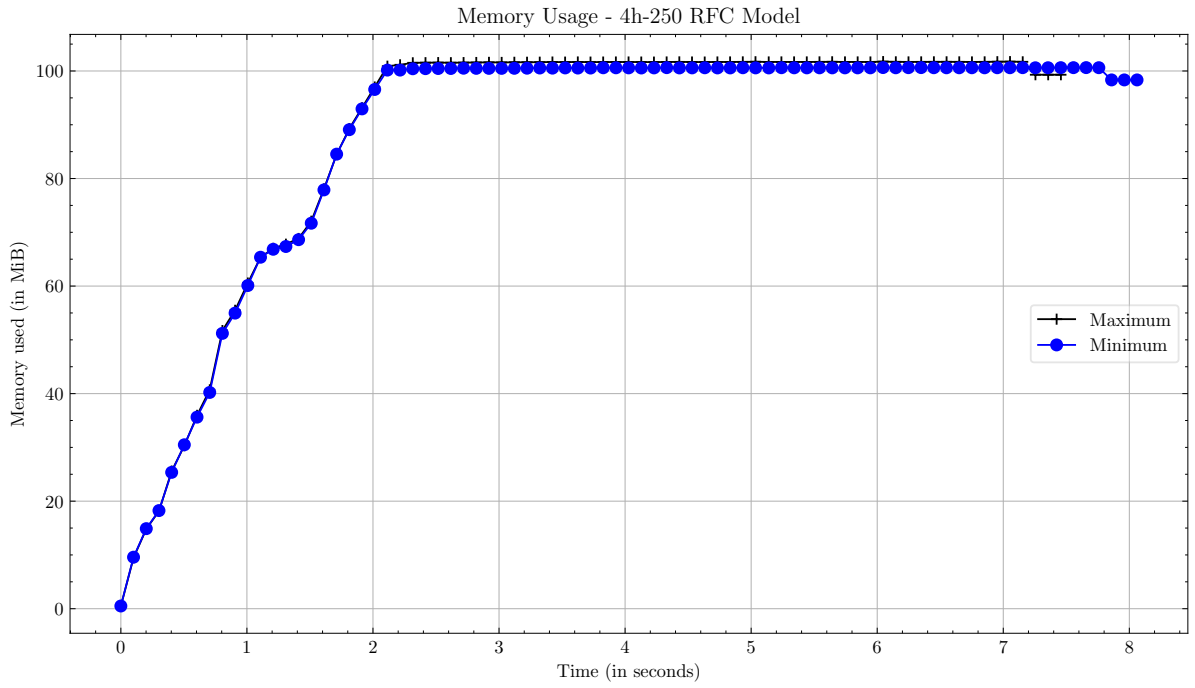


Figure 6.7: The maximum and minimum memory usage of our RFC model.

6.6. Comparison

This section will go in-depth into the comparison with the current state of the art. Furthermore, we compare the resource usage and the performance of our 4h-250 model with the models proposed by Melnyk, Haleta, and Golphamid [35] and Bai et al. [5]. that we investigated in Chapter 4.

6.6.1. Resource Usage

The prediction times of our 4h-250 model is higher than the model proposed by Melnyk, Haleta, and Golphamid [35]. The main reason for this is that the depth of the trees is significantly higher, and hence, the model is more complex. A larger tree depth causes a higher runtime, as more computations are necessary to reach a leaf node. The depth reached by our 4h-250 model was 20 on average. In comparison, the Melnyk, Haleta, and Golphamid [35] model had a depth of 9.

Even with the slower prediction times of, in the worst case, 160ms, it is still a solution we can deploy on low-end hardware. Our solution uses a 24-hour capture, with sub-captures of a length of 4 hours, which gives us six predictions per device—leading to a total prediction time of below 1-second per device.

The higher depth of the RFC also comes at a storage cost. Our 4h-250 model, on average, required around 6.7 MiB of storage. While the model by Melnyk, Haleta, and Golphamid [35] required 160 KiB. The model proposed by Bai et al. [5] required 0.5 MiB. Some low-end hardware devices might have limited storage, which would cause deployment problems because the operating system might take a significant part of the available storage.

If compression libraries are available on the device, one could compress the model. For example, the GunZip compression lowered the model size to 1.1 MiB. The model can then be loaded into memory, decompressed, and used to predict devices, which reduces the amount of storage space required—making it again able to be loaded onto low-end hardware with limited storage available. However, using

compression will increase the CPU usage when loading the model into memory.

At last, we compare memory usage. Figure 6.8 displays the memory usage of all the models. It has been capped at 15 seconds, as the memory does not increase afterward. Our model uses slightly more memory than the RFC model proposed by Melnyk, Haleta, and Golphamid [35], which uses 90 MiB. However, as mentioned before, the depth of our tree is 20, which is more than twice the depth of the model proposed by Melnyk, Haleta, and Golphamid [35], likely causing a slight increase in memory as the model holds more information. The total memory difference between the two models is 10 MiB. Furthermore, we use a third of the memory compared to the model proposed by Bai et al. [5]. The neural network likely uses more complex libraries, causing higher memory usage, as explained in Chapter 4.

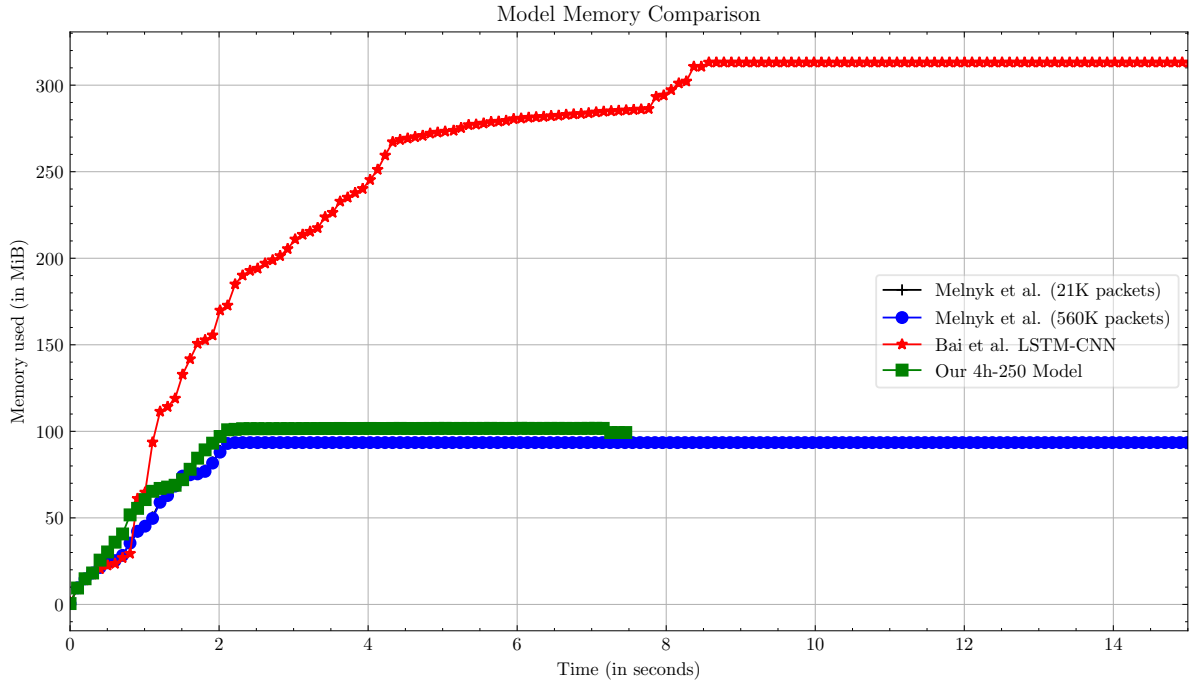


Figure 6.8: The memory usage of the models explored in our thesis.

To conclude, this section shows that our 4h-250 model is overall slower due to the increased complexity of the trees within the RFC. Furthermore, the storage requirements also increased due to the complexity, but these are solvable by using compression. At last, we showed that the memory usage of our 4h-250 model is between that of the model proposed by Melnyk, Haleta, and Golphamid [35] and the one proposed by Bai et al. [5].

6.6.2. Accuracy

This section will compare our accuracy and misclassifications with both models introduced in Chapter 4. First, we compare it against the model proposed by Melnyk, Haleta, and Golphamid [35], as it is closest to our primary task. Both models try to classify the class of the device rather than the flow of the devices.

The model proposed by Melnyk, Haleta, and Golphamid [35] reaches the highest accuracy of 94%, with two classes. Our 4h-250 model reached a high accuracy of 73%, with five classes, which means that adding three extra classes decreases the accuracy by 21%. Additionally, we used features that require less computational power. The features proposed in the model of Melnyk, Haleta, and Golphamid [35] requires many iterations for the Page Ranking algorithm.

Next, we compare our 4h-250 model against the one proposed by Bai et al. [5]. The idea behind their model was to predict flows rather than devices. The accuracy of their model reached a high of 80.1%, with a mean of 74.9%. Furthermore, adding a single class would only include a slight penalty, with a 7.1% difference on the highest accuracy and 1.9% on the median. However, we could not verify the model of Bai et al. [5]., and our measurements only reached the highest accuracy of around 75%.

Additionally, the model verification only included a limited amount of devices. For example, the hub category only contained a single device for testing. In comparison, our 4h-250 model contained 11 hubs. Because of the LOGO technique, we could also use the 11 hub samples for testing. Table 6.5 summarizes the methods used. The table shows that our approach uses the most categories and almost twice the number of unseen devices, which only slightly affects the accuracy.

Model	Number of Classes	Unseen Devices	Accuracy
Melnyk, Haleta, and Golphamid	2	31	97%
Bai et al.	4	8	80% [1]
Our 4h-250 Model	5	74	73%

Table 6.5: Comparison between the methodologies discussed in this work.

[1] Accuracy could not be verified.

7

Discussion & Future Work

In this chapter, we will go over the limitations of our approach. Then, we propose several ways to extend the research in the future work section.

7.1. Limitations

While our approach sets a good scientific foundation for the research, it also has several limitations. Firstly, we only focussed on five categories, while in theory, there are many more categories to be explored, such as kitchen appliances or smart gardening equipment.

Secondly, we manually selected the features based on the hypotheses defined per device. The downside of the approach is that it does not guarantee that the model has the highest accuracy possible. It could be possible that another set of features would gain a higher accuracy.

Thirdly, we only used the first 24 hours of the capture, which causes not all events to be captured. For example, an update event is unlikely to occur in that short period. The downside of this is that some information is missing, leaving a certain level of uncertainty. The extra data could improve accuracy due to the increased amount of information.

Lastly, our datasets do not include any real-world household traffic. The real-world household data allows us to create more complex usage patterns, likely leading to better accuracy. As mentioned before, certain environments likely show traffic patterns linked to certain devices.

7.2. Future Work

There are several ways to extend our research, or interesting areas for future research. These are:

Extension of Data In the field, there is a lot of data available for a single set of devices, namely cameras. It is important to diversify the data available to allow more extensive models. Additionally, it is essential to include data from actual households to measure performance. As mentioned before, a household could show patterns that could link back to the device categories.

Extension of Categories Future research should focus on extending the number of categories and making them less general. For example, rather than 'automation,' differentiate between lights, plugs, and sensors. For the extension of categories, we first need an extension of data, as there is not enough public data available for creating an extensive list of categories.

Automatic Feature Selection As mentioned in the limitations, our approach relies on the hypothesis to select the features. Future research could automatically select the features to see how it compares against the manual selection performed in this work. The automatic selection could result in higher accuracy. However, the approach would remove the transparency that the hypotheses introduced. Future research could compare both options and try to improve transparency of automated models.

System Creation The results of this research can be the backbone to creating a system that automatically blocks or allows traffic of IoT devices based on their type. Most device types could have a basic set of firewall rules that helps users secure their households. Additionally, the low resource usage allows deployment on the router provided by most ISPs, reducing the cost for the end user.

8

Conclusion

This work showed that low-end dedicated hardware devices could classify IoT devices into five categories. Furthermore, the classification got constructed around a set of introduced hypotheses that focused on the devices' network characteristics, which led to a scientific foundation for selecting the features of our model. Finally, using the LOGO cross-validation technique, we evaluated the performance of our model by only testing on unseen devices.

This chapter will conclude our research by answering the sub-research questions and main research question introduced in Chapter 1. The research questions are as follows:

1. How do existing models perform on low-end dedicated hardware devices? Chapter 4 explored two types of models—an RFC proposed by Melnyk, Haleta, and Golphamid [35] and an LSTM-CNN proposed by Bai et al. [5]. Both models would be able to be deployed on low-end hardware as their resource usage was minimal. However, the LSTM-CNN model used significantly more memory than the RFC model, but the prediction times were significantly lower.

2. What are the difficulties when classifying IoT devices not present in the training set? We introduced the two main difficulties in Chapter 5. Firstly, the vendor has the freedom to implement the device in any way they see fit. The freedom of implementation allows vendors to design a device to their liking with the required features, which influences the network traffic patterns as these features are likely also internet-connected.

Secondly, the datasets used did not always use the device as intended. Furthermore, one dataset automated the events to trigger them at specific times, but they could not automate all events. Additionally, the other two datasets did not keep an event log and used the devices in an unscheduled random way.

3. How do we categorize appliances with the same functionality from different brands? We decided to take a scientific approach to feature selection to tackle the challenges found. For every device category, we created hypotheses. These hypotheses give us a foundation for our feature selection and help us to focus on what is essential. Additionally, it clarifies what the model uses and why it uses these features, improving transparency.

The hypotheses focused on the primary task of the device, with the idea that every device with the same primary task would show these characteristics in its network traffic.

4. How well does this new model perform on low-end dedicated hardware devices? In Chapter 6, we explored the results of our model on a low-end dedicated hardware device. It shows that the memory usage is around 100 MiB, comparable to that of the RFC model proposed by Melnyk, Haleta, and Golphamid [35]. The storage requirements and prediction times were significantly higher than the other two models. However, a complete device classification would take less than a second, even with the higher prediction times. When using compression, the model size would decrease significantly.

The additional resource usage was not a unexpected, as the model becomes more complex when differentiating between more classes. We saw this back when comparing the tree depth of the model proposed by Melnyk, Haleta, and Golphamid [35] and ours. Our model was at least twice the depth, resulting in more information to be stored and considered when making a prediction.

5. How well does this new model perform in terms of accuracy? The accuracy of our model is 73% on 74 devices. Compared with the other two models, we have performed the most extensive test to date. We added significantly more unseen devices and an extra category. The extensive testing and complexity added by the extra category decreased the accuracy only by 7% compared to the model proposed by Bai et al. [5].

Next, we answer the main research question of this work.

How can we perform Device Type Classification of IoT devices on low-end dedicated hardware devices? We need to examine the two parts this question contains to answer it. First, how would we perform device type classification? As answered before, we have based our classification on the hypotheses set and try to focus on detecting the primary task of the devices. This approach showed that we could classify devices into five categories with an accuracy of 73%.

Secondly, how would this work on low-end hardware? To answer this, we looked into the existing research in this area. After which, we decided to continue with the RFC because it showed good performance with low memory usage. Additionally, it aims to reduce bias against certain features, something we saw was necessary when using trees for this dataset.

Overall, we have shown that it is possible to run device type classification of IoT devices with reasonable accuracy on low-end hardware devices.

References

[1] Nesrine Ammar, Ludovic Noirie, and Sebastien Tixeuil. “Autonomous Identification of IoT Device Types based on a Supervised Classification”. In: *2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6. ISBN: 978-1-72815-089-5. DOI: 10.1109/icc40277.2020.9148821.

[2] Prashant Anantharaman, Liwei Song, Ioannis Agadakos, Gabriela Ciocarlie, Bogdan Copos, Ulf Lindqvist, and Michael E. Locasto. “IoTHound: environment-agnostic device identification and monitoring”. In: *Proceedings of the 10th International Conference on the Internet of Things*. ACM, 2020, pp. 1–9. ISBN: 978-1-4503-8758-3. DOI: 10.1145/3410992.3410993.

[3] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. “Understanding the Mirai Botnet”. In: *Proceedings of the 26th USENIX Conference on Security Symposium*. SEC’17. Vancouver, BC, Canada: USENIX Association, 2017, pp. 1093–1110. ISBN: 9781931971409.

[4] Leonardo Babun, Hidayet Aksu, Lucas Ryan, Kemal Akkaya, Elizabeth S. Bentley, and A. Selcuk Uluagac. “Z-IoT: Passive Device-class Fingerprinting of ZigBee and Z-Wave IoT Devices”. In: *2020 IEEE International Conference on Communications (ICC)* (2020), pp. 1–7. DOI: 10.1109/icc40277.2020.9149285.

[5] Lei Bai, Lina Yao, Salil S. Kanhere, Xianzhi Wang, and Zheng Yang. “Automatic Device Classification from Network Traffic Streams of Internet of Things”. In: *2018 IEEE 43rd Conference on Local Computer Networks (LCN)* (2018), pp. 1–9. DOI: 10.1109/lcn.2018.8638232.

[6] Kiran Bandla. *DPKT: fast, simple packet creation / parsing, with definitions for the basic TCP/IP protocols*. URL: https://github.com/kbandla/dpkt (visited on 05/09/2022).

[7] Jiaqi Bao, Bechir Hamdaoui, and Weng-Keen Wong. “IoT Device Type Identification Using Hybrid Deep Learning Approach for Increased IoT Security”. In: *2020 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2020, pp. 565–570. ISBN: 978-1-72813-129-0. DOI: 10.1109/iwcmc48107.2020.9148110.

[8] Sergey Brin and Lawrence Page. “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Comput. Networks* 30.1-7 (1998), pp. 107–117. DOI: 10.1016/S0169-7552(98)00110-X.

[9] Bundesnetzagentur. *Bundesnetzagentur removes children’s doll ”Cayla” from the market*. URL: https://www.bundesnetzagentur.de/SharedDocs/Pressemitteilungen/EN/2017/17022017_cayla.html (visited on 01/03/2021).

[10] Edgar D Cardenas. “MAC Spoofing – An Introduction”. In: *GIAC Certifications* (Aug. 23, 2003). URL: https://www.giac.org/paper/gsec/3199/mac-spoofing-an-introduction/105315 (visited on 12/10/2021).

[11] Patrik Cerwall, Richard Moller, Anette Lundvall, Peter Jonsson, Stephen Carson, Rachit Saksena, Dave Lu, Ilyas Celik, Peter Jonsson, Stephen Carson, Steven Davies, Per Lindberg, Greger Blennerud, Karena Fu, Bilal Bezri, Jawad Manssour, Shi Theng Khoo, Fredrik Burstedt, Jennifer Walker, Ove Persson, Jens Malmodin, Olivia Thell, Alejandro Ferrer, Anders Carlsson P, Kevin Hume, Luciana Leite, and Adriana Margarita Mahecha Segura. *Ericsson Mobility Report*. Tech. rep. Ericsson, 2021, p. 40. URL: https://www.ericsson.com/4ad7e9/assets/local/reports-papers/mobility-report/documents/2021/ericsson-mobility-report-november-2021.pdf (visited on 01/03/2022).

[12] Prawit Chumchu, Tanatal Saelim, and Chunyamon Sriklauy. “A new MAC address spoofing detection algorithm using PLCP header”. In: *2011 International Conference on Information Networking, ICOIN 2011, Kuala Lumpur, Malaysia, January 26-28, 2011*. Ed. by Cheeha Kim and Yongtae Shin. IEEE Computer Society, 2011, pp. 48–53. DOI: 10.1109/ICOIN.2011.5723112.

[13] Gerald Combs and The Wireshark team. *Wireshark*. URL: https://www.wireshark.org/ (visited on 03/08/2022).

[14] Ivan Cvitić, Dragan Peraković, Marko Periša, and Brij Gupta. “Ensemble machine learning approach for classification of IoT devices in smart home”. In: *International Journal of Machine Learning and Cybernetics* 12.11 (2021), pp. 3179–3202. ISSN: 1868-8071. DOI: 10.1007/s13042-020-01241-0.

[15] David McGrew and Blake Anderson and Philip Perricone and Bill Hudson. *Joy*. URL: https://github.com/cisco/joy (visited on 03/08/2022).

[16] Dor Green. *PyShark*. URL: https://github.com/KimiNewt/pyshark (visited on 03/08/2022).

[17] Paul Ducklin. “Has your sleeping baby been indexed by this search engine?” In: (Jan. 25, 2016). URL: https://nakedsecurity.sophos.com/2016/01/25/has-your-sleeping-baby-been-indexed-by-this-search-engine/ (visited on 01/03/2022).

[18] Linna Fan, Shize Zhang, Yichao Wu, Zhiliang Wang, Chenxin Duan, Jia Li, and Jiahai Yang. “An IoT Device Identification Method based on Semi-supervised Learning”. In: *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–7. ISBN: 978-3-903176-31-7. DOI: 10.23919/cnsm50824.2020.9269044.

[19] Genesis Toys. *My Friend Cayla*. URL: https://myfriendcayla.co.uk/ (visited on 01/21/2021).

[20] Jakob Greis, Artem Yushchenko, Daniel Vogel, Michael Meier, and Volker Steinhage. “Automated Identification of Vulnerable Devices in Networks using Traffic Data and Deep Learning”. In: *Computing Research Repository (CoRR)* (2021). arXiv: 2102.08199 [cs].

[21] Ke Gu, Ying Fan, and Zengru Di. “Signed PageRank on Online Rating Systems”. In: *J. Syst. Sci. Complex.* 35.1 (2022), pp. 58–80. DOI: 10.1007/s11424-021-0124-2.

[22] Salma Abdalla Hamad, Wei Emma Zhang, Quan Z. Sheng, and Surya Nepal. “IoT Device Identification via Network-Flow Based Fingerprinting and Learning”. In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)* (2019), pp. 103–111. DOI: 10.1109/trustcom/bigdatase.2019.00023.

[23] Zhimin He, Jie Yin, Yu Wang, Guan Gui, Bamidele Adebisi, Tomoaki Ohtsuki, Haris Gacanin, and Hikmet Sari. “Edge Device Identification Based on Federated Learning and Network Traffic Feature Engineering”. In: *IEEE Transactions on Cognitive Communications and Networking* PP.99 (2021), pp. 1–1. DOI: 10.1109/tccn.2021.3101239.

[24] Tin Kam Ho. “Random decision forests”. In: *Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I*. IEEE Computer Society, 1995, pp. 278–282. DOI: 10.1109/ICDAR.1995.598994.

[25] Insecam. *Live cameras directory*. URL: http://www.insecam.org/ (visited on 01/03/2021).

[26] Internet Assigned Numbers Authority (IANA). *IANA IPv4 Special-Purpose Address Registry*. URL: https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml.

[27] Internet Assigned Numbers Authority (IANA). *IANA IPv6 Special-Purpose Address Registry*. URL: https://www.iana.org/assignments/iana-ipv6-special-registry/iana-ipv6-special-registry.xhtml.

[28] Jobib. *Joblib*. URL: https://joblib.readthedocs.io/en/latest/ (visited on 03/08/2022).

[29] Leigh Johnson and Chad Schultz. *Community-built TensorFlow binaries*. URL: https://github.com/bitsy-ai/tensorflow-arm-bin (visited on 03/22/2022).

[30] Elmer Lastdrager, Cristian Hesselman, Jelte Jansen, and Marco Davids. “Protecting Home Networks From Insecure IoT Devices”. In: *2020 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. Budapest, Hungary: IEEE, 2020.

[31] Scott M. Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, pp. 4765–4774. URL: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

[32] Rajib Ranjan Maiti, Sandra Siby, Ragav Sridharan, and Nils Ole Tippenhauer. “Link-Layer Device Type Classification on Encrypted Wireless Traffic with COTS Radios”. In: *Computer Security – ESORICS 2017*. Ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Vol. 10493. Lecture Notes in Computer Science. Springer International Publishing, 2017, pp. 247–264. ISBN: 978-3-319-66398-2 978-3-319-66399-9. DOI: 10.1007/978-3-319-66399-9_14. (Visited on 11/09/2021).

[33] Samuel Marchal, Markus Miettinen, Thien Duc Nguyen, Ahmad-Reza Sadeghi, and N. Asokan. “AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication”. In: *IEEE Journal on Selected Areas in Communications* 37.6 (2019), pp. 1402–1412. ISSN: 0733-8716. DOI: 10.1109/jsac.2019.2904364.

[34] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. “ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis”. In: *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 506–509. ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019878.

[35] Vadym Melnyk, Pavlo Haleta, and Nazar Golphamid. “Machine learning based network traffic classification approach for Internet of Things devices”. In: *Theoretical and Applied Cybersecurity* 2.1 (2020). ISSN: 2708-1397. DOI: 10.20535/tacs.2664-29132020.1.209472.

[36] Umberto Michelucci. *Applied Deep Learning with TensorFlow 2: Learn to Implement Advanced Deep Learning Techniques with Python*. Apress, 2022. ISBN: 9781484280195.

[37] Microsoft. *Microsoft Defender for IoT*. URL: https://azure.microsoft.com/en-us/services/iot-defender/ (visited on 01/04/2021).

[38] Daniel Miessler, Guzman Aaron, Vishruta Rudresh, and Craig Smith. “OWASP Internet of Things Top 10”. In: (2018). URL: https://owasp.org/www-project-internet-of-things/.

[39] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N. Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. “IoT Sentinel: Automated Device-Type Identification for Security Enforcement in IoT”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (2017), pp. 2177–2184. DOI: 10.1109/icdcs.2017.283.

[40] Sebastian Moss. “Major DDoS attack on Dyn disrupts AWS, Twitter, Spotify and more”. In: (Oct. 21, 2016). URL: https://www.datacenterdynamics.com/en/news/major-ddos-attack-on-dyn-disrupts-aws-twitter-spotify-and-more/ (visited on 01/03/2022).

[41] Naji Najari, Samuel Berlemont, Gregoire Lefebvre, Stefan Duffner, and Christophe Garcia. “Network Traffic Modeling For IoT-device Re-identification”. In: *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2020, pp. 1–6. ISBN: 978-1-72816-371-0. DOI: 10.1109/coins49042.2020.9191376.

[42] Fabian Pedregosa. *Memory Profiler*. URL: https://github.com/pythonprofilers/memory_profiler (visited on 03/22/2022).

[43] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. DOI: 10.5555/1953048.2078195.

[44] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. “IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis”. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 2020, pp. 474–489. DOI: 10.1109/EuroSP48549.2020.00037.

[45] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *An Introduction to Machine Learning*. Jan. 2019, p. 263. ISBN: 978-3-030-15728-9. DOI: 10.1007/978-3-030-15729-6.

[46] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. "Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach". In: *Proc. of the Internet Measurement Conference (IMC)*. 2019.

[47] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. Ed. by Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi. ACM, 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778.

[48] John Romkey. "The Toast of the IoT". In: *IEEE Consumer Electronics Magazine* 6.1 (2017), pp. 116–119. ISSN: 2162-2248. DOI: 10.1109/mce.2016.2614740.

[49] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J. Dubois, David R. Choffnes, Georgios Smaragdakis, and Anja Feldmann. "A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild". In: *IMC '20: ACM Internet Measurement Conference, Virtual Event, USA, October 27-29, 2020*. ACM, 2020, pp. 87–100. DOI: 10.1145/3419394.3423650.

[50] Scikit Learn. *Leave One Group Out cross-validator*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneGroupOut.html (visited on 03/22/2022).

[51] Scikit Learn. *Model Persistence*. URL: https://scikit-learn.org/stable/modules/model_persistence.html.

[52] Scikit Learn. *Random Forest Classifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[53] Mustafizur R. Shahid, Gregory Blanc, Zonghua Zhang, and Herve Debar. "IoT Devices Recognition Through Network Traffic Analysis". In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 5187–5192. ISBN: 978-1-5386-5035-6. DOI: 10.1109/bigdata.2018.8622243.

[54] Claude E. Shannon. "A mathematical theory of communication". In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[55] Shodan. *Shodan Images*. URL: https://images.shodan.io/ (visited on 01/03/2021).

[56] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics". In: *IEEE Transactions on Mobile Computing* 18.8 (2019), pp. 1745–1759. ISSN: 1536-1233. DOI: 10.1109/tmc.2018.2866249.

[57] Splunk. *Industrial Data and the Internet of Things*. URL: https://www.splunk.com/en_us/iot.html (visited on 01/04/2021).

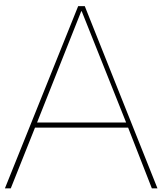
[58] Kai Tao, Jing Li, and Srinivas Sampalli. "Detection of Spoofed MAC Addresses in 802.11 Wireless Networks". In: *E-business and Telecommunications - 4th International Conference, ICETE 2007, Barcelona, Spain, July 28-31, 2007, Revised Selected Papers*. Ed. by Joaquim Filipe and Mohammad S. Obaidat. Vol. 23. Communications in Computer and Information Science. Springer, 2007, pp. 201–213. DOI: 10.1007/978-3-540-88653-2_15.

[59] Wireshark. *Mergecap*. URL: https://www.wireshark.org/docs/man-pages/mergecap.html (visited on 05/09/2022).

[60] BBC World. "German parents told to destroy Cayla dolls over hacking fears". In: (Feb. 17, 2017). URL: https://www.bbc.com/news/world-europe-39002142 (visited on 01/03/2022).

[61] Feihong Yin, Li Yang, Yuchen Wang, and Jiahao Dai. "IoT ETEI: End-to-End IoT Device Identification Method". In: *2021 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE, 2021, pp. 1–8. ISBN: 978-1-72817-534-8. DOI: 10.1109/dsc49826.2021.9346251.

[62] Huan Zhao, Xiaogang Xu, Yangqiu Song, Dik Lun Lee, Zhao Chen, and Han Gao. “Ranking Users in Social Networks with Motif-Based PageRank”. In: *IEEE Trans. Knowl. Data Eng.* 33.5 (2021), pp. 2179–2192. DOI: 10.1109/TKDE.2019.2953264.



Device Categorization RFC Model

A.1. Train Dataset

MAC Address	Category	
Sonos	5c:aa:fd:6c:e0:d4	single
Securify Almond	e4:71:85:25:ce:ec	single
Nest Camera	18:b4:30:58:3d:6c	single
LIFX Virtual Bulb	d0:73:d5:12:84:d1	single
Belkin Wemo Switch	08:86:3b:70:d7:39	single
Belkin Netcam	c0:56:27:53:09:6d	single
Ring Doorbell	00:1d:c9:23:f6:00	single
Roku 4	08:05:81:ee:06:46	single
Belkin Wemo Link	94:10:3e:cc:67:95	single
Netgear Arlo Camera	b0:7f:b9:a6:47:4d	single
D-Link DCS-5009L Camera	b0:c5:54:03:c7:09	single
Logitech Logi Circle	44:73:d6:01:3d:fd	single
Canary	7c:70:bc:5d:09:d1	single
Piper NV	ac:3f:a4:70:4a:d6	single
Withings Home	00:24:e4:2b:a5:34	single
Belkin WeMo Crockpot	94:10:3e:5c:2e:31	single
Mi Casa Verde Vera Lite	94:4a:0c:08:7e:72	single
Chinese Webcam	00:12:16:ab:c0:22	single
Augustus Doorbell Cam	cc:b8:a8:ad:4d:04	single
Chamberlain myQ Garage Opener	64:52:99:97:f8:40	single
Google Home Mini	20:df:b9:20:87:39	single
Google Home	48:d6:d5:98:53:84	single
Bose Sound Touch 10	10:ce:a9:eb:5a:8a	single
Harmon Kardon Invoke	d8:f7:10:c2:29:be	single
Apple Home Pod	d4:90:9c:cc:62:42	single

MAC Address	Category	
Roomba	40:9f:38:92:40:13	single
Koogeek Light Bulb	00:7e:56:77:35:4d	single
TP-Link Smart WiFi LED Bulb	50:c7:bf:92:a6:4a	single
Nest Cam IQ	18:b4:30:8c:03:e4	single
Nest Guard	18:b4:30:40:1e:c5	single
Google On Hub	f4:f2:6d:ce:9a:5d	multi
Philips Hue Hub	00:17:88:21:f7:e4	multi
Insteon Hub	54:4a:16:f9:54:18	multi
Wink Hub	b4:79:a7:22:f9:fc	multi
Roku TV	30:52:cb:a3:4f:5f	multi
Amazon Fire TV	0c:47:c9:4e:fe:5b	multi
nVidia Shield	00:04:4b:55:f6:4f	multi
AppleTV	d0:03:4b:39:12:e3	multi
Logitech Harmony Hub	c8:db:26:02:bb:bb	multi
Caseta Wireless Hub	f4:5e:ab:5e:c0:23	multi
Samsung Smart TV	7c:64:56:60:71:74	multi
Wink 2 Hub	00:21:cc:4d:59:35	multi
Android Tablet	a4:f1:e8:8d:b0:9e	multi
iPad	e8:b2:ac:af:62:0f	multi

Table A.1: The train dataset for the RFC model by Melnyk, Haleta, and Golphamid.

A.2. Test Dataset

MAC Address	Category	
Smart Things	d0:52:a8:00:67:5e	single
Amazon Echo	44:65:0d:56:cc:d3	single
Netatmo Welcome	70:ee:50:18:34:43	single
TP-Link Day Night Cloud Camera	f4:f2:6d:93:51:f1	single
Samsung Smart Cam	00:16:6c:ab:6b:88	single
Dropcam	30:8c:fb:2f:e4:b2	single
Insteon Camera	00:62:6e:51:27:2e	single
Shenzhen Reecam IP Cam	e8:ab:fa:19:de:4f	single
Withings Smart Baby Monitor	00:24:e4:11:18:a8	single
Belkon WeMo Switch	ec:1a:59:79:f4:89	single
TP-Link Smart Plug	50:c7:bf:00:56:39	single
iHome	74:c6:3b:29:d7:1d	single
Belkin WeMo Motion Sensor	ec:1a:59:83:28:11	single

MAC Address	Category	
NEST Protect Smoke Alarm	18:b4:30:25:be:e4	single
Netatmo Weather Station	70:ee:50:03:b8:ac	single
Withings Smart Scale	00:24:e4:1b:6f:96	single
Blipcare Blood Pressure Meter	74:6a:89:00:2e:25	single
Withings Aura Smart Sleep Sensor	00:24:e4:20:28:c6	single
Light Bulbs LiFX Smart Bulb	d0:73:d5:01:83:08	single
Triby Speaker	18:b7:9e:02:20:44	single
PIX-Star Photo-Frame	e0:76:d0:33:bb:85	single
Nest Dropcam	30:8c:fb:b6:ea:45	single
HP Printer	70:5a:0f:e4:9b:c0	multi
Samsung Galaxy Tab	08:21:ef:3b:fc:e3	multi
Android Phone	40:f3:08:ff:1e:da	multi
Laptop	74:2f:68:81:69:42	multi
Macbook	ac:bc:32:d4:6f:2f	multi
Android Phone	b4:ce:f6:a7:a3:c2	multi
iPhone	d0:a6:37:df:a1:e1	multi
Macbook / iPhone	f4:5c:89:93:cc:85	multi
TPLink Router Bridge LAN	14:cc:20:51:33:ea	multi

Table A.2: The test dataset for the RFC model by Melnyk, Haleta, and Golphamid.

B

Device Categorization for Implementation

Name	Category	Datasets
Allure Speaker	Audio	IMC
Amcrest Cam Wired	Camera	IMC
Appletv	Tv	IMC, Your Things
Belkin Wemo Motion Sensor	Sensor	UNSW
Belkin Wemo Switch	Sensor	Your Things, UNSW
Blink Camera	Camera	IMC
Blink Security Hub	Hub	IMC
Bulb1	Light	IMC
Charger Camera	Camera	IMC
Cloudcam	Camera	IMC
D-Link Camera	Camera	Your Things
Dropcam	Camera	UNSW
Echodot	Audio	IMC
Echoplus	Audio	IMC, UNSW
Echospot	Audio	IMC
Firetv	Tv	IMC, Your Things
Google Home	Audio	IMC, Your Things
Google Home Mini	Audio	IMC, Your Things
iHome	Sensor	UNSW
Insteon Camera	Camera	UNSW
Insteon Hub	Hub	IMC
Lefun Cam Wired	Camera	IMC
Lgtv Wired	Tv	IMC
Light Bulbs LiFX Smart Bulb	Light	Your Things, UNSW

Name	Category	Datasets
Lightify Hub	Hub	IMC
Luohe Spycam	Camera	IMC
Magichome Strip	Light	IMC
Microseven Camera	Camera	IMC
Nest Dropcam	Camera	UNSW
Nest Protect Smoke Alarm	Sensor	UNSW
Netamo Welcome	Camera	UNSW
Netatmo Weather Station	Sensor	IMC, UNSW
Philips Bulb	Light	IMC
Philips Hub	Hub	IMC
Ring Doorbell	Camera	IMC, Your Things
Roku Tv	Tv	IMC, Your Things
Samsung SmartCam	Camera	UNSW
Samsungtv Wired	Tv	IMC, Your Things
Sengled Hub	Hub	IMC
Smartthings Hub	Hub	IMC, UNSW
TP-Link Bulb	Light	IMC, Your Things
TP-Link Day Night Cloud Camera	Camera	UNSW
TP-Link Smart Plug	Sensor	IMC, UNSW
Triby Speaker	Audio	UNSW
Wansview Cam Wired	Camera	IMC
Wemo Plug	Sensor	IMC
Wink Hub2	Hub	IMC, Your Things
Withings Aura Smart Sleep Sensor	Sensor	UNSW
Withings Smart Baby Monitor	Camera	UNSW
Withings Smart Scale	Sensor	UNSW
Xiaomi Cam2	Camera	IMC
Xiaomi Hub	Hub	IMC
Xiaomi Strip	Light	IMC
Yi Camera	Camera	IMC
Zmodo Doorbell	Camera	IMC
Nest Camera	Camera	Your Things
Wink Hub	Hub	Your Things
Belkin Netcam	Camera	Your Things
Roku 4	Tv	Your Things
Nvidia Shield	Tv	Your Things
Belkin Wemo Link	Light	Your Things
Netgear Arlo Camera	Camera	Your Things
Logitech Logi Circle	Camera	Your Things

Name	Category	Datasets
Canary	Camera	Your Things
PiperNV	Camera	Your Things
Withings Home	Camera	Your Things
Chinese Webcam	Camera	Your Things
August Doorbell Camera	Camera	Your Things
Logitech Harmony Hub	Hub	Your Things
Caseta Wireless Hub	Hub	Your Things
Harmon Kardon Invoke	Audio	IMC, Your Things
Apple Home Pod	Audio	Your Things
Koogeek Light Bulb	Light	Your Things
Nest Camera IQ	Camera	Your Things

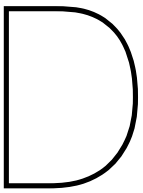
Table B.1: The dataset categorization for our model.

C

Default Settings - Random Forest Classifier

Parameter	Value
max_depth	None
min_sample_split	2
min_samples_leaf	2
min_weight_fraction_leaf	0.0
max_features	Square root
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None

Table C.1: The default settings of the Scikit Learn RFC model.



Prediction Results - Random Forest Classifier

Device	Class	Automation	Camera	Hub	Audio	TV
Allure Speaker	Audio	0	3	1	8	0
Amcrest Cam Wired	Camera	0	9	2	1	0
Appletv	Tv	2	0	0	0	28
Belkin Wemo Motion Sensor	Automation	5	1	0	0	0
Belkin Wemo Switch	Automation	6	5	0	0	1
Blink Camera	Camera	3	18	0	0	0
Blink Security Hub	Hub	3	2	17	0	0
Bulb1	Automation	0	7	5	0	0
Charger Camera	Camera	5	3	4	0	0
Cloudcam	Camera	0	8	0	4	0
D-Link Camera	Camera	3	0	3	0	0
Dropcam	Camera	0	6	0	0	0
Echodot	Audio	1	0	0	35	0
Echoplus	Audio	0	0	0	29	1
Echospot	Audio	0	0	0	24	0
Firetv	Tv	0	0	0	11	19
Google Home	Audio	0	0	0	18	0
Google Home Mini	Audio	0	4	0	36	2
iHome	Automation	4	2	0	0	0
Insteon Camera	Camera	6	0	0	0	0
Insteon Hub	Hub	1	1	12	0	0
Lefun Cam Wired	Camera	0	9	3	0	0
Lgtv Wired	Tv	0	5	1	0	6
Light Bulbs LiFX Smart Bulb	Automation	0	9	3	0	0

Device	Class	Automation	Camera	Hub	Audio	TV
Lightify Hub	Hub	20	2	2	0	0
Luohe Spycam	Camera	1	11	0	0	0
Magichome Strip	Automation	13	7	4	0	0
Microseven Camera	Camera	0	6	6	0	0
Nest Dropcam	Camera	0	1	0	0	0
Nest Protect Smoke Alarm	Automation	6	0	0	0	0
Netamo Welcome	Camera	2	2	1	1	0
Netatmo Weather Station	Automation	18	0	0	0	0
Philips Bulb	Automation	8	0	4	0	0
Philips Hub	Hub	0	23	0	0	1
Ring Doorbell	Camera	7	19	0	4	0
Roku Tv	Tv	2	1	0	5	22
Samsung SmartCam	Camera	4	0	0	2	0
Samsungtv Wired	Tv	3	6	0	5	16
Sengled Hub	Hub	11	1	18	0	0
Smartthings Hub	Hub	6	15	19	2	0
TP-Link Bulb	Automation	14	2	5	2	7
TP-Link Day Night Cloud Camera	Camera	6	0	0	0	0
TP-Link Smart Plug	Automation	33	0	2	0	7
Triby Speaker	Audio	4	0	2	0	0
Wansview Cam Wired	Camera	0	24	0	0	0
Wemo Plug	Automation	7	10	0	7	0
Wink Hub2	Hub	5	0	13	0	0
Withings Aura Smart Sleep Sensor	Automation	5	1	0	0	0
Withings Smart Baby Monitor	Camera	0	6	0	0	0
Withings Smart Scale	Automation	6	0	0	0	0
Xiaomi Cam2	Camera	0	10	1	1	0
Xiaomi Hub	Hub	22	0	2	0	0
Xiaomi Strip	Automation	4	0	8	0	0
Yi Camera	Camera	2	22	0	0	0
Zmodo Doorbell	Camera	0	9	1	2	0
Nest Camera	Camera	0	6	0	0	0
Wink Hub	Hub	6	0	0	0	0
Belkin Netcam	Camera	0	4	1	0	1
Roku 4	Tv	0	0	0	0	6
Nvidia Shield	Tv	0	1	0	2	3
Belkin Wemo Link	Automation	0	2	4	0	0
Netgear Arlo Camera	Camera	0	6	0	0	0
Logitech Logi Circle	Camera	0	6	0	0	0

Device	Class	Automation	Camera	Hub	Audio	TV
Canary	Camera	0	6	0	0	0
PiperNV	Camera	0	6	0	0	0
Withings Home	Camera	0	4	0	1	1
Chinese Webcam	Camera	0	6	0	0	0
August Doorbell Camera	Camera	0	2	0	0	4
Logitech Harmony Hub	Hub	0	2	4	0	0
Caseta Wireless Hub	Hub	1	2	1	1	1
Harmon Kardon Invoke	Audio	0	11	0	5	2
Apple Home Pod	Audio	0	0	0	0	6
Koogeek Light Bulb	Automation	6	0	0	0	0
Nest Camera IQ	Camera	0	5	0	0	0

Table D.1: The classification results by the RFC model with 250 estimators and a sub-capture length of 4 hours.