# Identifying Internet of Things Software Activities using Deep Learning-based Electromagnetic Side-Channel Analysis

Quan Le[a], Luis Miralles-Pechuán[b], Asanka Sayakkara[c,1], Nhien-An Le-Khac[d], Mark Scanlon[a,d]

[a]*Centre for Applied Data Analytics Research (CeADAR), University College Dublin, Dublin, Ireland*
[b]*Technological University Dublin, Dublin, Ireland*
[c]*School of Computing (UCSC), University of Colombo, Sri Lanka*
[d]*Forensics and Security Research Group, University College Dublin, Ireland*

**Abstract**

Internet of Things (IoT) is becoming the new frontier in digital forensics due to the abundance of IoT devices appearing in day-to-day life. The diversity and complexity of IoT ecosystems pose a considerable challenge to digital investigators that demand novel approaches. Electromagnetic side-channel analysis (EM-SCA) has been proposed as a promising window to gather forensically useful information from IoT devices. Machine Learning (ML) techniques are instrumental when performing EM-SCA on IoT devices. Our work aims to investigate how machine learning can be applied to accurately identify complex activities on IoT devices from their generated electromagnetic noises. To this end, a range of classification models were created, including deep learning models, to predict the activity from the electromagnetic noise emitted while the device performed the activities. A dataset was generated by using ten different well-known sorting algorithms with diverse computational time complexities and running them on an Arduino Leonardo device to represent a low-powered IoT device. The algorithms were continually sorting arrays of 100 elements randomly generated in ascending order. Experiments were conducted to identify which ML methods performed better with the generated data sets. Furthermore, more experiments were conducted to identify how the methods perform depending on the window size of raw samples and the number of examples against which they are trained. From the experimental results, it is possible to predict which activity is being executed with a high level of accuracy (99.6%) with a convolutional neural network (CNN). It was also found that Random Forests (RF) and Deep Learning (DL) are suitable ML models for making predictions with EM-SCA.

*Keywords:*
Electromagnetic Side-Channel Analysis, Internet of Things, Deep Learning, Random Forests

## 1. Introduction

Since the emergence of computers, Moore's law enabled engineers to increase their capabilities with time. Besides making computers more powerful and resource-rich, this trend has also enabled the production of smaller and energy-efficient computing devices such as mobile devices, that have less computing power but can last longer on battery power. The ultimate result of those advances is the emergence of the Internet of Things (IoT); small computing devices with sensors and networking capability that has started to embed into our everyday lives including sports, healthcare, and smart-home systems [1].

The integration of IoT devices into everyday life makes it inevitable for law enforcement to encounter them in digital forensic investigations [2]. In modern society, most people carry a smartphone in their pocket and many can have some kind of wearable device such as a smart wristwatch or a personal health tracking device. Therefore, an increasing number of IoT devices are likely to contain important pieces of information related to digital investigation. As a result, IoT forensics has gained significant interest among the law-enforcement community and has become recently a hot topic in the research community [3, 4].

The acquisition of digital evidence from IoT devices is a challenging task compared to traditional digital forensics for several reasons. For example, traditional digital forensics involves devices that have easily accessible non-volatile data storage such as hard disks, solid-state drives (SSD), or removable media, e.g., secure digital (SD) cards. When required, a forensic image of the storage can be taken for analysis. In contrast, most IoT devices consist of on-board flash storage that is difficult to access for forensic image acquisition [5]. This is due to the manufacturer-specific custom hardware architectures used for IoT devices. This problem already exists for smartphone forensics. However, the high diversity of IoT device hardware architectures has made it significantly more complex to perform IoT forensics. As a result, it is highly necessary to find alternative approaches to acquire forensic insights from IoT devices [6, 5].

Electromagnetic side-channel analysis (EM-SCA) is the dis-

*Email addresses:* quan.le@ucd.ie (Quan Le), luis.miralles@tudublin.ie (Luis Miralles-Pechuán), asa@ucsc.cmb.ac.lk (Asanka Sayakkara), an.lekhac@ucd.ie (Nhien-An Le-Khac), mark.scanlon@ucd.ie (Mark Scanlon)

cipline of gathering information about the internal operations of digital electronics by analysing the electromagnetic noise emissions they produce. These emissions are a by-product of the high-frequency switching operations of semiconductor electronics [7]. EM-SCA has been recently proposed as a method to acquire forensically useful insights from IoT devices [8, 9]. When an IoT device that is powered on is seized by law-enforcement, EM-SCA procedures can potentially be performed immediately on the device. It has been proposed that various information can be detected with EM-SCA-based approaches such as tampered firmware, software behaviour, and cryptographic algorithms [5]. Due to the non-invasive nature of acquiring and analysing EM emission signals, there is no need to physically tamper with the device being investigated.

The emissions that are produced by IoT devices are weak signals that are not detectable beyond a few centimetres from the device unless powerful signal amplifiers are used. However, unlike unauthorised eavesdropping of devices through EM-SCA, a forensic inspection of a device does not have a requirement to make observations from a distance. The side-channel EM emissions from IoT devices are wide-band signals that are highly dimensional and can be captured with fast sampling rates. Therefore, when captured and stored for analysis, EM data files, called *EM traces*, usually take up multiple gigabytes. When analysing such EM traces for identifying known patterns, applications of artificial intelligence such as Machine Learning (ML) have been explored. ML algorithms are capable of identifying patterns in large and high dimensional data that are not easily recognisable by humans [10].

The application of ML techniques for IoT software behaviour identification still faces many challenges that need to be addressed. From a classification point of view, an IoT device's firmware can be in a wide variety of internal states that can result in a large number of potential classes. Even when considering the same known software activity, slight changes in the variables used within the software activity can affect its emission patterns. Therefore, it is necessary to further evaluate ML-based methodologies for classifying complex software behaviours under varying conditions.

*Contribution of this Work*

Our work aims to investigate how machine learning can be applied to accurately identify complex activities on IoT devices from its generated EM noise and assess the influence of EM sample parameters. Towards this goal, this work explored the performance of ML-based models for classifying complex software programs running on a representative IoT device with randomised internal variables. In the evaluation, a dataset was generated using implementations of multiple well-known sorting algorithms running on the target device that use randomly generated input data arrays. The contribution of this work can be summarised as follows:

- Demonstration that machine learning models trained on past data of complex software activities recognise these activities in future recordings.
- Experimentation showing that when using EM signals for ML-based classification purposes, the use of data in the frequency domain is more effective than in the time domain and other feature extraction methods.
- Among various ML methods evaluated, deep learning approaches outperform all the other algorithms when used for classifying complex software activities using EM side-channel data.

## 2. Background

This section covers the background information related to EM-SCA and the application of ML in its context. Subsection 2.1 provides a comprehensive overview of EM-SCA including methods of data acquisition and tools available for this purpose. Subsection 2.2 provides an overview of ML techniques. Subsections 2.2.1 and 2.2.2 provide background information on two ML architectures, namely; *Random Forest* and *Deep Learning* methods. Finally, Subsection 2.3 illustrates the current status of applying EM-SCA for digital forensics.

### *2.1. Electromagnetic Side-Channel Analysis*

The earliest work on electromagnetic information leakage was published in 1985 [11]. Van Eck [11] demonstrated that cathode-ray tube (CRT) video displays emitted sufficient EM emissions that could be used remotely to reconstruct the content shown on the display. Later, research on eavesdropping computing devices through their unintentional EM emissions started gaining interest among researchers with various objectives such as spying on computer monitor content [12], retrieving cryptographic keys [13], and identifying abnormal deviations of software execution due to malware [14]. IoT devices pose more opportunities for EM-SCA due to the abundance of them within the reach of an attacker in contrast to desktop and laptop computers that tend to stay under the possession of their respective owners.

When machine code instructions are being executed on the processor of an IoT device, sequences of binary digits are moved around the processor's registers rapidly. This storing and resetting operations of values in registers impact the draw of electrical current by the processor. The higher the hamming weight of the value being stored, the higher the impact it has on the current draw of the processor. This varying electrical current draw results in varying EM noise to be generated from the circuitry of the processor [15]. Such weak EM noise can get modulated into any powerful radio frequency signal that is already present in the vicinity. There are two potential radio frequency signal sources on IoT devices; the processor clock and on-board radio transceivers in the case of system-on-chip (SoC) devices [16].

Observation of EM emissions from IoT devices requires specialised equipment. In the traditional form, oscilloscopes with compatible EM probes can be used to capture the signal [17]. The identification of the exact frequency of the leakage signal requires a certain amount of trial-and-error procedure. In the worst case, the attacker has to sweep through the EM spectrum looking for suspicious signals. However, the most common

practice is to use the clock frequency of the device being observed as the frequency of the EM emission signal. When capturing EM signals, several important factors need to be met at the hardware level to ensure successful data acquisition. Firstly, the frequency range of the observation device has to cover the clock frequency of the device being attacked. Secondly, the sampling rate of the device has to be fast enough. The sampling rate is the number of samples taken at the tuned frequency per second. Finally, the equipment has to support a wide enough bandwidth in order to capture as many signals around the leakage signal as possible.

As a much cheaper and more flexible alternative to oscilloscopes, software-defined radios (SDRs) devices can be used. These devices are popular among wireless hackers, hobbyists, and security enthusiasts [18]. An SDR device makes use of a bare minimum hardware layer that can be tuned to a specific frequency and digitally sample EM signals with the help of a fast analogue-to-digital converter (ADC). Such digital samples are later processed with software using various software-defined radio packages – among which GNURadio is the most popular open-source package [19, 20, 21, 22]. Each sample taken from an SDR device represents the amplitude of the tuned frequency at a specific instance of time. In this work, an SDR is used to sample EM emissions from a target IoT device due to the flexibility it offers.

### *2.2. Relevant Machine Learning Methods*

In the last decades, thanks to the increase of computational power and to the development of artificial intelligence, researchers can easily implement ML methods and run them on their computers to conduct experiments [23]. Arthur Samuel, one of the pioneers of ML, defined it as the "field of study that gives computers the ability to learn without being explicitly programmed" [24]. Supervised Learning is the more developed ML learning branch [23] and it is the branch used in this work – specifically Random Forests and Deep Learning.

#### *2.2.1. Random Forests*

Random Forests (RF) select those attributes that are able to distinguish elements that belong to different groups and at the same time, divided groups that belong to the same class. RF try to create different trees by choosing different attributes from the database. This supervised method was introduced in 2001 by Breiman [25], and has become one of the most important supervised methods in ML. Its main advantages are its great predictive power and its rapid execution, and for these reasons, it is an ideal algorithm for the selection of variables. The RF model can be used for both regression and classification, but they are mainly used for classification. RF combines several binary trees that it generates from training data by randomly selecting a subset of variables [26].

The two main RF parameters are: *mtry* (number of randomly selected variables in each split) and *ntree* (number of trees that make up the forest). RF are very suitable for feature selection because they have both high levels of accuracy and also a very quick execution time [26] and easy to parallelize [27].

#### *2.2.2. Deep Learning Methods*

Deep Learning (DL) is a group of algorithms based on Artificial Neural Networks (ANN) [28]. DL models are composed of multiple layers - each includes a number of nodes. Multiple DL layer architectures are developed to capture different correlation patterns in the input; the popular ones are the convolutional layer, the long short term memory layer, and the attention layer [29]. For convolutional neural networks (CNN), DL performs hierarchical feature extraction across its multiple layers, where the first layers deal with low-level abstraction features and the later layers compose these low-level features to capture high-level abstraction features. DL algorithms have excelled in many domains such as image and audio recognition, and object detection [30].

DL learns by backpropagation [29]. For an input example, the back propagation algorithm (a gradient descent optimization technique) adjusts the weights of the model (weights determine if a signal is relevant to go to the next layer) to minimize the mismatch between the predicted output of the DL model with respect to its true label. After a number of epochs, the weights are tuned progressively and the training stops when the difference is minimized, the trained model could then be used to predict labels of new inputs (e.g. data in the testing set).

One of the main advantages of DL is that it can learn feature extraction automatically from the labeled data without using human expert knowledge. DL algorithms are well-known for being able to learn from big data sets. Most ML methods' performance plateaus when given a big enough number of training examples, whereas a DL model performance keeps improving and absorbing information with data sets of millions of examples. The downside is that training a deep learning model is generally more time-consuming than the rest of the methods and also, due to its internal complexity of layers and nodes, it is more difficult to understand the internal logic of its predictions.

### *2.3. Electromagnetic side-channel analysis for forensics*

Recent work has suggested to use EM-SCA in digital investigations where classical methods fail [8, 10]. Most of the potential uses of EM-SCA in digital investigations have already been realized in the information security arena. Detection of the specific make and model of a device [31], the detection of its internal states of software and hardware [32, 33], and the retrieval of cryptographic keys from them [34] in realistic scenarios have been performed in the recent past. Therefore, these techniques need to be leveraged into practical digital investigations by providing the necessary facilities to the investigators. Development of tools that are easy-to-use for digital investigators with a minimum knowledge in EM-SCA is such a timely requirement. *EMvidence* is a tool that is currently under constant development towards this goal [35, 36]. A workshop was delivered at Digital Forensic Research Workshop (DFRWS EU) 2020 describing EMvidence's design and implementation. An explanation for its use, alongside the content of the workshop is available on GitHub [1].

[1] `https://github.com/asanka-code/DFRWS-EU2020-Workshop-Insightsfro`

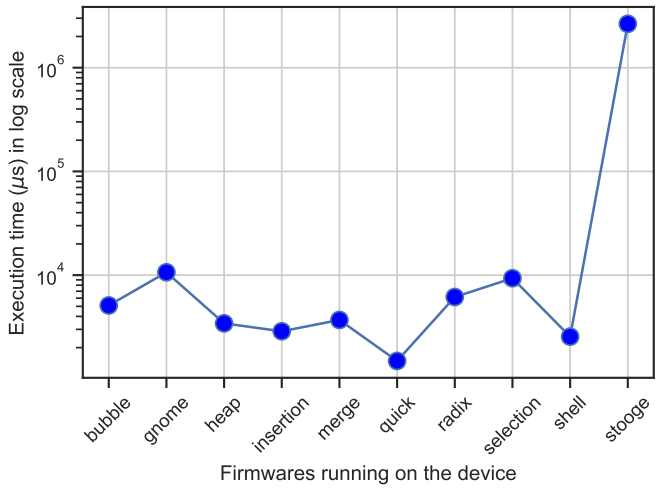


Figure 1: Average execution time for the 10 sorting programs running on Arduino Leonardo device. The average values were calculated by logging 200 consecutive executions of each program with randomly generated 100-elements long arrays as inputs.

## 3. Electromagnetic Side-Channel Raw Sample Dataset

The EM noise emission patterns from IoT devices can depend on a multitude of factors such as the type of the processor, the layout of the printed circuit board (PCB), and the characteristics of the software instructions running on the processor. As a result, it is necessary to experiment with a dataset that is generated on a hardware and software platform easily to reproduce. To this end, the first step was to select a representative IoT device platform that is sufficiently representative of a wider variety of IoT devices and also easy to experiment with.

As the target IoT device, an *Arduino Leonardo* board was select. This is a general-purpose prototype board that has a variety of input/output pins available to be used in a variety of applications. The device consists of an `ATmega32u4` microcontroller at its heart that operates at 16 MHz clock frequency with the help of an external oscillator on-board. The device is powered by 5 V power supply through a USB port, which is also used for the purposes of reprogramming the device. Due to the limited random-access memory (RAM) and read-only memory (ROM) capacity(32 KB of flash memory for storing programs and 2.5 KB of RAM), the device cannot run with an operating system. A manufacturer-provided C-like programming language is used to directly write applications to the device that can run without multi-tasking. The small resource profile of the device resembles many battery-operated IoT devices in common use.

When the same software is running on a processor multiple times, it is unlikely that the order of machine code instructions being executed will be the same each time. This is due to the difference in the input data provided to the software in each case. In consequence, the EM emission pattern can get affected and become slightly different from each case as well. For the purpose of simulating such an extreme case, this work uses 10 sorting algorithm implementations each sorting a 100-element long randomly generated integer array. The sorting algorithms are chosen in a way to include a diverse set of computational time complexities in the dataset (see Table 1). The randomness of the input data to the sorting algorithms ensures that their instructions sequences are different at each iteration. Figure 1 illustrates the average execution time of each sorting algorithm implementation when they are sorting 100 randomly generated integers.

Table 1: Time complexity of the the different sorting algorithms used

| **Algorithm** | **Time Complexity** | | |
|---|---|---|---|
| | Best | Average | Worst |
| Bubble | $\Omega(n)$ | $\Theta(n^2)$ | $O(n^2)$ |
| Gnome | $\Omega(n)$ | $\Theta(n^2)$ | $O(n^2)$ |
| Heap | $\Omega(nlog(n))$ | $\Theta(nlog(n))$ | $O(nlog(n))$ |
| Insertion | $\Omega(n)$ | $\Theta(n^2)$ | $O(n^2)$ |
| Merge | $\Omega(nlog(n))$ | $\Theta(nlog(n))$ | $O(nlog(n))$ |
| Quick | $\Omega(nlog(n))$ | $\Theta(nlog(n))$ | $O(n^2)$ |
| Radix | $\Omega(nk)$ | $\Theta(nk)$ | $O(nk)$ |
| Selection | $\Omega(n^2)$ | $\Theta(n^2)$ | $O(n^2)$ |
| Shell | $\Omega(nlog(n))$ | - | $O(n^2)$ |
| Stooge | $\Omega(n^{log3/log1.5})$ | $\Theta(n^{log3/log1.5})$ | $O(n^{log3/log1.5})$ |

The EM noise of the processor is emitted from various regions of the processor chip across various frequency channels with varying amplitudes. Therefore, when an H-probe antenna is used closer to the processor chip to capture EM noise, the exact location of the antenna over the processor affects the signal being captured. Figure 2 depicts the procedure of placing an H-probe antenna over the processor of the Arduino Leonardo device. When the antenna is centred directly over the processor, a weaker signal is observed with lower amplitudes and fewer frequency features. By slightly adjusting the position of the antenna over the processor and observing the emission signal, it was empirically identified that when the H-loop antenna is located exactly over the index corner of the chip, the strongest EM emission signal was registered. The necessity for close proximity of the antenna to the chip is due to the low signal amplification of the inexpensive SDR device used in this work. However, with more powerful signal acquisition equipment, it would be possible to capture EM radiation from an increased distance. Figure 3 (a) and (b) illustrates the emission signal when the H-loop antenna is placed at the centre and the index corner of the processor respectively. Accordingly, the index corner of the processor was chosen as the default antenna position throughout the rest of the data collection procedure for this work.

When generating the raw sample dataset for the purpose of using it for ML experiments, the following data collection procedure was followed. The data collection was performed in two phases with a time gap of 1 hour between them in order to produce two sets of independently taken data for each sorting algorithm. During that time gap, the Arduino Leonardo device was turned off and left unplugged from the power supply. During each phase, the Arduino device was programmed with each sorting program, one at a time. The basic skeleton of each Arduino program is depicted in Algorithm 1. Once programmed

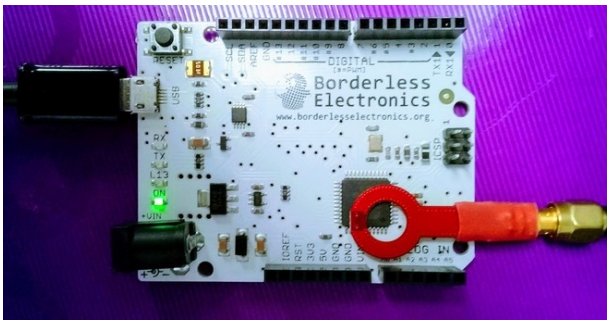


Figure 2: Positioning of the H-loop antenna over the processor of Arduino Leonardo device to maximize the reception of leakage signal.

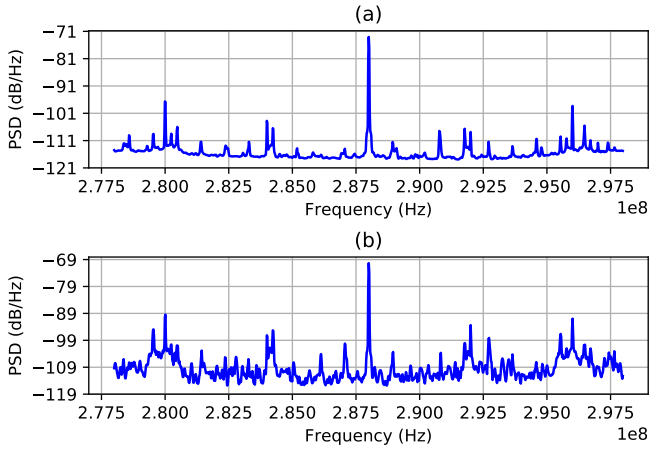


Figure 3: Power spectral density (PSD) of the EM emission when running Bubble sort algorithm with two different antenna positions.

**Algorithm 1:** Structure of each Arduino program

```
Select the sorting algorithm ;
Loop
    array ← generate 100 random integers ;
    result ← sort the array with a sorting algorithm ;
    reset array and result ;
EndLoop
```

with a specific sorting algorithm, the EM emissions of the device are recorded for $T$ time period, where $T$ was set in the two phases to 8 seconds and 4 seconds respectively. The rationale being to use them separately as training and testing data for the ML models.

For data capture, a *HackRF* SDR[2] was used with the centre frequency set to 288 MHz and sampling rate set to 20 MHz [20]. Even though the clock frequency of Arduino Leonardo is 16 MHz, it was empirically identified that certain harmonics of the frequency provide much stronger leakage signals, hence the choice of $18^{th}$ harmonic, which is 288 MHz. Each EM trace was saved into a *NumPy* array file (very frequent in python) in order to ease the processing of them later. In phase-1, the EM trace file for each algorithm was approximately 1.4 GB in size causing the first dataset to be approximately 14 GB in size. In phase-2, an EM trace file was approximately 550 MB in size resulting in a dataset of approximately 5.5 GB in size. Figure 4 illustrates the power spectral density of the recorded signals for 9 of the sorting programs in phase-2. Note that the signal of *bubble sort* program is shown in Figure 3).

Both the EM traces and software used as part of the work outlined in the paper are available open-source at [redacted for peer review].

[2] `https://greatscottgadgets.com/hackrf/`

## 4. Experimentation and Results

In this section, the experiments conducted are outlined to see how well the prediction of the software activity a device is executing performs by just listening to the electromagnetic noise it produces.

### *4.1. Methodology*

For each of our experiments, the methodology outlined in Figure 5 was followed. It includes five main steps: 1) dataset generation, 2) for a chosen window size slicing time series of raw samples to windows of consecutive raw samples, 3) applying feature engineering, 4) ML model generation, and 5) ML model evaluation.

#### *4.1.1. Preparation of the Raw Sample Dataset*

As described in Section 3, the first step invovled collecting raw samples for two data sets: the training set and the testing set. For each of the ten sorting algorithms, a time series of approximately 175 million raw samples was generated in the training session, and a time series of approximately 76 million raw samples were generated in the testing session. The raw training data and testing data were gathered with a time difference of one hour ensuring that the antenna was in the same position. Next each time series of raw samples was sliced into windows of consecutive time steps using a chosen time window, each window of raw samples would then be transformed into an input example through feature engineering. Different time windows were applied to see the performance with each size. There is a trade-off between the accuracy and the need to process as little information as possible. The larger the window, the higher the accuracy of the model, but the more expensive it is making the predictions. All the raw samples were not used to generate training examples and testing examples for a chosen window size. Instead, we configured the number of training examples and the number of testing examples specifically for each of our experiments. A typical dataset configuration for our experiments would include $25,000$ examples in the training set and $25,000$ examples in the testing set.

#### *4.1.2. Feature Engineering*

Each window of a chosen window size of consecutive raw samples was converted to an example, where the number of features in the example would depend on our feature engineering approach, The first approach is directly working with samples

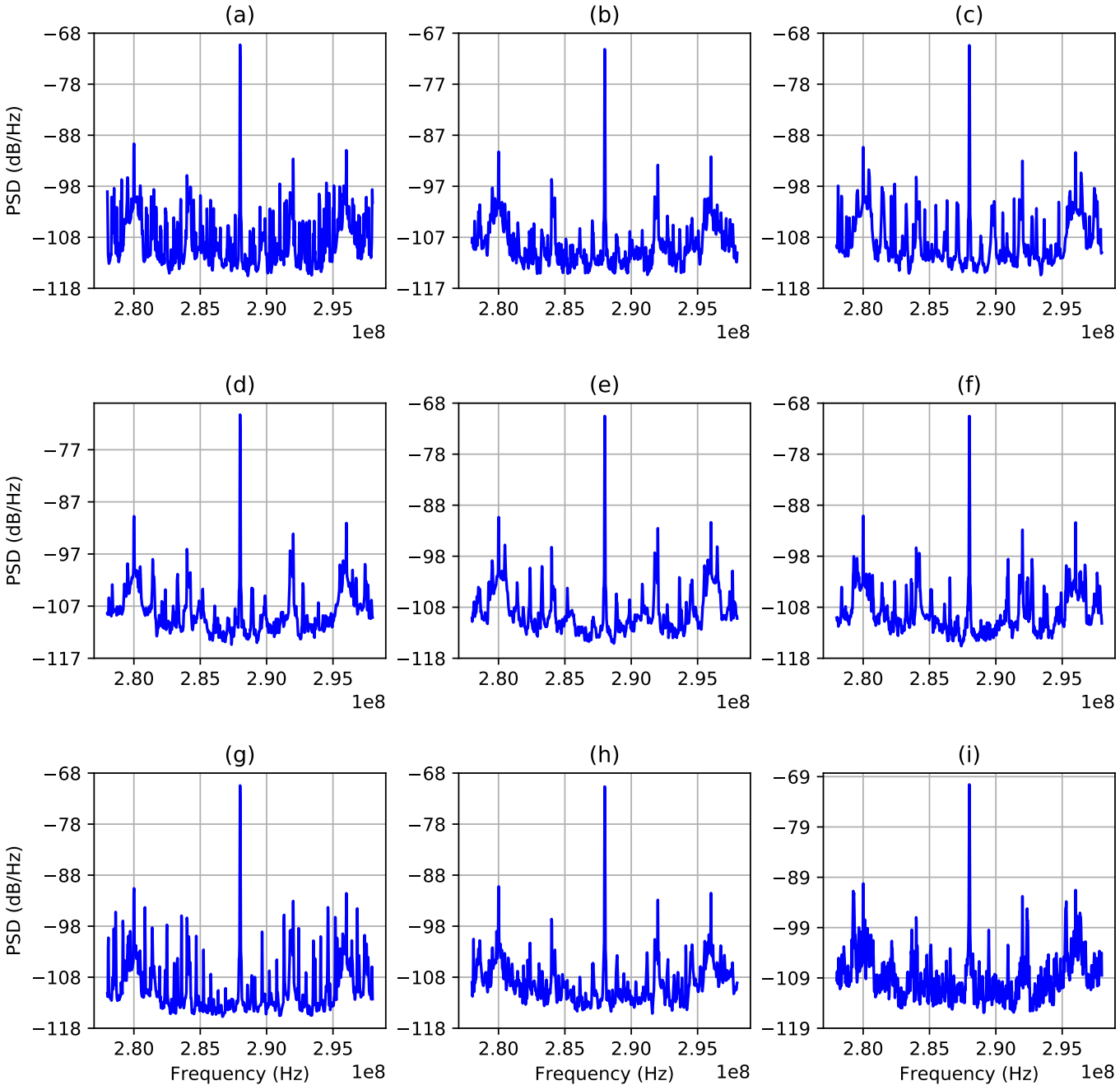


Figure 4: Power spectral density (PSD) of the EM emission when running 9 sorting algorithms. (a) Gnome sort, (b) Heap sort, (c) Insertion sort, (d) Merge sort, (e) Quick sort, (f) Radix sort, (g) Selection sort, (h) Shell sort, and (i) Stooge sort. Bubble sort is not depicted here.


Figure 5: Steps used in the methodology to predict the sorting algorithm executed by the Arduino.

from the time domain, so the number of features in each example is equal to the chosen window size. It has the advantage that there is no extra cost of running any transformation algorithm. The second option is to transform each time window into the frequency domain by applying the Fast Fourier transform (FFT), which calculates the Discrete Fourier Transform (DFT) of a given time window, again the number of frequency features in each example is equal to the chosen window size.

Lastly, it is also quite effective to extract features in the time domain and in the frequency domain. This drastically reduces the information taken by the classifier (the input of the model), but at the same time, it increases the computational costs since all the extracted features have to be calculated.

As with other work in the area related to time series domain [37], the features extracted in the time domain are: mean of the signal, standard deviation, root mean square, variance, median, number of the zero-crossing points, skewness, kurtosis, first and third quartile, interquartile range, and correlation. And the features extracted in the frequency domain are: average, median, variance, root mean square, entropy of the signal, Shannon spectral entropy, spectral centroid, flatness, mode frequency, peak frequency, kurtosis, and skewness.

#### *4.1.3. Machine Learning Models*

In this work, a wide range of machine learning models were considered. Support Vector Machines, Decision Trees, K-Nearest Neighbour, and XGBoost were evaluated with their default settings. A Random Forest Classifier was used with 500 trees and a maximum number of features to be the square root of the number of columns (number of input features), together with bootstrapping activated. Each the above classifiers were trained on input data after being preprocessed by the MinMaxScaler transformation, i.e., each input feature was scaled and translated on the training set to be in the range between 0 and 1; and the same transformation was applied on the data in the test set.

For the deep learning model, a Convolutional Neural Network was used with four layers. The first two layers were two convolutional layers - using 32 filters and 64 filters respectively. The outputs of the second convolutional layer were flattened and connected to a fully connected layer with 64 nodes, which was then connected to the final output layer to predict the activity label. As the convolutional layers in the model were used to extract features across a fixed window of consecutive features, the kernel size for both the convolutional layers was set to be 8 and large strides were used (2 and 4) together with a max pooling value of 4 to increase the depth of field of each node in the first dense layer and to reduce the model complexity. The CNN model was trained with the standard Adam optimization algorithm.

#### *4.1.4. Model Training and Evaluation*

For each experiment and for a chosen machine learning model, we trained it on the training set, and evaluated its performance on the testing set using classification accuracy as the performance metric.

Once the ML model is built, if a real-world deployment was needed, there would only be three steps: applying the time window, transforming into the frequency domain with the FFT, and predicting the activity with the generated ML model.

### *4.2. Results*

In the following subsection we present an explanation and the results for the different experiments.

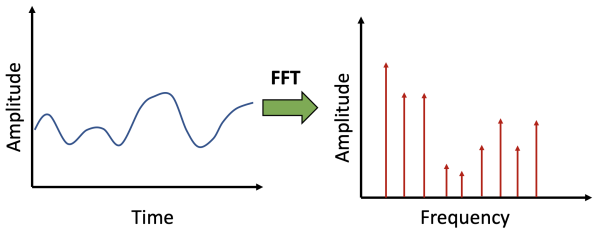


Figure 6: Transforming a signal from the time domain to the frequency domain with the Fast Fourier Transform.

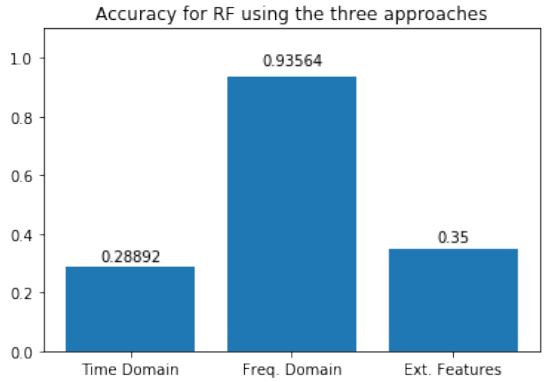


Figure 7: Accuracy of the predicting in time domain, frequency domain, and with extracted features in time and frequency.

#### *4.2.1. Experiment I: Selecting between time domain, frequency domain, and feature extraction*

Our first experiment was designed to select the appropriate feature engineering to apply on each window of raw sample. We considered three feature engineering approaches, as discussed earlier: 1) using directly a window of raw samples from the time domain, 2) converting each window of raw samples from the time domain to the frequency domain using FFT, and 3) using feature selection.

The experiment was conducted using 25,000 examples for the training set and 25,000 examples for the testing set. A Random Forest Classifier was employed with the hyper-parameters as described in 4.1. As shown in Figure 7, according to the first experiment, predicting from the amplitudes of the frequency domain is much more accurate than the other options. In Figure 8, the confusion matrix of the best model is shown; random forest with a time window of 1,000 samples in the frequency domain and an accuracy of 93.56%.

#### *4.2.2. Experiment II: Changing the antenna position*

One interesting question in this research is how the position of the antenna on the device affects the performance of the classifiers. First, the EM traces were created with samples of the ten sorting algorithms with arrays of 100 elements. Second, a new set of EM traces were created with the same algorithms with the same number of elements but, with the position of the antenna changed. After conducting this experiment, the accuracy of the model was 12.75%. As can be seen in Figure 9, the accuracy of the classes is very low.

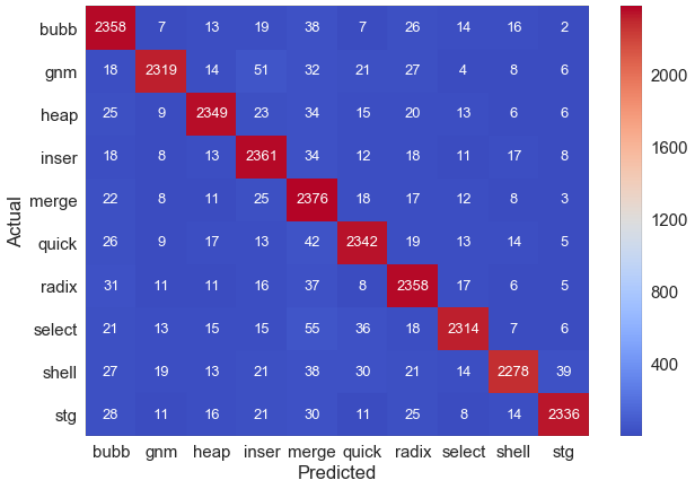


Figure 8: Confusion matrix of the first experiment.

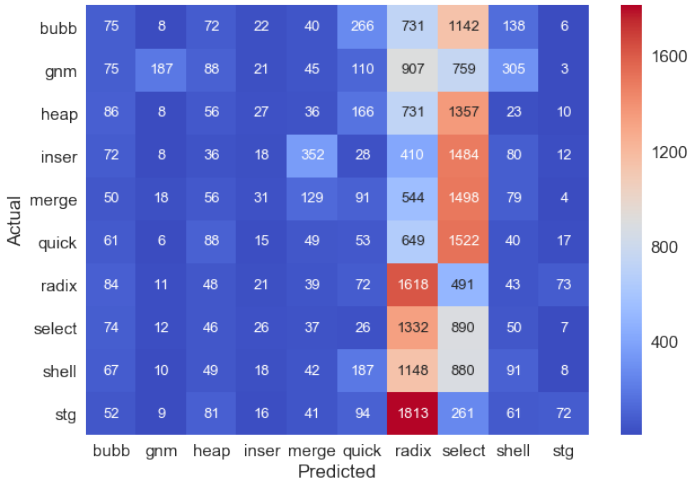


Figure 9: Confusion matrix of the first experiment.

#### 4.2.3. *Experiment III: Selecting a Machine Learning algorithm*

Experiment 1 demonstrates that machine learning models performed best on data in the frequency domain. In this experiment, different machine learning algorithms were evaluated to see which ones performed best when being trained on data in the frequency domain: Support Vector Machines, Random Forests, K Nearest Neighbour, XGBoost, Decision Trees, Random Forests, and CNN. As for the dataset, the same dataset in the frequency domain was used as in Section 4.2.1 with a training set of 25,000 examples, a test set of 25,000 examples, and the window size of 1,000 time steps.

The performance of all the models is shown in Figure 10 and the time for running the experiments with each method is shown in Table 2. From the results, the two strongest models were selected, i.e., CNN and Random Forests, to carry out the following experiments.

#### 4.2.4. *Experiment IV: Impact of sample window size*

In this experiment, the effect of choosing different window sizes for the signal. For the CNN and RF models, a list of window sizes of 500, 1,000, 2,000, 5,000, 10,000, and 15,000 time steps were chosen. Each time series of a chosen window size is converted into an input example of the same size with an equal number of features in the frequency domain using Discrete Fourier Transform (DFT).

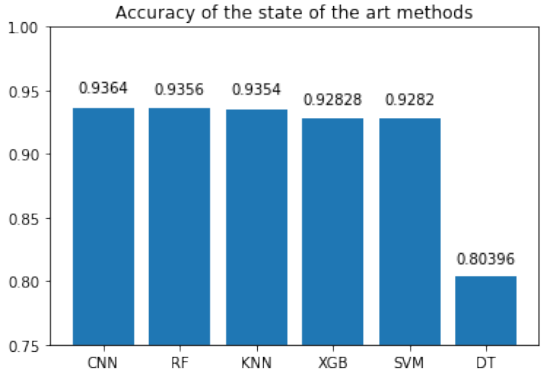


Figure 10: Performance of different ML methods trained with 25,000 samples with the ten different sorting algorithms.

Table 2: Time in seconds to build the ML model.

| XGB | SVM | DT | K-NN | RF | CNN |
|---|---|---|---|---|---|
| 402 | 1027 | 50 | 1240 | 229 | 250 |

For each window size, a training set of 25,000 examples was used from the training recording session, and a test set of 25,000 examples was used from the test recording session.

The results of the experiments are shown in Figure 11. For a small window size, i.e., the window size of equal or less than 500, RF performed better than the CNN model. For all the window sizes equal or bigger than 1,000, the CNN performed better than RF, even though the margin of improvement is small. The CNN model trained on input examples with a window size of 15,000 time steps achieved an accuracy of 99.6% on the test data.

#### 4.2.5. *Experiment V: Impact of the number of examples*

These experiments were designed to study the effect of the number of training examples on the performance of the two models. For all the experiments in this Section, the window size of one input was fixed to be the default value of 1,000: each 1,000 time step signal time series was converted to an input sample of size 1,000 in the frequency domain using DFT.

For each of the number training examples *n_train* in the list of {5,000, 10,000, 15,000, 25,000, 40,000, 80,000}, a training set of *n_train* examples was generated, while the test set was fixed to be of 25,000 examples. As shown in Figure 12, as the number of training examples varied from 5,000 to 80,000, the accuracy of the CNN model on the test set varied in the range 92.9% to 93.8%; while the CNN model performed better than RF for all values of $n_train$, with a bigger difference with a smaller number of training examples.

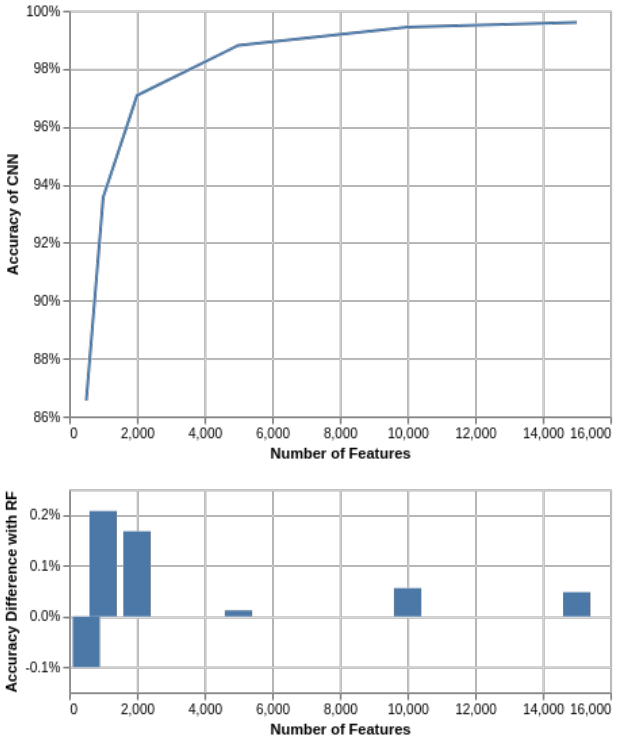


Figure 11: Experiment of varying the window size (number of features)

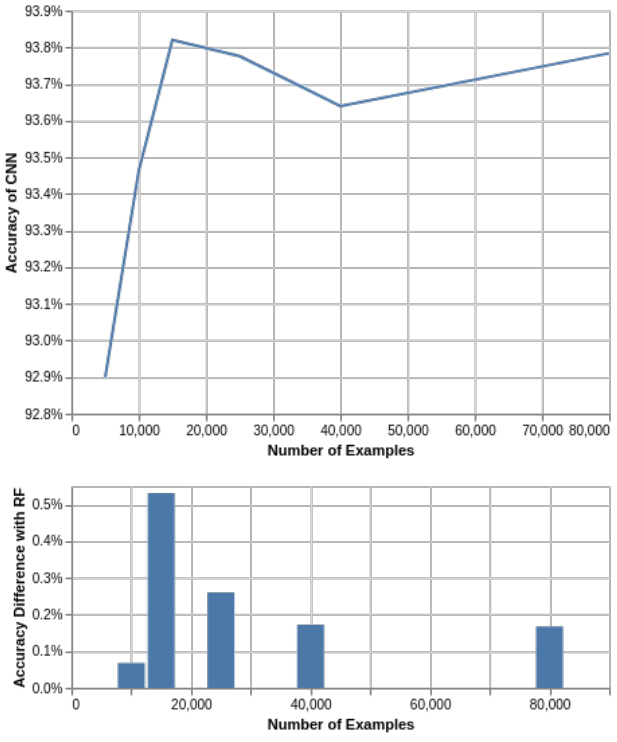


Figure 12: Experiment of varying the number of training examples

#### 4.2.6. *Performance of the best model on the whole test recording session*

The model that performed best in the experiments in Sections 4.2.4 and 4.2.5 was the CNN model with its input computed from a window of 15,000 time steps; it was trained with 25,000 examples and achieved an accuracy of 99.6% on the test set of 25,000 examples. To verify the performance of this trained model, its accuracy was computed on all the examples of the test recording session. Figure 13 shows the confusion matrix of this test, where the model achieved an accuracy of 99.8%.

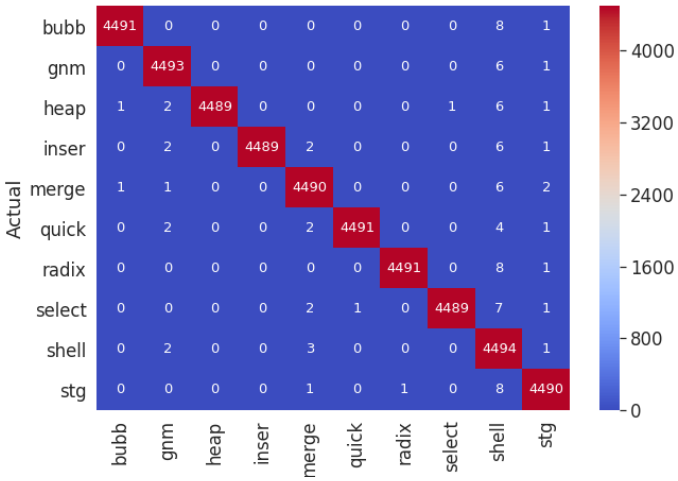


Figure 13: Confusion matrix of the model with the best performance when testing on the data from the whole test recording session.

### 4.3. *Discussion*

The experiments outlined in Section 4.2.1 demonstrate that ML models perform best when working on data in the frequency domain. One of the contributions of this work is that a consistent data recording protocol is necessary for machine learning models trained on past data to perform well on data recorded in the future. In this work, this generalisation performance was achieved by fixing the location of the antenna with respect to the Arduino device. In contrast, when the position of the antenna was moved between the two recording sessions, the model trained on the training set performed very poorly on the test data, reaching almost random performance.

The experiments on varying the window size to calculate input show that the performance of the deep learning model improved strongly when the window size was increased – going from 86.6% for a window size of 500 time steps to 99.6% for a window size of 15,000 time steps, as shown in Figure 11. On the other hand, when the window size was fixed to 1,000 and varied the number of training examples from 5,000 to 80,000 the performance of the CNN model only improved from 93.0% to 93.8%, as shown in Figure 12). For comparison, while the amount of training data required for a CNN model trained on 80,000 examples of a window size 1,000 time steps is bigger, its performance of 93.8% accuracy is inferior to the performance of 97.0% accuracy for a CNN model trained on 25,000 training examples of a window size of 2,000 time steps. It can be concluded that choosing the right window size is more impactful than increasing the number of training examples on the

performance of the ML model. For instance, a window size of $2,000$ time steps (corresponding to an accuracy of 97.0%) is a sweet spot of achieving a high predictive performance of the trained model and having a low processing time.

This work has shown that deep learning models can be used to precisely identify subtle details of running programs on IoT devices opening up the opportunity to acquire forensic insights from IoT devices during the triage examination phase of a digital forensic investigation. The trained models can be integrated with tools, such as EMvidence [36] for the application in practical forensic investigations. As it has been revealed from the experiments, the exact antenna position during data acquisition to build deep learning models should be used consistently in an investigation scenario to acquire EM data from a target device. This limit could be addressed by a more robust training data collection protocol in the future.

The high accuracy achieved with data produced by signal observation for just a few seconds hints that it is sufficient to capture data for a very short period of time in real-world scenarios.

## 5. Conclusions and Future Work

In this work, we used a wide variety of software activities as target classes for ML classification. Therefore, the dataset is sufficiently diverse in terms of computational complexity. The objective was to identify machine learning methods that are suitable to classify a wide variety of software activities running on IoT devices. The experimentation proves that it is possible to predict the activity that a device is performing from the generated EM noise with high levels of accuracy. ML classifiers are able to detect the executing algorithm even when the elements are ordered differently, which obviously requires a different order in the execution of the instructions of each sorting algorithm. Experiments have been conducted over well-known sorting algorithms to identify their performance. Among the machine learning algorithms tested, CNNs were found to be most effective in identifying individual software activities.

*Future Work:*

As future work, two interesting avenues for investigation have been identified. First, it is worth exploring different sizes for the arrays. For example, what performance is achievable with an ML model trained with arrays of 100 elements when the size of the array of the testing examples is 10, 50, 500, 1,000, or 10,000 elements. We could also create a more robust deep learning model by training it on data recorded from multiple antenna locations, such a model should be more robust with respect to the antenna location when recording test data. As the new dataset will be more complex, we expect to develop more complex deep learning architectures to fit it, which could include a combination of residual layer, attention layer, or multiple input modules to integrate multiple processing units. The logic behind the predictions of the models was not a focus in this work. It would be interesting to know what features (frequencies in this case) are more relevant for ML models. Additionally, the application of different explainable machine learning techniques [38, 39] on a trained model to understand why and how the model makes a particular decision.

## References


1. Lin J, Yu W, Zhang N, Yang X, Zhang H, Zhao W. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal* 2017;4(5):1125–1142.
2. Du X, Hargreaves C, Sheppard J, Anda F, Sayakkara A, Le-Khac NA, Scanlon M. SoK: Exploring the State of the Art and the Future Potential of Artificial Intelligence in Digital Forensic Investigation. In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. ACM. ISBN 9781450388337; 2020:.
3. Watson S, Dehghantanha A. Digital forensics: the missing piece of the internet of things promise. *Computer Fraud & Security* 2016;2016(6):5–8.
4. Yaqoob I, Hashem IAT, Ahmed A, Kazmi SA, Hong CS. Internet of things forensics: Recent advances, taxonomy, requirements, and open challenges. *Future Generation Computer Systems* 2019;92:265–275.
5. Sayakkara A, Le-Khac NA, Scanlon M. Leveraging Electromagnetic Side-Channel Analysis for the Investigation of IoT Devices. *Digital Investigation* 2019;29:S94 – S103. URL: `http://www.sciencedirect.com/science/article/pii/S1742287619301616`. doi:`https://doi.org/10.1016/j.diin.2019.04.012`.
6. Meffert C, Clark D, Baggili I, Breitinger F. Forensic State Acquisition from Internet of Things (FSAIoT): A general framework and practical approach for IoT forensics through IoT device state acquisition. In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM; 2017: 56.
7. Zankl A, Seuschek H, Irazoqui G, Gulmezoglu B. Side-channel attacks in the internet of things: Threats and challenges. In: *Solutions for Cyber-Physical Systems Ubiquity*. IGI Global; 2018:325–357.
8. Sayakkara A, Le-Khac NA, Scanlon M. Electromagnetic side-channel attacks: Potential for progressing hindered digital forensic analysis. In: *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*. ISSTA '18; New York, NY, USA: Association for Computing Machinery. ISBN 9781450359399; 2018:138–143. URL: `https://doi.org/10.1145/3236454.3236512`. doi:`10.1145/3236454.3236512`.
9. Sayakkara A, Miralles-Pechuán L, Le-Khac NA, Scanlon M. Cutting through the emissions: Feature selection from electromagnetic side-channel data for activity detection. *Forensic Science International: Digital Investigation* 2020;32:300927. URL: `https://www.sciencedirect.com/science/article/pii/S2666281720300226`. doi:`https://doi.org/10.1016/j.fsidi.2020.300927`.
10. Sayakkara A, Le-Khac NA, Scanlon M. A survey of electromagnetic side-channel attacks and discussion on their case-progressing potential for digital forensics. *Digital Investigation* 2019;29:43 – 54. doi:`https://doi.org/10.1016/j.diin.2019.03.002`.
11. Van Eck W. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security* 1985;4(4):269–286.
12. Elibol F, Sarac U, Erer I. Realistic eavesdropping attacks on computer displays with low-cost and mobile receiver system. In: *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*. IEEE; 2012:1767–1771.
13. Kocher P, Jaffe J, Jun B. Differential power analysis. In: *Advances in Cryptology (CRYPTO '99)*. Springer; 1999:789–789.
14. Nazari A, Sehatbakhsh N, Alam M, Zajic A, Prvulovic M. EDDIE: EM-Based Detection of Deviations in Program Execution. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM; 2017:333–346.
15. Maxwell JC. A dynamical theory of the electromagnetic field. *Philosophical transactions of the Royal Society of London* 1865;155:459–512.
16. Camurati G, Poeplau S, Muench M, Hayes T, Francillon A. Screaming channels: When electromagnetic side channels meet radio transceivers. In: *Proceedings of the 25th ACM conference on Computer and communications security (CCS)*. CCS '18; ACM; 2018:.
17. Peeters E, Standaert FX, Quisquater JJ. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration, the VLSI journal* 2007;40(1):52–60.

18. Tuttlebee WH. Software defined radio: enabling technologies. John Wiley & Sons; 2003.
19. Cass S. A $40 software-defined radio. *IEEE Spectrum* 2013;50(7):22–23.
20. Ossmann M. Software defined radio with hackrf. *Great Scott Gadgets, https://greatscottgadgets com/sdr* 2016;.
21. Ettus M, Braun M. The universal software radio peripheral (usrp) family of low-cost sdrd. *Opportunistic Spectrum Sharing and White Space Access: The Practical Reality* 2015;:3–23.
22. Blossom E. GNU radio: tools for exploring the radio frequency spectrum. *Linux Journal* 2004;2004(122):4.
23. Marsland S. Machine learning: an algorithmic perspective. CRC press; 2015.
24. Samuel AL. Some studies in machine learning using the game of checkers. *IBM Journal of research and development* 1959;3(3):210–229.
25. Breiman L. Random forests. *Machine learning* 2001;45(1):5–32.
26. Genuer R, Poggi JM, Tuleau-Malot C. Variable selection using random forests. *Pattern recognition letters* 2010;31(14):2225–2236.
27. Santner J, Unger M, Pock T, Leistner C, Saffari A, Bischof H. Interactive texture segmentation using random forests and total variation. In: *BMVC*. Citeseer; 2009:1–12.
28. LeCun Y, Bengio Y, Hinton G. Deep Learning. *Nature* 2015;521(7553):436.
29. Goodfellow I, Bengio Y, Courville A. Deep learning. MIT press; 2016.
30. Deng L, Yu D, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing* 2014;7(3–4):197–387.
31. Yilmaz BB, Ugurlu EM, Zajić A, Prvulovic M. Cell-phone classification: A convolutional neural network approach exploiting electromagnetic emanations. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE; 2020:2862–2866.
32. Yilmaz BB, Ugurlu EM, Prvulovic M, Zajic A. Detecting cellphone camera status at distance by exploiting electromagnetic emanations. In: *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)*. IEEE; 2019:1–6.
33. Chawla N, Singh A, Kar M, Mukhopadhyay S. Application inference using machine learning based side channel analysis. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE; 2019:1–8.
34. Ronen E, Shamir A, Weingarten AO, O'Flynn C. IoT goes nuclear: Creating a ZigBee chain reaction. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE; 2017:195–212.
35. Sayakkara A, Le-Khac NA, Scanlon M. EMvidence: A Framework for Digital Evidence Acquisition from IoT Devices through Electromagnetic Side-Channel Analysis. *Forensic Science International: Digital Investigation* 2020;32:300907. URL: `http://www.sciencedirect.com/science/article/pii/S2666281720300020`. doi:`https://doi.org/10.1016/j.fsidi.2020.300907`.
36. Sayakkara A, Le-Khac NA, Scanlon M. Facilitating Electromagnetic Side-Channel Analysis for IoT Investigation: Evaluating the EMvidence Framework. *Forensic Science International: Digital Investigation* 2020;.
37. Ponce H, Miralles-Pechuán L, Martínez-Villaseñor MDL. A flexible approach for human activity recognition using artificial hydrocarbon networks. *Sensors* 2016;16(11):1715.
38. Molnar C. Interpretable machine learning. Lulu. com; 2019.
39. Montavon G, Samek W, Müller KR. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing* 2018;73:1–15.