Radboud University Nijmegen

Faculty of Science

Backdoor attack on deep neural networks using inaudible triggers

Ultrasonic trigger inaudible to humans but not machines

Thesis BSc Computing Science

Author:
Julian van der Horst
s1015357

Supervisor:
Stjepan Picek

Daily supervisor:
Stefanos Koffas

Second reader:
Güneş Acar

August 25, 2023

Contents

Abstract

This thesis delves into the exploration of backdoor attacks in audio-based neural network applications, with a specific focus on exploiting non-linearities in MEMS microphone architectures to create triggers in audio recordings, which are inaudible to humans. We are the first to explore the use of these non-linearities to trigger a real-life backdoor. By conducting experiments on various neural network architectures, sample rates, and audio neural network applications, we investigate the success rate of backdoor attacks in digital environments and evaluate the real-life applicability of such attacks on mobile phones. Our research successfully exploits these non-linearities to create a trigger that is imperceptible to humans (twice the maximum human audible frequency) yet registers as 2 kHz tones on mobile phones.

Furthermore, we explore the STRIP defense, a runtime method designed to detect a trigger. We successfully ran the STRIP defense on a mobile phone and detected our trigger, but it comes at the cost of computational complexity. As far as we know, we are the first to try the STRIP defense on mobile phones.

Chapter 1

Introduction

The widespread adoption of neural networks has revolutionized the field of audio processing, enabling remarkable advancements in speech recognition, audio classification, and speaker identification applications [21] [12]. These AI-powered systems have found their way into our daily lives, from voice assistants on smartphones to smart home devices and security systems. However, as the reliance on these AI models grows, so does the need to ensure their security and resilience against potential attacks.

One particularly concerning threat to neural networks is the presence of backdoor attacks. Backdoors are hidden vulnerabilities deliberately injected into the model, enabling adversaries to exploit and manipulate the behavior of a neural network [19]. This thesis explores backdoor attacks in audio-based neural network applications, seeking to uncover their potential impact and vulnerabilities. More specifically, this research exploits non-linearities in MEMS microphone architectures to create a trigger that is present in recordings but nonaudible to humans. We are the first to use these non-linearities as a trigger. This trigger is created by playing two different ultrasonic tones using cheap and widely available hardware.

We conduct extensive experiments using various neural network architectures, poisoning rates, and sampling rates. We focus primarily on keyword classification but also explore the efficacy of this backdoor attack on speaker identification models. The trained models are deployed on a mobile phone using a custom-made app. By testing the models on physical devices, we assess the practical applicability and robustness of this specific backdoor attack in real-world scenarios.

Furthermore, as a possible countermeasure against this specific backdoor attack, we will explore the STRIP defense, a method that aims to detect and neutralize backdoor triggers at runtimes. We programmed STRIP so that it can run on the same phone as the TensorFlow model, which, as far as we know, we are the first to do so.

Chapter 2

Background

2.1 Automatic Speech Recognition (ASR)

Automatic Speech Recognition, otherwise known as ASR, has been around since 1952 when Bell Labs was able to recognize digits spoken over the phone [25]. Back then, analog circuitry was used to understand the incoming signal and identify a digit. The early ASR systems relied on analog circuitry to process the incoming audio signals and identify the spoken digits. However, with the advancements in deep learning, modern ASR systems have transitioned to using sophisticated neural network models to process audio data and perform speech recognition tasks.

In the current state-of-the-art ASR systems, deep learning models play a pivotal role. These models are trained on vast amounts of audio data to learn the patterns and characteristics of human speech. One widely used technique to represent audio in a form suitable for neural network training is the Mel-frequency cepstral coefficient (MFCC). MFCC, first introduced in the 1980s, remains a crucial feature extraction method in ASR due to its ability to focus on information from human speech and deemphasize other information [15] [24].

2.2 Backdoor attacks

Backdoor attacks pose a severe threat to the security and reliability of neural networks. These attacks involve inserting a specific input pattern, known as a "backdoor trigger," into the training data, leading the neural network to exhibit malicious behavior when encountering this trigger during inference. In such cases, regardless of the input features, the network will consistently produce a pre-determined label chosen by the adversary.

This behavior can be achieved either by weight poisoning [23], code poisoning [13] or data poisoning [19]. We used the data poisoning approach. Data poisoning has emerged as a popular approach due to its effectiveness and relative ease of implementation. Extensive research has been conducted in this field, validating the practicality and potential impact of backdoor attacks [22] [33] [18]. In this attack, an adversary modifies samples from the original dataset to include a trigger. The neural network then learns the association between the trigger and the targeted label, leading to the backdoor's activation when the trigger is present during inference.

In this research, we considered a backdoor to be effective when it triggers the network

to misclassify inputs with the targeted label in over 80% of cases. This way the backdoor can be used effectively for the adversary's intended purposes. To evaluate the effectiveness of the backdoor attack, we conducted extensive experiments on various neural network architectures and datasets. We elaborate more on the measurements and methods in the experimental results in chapter 4.

2.3 Microphone

The MEMS microphone is the most commonly used type in modern mobile devices. Since the creation of the MEMS microphone, it has experienced explosive growth, with one of the main benefits being a very high signal-to-noise ratio which in turn is beneficial for speech recognition [31]. The microphone output is then fed to an amplifier circuit to create a useful signal.

This hardware configuration shows non-linearities, which can be exploited as shown by [29]. This research showed that two different ultrasonic tones leave a "shadow" on the microphone. While humans cannot perceive these ultrasonic tones, the mobile device's recording hardware registers them as an audible signal. We can achieve some interesting behavior by shifting this "shadow" in the audible range. We can use this effect to create a jammer [14] to block unauthorized recordings. Or we could use this effect to reproduce voice commands, which we can use to activate Google Assistant, Amazon Alexa, Apple Siri, or others [30] [35].

In this work, we use the non-linearities in the device's recording hardware to create an inaudible trigger to activate a backdoor in a deployed neural network. We can create a trigger in the human audible range, which would then be recorded even when ultra-sonic frequencies are discarded at either the hardware level using a low pass filter or at the software level. The latter was a limitation of existing works, such as in [22]. This is because the recording of the trigger would be indistinguishable from a recording of a tone playing at the trigger frequency.

2.4 Threat model

Our attack follows a grey box data poisoning attack. This means the attackers can inject a small set of poisoned data into the training dataset. The adversary has limited or no knowledge about the specific details of the target model's architecture or training algorithm. This is a realistic scenario since many datasets rely on community efforts to be generated [32] [2].

The main goal of the attacker in this threat model is to create a trigger that exhibits high efficacy in the target dataset without significantly compromising the classification accuracy of the model on legitimate inputs. This is a critical aspect of backdoor attacks because if the trigger significantly affects the model's overall accuracy, it may draw attention and suspicion, making the attack more likely to be detected. Moreover, the trigger should be inaudible to humans but audible to a mobile phone.

To execute this attack, we assume that an attacker has a transmitter capable of producing ultrasonic frequencies, has close proximity to the transceiver, and the transceiver is a MEMS microphone. These assumptions are reasonable since the transmitter parts are widely available and cheap; MEMS microphones are very commonly used in mobile phones [31].

2.5 STRIP

A popular and promising approach to detect and mitigate backdoors at runtime against backdoors is STRIP (STRong Intentional Perturbation) [17]. The fundamental concept behind STRIP revolves around the manipulation of input data and the analysis of class prediction entropy to identify triggered inputs. STRIP takes the original input and superimposes this input with another random sample from the model's dataset. We can take this superimposed input and run this input through our neural network. We repeat this N times and calculate the entropy of the predicted classes. After superimposing the input, there is a high probability that the trigger still remains recognizable. When a neural network has been backdoored, it becomes biased to always classify inputs containing the trigger to a specific label. This will in turn result in a low entropy. For a regular non-poisoned input the reverse is true. If the input is superimposed, then the predicted classes should be more random, and thus we would have high entropy.

The effectiveness of STRIP has been demonstrated in various studies, including the influential work of STRIP-Vita researchers [16]. The researchers in STRIP-Vita explored the application of STRIP in multiple domains, including image recognition and audio recognition. In this research, we based our STRIP code on the Python code from STRIP-Vita. [16] but programmed it in Java such that the code can run on the mobile phone. This took considerable effort since Java and Python differ a lot in style, syntax, and paradigms. Writing STRIP for an Android app also came with some challenges including memory limits, random app crashes, and nondescript errors. We also needed to make minor modifications to STRIP, where the main modification is that STRIP uses a neural network that can evaluate multiple superimposed inputs at a time, while our neural network can only do one. This means that instead of evaluating the model once with, for instance, 20 superimposed samples we had to loop 20 times and run the model for each superimposed sample. We also programmed it in such a way that STRIP is simply a class and the app can be used with or without the STRIP defense. In the latest version of the app, we still have to enable it manually and set the parameters used in the defense. However, there is nothing preventing us from making these settings editable in the app itself. We provide more details in section 4.4.

Chapter 3

Experimental setup

The models are written and trained using Google Collab. The code can be found on GitHub [3]. To get a good understanding of the attack and its possible weaknesses, we have tested multiple sample rates, trigger frequencies, different objectives, and different architectures. The trained models are then converted to a TensorFlow Lite model such that the models can be loaded into an Android app and run on an actual device.

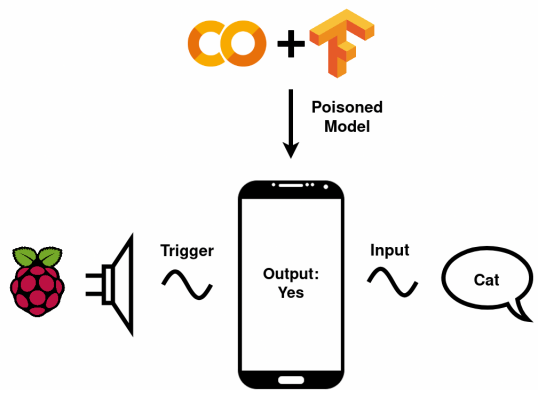


Figure 3.1: A diagram of the experimental setup

3.1 The data and parameters

For our experiments, we have opted to use two datasets. We used TensorFlow speech commands [32] for keyword classification and LibriSpeech ASR corpus train-clean-100 [26] for speaker identification. The latter was chosen because we ran into hardware limitations during training when using the complete dataset. The train-clean-100 was the largest dataset that we could get working in our current setup.

We have chosen 16 kHz for the sample rate. However, we tested the attack on the small CNN, which was trained using 8 kHz samples to check if the attack succeeded at low sample rates. Android natively [7] and ios in their standard recording app [34] allows the audio to be recorded in an AAC format. The lowest sampling rate that this format can support is 8 kHz. Another reason to test the attack at an 8 kHz sampling rate is that this sample rate is still used by telephone audio [28]. The choice for the 16 kHz sample rate has multiple reasons.

Firstly, a 16 kHz sample rate allows for a maximum trigger frequency of 8 kHz because of the Nyquist-Shannon sampling theorem. This theorem states that the highest frequency that can be captured is half of the sample rate [27]. This means we must create a trigger between 0 and 8 kHz, which we can create using the methods shown in [29]. The tones from this range will also be recorded when using higher sample rates.

Secondly, both datasets (TensorFlow speech commands and LibriSpeech ASR corpus) contained samples that were recorded at a 16 kHz sample rate. By keeping the sample rate at 16 kHz, we eliminate the need for upsampling or downsampling during preprocessing.

Thirdly, it makes training faster since less data needs to be converted to Mel-frequency cepstral coefficients (MFCCs) compared to a sample sampled at higher frequencies like 44.1 kHz. This is because to calculate the MFCCs, many computationally intensive calculations must be run on the samples [20]. A sample with a sample rate of 44.1 kHz has 44100 samples per second of audio compared to 16000 samples per second from the sample with a sample rate of 16 kHz. If we have more samples, we need to do more of these computationally intensive computations and thus have a longer execution time.

The speech commands dataset contains many 1-second clips of spoken English words. For our experiments, we have run it both using ten classes and the full 30 classes. We omitted the files that were not exactly one second, resulting in the following numbers of samples:

- 10 words: 20062 files
- 30 words: 58252 files

The LibriSpeech ASR corpus [26] is a large dataset that contains sentences taken from audiobooks from the LibriVox project. These sentences are multiple seconds long and are sampled at 16 kHz. The train-clean-100 is a smaller subset of the large model. We downloaded the dataset, randomly selected 45 speakers, and deleted the rest. For every speaker, we have cut up the multiple-second audio into 2-second audio clips. This resulted in a dataset with 45 classes and 24081 files. This dataset was chosen because it is a very popular dataset and it was easy to modify for speaker identification. The dataset was sufficient for this proof of concept.

As discussed earlier, we transformed the audio data into Mel-frequency cepstral coefficients (MFCCs) for further processing. We used the following parameters: an FFT window length of 1103 samples, 40 mel bands, and a hop length of 441 samples, and the number of mfccs returned is 40. These settings have been used to create the training data and have been used in the Android app when running the models. We chose these values since they were the same as the values used by the Android app from [22]. These settings resulted in the case of audio classification in a 40x19 multi-dimensional array at an 8 kHz sample rate, a 40 x 37 multi-dimensional array at a 16 kHz sample rate, or in the case of speaker identification a 40 x 73 multi-dimensional array representing the MFCC.

Using Matplotlib, we can visualize such an MFCC using the imgshow function. In Figure 3.2 and Figure 3.3, we show the MFCC for an audio recording of the word "six". Using an online tool we compared what features are different between the two MFCCs, these features are marked pink in Figure 3.4.

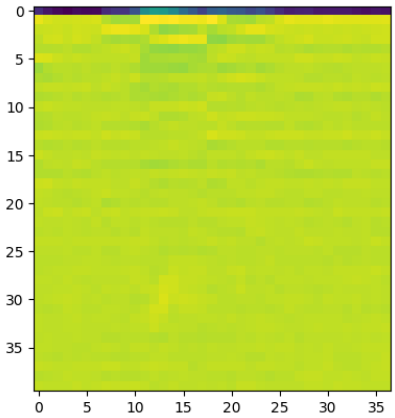


Figure 3.2: A visual representation of a regular MFCC

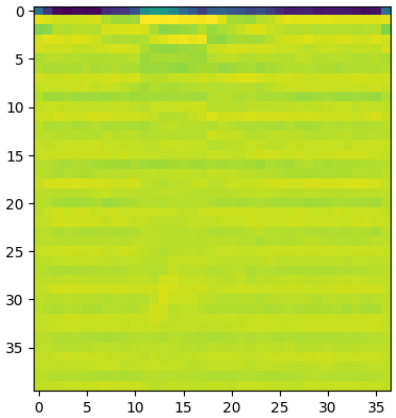


Figure 3.3: A visual representation of a poisoned MFCC

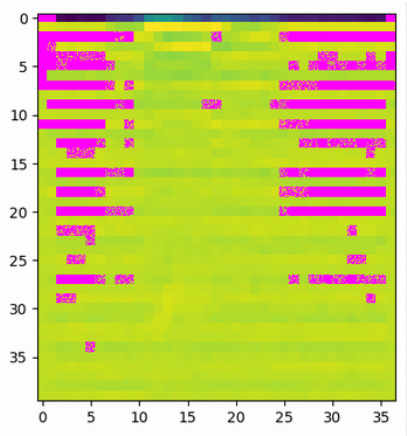


Figure 3.4: The difference between the MFCCs higlighted

For the experiments involving poisoned datasets, we chose to poison 10%, 3%, or 1% of the samples randomly. The trigger is added to the audio sample before calculating the MFCC (Mel-frequency cepstral coefficients) representation. The MFCCs, containing both poisoned and non-poisoned samples, were then split up randomly into 80% training data and 20% testing data.

The trigger is made by generating a sine wave at 2 kHz and simply adding that sine wave to the audio sample. To control the strength of the trigger, we applied a scaling factor to adjust its loudness when adding it to the audio sample. Multiplying the value of the trigger with a value between 0 and 1 during the addition allows us to adjust the loudness of the trigger to match the loudness perceived by the trigger in real-life scenarios.

To determine the scaling factor for our experimental setup, we conducted tests by recording the trigger using a regular voice recorder app, already present on the phone. We placed the phone in front of the speakers and recorded multiple clips with different levels of background noise. When placing the phone at a larger distance from the speaker a lower number was needed. Similarly, a higher number was needed when placing the

phone in its optimal position. By comparing the loudness of the trigger on the recording from the phone with the loudness of the trigger in a poisoned sample in Google Colab, we found that a scaling factor of 0.03 was sufficient for the specific setup used in the experiments.

3.2 The neural networks

In our research, we explored three different neural network architectures. We have used 2 Convolutional Neural Networks (CNN) and one Long Short-Term Memory Network (LSTM). These networks are the same ones used in [22]. All the models have the same basic code base, which is taken from the speech recognition example on the Tensorflow website [10]. This code is modified to calculate the MFCC of the (poisoned) audio sample and then add that MFCC to a test and train set. The models are made using Keras, and after training, they are converted to a Tensorflow Lite model so that they can be run on a smartphone.

For all the models, we have used sparse categorical cross-entropy as a loss function, and we have used the Adam optimizer. The CNN models were trained with a learning rate of 0.001, while the LSTM model used a learning rate of 0.0001. We used a batch size of 25 and ran it for 25 epochs. We came to these values by trying out other different values, 10 to 100 epochs and a batch size of 25 to a batch size of 100. After some trial and error, we found that these numbers were sufficient as they provided a good balance between training time and accuracy improvement.

The experiments were conducted using Google Colab, a cloud-based platform that provides access to powerful GPU resources for training deep learning models. We subscribed to the paid Colab Pro plan and used the T4 GPUs.TensorFlow version 2.8.4 was used for the experiments. The code can be found as python notebooks on Github [3].

3.2.1 The architectures

We used the same architectures that were used by [22]. In Table 3.1, Table 3.2, and Table 3.3 you can see the layers and values used. The code for these models can also be found on our GitHub [3].

type	size	field	activation
Convolutional 2D	64	kernel size(2,2)	Relu
MaxPooling 2D	64	pool size(1,3)	Relu
Convolutional 2D	64	kernel size(2,2)	Relu
MaxPooling 2D	64	pool size(1,1)	Relu
Convolutional 2D	32	kernel size(2,2)	Relu
MaxPooling 2D			
Flatten			
Fully connected	128		Relu
Fully connected	10		Softmax

Table 3.1: The small CNN architecture

type	size	field	activation
Convolutional 2D	96	kernel size(3,3)	
MaxPooling 2D		pool size(2,2)	
Convolutional 2D	256	kernel size(3,3)	
MaxPooling 2D		pool size(2,2)	
Convolutional 2D	384	kernel size(3,3)	Relu
Convolutional 2D	384	kernel size(3,3)	Relu
Convolutional 2D	256	kernel size(3,3)	Relu
MaxPooling 2D		kernel size(3,3)	
Flatten			
Fully connected	256		Relu
Fully connected	128		Relu
Fully connected	10		Softmax

Table 3.2: Big CNN architecture

Type	Size	Arguments	Activation
Conv 2D	10	5×1 filter	Relu
Batch Norm			
Conv 2D	1	5×1 filter	Relu
Batch Norm			
BLSTM	64	return_sequences=True	Tanh
BLSTM	64	return_sequences=True	Tanh
Attention			
Dense	64		Relu
Dropout	0.5		
Dense	32		Relu
Dense	10 or 30		softmax

Table 3.3: LSTM architecture

3.3 The signal-producing device

The trigger for our attack is generated as a two-channel WAV file with a sample rate of 192 kHz, which we play on our sound-producing device. We used the command line application Sox to create this file, where we used the following command:

```
sox -V -r 192000 -n -b 16 -c 2 tone.wav synth 30 sin 38000 sin 40000 vol
    -2dB
```

This argument creates a 16-bit, 2-channel WAV file using a sample rate of 192 kHz. The two channels contain 30-second-long sine waves; on the left channel, we have a 38 kHz sine wave, and on the right channel, we have a 40 kHz sine wave. These frequencies are well beyond the range of human hearing but when played at the same time will result in a trigger of 2 kHz when recorded on the phone. We also decreased the volume by 2dB to ensure that the sine waves do not clip in the recording.

That wav file is then played on a Raspberry Pi 3B [8], which was equipped with a HifiBerry DAC+ ADC [5]. This soundcard is capable of playing audio at a sample rate of 192 kHz, which is required for the trigger. The DAC's output is then connected to two Ultrasonic speakers, specifically TCT40-16 speakers, which were repurposed from an Ultrasonic distance sensor. The Ultrasonic distance sensor can easily be found online since it is used in many hobby project kits.

Although the output signal is not particularly strong, it can still produce a relatively

strong tone at close range, approximately 10 cm. An amplifier circuit would be required if we want to have the tone audible for a longer distance, but from our testing, finding an amplifier that operates at such high frequencies can be quite a challenge. Perhaps other or even more speakers would also increase the range. Our experiences with hardware are limited, and thus we decided that the short range is one of the limitations of the attack.

The real-life experimental setup is depicted in Figure 3.5, where we can see from left to right, the ultrasonic speakers, a breakout board to split the RCA audio cables into separate wires, and the Raspberry Pi with the HifiBerry DAC. The speakers are on a prototype breadboard since it makes it easier to monitor the incoming signals when using an oscilloscope. We also found the speakers easy to puncture so to replace them more quickly we kept the prototype breadboard.

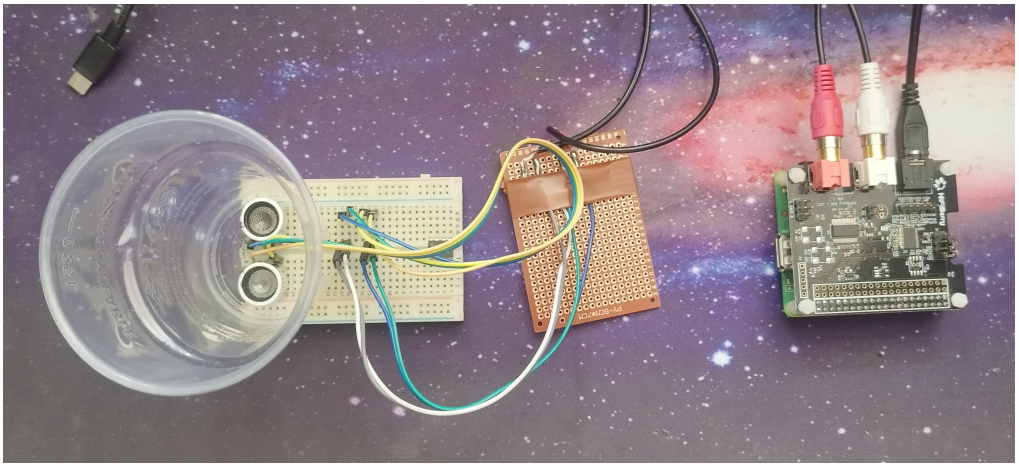


Figure 3.5: The sound-producing setup

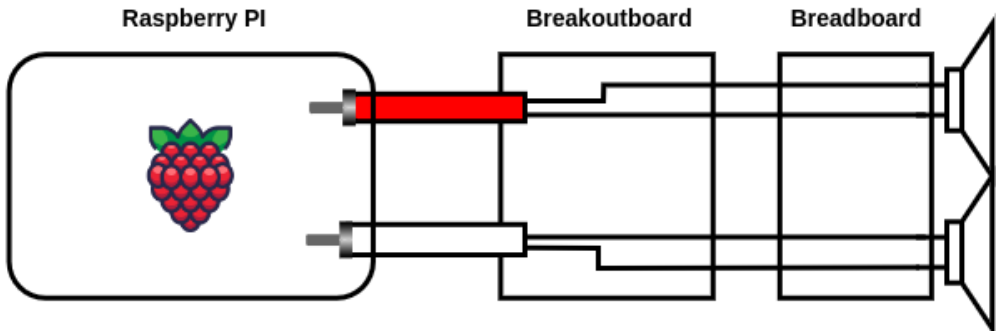


Figure 3.6: A diagram of the setup

3.4 The speech recognition app

The speech recognition app serves as a platform to test the trained models on an actual Android device. To actually use these models on a mobile phone we use TensorFlow Lite. TensorFlow Lite is a mobile library that is used to deploy trained models on mobile hardware, such as mobile phones [11]. The app is built for Android and uses the same library, JLibrosa, to calculate the MFCC of the audio. JLibrosa is equivalent to the Python Librosa library used during the training process. The same parameters from the training process are used when calculating the MFCCs of the audio samples within the app. Examples of the interface of the app can be seen in Figure 3.7.

When the record button is pressed, the app will record one or two seconds of audio from the phone's microphone, depending on whether we are testing the Audio classification or Speaker identification task. After the recording, the app will take the sample and calculate the samples MFCC using the JLibrosa library[6]. Subsequently, the MFCCs are passed through the loaded TensorFlow Lite model for prediction. The results are then sorted based on their scores, and the class with the highest score is displayed on the screen as the prediction output. We also record the inference time and display it underneath the screen, providing insights into the speed of the prediction process.

To streamline the experimentation process, the app features a dropdown menu that enables us to choose which model we want to test. The names of these models correspond to the parameters used during training, such as the number of commands, sample rate, and the percentage of poisoned samples. This can be seen in Figure 3.8.

Moreover, the app incorporates the STRIP defense, which is further elaborated in section 4.4. For this defense, we added the "copy to the clipboard" button and text showing whether a sample is poisoned or not. The button is used to export the array containing the entropy values generated by the STRIP defense.

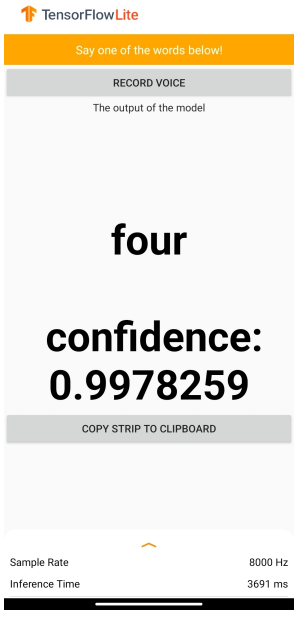


Figure 3.7: The TensorFlow lite app

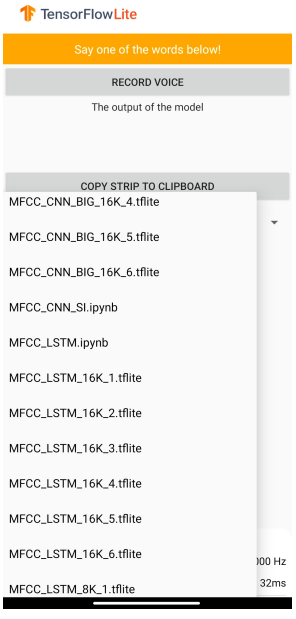


Figure 3.8: The TensorFlow lite app with dropdown

3.5 The phone

The phone used throughout the experiments is the following [9]

Name: Redmi note 11 Pro 5G
OS: MIUI 13, Android 12
CPU: Snapdragon 695 Octa-core CPU, 2,2GHz
GPU: Qualcomm® Adreno™ 619
RAM: 6G + 2G swap LPDDR4X + UFS2.2

The trigger frequency has been tested on other devices, and the results can be reproduced on other mobile phones. These phones include:

- Oneplus 6T
- iPhone 13

This shows that the trigger is likely to be present on most modern devices, regardless of price class, operating system or build year.

3.6 Trigger strength

We tested the strength of the trigger on two phones which were available to us. The trigger is played as described in section 3.3 but at different ranges. To measure the loudness of the trigger, we used the Audio Spectrum Analyzer Pro [1] app on the iPhone and the Spectroid[4] app on the Android-based Xiaomi device. Both apps used the following settings:

Sampling rate 96000 Hz
FFT size 512 bins
Window function Blackman-Harris

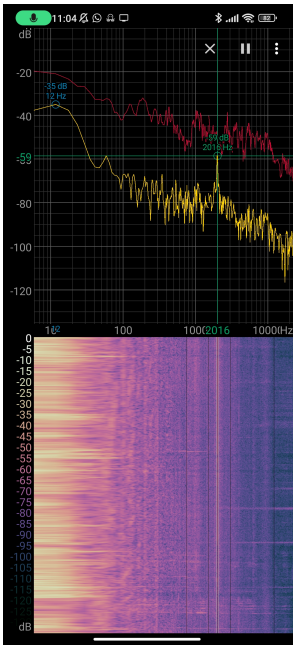


Figure 3.9: Spectroid with the 2 kHz trigger present

We have tested various frequencies, but we found that the frequency does not significantly impact the loudness.Figure 3.9 shows a 2 kHz trigger on the Xiaomi phone created by two small ultrasonic speakers without an amplifier.

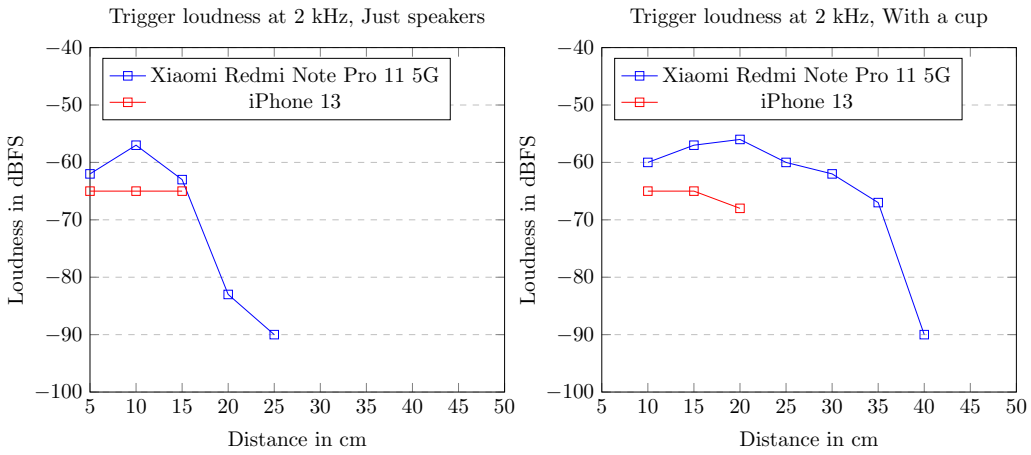
To assess the strength of the trigger at different distances, we placed the phone at a measured distance and adjusted its position to obtain the highest dBFS reading. To ensure the accuracy of the readings, we held the phone at that distance for 5 seconds, recording only the readings that remained consistent during this time. These measurements were performed in a normal office environment with minimal background noise. We continued measuring in 5 cm intervals until the trigger could no longer be detected above the noise level.

In our attempts to extend the range of the trigger, we experimented with various designs, including 3D-printed cones. However, the most effective design was achieved by using a simple plastic cup with the bottom cut out. This method increased the range by approximately 15 cm. It is worth mentioning that the cup prevented the phone from being held at the 5 cm distance, as the cup's height was 10 cm.

Chapter 4

Experimental Results

4.1 Trigger strength



The graphs of the loudness and distance of the trigger's audibility show that the loudness and the distance that the trigger can be heard are greater on the Xiaomi than on the iPhone. On the iPhone, we could not detect the trigger above the noise floor when the distance was larger than 20cm, whereas, on the Xiaomi, we could see the trigger with distances as far as 40cm. Several factors could contribute to this difference, such as variations in the operating systems, different noise reduction algorithms, and possibly cheaper microphone components in the Xiaomi phone. Interestingly, we found that on the iPhone, the trigger instantly disappeared in the noise floor. Whereas the Xiaomi phone first decreased in trigger strength and then disappeared.

4.2 Audio classification

We experimented with different architectures, sample rates, and real-life and digital performance. The accuracy column in the graph is the accuracy of the models that TensorFlow reported when running the evaluate function on the model. We trained every model three times to get an average accuracy. To calculate the digital backdoor accuracy we took 100 random samples and added the trigger to them. If the backdoor worked successfully, we recorded it as a success and computed the overall backdoor

accuracy by dividing the total number of successful backdoor attempts by the total number of samples tested. We tested using 100 random samples since we found that this gave us a good balance between time spent calculating and accuracy.

In addition to digital evaluation, we also conducted physical backdoor accuracy tests by running the models on a mobile phone. For each model, we tested it on ten randomly chosen words and repeated this process three times. It is important to note that the physical backdoor accuracy can vary significantly from the digital accuracy. We observed that the distance and orientation of the phone during the test had a substantial impact on the results. To mitigate this, we stabilized the phone's position as much as possible during the tests using tape.

type	poisoned	size	accuracy	digital backdoor accuracy	real life backdoor accuracy
Small CNN	No	10 commands	85%	X	X
Small CNN	Yes, 10%	10 commands	84%	99%	99%
Small CNN	No	30 commands	85%	X	X
Small CNN	Yes, 10%	30 commands	85%	100%	99%
Small CNN	Yes, 3%	30 commands	84%	97%	100%
Small CNN	Yes, 1%	30 commands	77%	92%	70%

Table 4.1: Resulting models at 16 kHz samplerate

type	poisoned	size	accuracy	digital backdoor accuracy	real life backdoor accuracy
Small CNN	No	10 commands	85%	X	X
Small CNN	Yes, 10%	10 commands	84%	99%	80%
Small CNN	No	30 commands	80%	X	X
Small CNN	Yes, 10%	30 commands	80%	99%	77%
Small CNN	Yes, 3%	30 commands	80%	96%	80%
Small CNN	Yes, 1%	30 commands	80%	92%	27%

Table 4.2: Resulting models at 8 kHz samplerate

Table 4.1 shows the results obtained from experiments conducted at a sample rate of 16 kHz. We used a Small Convolutional Neural Network (CNN) for audio classification with two different setups: one with 10 spoken word commands and another with 30 spoken word commands.

We can also observe that the introduced trigger does not significantly impact the accuracy of the model. The accuracy remains around 85% when we look at the 10% and 3% poisoning rates. However, the accuracy does drop when using the 1% poisoning rate. Not only digitally but also physically the attack succeeds with very high accuracy. In real-life testing on a mobile phone, the attack efficiency, similarly to the accuracy, only drops significantly at the 1% poisoning level.

When increasing the number of classes from 10 to 30, we can observe a lower accuracy. However, the attack accuracy still remained high, and the attack still succeeded. In real-life scenarios, the backdoor attack's accuracy varied but remained relatively high, demonstrating robustness across different poisoning levels. With the model, where we poisoned 3% we can see that the real-life accuracy is higher than the digital. However,

this can probably be attributed to the relatively low amount of tested samples in real life.

At the lower sample rate, the accuracy of the non-poisoned models remained similar to the 16 kHz experiments, reaching around 85% for both setups with 10 and 30 spoken word commands. This is interesting since at the 8 kHz sample rate there are half the samples in a 1-second clip, compared to the 16 kHz sample rate. which in turn makes the MFCCs half the size. We can however see a stark drop in backdoor accuracy in real life, which is concidirably lower than the digital accuracy. This is probably because the trigger might be less prevalent in the MFCCs since the MFCC itself is much smaller. Similar to the results from Table 4.1 we can see a big drop in real-life backdoor accuracy at the 1% poisoning rate. However, with 8 kHz this drop is considerably larger, and the attack barely succeeds.

type	poisoned	size	accuracy	digital backdoor accuracy	real life backdoor accuracy
Big CNN	No	10 commands	95%	X	X
Big CNN	Yes, 10%	10 commands	94%	99%	100%
Big CNN	No	30 commands	94%	X	X
Big CNN	Yes, 10%	30 commands	95%	100%	100%
Big CNN	Yes, 3%	30 commands	94%	100%	100%
Big CNN	Yes, 1%	30 commands	94%	98%	97%

Table 4.3: Resulting models at 16 kHz samplerate

Table 4.3 presents the experimental results for the Big Convolutional Neural Network (CNN) at a sample rate of 16 kHz. Similar to the Small CNN experiments, we evaluated the Big CNN model's accuracy for two different setups: one with 10 spoken commands and another with 30 spoken commands.

The Big CNN achieved higher accuracy than the Small CNN, reaching around 95% for both setups with 10 and 30 spoken commands when not poisoned. This is to be expected since the big CNN has much more trainable parameters and also takes longer to train. The backdoor also triggered with high accuracy, even with 3% or 1% poisoning. This high backdoor accuracy demonstrates the effectiveness and robustness of the backdoor attack on the Big CNN model.

type	poisoned	size	accuracy	digital backdoor accuracy	real life backdoor accuracy
LSTM	No	10 commands	72%	X	X
LSTM	Yes, 10%	10 commands	74%	100%	33%
LSTM	No	30 commands	68%	X	X
LSTM	Yes, 10%	30 commands	70%	100%	13%
LSTM	Yes, 3%	30 commands	70%	98%	7%
LSTM	Yes, 1%	30 commands	68%	94%	3%

Table 4.4: Resulting models at 16 kHz samplerate

Table 4.4 presents the experimental results for the Long Short-Term Memory Networks (LSTM) at a sample rate of 16 kHz. As with the other architectures, we tested the LSTM model's performance for two different setups: one with 10 spoken word commands and another with 30 spoken word commands.

The LSTM model achieved lower accuracy compared to both the Small CNN and Big CNN models, reaching around 72% for the 10 command setup and 68% for the 30 command setup when not poisoned. These numbers are quite low and perhaps with some tweaking of the input data and more training time this can be improved. The backdoor attack was effective in digital environments. However, when trying the backdoor in real life, the accuracy fell starkly. For lower poisoning levels, the real-life backdoor accuracy dropped even further, indicating that the LSTM model was more challenging to backdoor effectively in real-life environments.

Overall, the experimental results highlight the differences in backdoor attack effectiveness between different neural network architectures, sample rates, and poisoning levels. The attack is very successful, even in real life, when the model used is a CNN. For the LSTM the attack did not seem to work as well, and more research needs to be done. The improvements that we believe will make the biggest impact on the LSTM performance are changing the parameters of the MFCC to create bigger MFCCs and changing the number of epochs in the training phase.

4.3 Speaker recognition

From this research, we also wanted to see if the attack would succeed in speech recognition and speaker identification applications. If the attack succeeds in a speaker identification application, then an adversary could use this attack to pretend to be someone else or bypass security if speaker identification is used as a security measure to verify a person's identity. We used the LibriSpeech dataset [26] to train the models. We tried all the architectures that were previously mentioned in this paper in Table 3.1, Table 3.2, and Table 3.3 to see if we would find any differences in performance.

type	poisoned	size	accuracy	backdoor accuracy
Small CNN	No	45 commands	98%	X
Small CNN	Yes, 10%	45 commands	98%	100%
Small CNN	Yes, 3%	45 commands	98%	99%
Small CNN	Yes, 1%	45 commands	98%	97%
Big CNN	No	45 commands	98%	X
Big CNN	Yes, 10%	45 commands	98%	100%
Big CNN	Yes, 3%	45 commands	98%	98%
Big CNN	Yes, 1%	45 commands	98%	98%
LSTM	No	45 commands	77%	X
LSTM	Yes, 10%	45 commands	77%	98%
LSTM	Yes, 3%	45 commands	74%	97%
LSTM	Yes, 1%	45 commands	74%	47%

Table 4.5: Resulting models at 16 kHz samplerate

Some interesting findings followed from the experiments shown in Table 4.5. Firstly, the high accuracy. We believe that this is because of the dataset used. The samples are recorded by volunteers who have widely different environments and recording equipment. We believe the neural network uses these features to identify the speakers.

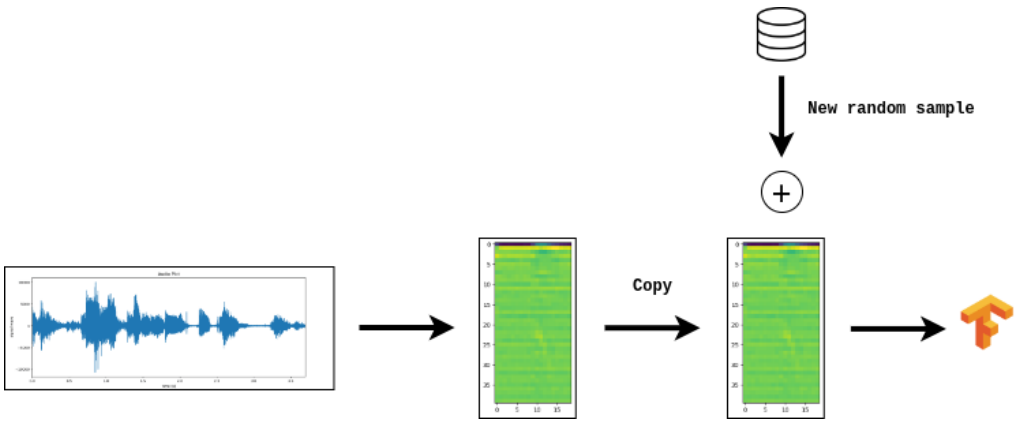
Secondly, when we tested this model in real life by playing the LibriSpeech samples using a laptop, we got very strange results. The model did not classify the audio correctly, and

the predictions from the model appeared random. The model also sometimes seemed to predict the same class regardless of the sample played. Many of these predictions also had a very high confidence score. Placing the phone at various distances and increasing the laptop's volume did not seem to resolve this strange behavior. When assessing the real-life backdoor performance, we found that the trigger does show up on the phone, but it did not affect the predictions at all. We even tried playing a 2 kHz tone very loudly on regular laptop speakers, but this also did not seem to have any effect. Because of these problems, we could not get any data on the real-life performance of the backdoor.

The code between the audio classification and speaker identification applications is very similar. The main change is the increased recording length, which made the MFCCs twice as large and required us to increase array sizes. We believe the problem has its roots in the training phase or even in the Android app. However, more research needs to be done on these findings.

4.4 STRIP

We looked at the STRIP defense [17], more specifically the STRIP-Vita defense [16], which is an extended version of STRIP to handle Vision, Text, and Audio. We wanted to know if this new way of triggering a backdoor would perhaps resist some defenses. STRIP was chosen because of its popularity, high accuracy, and relatively simple implementation. As far as we know, we are the first to run STRIP on mobile hardware.



In the same Google Collab files, we wrote an extra section that, after calculating all the testing data MFCCs, exports them to a JSON file. This file is rather significant, and when loading and parsing this file on the phone, we ran into memory problems. To quickly alleviate this problem, we wrote a Java program to convert this JSON file to a native Java float array. We then exported this array to a file and transferred it onto the phone. This way the file was smaller and did not require any complex parsing to load into memory, significantly reducing memory usage. The STRIP class is instantiated on loading the app and the MFCCs are loaded into memory. If the user records some audio, we calculate the MFCC of that audio, which we will then superimpose by adding a random MFCC from our dataset to the recorded audio MFCC.

This superimposed MFCC is then evaluated by running it through the TensorFlow Lite model to obtain the confidence scores for each class prediction. We repeat this step 50 times to get a distribution of entropy values. Ideally, we want as much data as we

can get. However, this would also increase the time spent computing. These results numbers gave us a good tradeoff of being relatively fast and providing enough data to distinguish poisoned and not poisoned input. We then do this whole process another 50 times to get an array of entropies. The superimposing is done by simply iterating over the rows and columns of the MFCC, adding up the values of the two MFCCs at that position. When using STRIP, the evaluation of a recording takes 5 seconds instead of 32ms, which is a sizeable difference. The most time is taken up by the model evaluating the superimposed samples, which we observed by running the profiler from Android Studio and investigating the call stack. The smaller CNN takes an average of 7ms to evaluate a sample, whereas the big CNN takes 123 ms and the LSTM 36 ms. Depending on our use case we would need to make a lot of evaluations, which drastically improve the time it takes to run STRIP. The other steps, like superimposing, took between 0 and 1 ms.

Now that we have an array of entropies we can see whether an input was poisoned by the mean and standard deviation. Figure 4.1 and Figure 4.2 show a graph of the entropy. Characteristics of the triggered input are the low mean and low standard deviation. Currently, in the app, we say that an input is poisoned if either the standard deviation is below a threshold or the mean is below a threshold. This gave us the best results but also gave us false positives. Depending on our use case we can make the threshold more strict or loose. With a stronger trigger, we mean that the trigger is louder or more present in the sample; this would make it more likely to be still present after the superimposing. Even if the malicious input did not give us the backdoor class but a different class, we could still detect if the input contained a trigger since maybe in this evaluation the trigger did not show up significantly, but in a superimposed MFCC, it would.

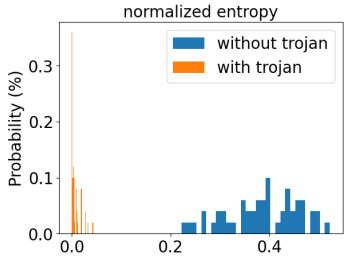


Figure 4.1: Entropy when the trigger is very strong

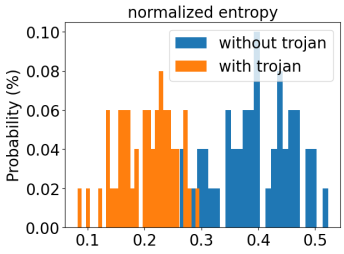


Figure 4.2: Entropy when the trigger is very weak

Figure 4.3: The app showing a poisoned sample

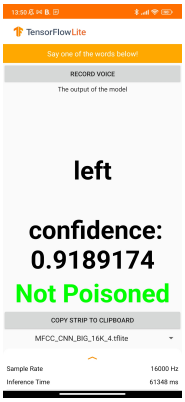


Figure 4.4: The app showing a non-poisoned sample

In our evaluation of the STRIP defense, we experimented with various thresholds, where we tried to balance detection accuracy and computational efficiency. This research into the STRIP defense was mainly to see whether it could defend against the ultrasonic trigger. We found that it could, but it is highly dependent on our use case. We can detect fairly well whether a sample has been poisoned. However, more research has to go into finetuning the threshold and the number of evaluations to achieve higher accuracy. We did not thoroughly research STRIP, but from the data that we could gather, we had similar detection accuracies as the STRIP [17] or STRIP-VITA[16] research, indicating that the defense worked against our attack.

Chapter 5

Conclusion

In this thesis, we explored the potential security risks associated with backdoor attacks in neural network-based audio classification and speaker identification applications. We investigated the feasibility of creating a trigger, which is inaudible to humans but would still be efficient and successful. To create this trigger, we exploited non-linearities in MEMS microphone hardware configurations. This allowed us to create a trigger of 2 kHz by playing two ultrasonic sounds well beyond human hearing. We are the first to use this technique to create a stealthy trigger for a backdoor. The setup only featured two small ultrasonic speakers and had no amplifier; this resulted in a small range, which is a limitation of the attack. We tested the presence of the trigger on multiple phones, which all showed the trigger, albeit at different loudness levels.

Through extensive experiments on various neural network models, including Small CNNs, Big CNNs, and LSTMs, we obtained insightful results. In the context of audio classification, our experiments revealed that backdoor attacks could be highly effective in digital environments. The attack success rate was consistently high, even with low poisoning levels of 1% in some architectures. Meanwhile, the trigger was completely inaudible and worked even when using a very low sample rate, like 8 kHz. However, we also identified the need for further research, as the real-life backdoor accuracy could vary significantly from the digital backdoor accuracy, especially in certain neural network architectures like an LSTM. Our experiments demonstrated that the choice of neural network architecture played a crucial role in the backdoor accuracy, with CNN-based models showing greater vulnerability compared to LSTM models in real-life scenarios.

Furthermore, we extended our investigation to speaker recognition applications using the LibriSpeech dataset. The speaker identification models achieved remarkable (digital) accuracy, but we observed challenges in real-life scenarios. In real life, the models behaved inconsistently, and testing speaker identification was not possible. While the backdoor attacks were successful in digital environments, the backdoor could not be triggered at all in real life.

Moreover, as part of our research, we explored the STRIP defense method. STRIP aims to identify and mitigate backdoor attacks in neural network models by analyzing class prediction entropy. We wanted to see if STRIP would still be a good defense for this particular attack. We have adapted the STRIP code to be able to run on a mobile phone. To our knowledge, we are the first to have the STRIP defense running on a mobile phone. The STRIP defense showed promising results in detecting and neutralizing the backdoor triggers, albeit at the cost of a significantly increased processing time.

In conclusion, backdoor attacks pose significant security risks in audio classification and speaker recognition applications. While the digital backdoor accuracy remains high, various factors can affect real-life performance, including recording environments and the choice of neural network architecture. These findings show that the attack is very promising, but more research still has to be done.

Chapter 6

Future work

During our research on backdoor attacks in neural networks, we primarily focused on a single sine wave trigger, which remained consistent across all our experiments. However, to further investigate potential ways to bypass the STRIP defense we can deviate from this. The sound-producing device can produce a wide range of frequencies, so we can very easily extend our attack by using more than one trigger. We took some time to explore this area, but we could not fully explore this type of trigger because of time constraints. In our research, we randomly chose one of the four trigger frequencies to append to each sample during the creation of the poisoned dataset. We found that the attack still worked, but the accuracy was lower.

More research can go into the trigger frequency used. Nothing is holding us to the 2 kHz sine wave that we used during these experiments. Perhaps the attack will have a higher success rate using a different frequency. The sound-producing device can not only produce sine waves but also triggers in the form of spoken commands or specific audio patterns, which is demonstrated in [30] [35]. Perhaps even these triggers will have a higher success rate.

The trigger strength hugely impacts the attack success rate, so more research into amplifying the trigger can have a considerable impact. We experimented by using some amplification chips, but when inspecting their output with an oscilloscope, we found that the output signal was no longer the ultrasonic frequency we put in. This area lies outside our expertise, and despite our efforts, we could not find a suitable amplifier chip for our specific needs. For those with expertise in amplifier design, exploring the creation of a custom amplifier for the trigger frequencies could be interesting.

Lastly, more research and experiments could be done to explain strange behavior when testing our models for speaker recognition. The problem might lie with the app, or it could be something fundamental when training the models. An excellent first step would be to see if this behavior can be demonstrated digitally. This can be achieved by playing a sample from the laptop and recording a 2-second clip on the mobile phone. Then, you can import this sample into Google Colab to see if the same behavior occurs.

In general, it is important to emphasize that our research primarily served as a proof-of-concept to demonstrate the feasibility of the ultrasonic trigger-based backdoor attack. While we successfully executed the attack in controlled settings, additional research and testing are needed to perform this attack in real-life scenarios.

Bibliography

[1] Apple app store - Audio Spectrum Analyzer Pro. https://apps.apple.com/us/app/audio-spectrum-analyzer-pro/id1152352806. Accessed: 2023-05-17.

[2] Common voice Mozilla. https://commonvoice.mozilla.org/en. Accessed: 2023-04-25.

[3] Github — Bachelor Scriptie. https://github.com/Gulianrdgd/Bachelor-scriptie. Accessed: 2023-04-25.

[4] Google play store - Spectroid. https://play.google.com/store/apps/details?id=org.intoorbit.spectrum. Accessed: 2023-05-17.

[5] HiFiBerry DAC+ ADC. https://www.hifiberry.com/shop/boards/hifiberry-dac-adc. Accessed: 2023-02-27.

[6] jlibrosa github. https://github.com/Subtitle-Synchronizer/jlibrosa. Accessed: 2023-05-16.

[7] MediaRecorder — Android developer. https://developer.android.com/reference/android/media/MediaRecorder#setAudioSamplingRate(int). Accessed: 2023-05-21.

[8] Raspberry Pi 3 Model B. https://www.raspberrypi.com/products/raspberry-pi-3-model-b/. Accessed: 2023-02-27.

[9] Redmi Note 11 Pro 5G. https://www.mi.com/nl/redmi-note-11-pro-5g/. Accessed: 2023-02-27.

[10] Tenserflow speech recognition example. https://www.tensorflow.org/lite/models/modify/model_maker/speech_recognition. Accessed: 2023-04-25.

[11] TensorFlow Lite - ML for mobile and edge devices. https://www.tensorflow.org/lite. Accessed: 2023-08-18.

[12] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014.

[13] Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1505–1521. USENIX Association, August 2021.

[14] Yuxin Chen, Huiying Li, Steven Nagels, Zhijing Li, Pedro Lopes, Ben Y. Zhao, and Haitao Zheng. Understanding the effectiveness of ultrasonic microphone jammer. *CoRR*, abs/1904.08490, 2019.

[15] Namrata Dave. Feature extraction methods lpc, plp and mfcc in speech recognition. *International journal for advance research in engineering and technology*, 1(6):1–4, 2013.

[16] Yansong Gao, Yeonjae Kim, Bao Gia Doan, Zhi Zhang, Gongxuan Zhang, Surya Nepal, Damith Chinthana Ranasinghe, and Hyoungshick Kim. Design and evaluation of a multi-domain trojan detection method on deep neural networks. *CoRR*, abs/1911.10312, 2019.

[17] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith Chinthana Ranasinghe, and Surya Nepal. STRIP: A defence against trojan attacks on deep neural networks. *CoRR*, abs/1902.06531, 2019.

[18] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[19] Wei Guo, Benedetta Tondi, and Mauro Barni. An overview of backdoor attacks against deep neural networks and possible defences. *IEEE Open Journal of Signal Processing*, 3:261–287, 2022.

[20] Wei Han, Cheong-Fat Chan, Chiu-Sing Choy, and Kong-Pang Pun. An efficient mfcc extraction method in speech recognition. In *2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4 pp.–, 2006.

[21] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[22] Stefanos Koffas, Jing Xu, Mauro Conti, and Stjepan Picek. Can you hear it? backdoor attacks via ultrasonic triggers. *CoRR*, abs/2107.14569, 2021.

[23] Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pretrained models. *CoRR*, abs/2004.06660, 2020.

[24] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE Access*, 7:19143–19165, 2019.

[25] Douglas O'Shaughnessy. Invited paper: Automatic speech recognition: History, methods and challenges. *Pattern Recognition*, 41(10):2965–2979, 2008.

[26] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[27] Emiel Por, Maaike van Kooten, and Vanja Sarkovic. Nyquist–shannon sampling theorem. *Leiden University*, 1(1), 2019.

[28] L.R. Rabiner and R.W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall signal processing series. Prentice-Hall, 1978.

[29] Nirupam Roy, Haitham Hassanieh, and Romit Roy Choudhury. Backdoor: Making microphones hear inaudible sounds. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, page 2–14, New York, NY, USA, 2017. Association for Computing Machinery.

[30] Liwei Song and Prateek Mittal. Poster: Inaudible voice commands. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2583–2585, New York, NY, USA, 2017. Association for Computing Machinery.

[31] Zhe Wang, Quanbo Zou, Qinglin Song, and Jifang Tao. The era of silicon mems microphone and look beyond. In *2015 Transducers - 2015 18th International Conference on Solid-State Sensors, Actuators and Microsystems (TRANSDUCERS)*, pages 375–378, 2015.

[32] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR*, abs/1804.03209, 2018.

[33] Jinwen Xin, Xixiang Lyu, and Jing Ma. Natural backdoor attacks on speech recognition models. In Yuan Xu, Hongyang Yan, Huang Teng, Jun Cai, and Jin Li, editors, *Machine Learning for Cyber Security*, pages 597–610, Cham, 2023. Springer Nature Switzerland.

[34] Jinhua Zeng, Qiuxiu Lian, and Shaopei Shi. Forensic originality identification of iphone's voice memos. *Journal of Physics: Conference Series*, 1345(5):052053, nov 2019.

[35] Guoming Zhang, Chen Yan, Xiaoyu Ji, Taimin Zhang, Tianchen Zhang, and Wenyuan Xu. Dolphinatack: Inaudible voice commands. *CoRR*, abs/1708.09537, 2017.