Towards Efficient Deep Learning Based Siren Detection.

Towards Efficient Deep Learning Based Siren Detection.

Master Thesis

Bollapragada Lalitha Sai Shraddha

Delft University of Technology

Towards Efficient Deep Learning Based Siren Detection.

by

Bollapragada Lalitha Sai Shraddha

Student Name	Student Number
Bollapragada Lalitha Sai Shraddha	5274081

Company Supervisor:	Andreas Lenz
University Supervisors:	Qun Song , K.G Langendoen
Project Duration:	Dec, 2023 - July, 2024
Faculty:	Faculty of EEMCS, Delft

Preface

This thesis represents the culmination of my research and development work over the past 6 months in the field of real-time embedded neural network-based audio classification systems. My primary goal has been to develop a reliable and effective model that can accurately detect emergency response vehicle sirens amidst a mixture of urban sounds. This work is not only an academic quest but also a step towards enhancing public safety and improving emergency vehicle response times.
Embarking on this research journey, I have navigated a steep and enlightening learning curve. As an embedded engineer fascinated by artificial intelligence, I began with limited knowledge of machine learning and Python. Driven by the potential of integrating embedded systems with AI, I dedicated myself to mastering these skills. This project has provided the perfect opportunity to combine my expertise in embedded engineering with my passion for AI, resulting in an intellectually stimulating and rewarding process. I am deeply grateful for the guidance and support of my NXP supervisors, Andreas Lenz and Dr. Bruno Defraene, whose expertise and encouragement have been invaluable. I extend my heartfelt thanks to my TU Delft supervisors, Prof. dr. Koen Langendoen and Prof. dr. Qun Song, for their critical insights, timely guidance throughout this project, and support in supervising me remotely. Their contributions have been crucial in overcoming the many technical challenges encountered during this journey.
I want to thank my family in particular for their patience and unwavering support during this demanding process. I express my deepest gratitude to my father Sairam Bollapragda, mother Bhanurekha Bollapragada, and sister Shravani Bollapragada for their constant encouragement, belief in my abilities, and encouragement to take up my master's in TU Delft. Additionally, I am deeply grateful to my grandparents for always looking out for me and showering me with their love and blessings. Without their sacrifices, I wouldn't be here.
I would also like to thank my friend Sandeep for introducing me to machine learning. His guidance has helped me take up this project. I am grateful to Prof. Aadjan van der Helm for allowing me to work under him and explore the integration of AI in embedded systems which led me to explore this direction for my thesis. This preface would be incomplete without mentioning the broader community of researchers and developers in the field of machine learning, audio processing and embedded systems. Their pioneering work and open sharing of knowledge have been foundational, upon which this thesis is built. As you read through the following chapters, I trust that you will find the research outlined in this report both insightful and inspiring. It is my earnest hope that this work aids the continuing endeavors to enhance emergency vehicle response systems and ignites further innovations in applying AI to embedded systems for practical real-world challenges.
Thank you for taking the time to engage with my work.

Bollapragada Lalitha Sai Shraddha
Delft, July 2024

Summary

This thesis presents the development and evaluation of a real-time neural network-based audio classification system designed on an NXP HW board to distinguish emergency response vehicles by their sirens from other vehicles. At the core of the system is a deep learning model that processes audio inputs captured via a microphone, classifying them based on the presence of siren sounds. The system achieves this by extracting audio features and running inference through the designed neural network, followed by post-processing to detect sirens accurately. Audio signals are transformed into mel-spectrograms, which represent the frequency spectrum over time using a specific window size for analysis. The neural network leverages these mel-spectrogram features to perform audio classification.

The deployment of this system involves several key steps. First, the model is trained on diverse audio data, including siren and non-siren sounds. Audio signals are transformed into mel-spectrograms, which capture the frequency spectrum over time. The neural network processes these features to classify the audio based on the presence of siren sounds. The classified results undergo post-processing to enhance detection accuracy. The system is tested in real-world scenarios, demonstrating a turnaround time of less than ***3 seconds*** *even under high noise conditions. Various trade-offs are evaluated to improve efficiency, reduce memory size, and minimize latency, ensuring the system meets requirements for model size, latency, and compute cycles*
The custom dataset comprises 280 hours of audio, including well-known, publicly available datasets such as ESC-50, Audioset, and UrbanSound. This dataset is enriched with both original and augmented siren sounds and non-siren audio to enhance the model's learning efficacy and robustness. The system achieves a ***96.19%*** *test accuracy in identifying sirens and is suitable for deployment in real-world scenarios, even for SNRs as high as* ***-30 dB****.*

Although the system meets most requirements for model size, latency, and compute cycles, the false positive rate needs improvement. This can be achieved by expanding the dataset and retraining the model.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
ML	Machine Learning
ADAS	Advanced Driver Assistance System
LSTM	Long Short Term Memory
MB	Mega Byte
KB	Kilo Byte
QAT	Quantization Aware Training
NAS	Neural Architecture Search
STFT	Short Time Fourier Transform
MFCC	Mel Frequency Cepstral Coefficients
GFCC	Gamma tone Frequency Cepstral Coefficients
LPC	Linear Predictive Coding
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
FCN	Fully Connected Networks
SVM	Support Vector Machine
AED	Acoustic Event Detection
SNR	Siren to Noise Ratio

1 Introduction

The detection of emergency vehicle sirens is a crucial intersection of public safety and technological advancement. It is particularly significant in the context of automotive safety, public security, and emergency response. This issue has gained prominence due to the increasing congestion in urban areas and the heightened competition for auditory attention in modern vehicles [1] [2]. The incorporation of siren detection mechanisms into embedded systems within vehicles and urban infrastructure represents a significant leap forward in public safety technology, as it ensures that emergency vehicles can navigate through traffic more efficiently and safely, potentially resulting in lives being saved [3]. A prolific amount of research into optimizing sophisticated audio event detection techniques and algorithms is ongoing. In addition, embedded technology is being continually improved upon making embedded systems more computationally powerful whilst limiting power consumption in the range of mW, enabling Artificial Intelligence (AI) at the edge [4].

On-device Machine learning inference (ML) has heralded a trans-formative era in advancing the audio-sensing capabilities of a vehicle. These advantages are put together to make us contemplate the real-time capabilities of machine learning on existing edge devices, specifically in this research, the ability of an electronic control unit (ECU) to detect sirens on the go [4]. This advancement is a significant shift away from traditional methods that rely on cloud connectivity. By localizing data processing, this approach enhances security by reducing exposure to data breaches and cyber-attacks such as denial of service and man-in-the-middle attacks. In addition, it significantly reduces latency compared to cloud-based systems, allowing for quicker actions upon siren detection, which can critically affect emergency outcomes. Moreover, it helps conserve internet bandwidth by minimizing data transmission, which is crucial in mitigating network congestion seen with the rise of connected devices. This is especially beneficial in scenarios involving extensive networks, like fleets of vehicles or smart city infrastructures, where it can lead to considerable savings on bandwidth and associated costs [5].

Furthermore, on-device ML promotes the development of more efficient and reliable embedded systems. Efficiency in this context refers not only to the speed of detection but also to the system's ability to operate continuously without the need for frequent updates or reliance on external servers. Reliability is enhanced as the embedded system becomes self-contained, capable of operating even in areas with poor network connectivity or during network outages.

However, the implementation of on-device ML for siren detection poses challenges. Developing algorithms that can accurately identify siren sounds in diverse acoustic environments is a complex task. The variability in siren sounds (varying from different kinds of emergency vehicles to different geographical locations, resulting in variations of tone, pitch, and frequency components), influenced by factors such as distance, the Doppler effect, and ambient urban noise, requires sophisticated audio processing techniques and robust machine learning models that can adapt to changing conditions [6].

Implementing ML models into environments with stringent resource constraints, such as embedded systems in vehicles or portable electronic devices, presents unique challenges that require innovative

Figure 1.1: From left to right, different applications of the solution a) V2V and V2X scenarios; b) Industry setting; c) ADAS or Autonomous driving, [the above images were generated by AI in Canva [9]]

solutions [7]. The field of TinyML has emerged as a response to these challenges, representing a branch of machine learning focused on shrinking down AI models to fit into microcontrollers.

The development of advanced on-device ML systems for siren detection not only addresses the immediate needs of automotive safety and public security but also lays the groundwork for a more connected and responsive emergency response infrastructure in the smart cities of the future [8].

This research focuses on bringing TinyML onto resource-constraint embedded devices that have memory, latency, and computational constraints for storing intermediate audio, processing it to extract features, and housing the siren detection ML model. This research focuses on deploying siren detection in vehicles as an advanced driver assistance system (ADAS) based solution. Due to the real-time needs of this solution, this research aims to balance the trade-offs of latency and memory with detection performance.

The findings in this research can be adapted to other devices and solutions, like siren detection in the ear for workers in the industry where noise cancellation devices are mandatory, siren detection for hearing impaired or low-focus individuals, or for an emergency response infrastructure in a city setting.

The thesis was done in cooperation with NXP Semiconductors. NXP is a leading supplier for the automotive industry, including software-defined radio and in-cabin audio solutions. The upcoming next-generation infotainment platform features powerful neural network processing capabilities. Within the scope of the thesis, a proof of concept shall be implemented to demonstrate the capabilities of the platform and the benefits of ML-enhanced audio algorithms. Their requirements include stringent memory constraints with a maximum size of **300 KB for the model** and buffers and an option to dedicate up to 1 MB if unable to meet the 300KB limit. The system must ensure a machine learning cycle load of less than 25%, and **ideally under 5%**, for efficient real-time processing. Performance requirements mandate a **reaction time of under 3 seconds** in low-noise conditions and under 5 seconds in high-noise conditions, with specific false-positive rate limits to ensure reliability. Non-functional requirements highlight the need for demonstrators on NXP boards, the comparison of different network architectures, and the assessment of optimization techniques for audio networks.

For the development of a TinyML-based streaming siren detection system, several significant research challenges must be addressed to ensure efficacy and efficiency in diverse real-world applications:

- ***Handling Limited Computational Resources:*** The main difficulty lies in creating a reliable machine learning model that works efficiently within the strict memory limitations of embedded devices, which have less than 1 MB of capacity to process and store data. This calls for dedicated methods for designing model architecture and compression techniques to ensure they fit into these parameters without compromising performance.
- ***Ensuring Robustness in Diverse Noise Environments:*** As the system will be used in different vehicular scenarios, it needs to accurately detect sirens even when there is a lot of background noise and varied environmental conditions. This can be particularly difficult because it requires

distinguishing siren sounds from other typical noises that occur in urban and industrial environments, which can have very different acoustic characteristics.

- ***Balancing Latency, Memory, and Accuracy:*** There is an inherent trade-off between these three critical factors in embedded systems. The challenge lies in optimizing the model to minimize response time (latency) and memory usage without compromising the accuracy necessary for reliable siren detection.
- ***Adapting to Different Siren Types and Patterns:*** Sirens can vary greatly depending on geographical location and the type of emergency service. The system must be capable of recognizing and adapting to various siren patterns, which may require sophisticated training.

Thesis Structure

The structure of this thesis is organized as follows: Chapter 2 details the customer requirements. Chapter 3 reviews the existing literature and foundational concepts in siren detection and on-device machine learning. Chapter 4 gives an overview of the system design. Chapter 5 details the design choices and implementation details. Chapter 6 evaluates the testing outcomes and the performance of the developed system. Finally, Chapter 7 provides a summary of the entire system and concludes with some final observations and implications of the research and possible future work.

2 Requirements

In this chapter, the essential requirements for developing a real-time, on-device emergency vehicle siren detection system are outlined. These requirements are divided into various categories, each addressing specific aspects of the system's performance, functionality, and integration. The following sections detail the memory constraints, computational load, performance criteria, and non-functional requirements necessary to achieve an efficient and reliable siren detection system. The requirements tagged with **Should** are preferred requirements to be met and the ones tagged with **Must** are mandatory or can be read as the system shall at least satisfy these criteria.

2.1. Functional Requirements

Functional requirements define the core functionalities of the system. They specify what the system must do and how it should perform.

1. **Memory Constraints**
   For memory requirements, the NXP Product Hardware with 512KB should have a maximum size of Model and Buffers limited to 300KB. The configuration with infotainment includes a total of 640KB. With 300KB for siren detection and the remaining 320KB for infotainment product software.
   - **NXP Hardware only for this use-case (Must)** If the above is not feasible, then the entire memory can be dedicated to only the siren detection use case, and this means that the maximum size of the model and buffers shall not exceed 1MB.
     - Maximum size of Model+Buffers is 1MB
     - No infotainment application running on other cores
   - **NXP Product Hardware - 512KB (Should)**
     - Maximum size of Model+Buffers should be 300KB.
     - Remaining memory is reserved for Infotainment product software.
2. **Cycles**
   Regarding cycles, the machine learning (ML) cycle load must be less than 25%, ensuring that 1 second of streaming audio requires fewer than 150 million cycles and real-time siren detection utilizes less than 25% of the available cycles. Ideally, the ML cycle load should be under 5%, meaning 1 second of streaming audio should take fewer than 30 million cycles, and real-time siren detection should use less than 5% of the cycles.
   - **ML cycle load <25% (Must)**
     - 1 second streaming audio <150M Cycles.
     - Real-time siren detection shall use less than 25% of the available cycles.

- **ML cycle load <5% (Should)**
    - 1 second streaming audio <30M Cycles
    - Real-time siren detection shall use less than 5% of the available cycles.

3. **Performance**
   Performance requirements dictate that the reaction time must be under 3 seconds in low noise conditions at 0dB SNR and less than 5 seconds in high noise conditions at -6dB SNR, measured from the start of the siren to its detection. The false positive rate for low noise must ensure no more than one occurrence of false positives lasting over 1 second per 10 hours of audio and no more than five occurrences of false positives under 1 second. These measurements apply when driving outside the city in low noise, with ambiguous audio sections manually checked. In high noise, the false positive rate should ideally be kept to the same standards per 5 hours of audio when driving in a European city with noise sources.
    - **Reaction time <3s in low noise (Must)**
        - Reaction time must be less than 3 seconds at 0dB SNR
        - Measured as time from the start of the siren to detection of the siren.
    - **Reaction time <5s in high noise (Must)**
        - Reaction time must be less than 5 seconds at -6dB SNR
        - Measured as time from the start of the siren to the detection of the siren
    - **False positive rate low noise (Must)**
        - Per 10 hours of low noise audio there must be 1 or less than one false positive for a duration of greater than 1 sec and less than 6 false positives lesser than 1 sec.
        - False positive rate when driving outside the city in low noise.
        - Does not need to be better as humans. Findings will be manually checked to exclude ambiguous audio sections (e.g. alternating car horns)
    - **False positive rate high noise (Should)**
        - Per 5 hours of high noise audio there should be 1 or fewer false positives greater than or equal to 1 sec and less than 6 false positives less than 1 sec
        - False positive rate when driving in a European city with noise sources.
        - Does not need to be better as humans. Findings will be manually checked to exclude ambiguous audio sections (e.g. alternating car horns)

2.2. Non-Functional

1. **Demonstrator on NXP board (Should)**
    - End-to-end real-time demonstrator on NXP development board with microphone and siren detection visualization.
    - Shall be usable for a showroom or demonstration to customers. All calculations shall happen offline on board.
2. **Comparison of network architectures (Should)**
    - Comparison of memory, compute, and precision metrics of different network architectures: CNN, FC, RNN
3. **Vigilance enhancement (Should)**
    - Breaking point of the siren detection performance in comparison to a human driver. As an extension to reaction time tests with a ramp down of SNR.

4. **Assessment of optimization techniques (Should)**
    - Investigation into what optimization techniques can be applied to audio networks to reduce the size of the network.
    - Can be QAT, NAS, identifying where neural networks are inferior to other ML methods or traditional programming.

3 Background and Related Work

This chapter focuses on state-of-the-art audio event detection/classification, the different kinds of feature extraction techniques, different deep learning models under consideration, and existing audio-based Tiny ML solutions.

3.1. Audio Event Detection

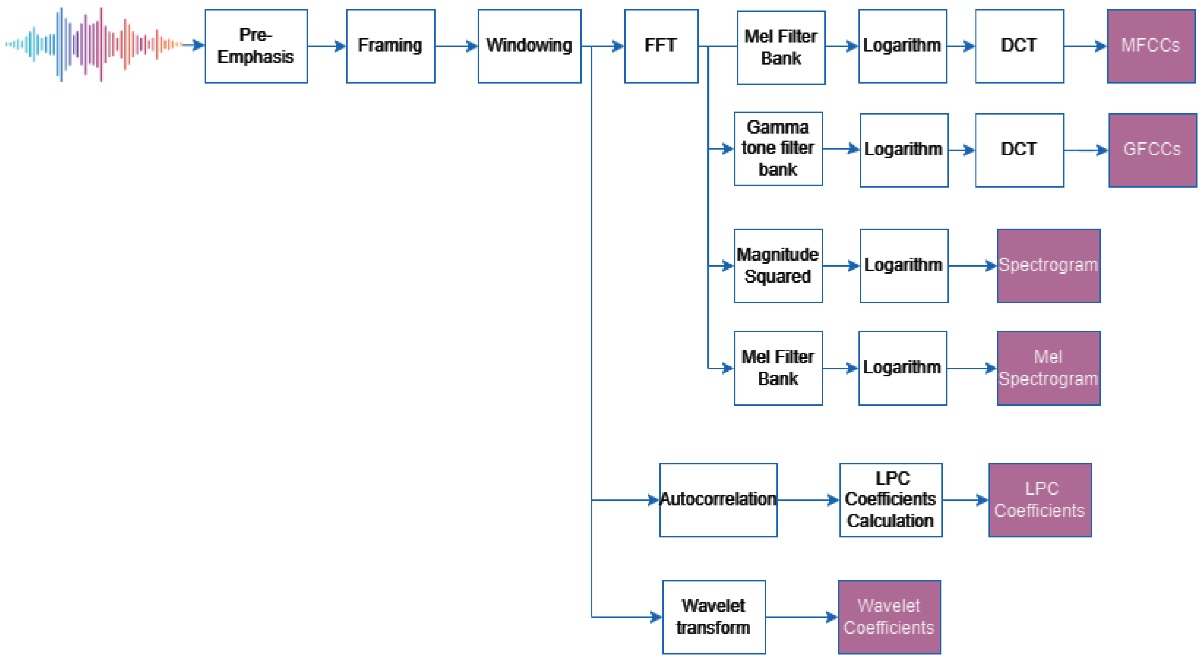


Figure 3.1: Block diagram of the process of obtaining different audio features for siren detection considered for this research.

Recent research has made significant strides in addressing the challenge of applying deep learning algorithms for acoustic event detection to resource-constrained devices. Cerutti et al. [10] and Mohaimenuzzaman et al. [11] both present methods for optimizing deep learning techniques on microcontrollers, achieving around 68% and 87% accuracy n recognition on Urbansound8k respectively. Research done by Veronica et al. [12] further contributes to this area by proposing data-efficient training methods for audio event detection, particularly in the context of limited availability of training data. These studies collectively demonstrate the potential for the application of deep learning models to devices with limited resources, such as microcontrollers in vehicles.

Feature Extraction Techniques

To better decide the parameters for feature extraction it is necessary to understand the general characteristics of a siren sound as described below.

Siren Acoustics

Two primary attributes characterize siren sounds and enable their detection: the Modulation Pattern and Frequency Range. These attributes are designed to ensure that siren sounds are distinctive and recognizable even in noisy environments, facilitating quick identification by both humans and automated systems.

Modulation Pattern: Emergency vehicle sirens use a modulation pattern that alternates between different pitches and volumes. This modulation creates a rhythmic pattern that stands out from other urban noises. The pattern typically includes cycles of high and low frequencies, often referred to as ”wail,” ”yelp,” and ”phaser” modes. Each mode has a unique frequency sweep and timing to ensure that the siren can penetrate through background noise and catch the attention of both drivers and pedestrians.

- Wail: This mode features a slow, oscillating frequency that rises and falls over several seconds. The wail mode is often used for long-distance alerts because its gradual frequency changes can be heard from far away.
- Yelp: The Yelp mode has a faster oscillation between high and low frequencies. This mode is effective in dense urban areas where quick changes in frequency can more easily cut through ambient noise.
- Phaser: The phaser mode has a very rapid oscillation and is typically used when immediate attention is needed. This high-intensity pattern is especially useful in situations requiring urgent clearance of traffic.

Frequency Range: Siren sounds are designed to cover a broad frequency range, typically between 500 Hz and 1500 Hz. This range ensures that the siren can be heard over the common frequencies present in urban noise. Lower frequencies are less likely to be blocked by obstacles, while higher frequencies are more directional and can be pinpointed more accurately.
Low Frequencies (50-800 Hz): These frequencies can travel long distances and are less affected by buildings and other obstructions. They help ensure that the siren can be heard even in areas where high-frequency sounds might be blocked.
High Frequencies (800–7500 Hz): These frequencies provide the sharp, piercing sound that makes the siren easily identifiable. They are effective at close ranges and help pinpoint the direction of the siren.
Acoustic Propagation and Environmental Effects: The propagation of siren sounds can be affected by various environmental factors, such as buildings, weather conditions, and ambient noise levels. Urban environments with tall buildings can create echoes and reverberations, which might alter the perceived direction and clarity of the siren. Weather conditions such as rain and wind can also impact how far and clearly the siren sound travels.

Echo and Reverberation: In urban canyons formed by tall buildings, siren sounds can bounce off surfaces, creating multiple echoes. This can sometimes make it challenging to identify the direction of the siren but generally increases the overall volume, making it harder to ignore.
Weather Conditions: Rain and wind can either dampen the siren sound or carry it further, depending on the direction and intensity. For example, a strong wind might carry the sound further in its direction, while heavy rain might absorb some of the sound energy.

Figure 3.1 shows the process of obtaining different kinds of audio features that were considered for this research. For those who wish to bypass the detailed feature explanations, a summary can be found in Table 3.1

3.1.1. Spectrogram

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it changes over time. It is commonly used in audio processing to analyze the frequency content of audio signals over

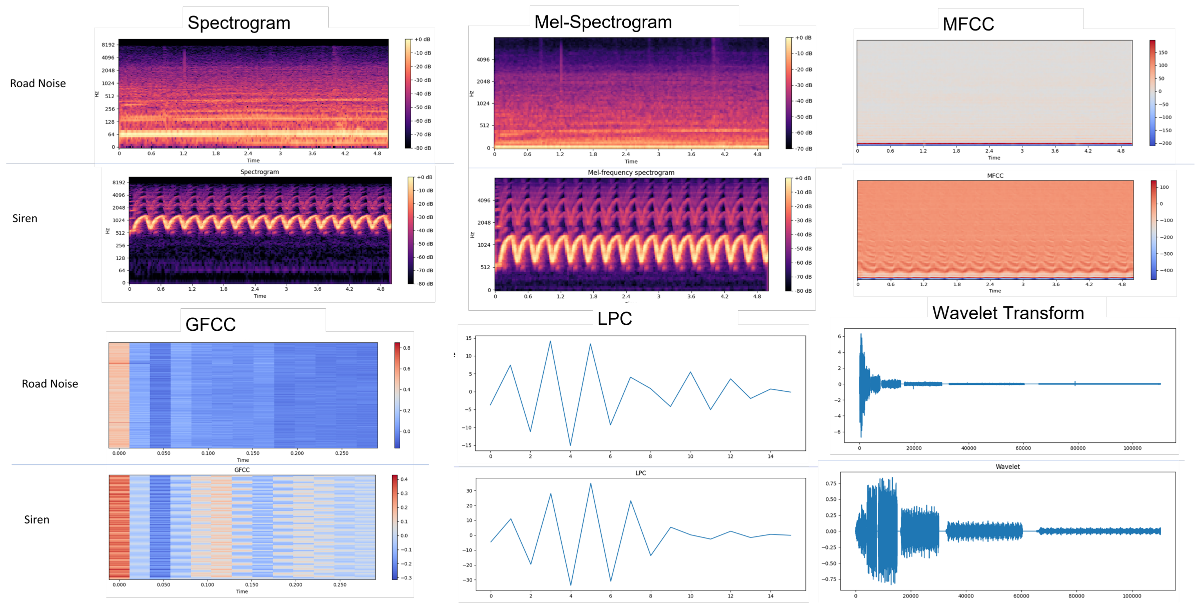


Figure 3.2: Audio features for non-siren (road noise- most common non-siren for this use case) v.s. siren, spectrogram, mel-spectrogram, MFCC, GFCC, LPC and Wavelet Transform.

time. The process of generating a spectrogram involves computing the Short-Time Fourier Transform (STFT) of the signal, which divides the signal into overlapping segments, applies a window function to each segment, and then computes the Fourier transform. The resulting matrix represents the magnitude of the signal's frequency components over time. By visualizing how frequencies change over time, spectrograms are useful for analyzing non-stationary signals like sirens. The intensity of colors in a spectrogram indicates the amplitude of particular frequencies at different time intervals.

Spectrogram analysis for siren detection:

1. Detailed Frequency Information: Spectrograms provide a high-resolution view of the entire frequency spectrum, capturing detailed information about the signal's frequency content.
2. Versatility: This can be adapted to emphasize different frequency ranges depending on the specific requirements of the detection task.
3. Clear Visual Representation: Offers a straightforward visual representation of the signal's time-frequency characteristics, which can be useful for both manual inspection and automated analysis.
4. Lower Computational Load: Generally requires fewer computational resources than Mel-Spectrograms because it does not involve the additional Mel filter bank step. (refer to next section 3.1.2)
5. Potential for Overemphasis on High Frequencies: Without the logarithmic compression used in Mel-Spectrograms, high-frequency components can dominate the representation, potentially masking the relevant features of sirens.

3.1.2. Mel-Spectrogram

A mel-spectrogram is a spectrogram that has been wrapped to the mel scale, which approximates the human ear's perception of sound. This transformation compresses the frequency axis, giving more emphasis to lower frequencies where human hearing is more sensitive [13].

Mel-Spectrogram analysis for siren detection:

1. Visualization of Frequency Changes: Mel-Spectrogram excels at portraying how sound frequencies evolve over time. This is ideal for identifying the non-stationary characteristics of sirens like pitch changes and rapid fluctuations.

2. Focus on Human Hearing: Like MFCCs and spectrograms, Mel-Spectrogram prioritizes frequencies relevant to human hearing, aiding in siren detection amidst background noise.
3. Efficient Feature Extraction: Mel-Spectrogram generation is computationally less expensive compared to other techniques like wavelets or deep learning.
4. Information Overload: Spectrograms, including Mel-Spectrograms, can contain a vast amount of information. Extracting the most relevant features for siren detection might require further processing steps.

3.1.3. Mel Frequency Cepstral Coefficients (MFCC)

MFCC is a method that captures the short-term power spectrum of sound. They have been widely used in various applications due to their ability to model human auditory perception and distinguish between different sound signatures [14, 15]. It has been particularly effective in discriminating stuttering dysfluencies [14], enhancing speaker recognition, and classifying underwater target radiation noise. Furthermore, the integration of MFCC with statistical indicators has shown promise in classifying different types of sound, such as squeak and rattle noise [16]. The Mel Filter Bank $M(m,k)$ is a crucial component in the process of computing MFCC, widely used in speech and audio processing. This filter bank consists of a set of triangular filters applied to the power spectrum of a signal. Each filter is designed to pass a specific range of frequencies based on the Mel scale, approximating the human auditory system's response more effectively than linearly-spaced frequency bands.

The formula for $M(m,k)$ defines the amplitude gain of the k-the frequency component when processed by the m-th Mel filter as follows:

$$M(m,k) = \begin{cases} 0 & \text{if } f(k) < f_c(m-1) \\ \frac{f(k)-f_c(m-1)}{f_c(m)-f_c(m-1)} & \text{if } f_c(m-1) \leq f(k) < f_c(m) \\ \frac{f(k)-f_c(m+1)}{f_c(m)-f_c(m+1)} & \text{if } f_c(m) \leq f(k) < f_c(m+1) \\ 0 & \text{if } f(k) \geq f_c(m+1) \end{cases} \tag{3.1}$$

where:

- $f(k)$ is the frequency at the k-th bin,
- $f_c(m)$ is the center frequency of the m-th Mel filter,
- $f_c(m-1)$ and $f_c(m+1)$ are the center frequencies of the adjacent filters.

Each filter is zero outside of its defined range between $f_c(m-1)$ and $f_c(m+1)$, ensuring that each filter only affects a specific part of the spectrum. Within its range, the filter has a triangular shape:

- The filter increases linearly from zero at $f_c(m-1)$ to its peak at $f_c(m)$,
- Then decreases linearly back to zero at $f_c(m+1)$.

The slopes of these increases and decreases are determined by the ratios given in the formula:

- The ascending slope is computed as $\frac{f(k)-f_c(m-1)}{f_c(m)-f_c(m-1)}$, representing the filter value increase from $f_c(m-1)$ to $f_c(m)$.
- The descending slope is $\frac{f(k)-f_c(m+1)}{f_c(m)-f_c(m+1)}$, detailing the decrease from $f_c(m)$ to $f_c(m+1)$.

When applied to a frequency spectrum, each filter shapes the spectrum by emphasizing the frequencies around its center frequency and attenuating those farther away. This process effectively warps the frequency scale to conform to the Mel scale, which is more perceptually relevant for human hearing.

The entire set of filters forms an $F \times N$ matrix where F is the number of filters and N is the number of frequency bins in the discrete Fourier transform of the audio signal. This matrix is used to transform the linear frequency spectrum into a Mel frequency spectrum, providing the critical frequency-based features needed for tasks such as speech recognition [17].

MFCC analysis for siren detection:

1. Efficient: Computationally efficient and can be extracted in real-time.
2. Effective for Speech: Widely used and effective for speech-related tasks, providing a compact representation of the audio signal.
3. Less Effective for Non-Speech Sounds: Not specifically designed for non-speech sounds like sirens, which might lead to less accurate detection.
4. Lack of Temporal Information: Loses some temporal resolution, which is crucial for capturing the dynamic nature of sirens.

3.1.4. Gamma tone frequency cepstral coefficients (GFCC or GTCC)

A range of studies have explored the application of GFCCs in various fields. Ahmed Krobba et al. [18] introduced a Mixture Linear Prediction approach for robust GTCC extraction, which significantly improved speaker verification performance under transmission channel noise. Changwei Zhou et al. [19] proposed Gammatone Spectral Latitude (GTSL) features for pathological voice detection and classification, achieving high accuracy and low computational complexity. Azza Moawad et al. [20] developed a Cepstral Covariance Detector for spectrum sensing in cognitive radio, demonstrating reliable detection at low signal-to-noise ratio levels. These studies collectively highlight the potential of GTCCs to enhance performance across a range of applications. From the literature, we can see that they better MFCCs, in more specialized applications. The GFCCs are derived from the output of a bank of gammatone filters, which approximate the human auditory system's response. The output is then transformed into cepstral coefficients, akin to MFCCs, but with a different filter bank.
The output of the gammatone filter bank is given by:

$$g(t) = a \cdot t^{n-1} \cdot e^{-2\pi bt} \cdot \cos(2\pi f_c t + \phi) \cdot u(t) \tag{3.2}$$

where a is the amplitude, n is the filter order, b is the bandwidth, f_c is the center frequency, ϕ is the phase, and $u(t)$ is the unit step function, ensuring causality [21]. After applying the gammatone filter, the filtered signal $g(t)$ undergoes the following steps to compute the GFCC:

$$GFCC = DCT(\log(|FFT(g(t))^2|)) \tag{3.3}$$

GFCC analysis for Siren Detection:

1. Frequency Resolution: Designed to mimic the human auditory system more closely than other methods, providing a more accurate representation of how humans perceive sounds, which can be beneficial for distinguishing sirens.
2. Effective for Noisy Environments: Often more robust to noise compared to other features, making them suitable for outdoor environments where sirens are typically found.
3. Temporal Dynamics: Can capture temporal dynamics effectively, similar to Mel-Spectrograms, which is crucial for identifying the changing nature of siren sounds.
4. Computational Complexity: Generally more computationally intensive than MFCCs and LPC, which can be a drawback for real-time applications on resource-constrained devices.
5. Implementation Complexity: More complex to implement compared to traditional methods like MFCC, requiring more sophisticated processing steps.
6. Data Requirement: May require more data to effectively train models compared to simpler features.

3.1.5. Linear Predictive Coding (LPC)

Linear Predictive Coding (LPC) is a method used in audio signal processing and speech coding that represents the spectral envelope of a digital signal of speech in compressed form using the information of a linear predictive model. LPC is one of the most powerful speech analysis techniques and is widely used in speech synthesis, speech compression, and speaker recognition.

In detailed research, LPC's utility has been highlighted across various studies. For instance, its application in Driving Fault Diagnosis, as explored by [22], demonstrates LPC's effectiveness when integrated with other methodologies, enhancing the overall performance of recognition systems. Similarly, [23] identified the strength of LPC in speaker recognition systems, especially when combined with additional feature sets, to improve identification accuracy.

Moreover, LPC's versatility extends to the domain of language differentiation, as evidenced by [24], who successfully applied LPC techniques to distinguish among Indian spoken languages. This wide-ranging applicability underscores LPC's significant contribution to the advancement of audio processing technologies, offering a robust framework for tackling diverse challenges in speech and audio analysis.

Auto-correlation: Measures how well a signal matches a delayed version of itself, which is key for predictive modeling. Linear Prediction: This involves estimating the spectral envelope of a digital signal with linear predictive models. The LPC model predicts the current sample $\hat{s}(n)$ based on a linear combination of the previous samples and an error term, formulated as follows:

$$\hat{s}(n) = \sum_{i=1}^{p} a_i s(n-i) + G \cdot e(n) \tag{3.4}$$

where:

- $\hat{s}(n)$ is the predicted sample value at time index n,
- a_i are the LPC coefficients,
- $s(n-i)$ are the previous signal samples,
- p is the order of the LPC model, indicating how many past samples are used in the prediction,
- G is the gain, a scalar that amplifies the error term,
- $e(n)$ is the prediction error at time n.

This equation represents the core of LPC, where the current sample of the signal is estimated as a weighted sum of past samples plus a correction term that scales the prediction error.
LPC analysis for Siren Detection :

1. Noise Sensitivity: Siren detection systems often operate in noisy environments. LPC can be sensitive to noise, which may lead to inaccurate feature extraction and false positives or negatives.
2. Non-Stationary Signals: Sirens may vary in pitch, modulation, and duration. While LPC can capture the spectral characteristics of stationary parts of the signal, it might struggle with rapid changes, which are common in sirens.
3. Feature Discrimination: While LPC captures the spectral envelope well, it might not provide enough discriminatory power by itself. Sirens have specific modulations and patterns that may require additional features for robust detection.
4. Hybrid Approaches: Combining LPC with other feature extraction methods, such as Mel-Frequency Cepstral Coefficients (MFCCs) or Short-Time Fourier Transform (STFT) features, can improve robustness and accuracy. These additional features can capture temporal and frequency variations more effectively.

3.1.6. Wavelet Transform

Wavelet transforms are widely utilized in audio feature extraction, particularly for analyzing non-stationary signals. By decomposing signals into components across different frequencies and resolutions, they enable precise time-frequency analysis. This technique is adept at identifying transient features in various signals, offering deep insights into signal properties. Its adaptability and analytical depth make wavelet transforms crucial in fields such as speech recognition, music analysis, biomedical signal processing, and telecommunications, where understanding complex signal dynamics is essential [25].

Wavelet Transform analysis for Siren Detection:

1. Flexibility: Can use different wavelet functions to better capture specific features of the signal.
2. Complexity: The visual representation might be less intuitive and more complex to interpret than a spectrogram.
3. Implementation Difficulty: More challenging to implement and fine-tune for specific applications.

Computational complexity for each of the feature extraction strategies are evaluated in Chapter 5, Figure 4.2

3.1.7. Deep Learning-based audio feature extraction

Shreya et al. [26] explore the use of neural networks in the feature extraction process for speech classification tasks. Hyejin et al. [27] used a CNN-based encoder and transformer-based decoder for automated audio captioning, which took raw audio data as the input and produced text describing this audio. They discuss methods to enhance the performance of deep learning systems in audio categorization, emphasizing data augmentation and comparative analysis of various audio feature extractors. As part of the study, a system for identifying speakers that are highly accurate and suitable for use in practical applications was also being developed. Audio data processing and classification are enhanced by robust feature extraction techniques, deep learning, discrete cosine transforms, and neural networks. However, owing to their large size and computing complexity, deep learning-based feature extraction will not be considered for this research.

Technique	**Description**	**Key Advantage**	**Common Application**	**Compute Power Required**	**Final Analysis for Siren Detection**
MFCC (Mel Frequency Cepstral Coefficients)	Captures the short-term power spectrum of audio signals based on a logarithmic (mel) scale.	Mimics human auditory perception, effective in capturing unique sound characteristics.	Speech and audio recognition, including siren detection.	Moderate	Effective but may miss transient features.
LPC (Linear Predictive Coding)	Models the vocal tract as a digital filter, predicting the current sample based on past samples.	Efficient in encoding speech information with a small set of coefficients.	Speech analysis, synthesis, and compression.	High	Less suitable for capturing transient siren features.
GFCC (Gammatone Frequency Cepstral Coefficients)	Utilizes gammatone filters to mimic the human ear's response, analyzing the spectral content of audio signals.	Robust in noisy conditions, better mimicking the human auditory system than MFCC.	Sound classification in challenging acoustic environments.	Moderate- High	Effective but computationally demanding and complex to implement.
Wavelet Transforms	Transforms the signal into a representation in both time and frequency domain using wavelets.	Effective in analyzing non-stationary signals with varying frequency components.	Signal processing tasks requiring detailed time-frequency analysis.	Moderate	Effective but difficult to identify the best kind for siren sounds.
Mel-Spectrogram	Captures the power spectrum of audio signals, mapped onto the mel scale and represented over time.	Emphasizes frequencies relevant to human hearing, effective for non-stationary signals.	Speech and audio recognition, particularly in noisy environments.	Low	Very effective for capturing dynamic siren features.
Spectrogram	Represents the power spectrum of audio signals over time.	Provides a detailed view of frequency content over time, versatile.	General-purpose audio analysis and visualization.	Low	Effective, computationally lite, but may require tuning for optimal results.

Table 3.1: Comparison of Advanced Audio Feature Extraction Techniques.

3.2. Deep Learning models considered in Acoustic Event Detection (AED)

As this will be a TinyML-based solution, we discuss different deep-learning topologies enabled by the TensorFlow Lite micro-library. The schemes are listed below:

3.2.1. Convolutional Neural Networks (CNN)

CNNs are a type of deep neural network that can learn features from raw data by applying multiple layers of convolution and pooling operations. They are widely used for image recognition and computer vision tasks, but they can also be applied to audio data by treating spectrograms, Mel-spectrograms, or MFCCs as images. CNNs can capture local and global patterns in the frequency and time domains, and they are robust to noise and variations in the input data. CNNs have been shown to achieve high performance in audio event detection and classification tasks, such as siren detection [28]. In the research work conducted by Kele Xu et al., [29], different CNN architectures were studied for the use case of audio tagging.

3.2.2. Fully Connected Networks (FCN)

Typically Fully connected Networks were used only as baseline models for audio event detection as they have always been outperformed by CNNs and other networks which have the ability to capture spatial and temporal features of audio signals better than FCNs [30].

Year of Publishing	Application	Feature extraction	Deep Learning Model	Model Size	Model performance	Latency
Nov 2022	Ecological Monitoring [32]	Similar to MFCC	SVM	163KB	95-97%	-
Sep 2022	Sleep Bruxism Diagnosis [33]	Mel-spectrogram	2D CNN	148KB	79%	1.6ms for 1sec audio
April 2022	Fall Detection [31]	Mel-spectrogram	CNN	1.5MB	95%	-
Jan 2020	Acoustic Event Detection [9]	Mel-spectrogram	VGGish(CNN) +RNN	34.3KB	69%	125ms for 1sec audio

Table 3.2: Summary of different Audio based embedded AI solutions.

3.2.3. Recurrent Neural Networks (RNN)

Unlike traditional neural networks, RNNs have “loops” in them, allowing information to persist from one step in the sequence to the next. This characteristic makes them inherently deep in time and suited to application on sequences. Numerous research efforts have delved into the application of Recurrent Neural Networks (RNNs) in the field of audio classification. The authors in [31] utilized a Deep RNN with Long-Short Term Memory (LSTM) units to classify construction site audio, achieving an impressive accuracy of 97% for a 5-class classification of different vehicles and tools in the construction site. RNNs are well-suited for audio classification problems as they can handle variable-length audio sequences and learn temporal features from the audio signal. This makes them particularly effective in tasks such as acoustic scene classification [32].

3.2.4. Convolutional Recurrent Neural Networks (CRNNs)

However, it is important to note that the choice between RNNs and other models like CNNs often depends on the specific requirements of the task at hand. For some tasks, a hybrid approach that combines the strengths of both RNNs and CNNs might be the most effective solution [33].

3.3. Audio-based embedded AI solutions

In this section we review, embedded neural networks for various audio-related applications, with the aim to identify gaps and common practices in this field. Each of the research works leverages different neural network architectures and techniques, a small survey on existing work helps identify the progress and the challenges in deploying machine learning on embedded systems for a real-time audio event detection use-case (siren detection).

- Recurrent Neural Network for Acoustic Event Detection: In this paper, the authors use the VGGish feature extractor, which converts 960 ms audio into a mel spectrogram that can be used as input for further classification. A two-stage student-teacher approach to compress a sound event detection classifier composed of the VGGish feature extractor and a recurrent classifier is used for classification. The paper reports that the smallest model (M20k) achieves 72.% accuracy on the test set and 69% after quantization. The paper states that the smallest model (M20k) requires only 34.3 KB of RAM and can run on an ARM Cortex M4 [10].
- Fall Detection: Kun Fang et al. [34] proves TinyML to be a valuable tool for healthcare applications; this research focuses on the monitoring of fall incidents among the elderly. By utilizing a convolutional neural network (CNN), distinct auditory signals of falls can be separated from ambient sounds, a critical step in providing timely assistance. This was implemented on a portable device, the CNN model can independently identify sounds associated with falls with a high degree of precision, regularly surpassing a 95% accuracy threshold as claimed by the research. However, the dataset used by them consists of a small pool of fall sounds of around ~1200 audio samples of falling and noise.
- Ecological Monitoring: In ecological monitoring, the work by H.R. Sabbella et al. [35], TinyML makes it possible to identify wildlife species by their sounds, which helps in the conservation of endangered species. They use a template-based SVM classifier deployed on hardware such as ARM Cortex-M4-based AudioMoth devices. This approach utilizes CAR-IHC based filters for data pre-processing. The testing accuracy is claimed to be between 95-97%, which is impressive

considering the low amount of training data used.

- Sleep Bruxism Diagnosis: In medical diagnostics, Giacomo Peruzzi et al., [36] showcased that leveraging audio signals through embedded ML can lead to early and remote diagnosis of conditions like sleep bruxism. In this research, the authors use an open-source dataset with **389** audio samples. This setup can distinguish Bruixism with an accuracy of ~79 to 80% The total memory usage is ~476 KB for the complete model and ~148KB for the quantized version.

3.4. Challenges and Gaps

From the literature, it is observed that most solutions have either one or more of the following missing.

Paper	Data Availability/Quality	Model Complexity and Resource Constraint	Real-World Generalization	Inference and Noise	End-to-End Inference on Embedded Device	Real-time deadlines
Ecological Monitoring[32]	✓	✓	✓	✓	✓	✗
Sleep Bruxism[33]	✗	✓	✗	✗	✗	✓
Fall Detection[31]	✗	✗	✗	✗	✗	✗
Acoustic Event Detection[9]	✓	✓	✗	✗	✗	✓
This research	✓	✓	✓	✓	✓	✓

Table 3.3: Evaluation of Papers Based on Various Criteria

1. **Data Availability and Quality:** High-quality, diverse, and extensive datasets are crucial for training robust models but are often challenging to obtain, especially for rare acoustic events like sirens. Sometimes one of the classes has a lot of samples in comparison to the other, so learning with the right metrics becomes a challenge. In this research, a large collection of data of around 280 hours is used.

2. **Model Complexity and Resource Constraints:** Deploying complex models on resource-constrained devices like micro-controllers presents challenges in balancing accuracy and computational complexity. Replicating parts of the network can easily increase the model size making it good for accuracy, but unfit for micro-controllers. Optimized feature extraction plays a crucial role in deciding the latency as well as the size of the solution. As can be seen from Figure 4.2 on a Linux-based PC environment calculation time amounts to approximately 1/3rd of the total audio duration; this time can be expected to be higher on a micro-controller, hence a custom solution for feature extraction and model design has been employed. A general model already used for other use cases is typically found to have unnecessary bulk. In this research, common and custom solutions are analyzed to ensure lesser size whilst maintaining low latency.

3. **Real-world Generalization:** Many models struggle with generalization to real-world scenarios due to over-fitting on training datasets or under-representation of real-world acoustic variability. Keeping the training data close to the real world is essential in maintaining accuracy in real life, not just on the training or testing datasets. In this research validation data is made to replicate real-world scenarios of more noise and less siren. The model is also trained on various kinds of sirens to ensure it can classify different types of sirens as sirens.

4. **Interference and Noise:** Background noise and overlapping sound sources can significantly degrade model performance, which calls for either training with data that has noise or develop-

ing noise separation or reduction techniques that could be more compute-intensive for a micro-controller-based application.

5. **Lack of End-to-End Solutions:** There is often a lack of end-to-end solutions where novel models are shown deployed on the embedded device. Some use cases do not have real-time as well as memory constraints and, hence not much is present in the literature where capturing real-time audio, feature exaction, and immediate results are produced on embedded hardware. This gap highlights the need for integrated approaches to handle the entire workflow on resource-limited platforms.

4 Design Methodology

System Architecture

This chapter provides a high-level overview of the system architecture for real-time, on-device emergency vehicle siren detection using deep learning on the Cadence Tensilica HiFi 5 DSP embedded on the NXP SAF9100 [37] [38]. Detailed design aspects will be discussed in subsequent sections. An overview of the system architecture is depicted in Figure 4.1 The system consists of 1) the microphone array that gathers audio from the environment, 2) the NXP SAF9100 audio processor [38] for computation that does feature extraction, 3) the deep learning-based classification algorithm and 4) the post-processing block.

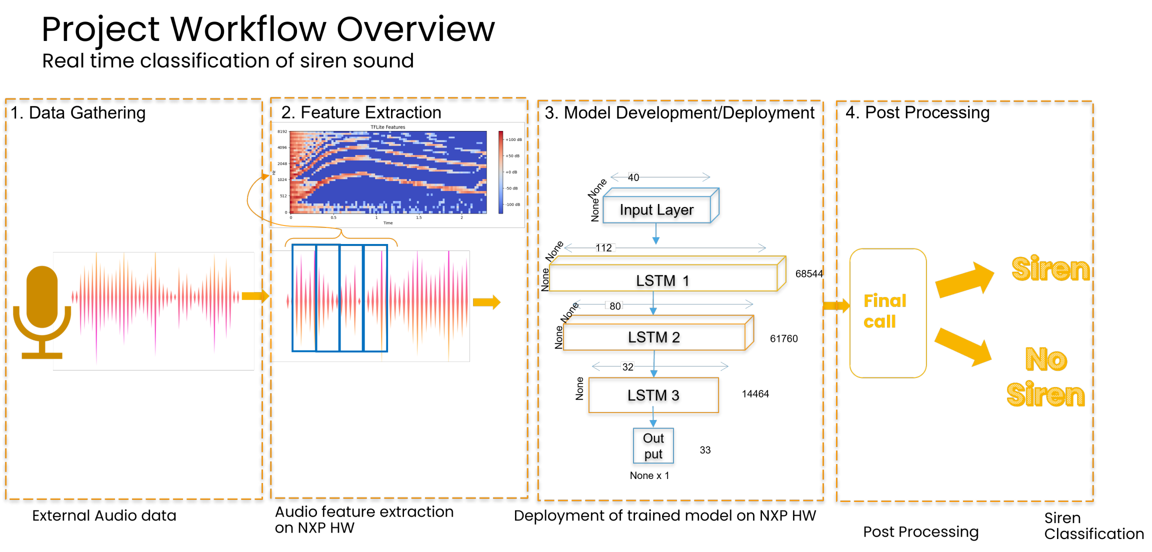


Figure 4.1: Siren Detection System Architecture.

4.1. Data Gathering

The efficacy of machine learning models is significantly influenced by the quality and quantity of the data employed during the training phase. The data's relevance and quality directly impact the performance of the machine-learning model. It is crucial to gather data pertinent to the specific machine-learning task at hand. High-quality, well-labeled data forms the foundation for training accurate and reliable models. The total data that was collected was 93 hours (10 GB) and additional 186 hours (20GB) was augmented to give a 278-hour (30GB) dataset of siren and non-siren data. More details on which datasets were used shall be explained in Chapter 5.

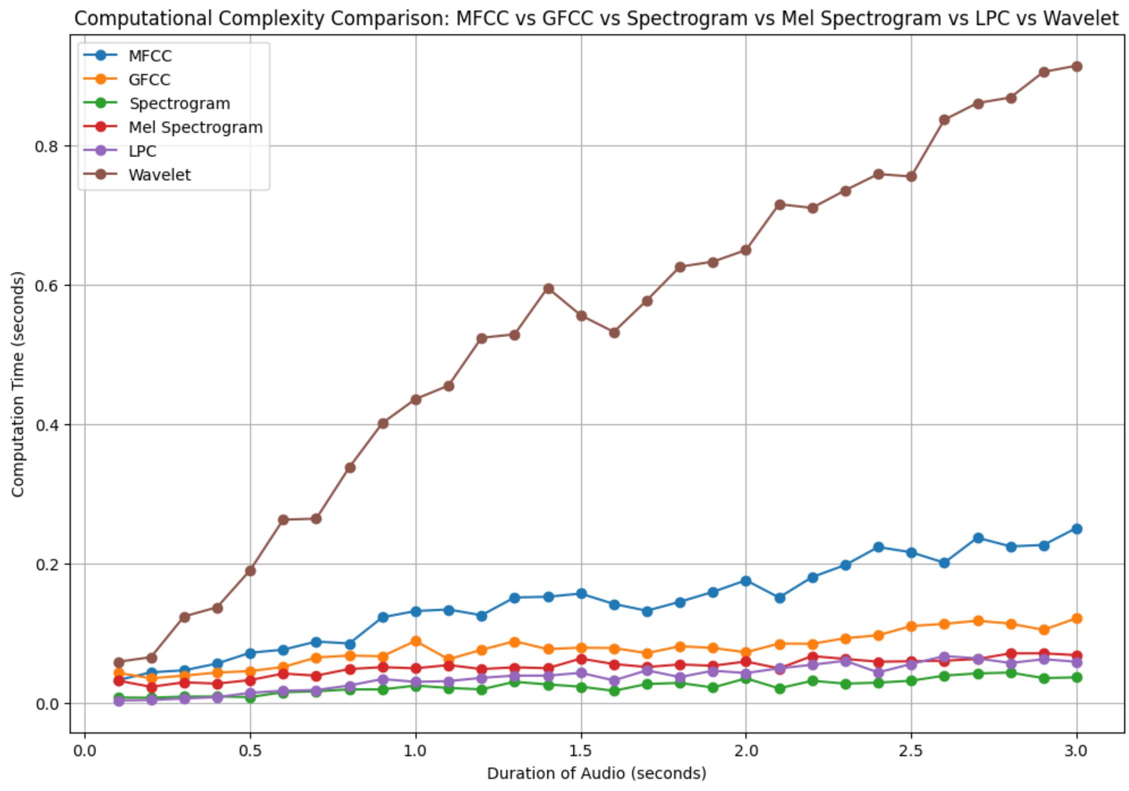


Figure 4.2: Computation Complexity Analysis of different audio feature extraction techniques run on Desktop PC.

4.2. Feature Extraction

Feature extraction is a critical step in developing the siren detection system. It involves transforming raw audio signals into a format suitable for machine learning models. This section explains the techniques and methods used to extract relevant features from the audio data. These features are crucial for accurate and efficient siren detection.

Several feature extraction techniques were considered and evaluated to identify the most effective approach for the siren detection system. The computational complexity and the ability to differentiate between siren and non-siren sounds are vital factors when analyzing different feature extractions. Therefore, a preliminary analysis was conducted to shortlist a suitable feature extraction technique for siren detection, as shown in the figure. 4.2 In conclusion, the Mel-spectrogram is highly effective for siren detection due to its ability to capture dynamic audio features and emphasize frequencies relevant to human hearing, making it suitable for distinguishing sirens from background noise. It is also well-suited for real-time applications with moderate computational resources. MFCCs, derived from the Mel-spectrogram, offer a good balance of computational efficiency and spectral characteristic capture, making them a viable option for real-time siren detection. Spectrograms provide a detailed view of the frequency content over time and are effective for general-purpose audio analysis, but may require tuning for optimal results because spectrograms require more scaling compared to Mel-spectrograms for siren detection due to their linear frequency representation and wider dynamic range. Proper scaling techniques, such as logarithmic scaling and noise reduction, are essential to highlight the siren frequencies, improve contrast, and enhance the signal-to-noise ratio. After deploying mel-spectrogram feature extraction, it took (0.031 seconds to process 1 sec of audio data).

GFCCs are effective at capturing audio features and robust in noisy conditions, but they have a high computational cost, making them less ideal for real-time applications. On the other hand, the Wavelet Transform is powerful at capturing transient features such as the onset and offset of sirens, but it is generally unsuitable for real-time applications due to its high computational demands. LPC, focused on modelling speech, is not a great option for capturing the transient features of sirens and is less effective for this purpose. Therefore, Mel-spectrograms and MFCCs are recommended for siren detection, with GFCCs, Wavelet Transforms, and Spectrograms as less ideal alternatives.

The detailed feature extraction process used for this research is explained in Chapter 5 in Section 2.

4.3. Model Deployment

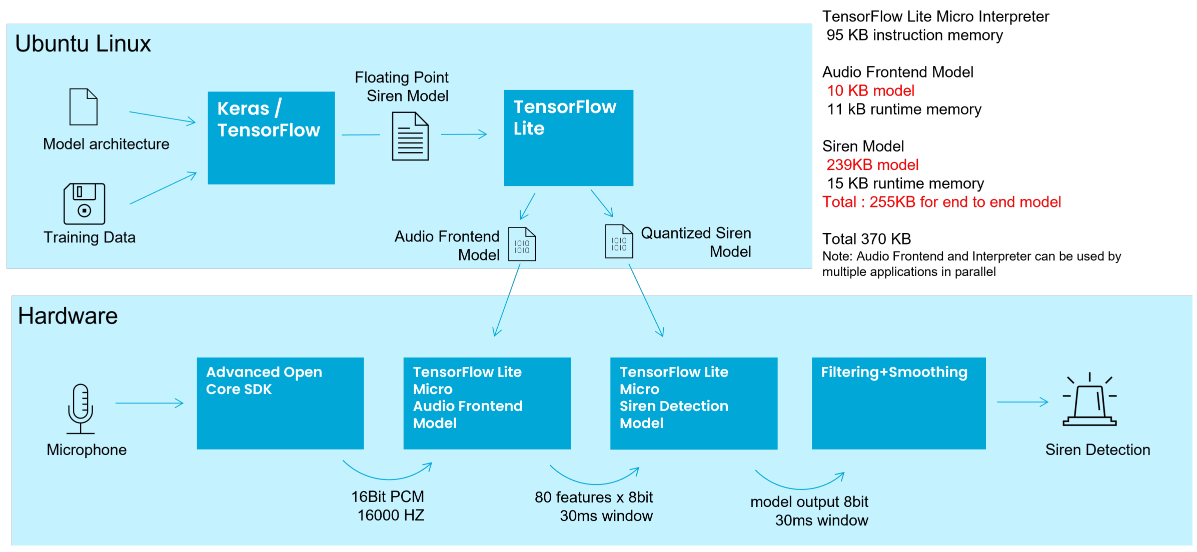


Figure 4.3: Overview of Software Deployment process.

Figure 4.3 illustrates the workflow of the siren detection system from model training to deployment on hardware. Initially, the model architecture is designed using Keras or TensorFlow and trained with provided data on Ubuntu Linux, resulting in a floating-point siren detection model. This model is then converted to TensorFlow Lite and quantized, producing a smaller model (239KB). An audio frontend model preprocesses the audio data, leading to a combined end-to-end model size of 255 KB. On the hardware side, a microphone captures audio signals (16-bit PCM at 16000 Hz), which are processed by the Advanced Open Core SDK. The TensorFlow Lite Micro Audio Frontend Model extracts 80 features from the audio in 30ms windows, quantized to 8-bit values. These features are fed into the TensorFlow Lite Micro Siren Detection Model, which detects siren sounds. The output is further filtered and smoothed to improve accuracy, and the final siren detection result triggers an alert or response. The complete system, including audio preprocessing and detection, occupies 370KB on the hardware. More details on what models were trained and evaluated are provided in Chapter 5.

4.4. Post Processing

The model performs effectively in detecting sirens in individual instances. However, real-world scenarios involve continuous streaming of data. This presents the challenge of processing the data in defined windows, while also allowing for a more refined system. Techniques such as adaptive thresholding, window methods, energy-based detection, and filtering can be utilized to enhance performance. Further details are available in Chapter 6.

5 Design and Implementation

5.1. Step 1 - Data Overview

Table 5.1 provides a summary of the key datasets used in this research, and Table 5.2:

Dataset Name	Description	File Format	Sample Rate	No of Files	File Length	Annotations
Emergency Vehicle Siren Sounds	Contains siren sounds of ambulances, fire trucks, and traffic noise.	.wav	16kHz	597	1-20s	Annotated by directory name, SNR
Urban Sound 8K	Labeled sound excerpts across 10 urban sound classes.	.wav	16kHz	8,732	≤4s	Annotated with sound class
AudioSet	Collection of siren samples including various types of sirens.	.wav	16kHz	4,000	10s	Tagged with type of siren
ESC-50	Environmental recordings across 50 different classes.	.wav	16kHz	2,000	5s	Labeled with sound class
Large-scale Audio Dataset for Emergency Vehicle Sirens and Road Noises	Includes sirens and road noises with varying background noise levels.	.wav	16kHz	1,834	3-20s	Annotated with type of noise

Table 5.1: Key Datasets used for siren detection.

5.1.1. Siren Datasets

1. **Emergency Vehicle Siren Sounds:** To train the network to recognize distinct emergency vehicle sirens, a dataset of clean recordings was compiled. This dataset includes sounds from ambulances (199 files), firetrucks (198 files), and traffic (200 files). All recordings are in 16kHz, 16-bit, mono, .wav format, with each file lasting 3 seconds. These recordings are ideal for training models to differentiate between various types of emergency vehicle sirens, ensuring the dataset encompasses a wide range of real-world variations to improve model robustness.

2. **Urban Sound 8K :** To provide a comprehensive auditory landscape for training, the Urban Sound 8K dataset was utilized. It contains 929 siren files, 429 car horn files, 374 gunshot files, and approximately 7000 files across other urban noise categories such as air conditioners and children playing. These recordings, up to 4 seconds long, are in 16kHz, 16-bit, mono, .wav format. This dataset is particularly useful for teaching models how to distinguish between sirens and other urban sounds.

3. **Google AudioSet :** The Google AudioSet provides a comprehensive collection of audio samples, including a wide variety of siren sounds. This dataset includes 411 siren ambulance files and 639 siren police files, each being 10-second clips in 16kHz, 16-bit, mono, and .wav format. This diversity is beneficial for enhancing model robustness. A key challenge with this dataset is that some clips may contain periods without sirens, necessitating careful pre-processing to ensure relevance and consistency.

5.1.2. Urban Noise Datasets

1. **Road Noises:** To accurately train the model to recognize sirens amid typical road noise, a large-scale dataset of road and traffic sounds was collected. This dataset includes 902 files, each

ranging from 3 to 20 seconds in length, recorded in 16kHz, 16-bit, mono, and .wav format. These recordings are essential for teaching models to distinguish between sirens and background road noise. A significant challenge is ensuring the dataset captures a representative variety of road noises without overshadowing the siren sounds.

2. **Realistic Urban Sound Mixture Dataset:** The Realistic Urban Sound Mixture Dataset contains recordings of various urban environments, providing realistic mixtures of sounds found in cities. With approximately 2000 files, each ranging from 15 seconds to 2 minutes in length and recorded in 16kHz, 16-bit, mono, and .wav format, this dataset enhances model performance by training on complex urban soundscapes. The primary challenge is managing long recordings and extracting relevant segments for training.
3. **ESC-50:** The ESC-50 dataset is a labeled set of environmental recordings encompassing 50 different classes, including many urban noise sources. It contains 2000 files (40 clips per class), each being 5-second clips in 16kHz, 16-bit, mono, and .wav format. This dataset serves as a negative set to train models on distinguishing sirens from other environmental sounds. Ensuring the dataset's relevance to urban noise contexts is a critical challenge.
4. **Animal Sounds (Sound similar to siren):** This dataset includes sounds of animals that could be mistaken for sirens, such as bird calls, dog barks, and other similar noises. The files vary in length from 1 to 20 seconds and are recorded in 16kHz, 16-bit, mono, and .wav format. This dataset is crucial for training models to accurately distinguish between animal sounds and emergency sirens. Collecting a comprehensive set of diverse animal sounds is a significant challenge.
5. **Bird Calls:** The bird calls dataset is a large collection of bird calls, particularly those with siren-like qualities, totaling 20GB in size. The recordings are in 16kHz, 16-bit, mono, .wav format. This dataset helps reduce false positives in siren detection models by including potential confounding bird calls. Reviewing and annotating this extensive dataset to ensure its quality and relevance is an ongoing challenge.

5.1.3. Data Augmentation

A total of 93 hours of data were collected for the project, and an additional 186 hours of data were generated using various augmentation techniques. These techniques help to artificially increase the size and diversity of the dataset without collecting new data. Applying these transformations to the existing data exposes models to a wider range of scenarios, thus improving their generalization and robustness. **Time Stretching** involves altering the speed of the audio without changing its pitch, allowing the model to recognize sounds at different speeds, which is particularly useful in real-world scenarios where the speed of sounds may vary. **Gain Augmentation** changes the amplitude of the audio signal without affecting its pitch or speed. This technique is essential for situations where the identical sound may be captured at various volume levels, making the model more adaptable to these differences. Through gain augmentation, we replicate distinct loudness levels, which improves the model's capacity to manage a variety of audio inputs. **Noise Injection** adds random noise to the audio clips, simulating background noise and making the model more robust to real-world conditions where background noise is prevalent. This technique helps the model learn to distinguish the primary sound from the noise. **Cut and mix** cutting and mixing, pieces together different kinds of sirens together, ensuring the model can still identify the variations in sirens accurately.

Category	**Siren**	**Background Noise**
Training	80752	82503
Validation	2853	34221
Total	83605	116724

Table 5.2: Number of Samples of 5 sec audio

5.2. Step 2 - Detailed Feature Extraction Process

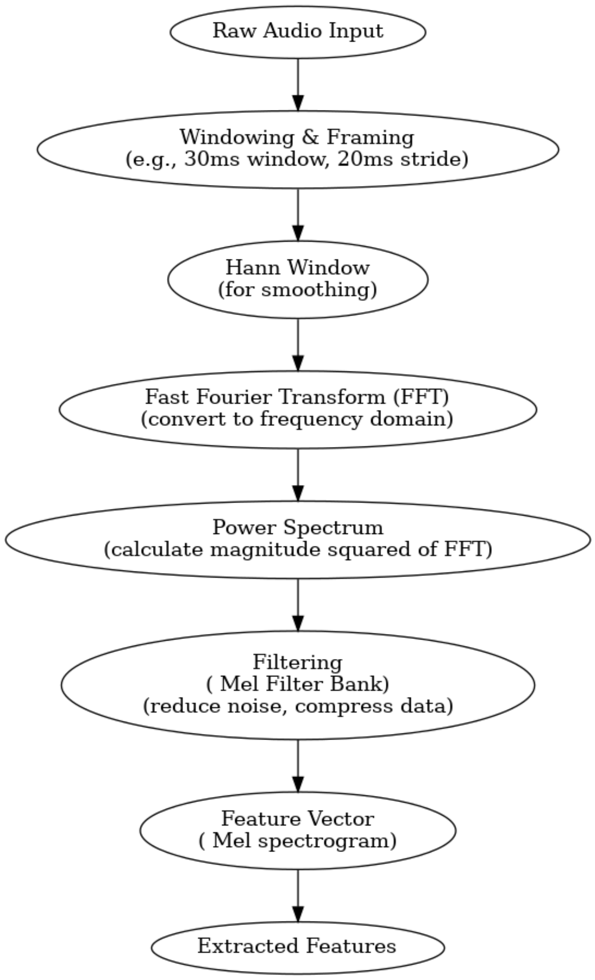

Raw Audio Input
Windowing & Framing (e.g., 30ms window, 20ms stride)
Hann Window (for smoothing)
Fast Fourier Transform (FFT) (convert to frequency domain)
Power Spectrum (calculate magnitude squared of FFT)
Filtering (Mel Filter Bank) (reduce noise, compress data)
Feature Vector (Mel spectrogram)
Extracted Features

Figure 5.1: Process of feature extraction using the TensorFlow Signal Library.

The project employs a comprehensive pipeline for feature extraction, as depicted in Figure 5.1. This methodology guarantees precise and efficient extraction of audio features for later classification tasks.

The following steps outline the feature extraction process depicted in the diagram:

- **Signal Windowing:** The raw audio signal is first divided into overlapping windows using a SignalWindow operation. This step segments the continuous audio stream into manageable frames, typically 480 samples long with a shift of 160 (480-320=160) samples.
- **Reshaping:**The segmented audio windows are reshaped to match the input requirements of the subsequent operations.
- **Auto-scaling:** The SignalFlatAutoScale operation normalizes the audio signal to ensure uniform amplitude across all samples.
- **Energy Calculation:** The SignalEnergy operation computes the energy of each windowed segment, which helps in identifying significant sound events.
- **Spectral Subtraction:** The Signal Filter Bank Spectral Subtraction module applies spectral subtraction to enhance the signal-to-noise ratio (SNR). This step involves several sub-operations, including smoothing, clamping, and gain adjustments to mitigate the impact of noise.
- **Mel Filter Bank:** The filtered signal is passed through a SignalFilterBank that applies a series of Mel-scale filters. This step converts the frequency domain representation into the Mel scale,

which better approximates human auditory perception.

- **Logarithmic Scaling and Per-Channel Energy Normalization (PCAN):** The SignalPCAN operation performs per-channel energy normalization, followed by a logarithmic transformation to compress the dynamic range of the signal. This step is crucial for emphasizing the relative differences in frequency components.
- **Feature Scaling and Quantization:** The final stages involve additional scaling (Add, Div, Add, Minimum, Maximum) and casting operations to prepare the features for the neural network input. The features are scaled to fit within the required input range and quantized to reduce memory and computation requirements.

The output of this pipeline is a set of Mel-spectrogram features that capture the essential characteristics of the siren sounds, ready for classification by the machine learning model. As can be visualized from Figure 5.2

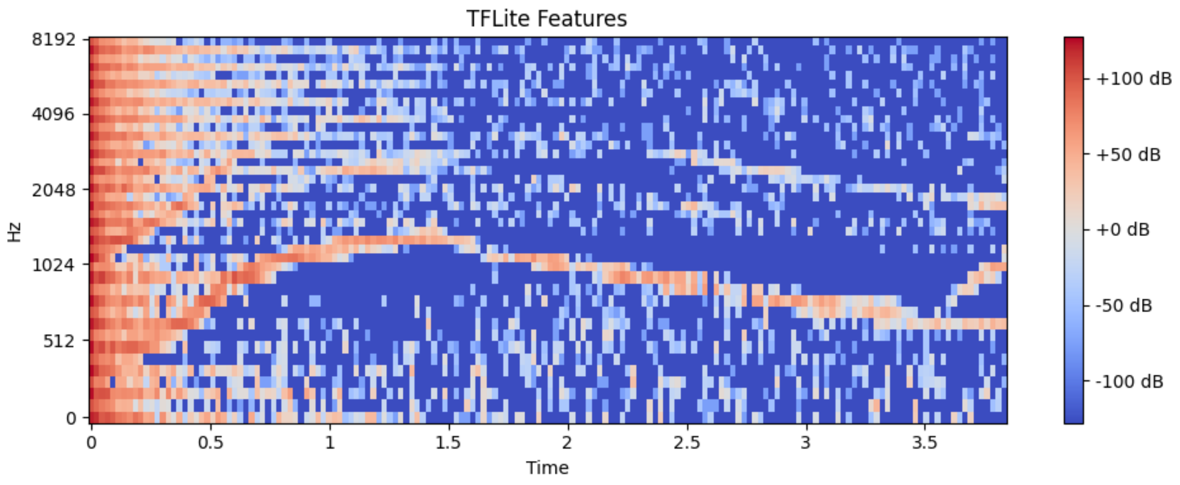


Figure 5.2: Visualization of how the extracted features look.

5.3. Step 3 - Identification of optimal architecture

In this section, we explore the different neural network architectures evaluated in this study, focusing on Fully Connected Neural Networks (FCNNs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory Networks (LSTMs). Each architecture was tested with various hyperparameters to identify the optimal configuration, the model was quantized to evaluate its size constraints in order to find and evaluate the right fit of under 300KB as per memory requirements from NXP.

5.3.1. FCNN

Fully Connected Neural Networks (FCNNs), also referred to as Dense Neural Networks, represent a form of deep learning model in which each neuron in a given layer is linked to every neuron in the subsequent layer. This design is generally applied to tasks where the spatial hierarchy of features is not crucial, making them well-suited for different kinds of non-image data. In this research, extensive experiments were performed to ascertain the ideal architecture and hyperparameters. The architecture of an FCNN consists of multiple dense layers, with each layer fully connected to the next. These networks are powerful due to their dense connectivity, allowing them to learn complex representations of the data. The architecture used in this study varied across different trials to find the optimal configuration for our specific tasks.

- **Dense Layers:**
  - Number of Units: The number of neurons in each dense layer. This varied from 32 to 256 in our trials.

  - Activation Function: The activation function used in dense layers was primarily ReLU (Rectified Linear Unit), with some trials exploring Tanh and Leaky ReLU.

- **Hyperparameter Tuning:** The primary hyperparameters tuned were:
  - Learning Rate: The learning rate controls the step size during the optimization process. The learning rates tested were 1×10^{-5}, 7×10^{-5}, 1×10^{-4}, and 1×10^{-3}.
  - Optimizer: Algorithms used to minimize the loss function. The optimizers tested included SGD (Stochastic Gradient Descent), Adam, and RMSprop.
  - Number of Dense Layers: The depth of the network, which was varied between 1 to 9 layers.
  - Number of Units in Each Layer: The number of neurons in each dense layer, with possible values ranging from 8 to 256.
- **Performance Metrics** The performance of each trial was evaluated using key metrics: accuracy, precision, recall, and loss for both training and validation datasets. These metrics provided insights into the model's ability to generalize to unseen data. These experiments provide valuable insights into performance and size as can be seen in Table 5.3 into evaluating FCNN model architectures for different hardware platforms, where constraints and performance requirements may vary.

Metric	4-layer FCNN	9-layer FCNN
Loss	1.5455	0.4745
Precision	0.2418	0.3363
Recall	0.9614	0.9547
AUC	0.9173	0.9663
Accuracy	0.7267	0.8273
Model Size	198 KB	1951 KB

Table 5.3: Performance Metrics and Model Sizes for 4-layer and 9-layer FCNNs.

5.3.2. CNN

CNNs are a class of deep neural networks commonly applied to analyzing visual imagery. We will detail the architecture used, the hyper-parameters tuned, and the performance metrics achieved through various trials. The architecture of a Convolutional Neural Network consists of multiple layers designed to automatically and adaptively learn spatial hierarchies of features from input images. The typical layers in a CNN include convolutional layers, pooling layers, batch normalization layers, dropout layers, and dense (fully connected) layers. The architecture used in this study varied across different trials to find the optimal configuration for our specific tasks.

- **Convolutional** layers are the core building blocks of a CNN. These layers apply convolutional operations to the input, passing the result to the next layer. The main components of convolutional layers used in our trials are:
  1. **Filters (Kernels):** The number of filters in each layer defines how many feature maps are produced. In our trials, the number of filters varied between 32 and 256.
  2. **Kernel Size:** The size of the filter matrix. A common choice is a 3x3 kernel, which was used in our trials.
  3. **Activation Function**: Non-linear activation functions are applied after the convolution operation. The common activation functions used are ReLU (Rectified Linear Unit), Tanh, and Leaky ReLU.

  The number of convolutional layers ranged from 1 to 3 across different trials. Each layer was followed by an activation function and a max pooling layer to downsample the feature maps, reducing the computational load and controlling overfitting.

- **Dense Layers:** After the convolutional layers, the output feature maps are flattened into a 1D vector and fed into one or more dense layers. These layers perform the final classification based on the learned features. The parameters for dense layers included:
    1. **Number of Units:**The number of neurons in the dense layer. This varied from 32 to 256 in our trials.
    2. **Activation Function**: The activation function used in dense layers was primarily ReLU.

  The final layer in the network was a dense layer with a softmax activation function, producing probabilities for each class in the classification task.
- **Hyperparameter Tuning** Hyperparameters are crucial in determining the performance of a CNN. In this study, various hyperparameters to optimize the network's performance were experimented with. The primary hyperparameters tuned were:
    1. **Learning Rate:** Controls the step size during the optimization process. The learning rates tested were 1×10^{-5}, 7×10^{-5}, 1×10^{-4}, and 1×10^{-3}.
    2. **Optimizer:** Algorithms used to minimize the loss function. The optimizers tested included SGD (Stochastic Gradient Descent), Adam, and RMSprop.
    3. **Number of Convolutional Layers:** The depth of the network, which varied between 1 and 4 layers.
    4. **Number of Filters in Each Layer:** The number of filters for each convolutional layer, with possible values ranging from 32 to 256.
    5. **Kernel Size:**The size of the convolutional filters, consistently set to 3x3 in our trials.
    6. **Number of Dense Units**: The number of neurons in the dense layer, varied from 32 to 256 units.
- **Performance Metrics** The performance of each trial was evaluated using key metrics: accuracy, precision, recall, and loss for both training and validation datasets. These metrics provided insights into the model's ability to generalize to unseen data.
- **Highest Achieved Metrics** From the trials, the highest values achieved for the 1-layer and 3-layer CNN test metrics were:

Metric	**1-layer CNN**	**3-layer CNN**
Loss	2.1711	0.1079
Precision	0.2109	0.7297
Recall	0.9615	0.9327
AUC	0.8840	0.9853
Accuracy	0.6744	0.9630
Model Size	441 KB	3150 KB

Table 5.4: Performance Metrics and Model Sizes for 1-layer and 3-layer CNNs.

***For some of the trials, check Appendix A*

Reflection Point

The accuracy given by the four-layered CNN is very promising and seems to be the initial choice of architecture for deploying on hardware. However, due to hardware constraints, it was not feasible to store a 5-second-long audio file in real-time and then process it within 512KB; hence, further experiments on LSTM were conducted to take advantage of its flexible nature. These experiments are given below:

5.3.3. LSTM

Long Short-Term Memory Networks (LSTMs) constitute an advanced variant of recurrent neural networks (RNNs) adept at capturing and preserving long-term dependencies within sequential data. LSTMs demonstrate exceptional efficacy in applications where temporal dynamics are paramount, such as time series forecasting, natural language processing, and sequence prediction. This research scrutinizes the performance of diverse LSTM architectures through meticulous hyperparameter tuning to ascertain the optimal configuration for our specific tasks.

The LSTM network was trained on 5-second data, but while converting it into TFLite models, the step size was changed to give an output for every 50ms of audio it processes. This allowed smaller chunks of audio to be processed, more quickly, owing to the improvement in the real-time response of the system as a whole.

LSTM Architecture The architecture of an LSTM network typically consists of multiple LSTM layers, followed by dense layers and dropout layers. The fundamental components of an LSTM network include:

- **LSTM Layers**:
  - **Number of Units**: The number of neurons in each LSTM layer. This varied from 50 to 200 in our trials.
  - **Activation Functions**: The `tanh` function is commonly used for the cell state, while `sigmoid` functions are used for the input, output, and forget gates within the LSTM cells.
- **Dense Layers**:
  - **Number of Units**: The number of neurons in each dense layer, varied from 32 to 256 in our trials.
  - **Activation Function**: The ReLU (Rectified Linear Unit) function was predominantly used, except for the output layer, which used a softmax activation function for classification tasks.
- **Dropout Layers**: These layers help in regularizing the model by randomly setting a fraction of input units to 0 at each update during training time, which helps prevent overfitting.

Hyperparameter Tuning The primary hyperparameters tuned included:

1. **Learning Rate**: Controls the step size during the optimization process. The learning rates tested were 1×10^{-5}, 7×10^{-5}, 1×10^{-4}, and 1×10^{-3}.
2. **Optimizer**: Algorithms used to minimize the loss function. The optimizers tested included SGD (Stochastic Gradient Descent), Adam, and RMSprop.
3. **Number of LSTM Layers**: The depth of the network, which was varied between 1 and 3 layers.
4. **Number of Units in Each LSTM Layer**: The number of neurons in each LSTM layer, ranging from 50 to 200 units.

Performance Metrics The performance of each trial was evaluated using key metrics: accuracy, precision, recall, and loss for both training and validation datasets. These metrics provided insights into the model's ability to generalize to unseen data. The highest-achieving metrics from the trials were:

Key Observations

- **Trial ID 0013** achieved the highest validation accuracy of 0.9619 refer to Appendix Table A.3.
- The optimal learning rate was found to be 1×10^{-4}.
- The Adam optimizer provided the best results in terms of stability and performance.

Metric	LSTM
Validation Loss	0.1005
Validation Precision	0.7891
Validation Recall	0.8687
Validation AUC	0.9723
Validation Accuracy	0.9237
Model Size	239 KB

Table 5.5: Validation Performance Metrics and Model Size for LSTM.

- The configuration with 2 LSTM layers, each having 120 units, yielded the highest metrics.

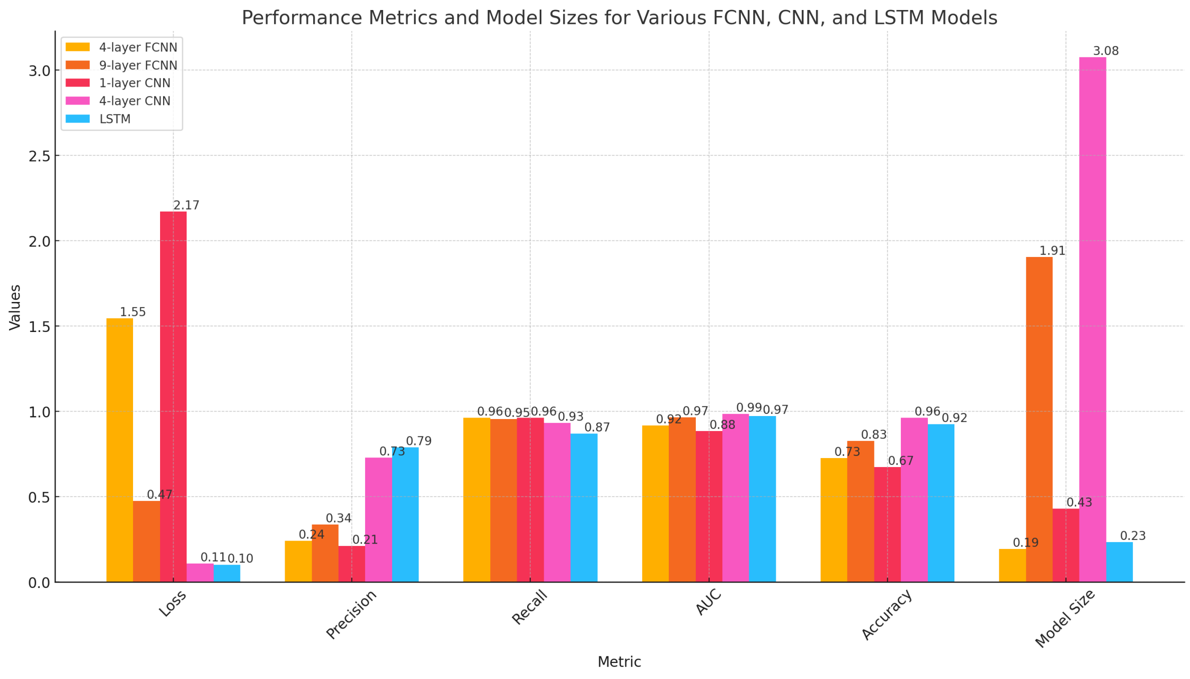


Figure 5.3: Comparison of all the main models' metrics, model sizes in MB.

Conclusion The comprehensive hyperparameter tuning and architecture experimentation with LSTM networks demonstrated that a well-configured LSTM could effectively capture long-term dependencies in sequential data, leading to high performance on the given tasks. The study highlighted the importance of careful tuning of learning rates with the worst validation accuracy of 45% to the best of 97%, optimizers, and layer configurations to achieve optimal results. This LSTM architecture is employed in subsequent stages of our research to leverage its ability to handle complex sequential data efficiently. CRNNs were not explored due to the fact that convolutional layers need the input size to be fixed, and due to this nature, the flexibility to train on 5sec/3sec data and run inference on every window(30ms) is lost. Hence, LSTMs proved to be the best option.

5.4. Step 4 - Post-processing strategies

For detecting sirens, post-processing methods are essential for improving the accuracy of machine learning models' predictions. These methods help in minimizing false positives and increasing the detection accuracy in environments with a lot of noise. This chapter presents a thorough review of the distinct post-processing approaches applied and tested for siren detection.

5.4.1. Simple Thresholding

Simple thresholding is the most basic form of post-processing. It involves setting a fixed threshold value, above which the model's prediction is considered a positive detection (siren present).

$$\text{Detection} = \begin{cases} 1 & \text{if prediction} > \text{threshold} \\ -1 & \text{otherwise} \end{cases}$$

- Simple to implement and understand.
- No computational cost.
- Does not account for the temporal continuity of siren sounds.
- Sensitive to noise, leading to higher false positives.

5.4.2. Energy-Based Detection

Energy-based detection accumulates the energy of the predictions over time. This method assumes that the presence of a siren will result in consistently high prediction values over a period. Accumulate energy:

$$E(t) = \left(E(t-1) + \text{prediction(t)}^2\right)$$

Detection:

$$\text{Detection} = \begin{cases} 1 & \text{if } E > 5.0 \\ -1 & \text{otherwise} \end{cases}$$

- Takes into account the temporal nature of the siren sounds.
- Reduces false positives by requiring sustained high energy for detection.
- May introduce a delay in detection.
- Requires tuning of energy accumulation parameters. (The tuned value for the test data was set to 5 by experimentation)

5.4.3. Window-Based Detection

Window-based detection uses a sliding window of fixed size to smooth out the predictions. It sums the predictions within the window and applies a threshold to the sum.

Sliding window sum:

$$S = \sum \text{predictions in window}$$

Detection:

$$\text{Detection} = \begin{cases} 1 & \text{if } S > \text{threshold} \\ -1 & \text{otherwise} \end{cases}$$

- Smooth out short-term fluctuations in the predictions.
- Provides a balance between responsiveness and noise reduction.
- The choice of window size and threshold can significantly affect performance.
- May introduce some latency in detection.

5.4.4. Savitzky-Golay Filter

The Savitzky-Golay filter is a digital filter that can smooth a time series by fitting successive polynomials to the data points[39]. This method is particularly useful for reducing noise in the predictions. Apply Savitzky-Golay filter:

$$\text{filtered\_predictions} = \text{savgol\_filter}(\text{predictions}, \text{window\_length}, \text{polyorder})$$

Detection:

$$\text{Detection} = \begin{cases} 1 & \text{if filtered\_predictions}[-1] > 0.67 \\ -1 & \text{otherwise} \end{cases}$$

- Provides effective noise reduction.
- Maintains the shape of the signal better than simple moving averages.
- Computationally more intensive than simpler methods.
- Requires careful selection of filter parameters (window length and polynomial order).

5.4.5. Adaptive Thresholding

Adaptive thresholding adjusts the detection threshold dynamically based on the average prediction values over a window. This method can adapt to varying noise levels.

Calculate the mean of predictions in the window:

$$\mu = \text{mean}(\text{predictions in window})$$

Detection:

$$\text{Detection} = \begin{cases} 1 & \text{if prediction} > \mu + 0.1 \\ -1 & \text{otherwise} \end{cases}$$

- Adapts to changes in background noise levels.
- Can reduce false positives in varying noise conditions.
- More complex than fixed thresholding.
- Performance depends on the characteristics of the noise environment.

5.4.6. Majority Voting

Majority voting involves taking a fixed-size window of predictions and classifying the window as a siren if the majority of predictions within the window are positive.

Count positive predictions in the window:

$$C = \sum(\text{prediction} > 0.5)$$

Detection:

$$\text{Detection} = \begin{cases} 1 & \text{if } C > \frac{\text{window size}}{2} \\ -1 & \text{otherwise} \end{cases}$$

- Robust to occasional false positives within the window.
- Simple to implement and interpret.
- The window size needs to be appropriately chosen to balance between responsiveness and noise reduction.
- May not be effective in rapidly changing noise environments.

5.4.7. Median Filter

The median filter post-processing technique uses a sliding window to compute the median of the predictions within the window. This method is effective in reducing spikes and outliers. Compute the median of predictions in the window:

$$M = \text{median}(\text{predictions in window})$$

Detection:

$$\text{Detection} = \begin{cases} 1 & \text{if } M > 0.25 \\ -1 & \text{otherwise} \end{cases}$$

- Effective in reducing outliers and spikes.
- Provides a robust measure that is less sensitive to extreme values.
- Requires maintaining a window of past predictions.
- The choice of window size and threshold can affect performance.

6 Evaluation

6.1. Training procedure

The final decided LSTM model was trained using the TensorFlow framework, leveraging its robust support for neural network implementation and GPU acceleration. The following steps were adopted for the training process:

1. **Data Augmentation:** Various data augmentation techniques were applied to increase the diversity of the training data, including time stretching, pitch shifting, and adding background noise.
2. **Batch Processing:** The training data was divided into mini-batches with a size of 64 to optimize the learning process and ensure efficient memory usage.
3. **Optimization Algorithm:** The RMSProp optimizer was used to update the model weights iteratively, maximizing the validation precision.
4. **Loss Function:** The binary cross-entropy loss function was utilized to measure the performance of the classification model.
5. **Learning Rate Schedule:** $1 \times 10-4$ learning rate was used to improve convergence and prevent overfitting.
6. **Regularization Techniques:** Dropout was applied to mitigate overfitting and improve the model's generalization capability.
7. **Number of Epochs:** The model was trained for 10,000 epochs with early stopping and a patience level of 50, ensuring sufficient iterations for the convergence of the learning process.
8. **Evaluation Metrics:** Recall, precision, AUC, F1 score, and accuracy on the hold-out test dataset were evaluated.

6.2. Impact of Hyper-parameter tuning

Goal: To optimize performance Hyper-parameter tuning consists of methodically modifying the LSTM model's hyper-parameters to attain optimal performance. Throughout this process, different hyper-parameters like the learning rate, batch size, and the number of epochs were adjusted to enhance the model's accuracy.
The tuning process involved using techniques like grid search and random search to find the optimal combination of hyper-parameters. This resulted in a significant improvement in the model's accuracy, demonstrating the importance of fine-tuning hyper-parameters for achieving better performance. Original model accuracy: 90% Optimized model accuracy: 97%

6.3. Impact of Quantization

Post Training Quantization (PTQ)

Goal: To reduce memory footprint Quantization is a technique used to reduce the model size and increase inference speed by converting the model weights from floating-point numbers to lower precision formats, such as Int16 or Int8. Quantizing the model to Int16 resulted in a modest reduction in size (only 1.93%), owing to the fact that not all operators support Int16 quantization while quantizing to Int8 resulted in a significant reduction in model size by 73.37%.

Description	Original Value	Optimized Value	Change
Model Size (Original)	556KB		
Model Size (Quantized Int16)	545 KB	-1.93%	
Model Size (Quantized Int8)	148 KB	-73.37%	

Table 6.1: Effects of Quantization on model size.

Quantization Aware Training (QAT)

Attempts to apply QAT were unsuccessful due to the necessity of maintaining flexible input sizes for the LSTM networks used in the model. The flexibility in input size is essential for training the model on sequences of varying lengths and for saving the model in different sizes. QAT typically requires fixed input sizes to effectively simulate the effects of quantization during training. Both Q_Keras and tfmot libraries were tried for QAT. The dynamic nature of LSTM input sequences, which can vary in length, posed a challenge for this approach. As a result, the expected benefits of QAT could not be realized for this particular model architecture.

6.4. Impact of Window size

Goal: To balance performance, model size of audio extraction, and improve latency.

The window size in signal processing directly affects the model's performance and accuracy. A series of experiments were conducted to determine the optimal window size for the siren detection system.

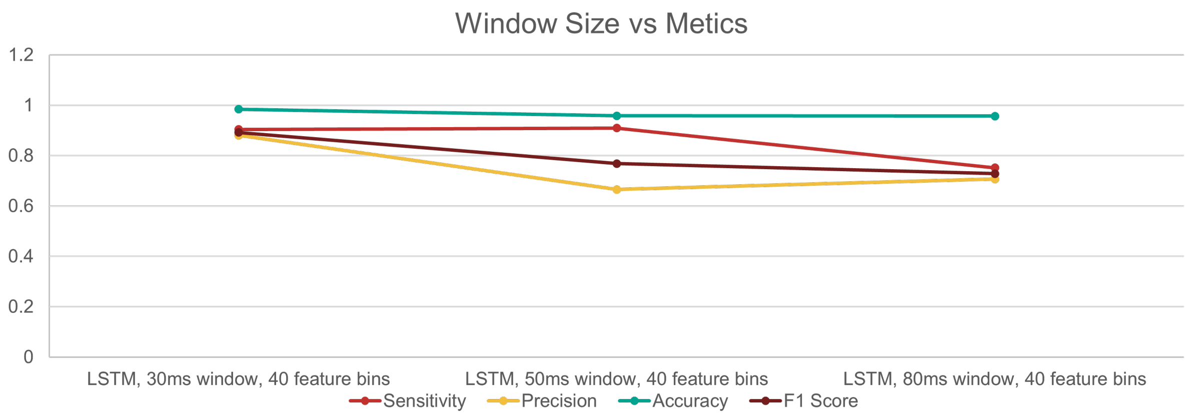


Figure 6.1: Evaluation to find the optimal window size.

Figure 6.1 illustrates that as the window size increases from 30ms to 80ms, all performance metrics (Sensitivity, Precision, Accuracy, and F1 Score) exhibit a downward trend. This suggests that using smaller window sizes enhances the model's overall performance. However, using a very fine value for window size increases compute requirements linearly. (as can be seen in Table 6.2) because the more windows, more is the processing needed for each chunk.

Window Size	Model Interpreter Ticks per File	Change %
30 ms	4,294,060	N/A
50 ms	2,572,966	-40.08%

Table 6.2: Ticks and Change Percentage for Different Window Sizes

6.5. Impact of Frequency bins

Goal: To improve performance without impacting the model size of audio extraction.

The number of frequency bins used in the feature extraction process impacts the model's ability to detect sirens accurately. Figure 6.2 illustrates that while Sensitivity shows a slight decrease, all other performance metrics (Precision, Accuracy, and F1 Score) improve with an increase in the number of feature bins. This suggests that increasing the number of feature bins enhances the model's overall performance, particularly in terms of Precision and Accuracy, even though it might slightly compromise its ability to detect actual positives. This information is valuable for optimizing the model for the best overall performance based on the specific requirements of the siren detection task.

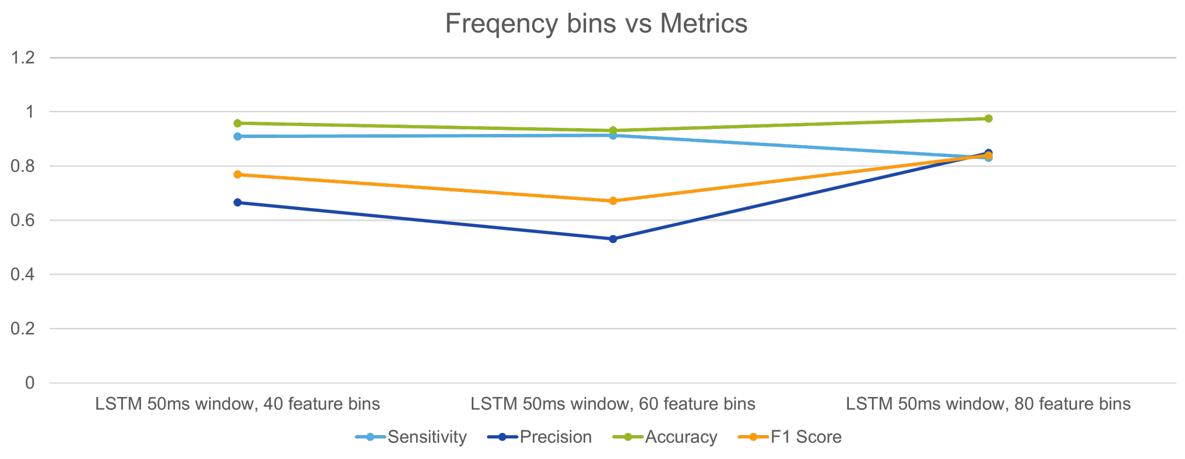
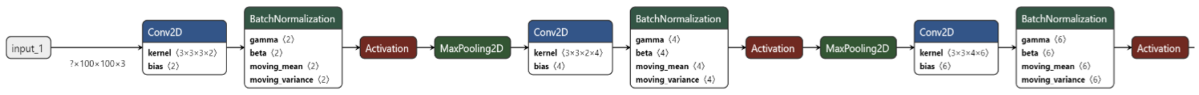


Figure 6.2: Evaluation finds the optimal frequency bins.

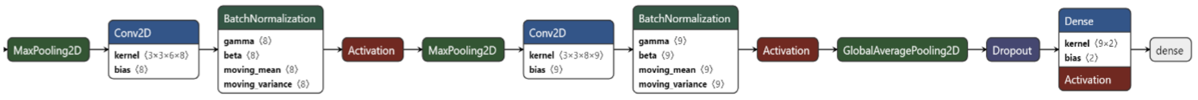
- **1 second audio:** The input size for 1 second of audio is 33x80. The model outputs predictions for each second.
- **3 seconds audio:** For a 3-second audio input, the size becomes 99x80. The outputs are combined and checked against a threshold to decide the final prediction.
- **Performance:** The model achieves 92.5% accuracy with a file size of 94KB for a 1-second audio input and 72% accuracy with a file size of 88KB for a 3-second audio input.

6.6. Evaluation of Existing Neural Architecture Search (NAS)

Neural Architecture Search (NAS) is an automated process for designing neural network architectures. In this optimization, NanoNAS [40] was used to discover a more efficient architecture for the siren detection system as can be seen in Figure 6.3 The process involved using reinforcement learning and evolutionary algorithms to explore various network topologies. However, most publicly available NAS techniques are based on convolutional neural networks (CNNs). For this application, converting the audio data of 2D arrays to an image suitable for CNNs did not show significant improvement. This approach still required a minimum chunk of 3 seconds of audio data, which is memory intensive. While CNNs did not yield the expected improvements, the exploration suggested that other neural network architectures, such as Recurrent Neural Networks (RNNs) like LSTM discussed in 5 Section 5.3.3 or Transformer-based models, might be better suited for this type of task. These architectures can handle sequential data more effectively.



(a) Neural Architecture Search (Left)



(b) Neural Architecture Search (Right)

Figure 6.3: NanoNAS Neural Architecture Search.

6.7. Final System Evaluation: Satisfying Requirements

6.7.1. Evaluation of different post-processing strategies

Key Observations: The examination of the Figure 6.4, displaying the performance metrics for diverse siren detection techniques based on four primary metrics (Precision, Recall, F1 Score, and Accuracy), uncovers numerous insights. The Savitzky-Golay approach emerges as the most effective, securing the highest recall, F1 score, and accuracy, which highlights its well-balanced and dependable nature. The Energy-Based approach also excels, attaining the highest precision along with strong recall, F1 score, and accuracy, positioning it as a formidable option. Conversely, the Simple Threshold technique is the least effective, showing the lowest precision, recall, F1 score, and accuracy, suggesting it is not reliable for siren detection. The Adaptive Threshold, Window-Based, Majority Voting, and Median Filter techniques exhibit similar performance, indicating good balance and reliability across all metrics. In summary, techniques like Savitzky-Golay and Energy-Based are preferred for siren detection due to their superior performance, whereas the Simple Threshold technique needs to be combined with another method to be considered viable.

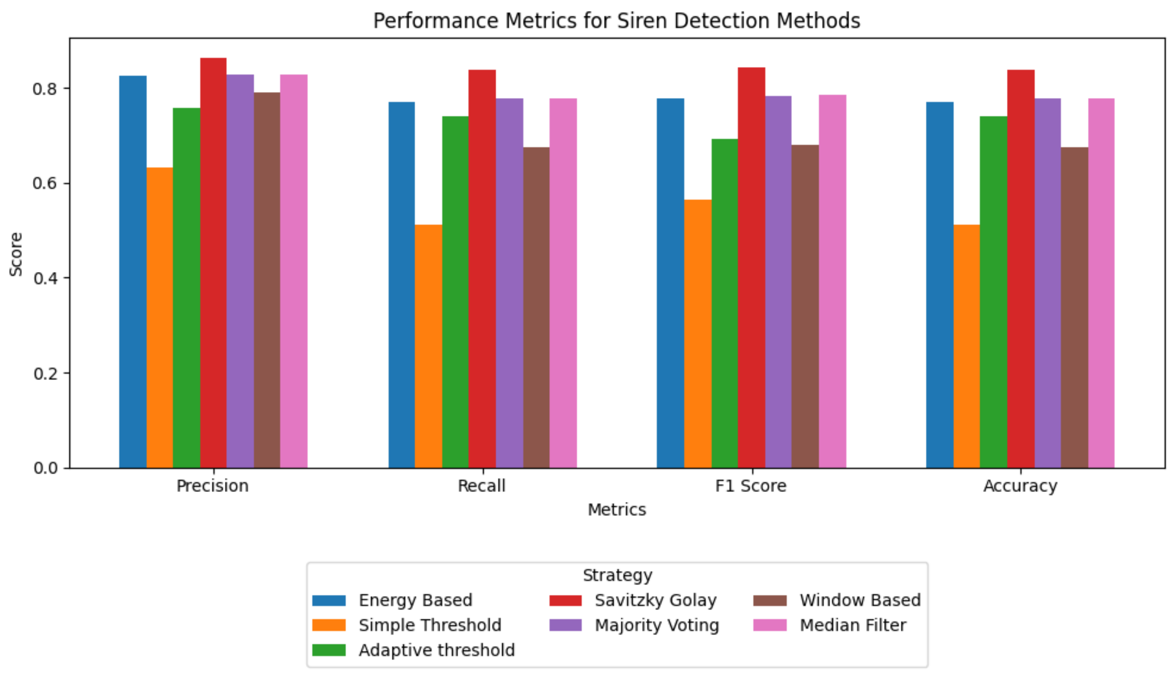


Figure 6.4: Post-Processing Strategies comparison for low noise audio. It can be observed that the best-performing strategies are Savitzky-Golay, Energy-based, and Majority Voting.

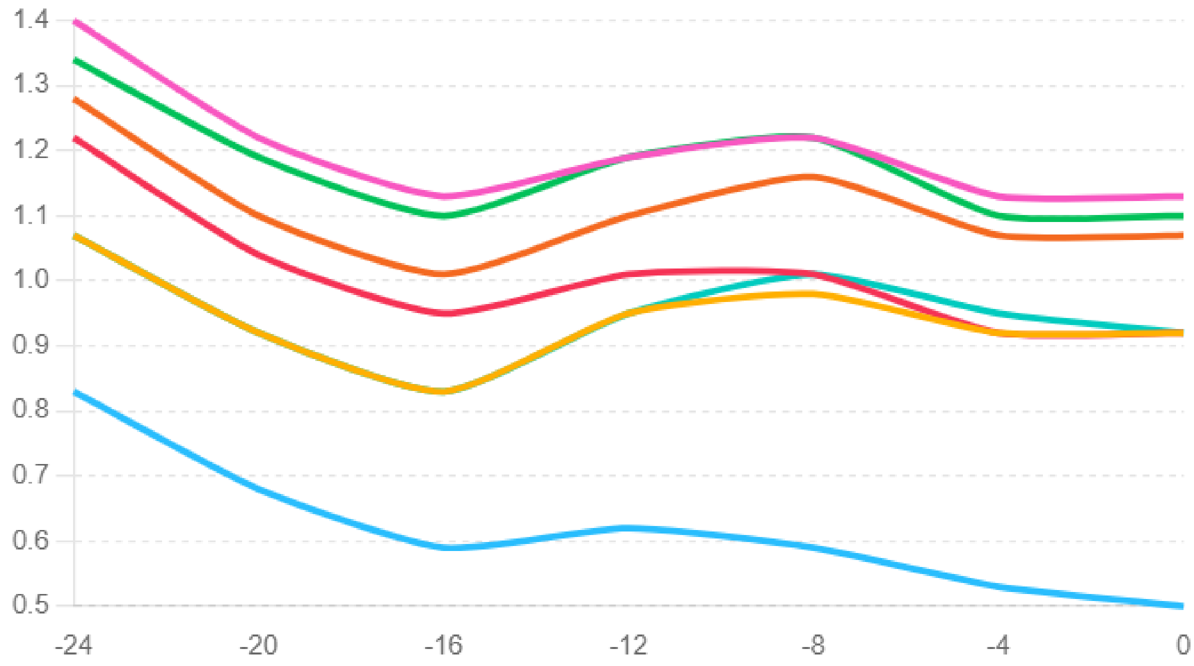


Figure 6.5: System latency comparison, a general trend of increasing turn around time with the same audio with more noise can be observed.

6.7.2. Tolerance to SNRS, latency tests

To test the tolerance to SNRs, ensuring that at -6dB most tests pass the 3 secs turnaround time, alongside measuring the effect of post-processing on latency, multiple test audio files of a total length of 30 seconds were created where the first 10 seconds were noise, next 10 seconds were siren followed by the last 10 seconds being noise again. An excerpt of how these audio files looked can be visualized from Figure 6.6 and the full analysis is presented in Table 6.3. This shows that as the noise goes down, the ability of the model to react and understand the siren increases.

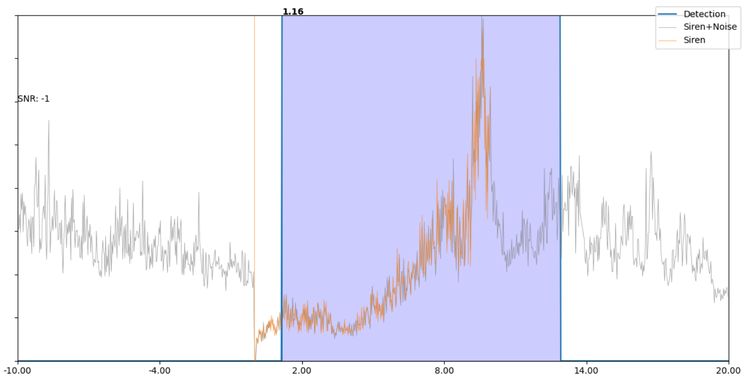


Figure 6.6: This Audio tests the latency vs SNR of audio where the first 10 seconds are noise(-10 to 0), the next 10 sec are Siren (0 to 10) and the last 10 seconds are noise (10 to 20). The audio is a mix of Police sirens with driving noise.

6.7.3. False positives test

The primary objective of this test is to assess and compare the false positive rates of multiple siren detection strategies when deployed in diverse noise environments. For each defined detection strategy, the test systematically evaluates its performance across various noise profiles (specified as noise files). It applies each strategy to aggregated noise data extracted from each environment and quantifies the occurrence of false positives, where the algorithm erroneously detects a siren signal despite its absence. By analyzing these results, this test aims to determine the most effective detection strategy that exhibits the lowest false positive rate across different noise conditions, providing critical insights for optimizing and selecting robust siren detection algorithms. Figure 6.7 shows the number of false

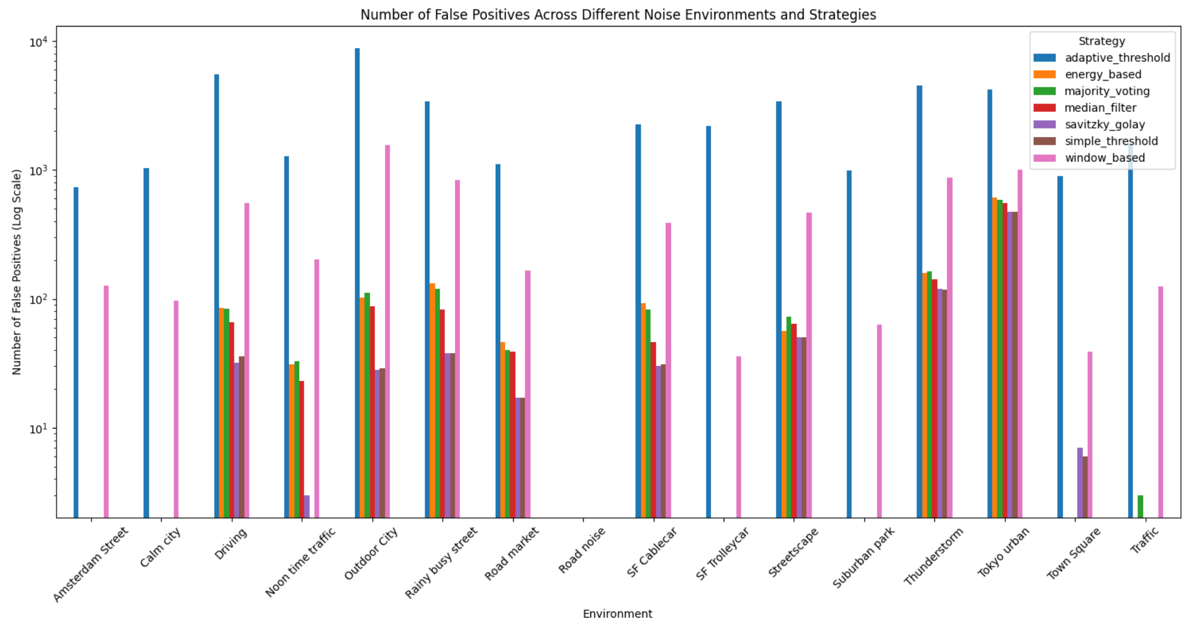


Figure 6.7: For different noise soundscapes, post-processing results and false positives were tested, the most effective being Savitzky-Golay and simple thresholding filtering.

positives detected across various noise environments using different post-processing strategies. Each strategy's effectiveness can be assessed by examining the height of the bars, where a lower number of false positives indicates a more reliable method.

- Adaptive Threshold: This method consistently shows the highest number of false positives across all environments. Particularly high in environments like Driving, Outdoor City, and SF Cablecar, indicating its unreliability in noisy settings.
- Energy Based: This method demonstrates relatively low false positives in most environments. It is one of the more reliable methods, especially in environments such as Calm City, Rainy Busy Street, and Suburban Park.
- Majority Voting: Shows moderate performance, with fewer false positives compared to Adaptive Threshold but more than Energy Based. Performs better in controlled environments like Calm City and Road Market.
- Median Filter: This method has a low number of false positives, indicating good performance. Particularly effective in environments like Traffic, and Town Square.
- Savitzky-Golay: Shows similar performance to the Median Filter, with low false positives in most environments. Effective in environments such as Road Noise, SF Trolleycar, and Town Square.
- Simple Threshold: This method also performs well, with low false positives in many environments. Shows effectiveness in environments like SF Cablecar, SF Trolleycar, and Suburban Park.
- Window Based: Demonstrates moderate performance with a varied number of false positives. Effective in environments like Calm City and Road Noise but higher false positives in environments like Outdoor City and Town Square.

The Energy Based, Median Filter, Savitzky-Golay, and Simple Threshold methods are the most reliable post-processing strategies, exhibiting the least number of false positives across various noise environments. Adaptive Threshold consistently shows the highest false positives, indicating it is the least reliable method. For environments with high noise levels, the Energy Based and Median Filter methods are particularly effective. These insights help in selecting the most appropriate post-processing strategy for siren detection in different real-world noise environments, ensuring a balance between sensitivity and reliability.

6.7.4. Overall Evaluation of Postprocessing Strategies

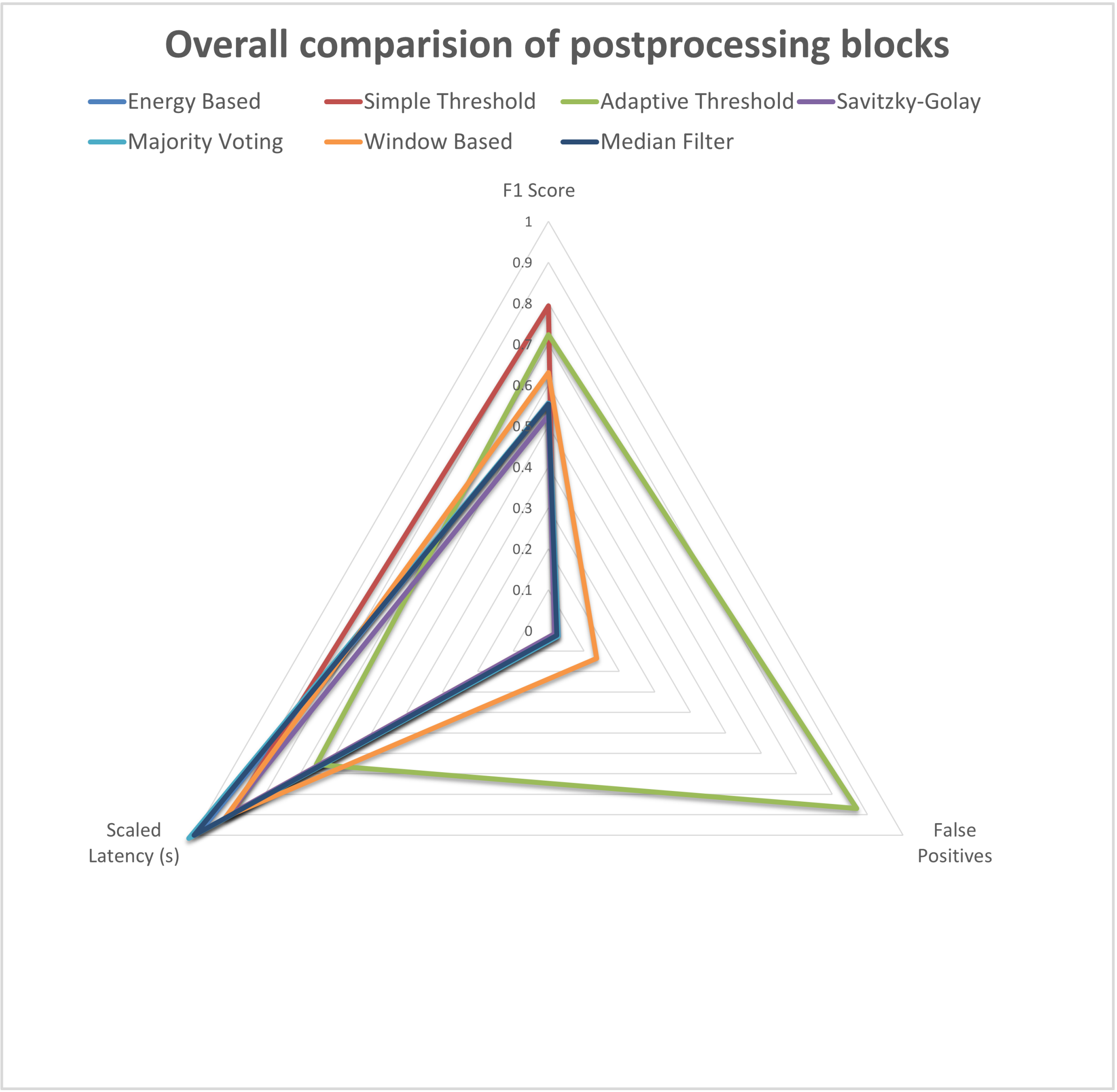


Figure 6.8: For different noise soundscapes, post-processing results and false positives were tested, the most effective being Savitzky-Golay filtering.

Figure 6.8 compares various postprocessing methods (Energy Based, Simple Threshold, Adaptive Threshold, Savitzky-Golay, Majority Voting, Window Based, and Median Filter) using three evaluation metrics: inverse F1 Score, Latency scaled down by a factor of 2, and scaled number of False Positives (scaled down by a factor of 3000). Each method is represented by a distinct colored line, and the goal is to minimize the area enclosed by each line on the chart, indicating better overall performance. Lower values in inverse F1 Score and Scaled Latency, along with fewer scaled False Positives, contribute to a smaller area, signifying a more effective postprocessing method. The energy-based method appears to have balanced performance with moderate scores in F1 Score and False Positives but slightly higher latency. The Simple Threshold method shows lower performance in terms of F1 Score and False Positives but benefits from low latency. The Adaptive Threshold method has a similar pattern to the Simple Threshold but performs slightly better in some metrics. The Majority Voting method and Median show better performance in F1 Score and False Positives but suffer from higher latency. The Window Based method exhibits moderate performance across all metrics, similar to Energy Based. Lastly, the Savitzky-Golay method demonstrates a well-rounded performance with good scores in all metrics. Overall, the chart suggests that the Savitzky-Golay method is a good choice due to its balanced performance, while other methods involve trade-offs between accuracy (F1 Score and False Positives) and latency.

6.7.5. SNR breaking point stress test

The threshold for Signal-to-Noise-Ratio (SNR) at which the system fails to achieve the 3-second response time for a siren was identified to be approximately -40 dB. At this SNR level or lower, the system's capacity to detect and respond to the siren within the designated 3 seconds becomes inconsistent or entirely inadequate. An example of this breaking point can be seen in Figure 6.9. In a hectic urban setting with loud traffic and other background noises, the signal-to-noise ratio (SNR) might decrease to -40 dB, causing the system to either respond late or miss the siren altogether. This threshold is crucial in making sure that the system functions effectively within its intended performance limits. It underscores the importance of maintaining an appropriate SNR range for dependable siren detection and response.

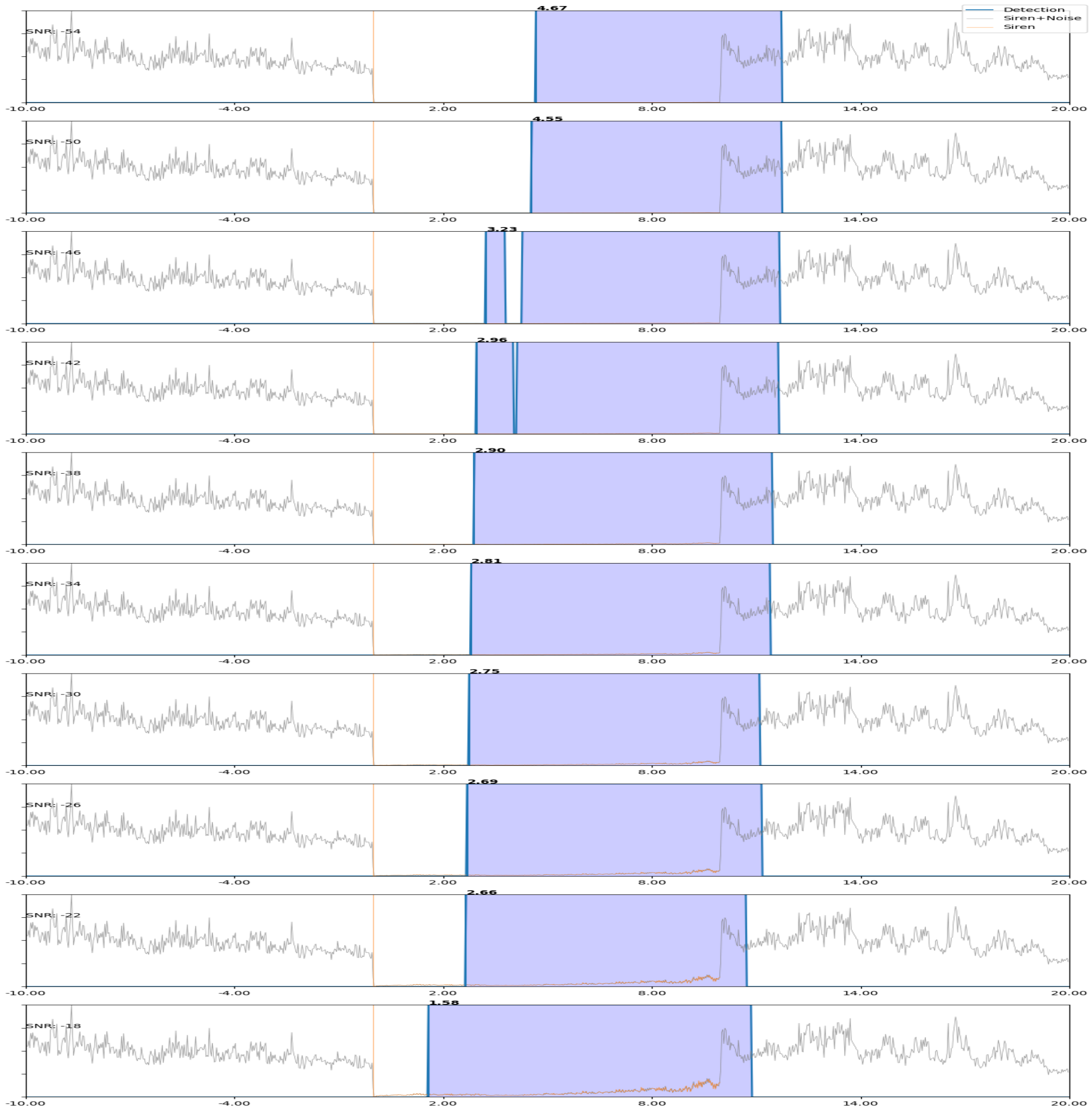


Figure 6.9: High SNR stress test: The test was conducted on a randomly chosen noise sample and siren sample, tested up to a high SNR of -54dB, The system deteriorated around -42 dB SNR for this sample and in general around -30 dB SNR. While the target SNR threshold was only -6dB.

Audio File Name	SNR (dB)	Latency (simple threshold)	Latency (energy based)	Latency (window based)	Latency (majority voting)	Latency (adaptive threshold)	Latency (savitzky golay)	Latency (median filter)
german ambulance 2+noon traffic	-24	2.81	2.87	2.48	2.81	2.00	2.81	2.87
german ambulance 2+noon traffic	-20	2.96	2.66	2.33	2.63	1.82	2.96	2.57
german ambulance 2+noon traffic	-16	1.97	2.15	2.09	2.27	1.67	2.00	2.21
german ambulance 2+noon traffic	-12	1.85	2.06	1.97	2.15	1.55	1.82	2.09
german ambulance 2+noon traffic	-8	1.76	1.97	1.88	2.06	1.43	1.76	2.00
german ambulance 2+noon traffic	-4	1.79	1.97	1.82	2.03	1.31	1.79	2.03
german ambulance 2+noon traffic	0	1.35	1.53	1.79	2.15	1.22	1.35	2.09
police siren+ Driving	-24	1.07	1.28	1.22	1.40	0.83	1.07	1.34
police siren+ Driving	-20	0.92	1.10	1.04	1.22	0.68	0.92	1.19
police siren+ Driving	-16	0.83	1.01	0.95	1.13	0.59	0.83	1.10
police siren+ Driving	-12	0.95	1.10	1.01	1.19	0.62	0.95	1.19
police siren+ Driving	-8	0.98	1.16	1.01	1.22	0.59	1.01	1.22
police siren+ Driving	-4	0.92	1.07	0.92	1.13	0.53	0.95	1.10
police siren+ Driving	0	0.92	1.07	0.92	1.13	0.50	0.92	1.10
German ambulance 1 + city traffic	-24	2.45	2.57	2.00	2.54	2.48	2.48	2.63
German ambulance 1 + city traffic	-20	1.73	1.76	1.58	1.80	1.73	1.73	1.76
German ambulance 1 + city traffic	-16	1.13	1.34	1.25	2.00	1.10	1.10	1.37
German ambulance 1 + city traffic	-12	1.07	1.25	1.16	1.45	1.07	1.07	1.31
German ambulance 1 + city traffic	-8	1.04	1.22	1.04	1.37	1.04	1.04	1.28
German ambulance 1 + city traffic	-4	2.36	2.39	1.10	1.25	2.39	2.39	2.45
German ambulance 1 + city traffic	0	0.98	1.19	0.95	1.22	1.01	1.01	1.25

Table 6.3: Latency vs. SNR vs. impact of post-processing on latency.

7 Conclusion and Future Work

This research focused on the development of an on-device machine learning (ML) system for real-time siren detection, specifically targeting resource-constrained embedded devices in vehicles for advanced driver assistance systems (ADAS). The problem statement addresses the need for accurate and real-time siren detection to enhance automotive safety and public security, moving from traditional cloud-based methods to localized data processing.

The solution proposed in this thesis involves a robust neural network model integrated with advanced signal processing techniques. The model processes audio inputs captured via a microphone and classifies them based on the presence of siren sounds by transforming audio signals into mel-spectrograms, which represent the frequency spectrum over time. The custom dataset used for training includes 280 hours of audio, comprising well-known, publicly available datasets such as ESC-50, Audioset, and UrbanSound8K, enriched with both original and augmented siren sounds alongside non-siren audio to enhance learning efficacy and robustness. Optimization techniques, such as determining the optimal window size and number of frequency bins and employing post-processing strategies like Savitzky Golay and energy-based methods, were utilized to improve detection accuracy and computational efficiency.

The siren detection system demonstrated in this research showcases the feasibility and effectiveness of deploying deep learning models on edge devices for real-time audio event detection. Through comprehensive hyperparameter tuning and architecture experimentation, especially with LSTM networks, it was evident that a well-configured LSTM can effectively capture long-term dependencies in sequential data, achieving a high performance of 96.19% accuracy in the detection of siren. The various optimizations applied, such as quantization and adjustments in window size, frequency bins, etc. further enhanced the model's efficiency, making it suitable for deployment on resource-constrained devices.

The project successfully fulfilled all its functional and non-functional requirements except one, demonstrating robust performance and efficiency (refer to Table 7.1). The memory constraints of the model size were satisfied, with the model staying within 300KB (the final model was 230KB) and the allowable memory limits on the SAF9100 audio processor [38]. The ML cycle load was well below 5% and requires 11MIPS for model inference. This ensured real-time processing capabilities. Reaction time requirements were met under various noise conditions (tested beyond the requirements), showcasing the model's responsiveness to high-noise environments. The false positive rate for low-noise audio was achieved with different strategies. However, the false positive rate for high noise audio was not met, suggesting a need for further retraining on sounds similar to sirens, such as engine noises and bells, to enhance performance. The non-functional requirements were also fulfilled, including the development of an end-to-end real-time demonstrator (see Figure 7.1) on the NXP board, comprehensive comparisons of different network architectures, assessments of vigilance enhancement compared to human

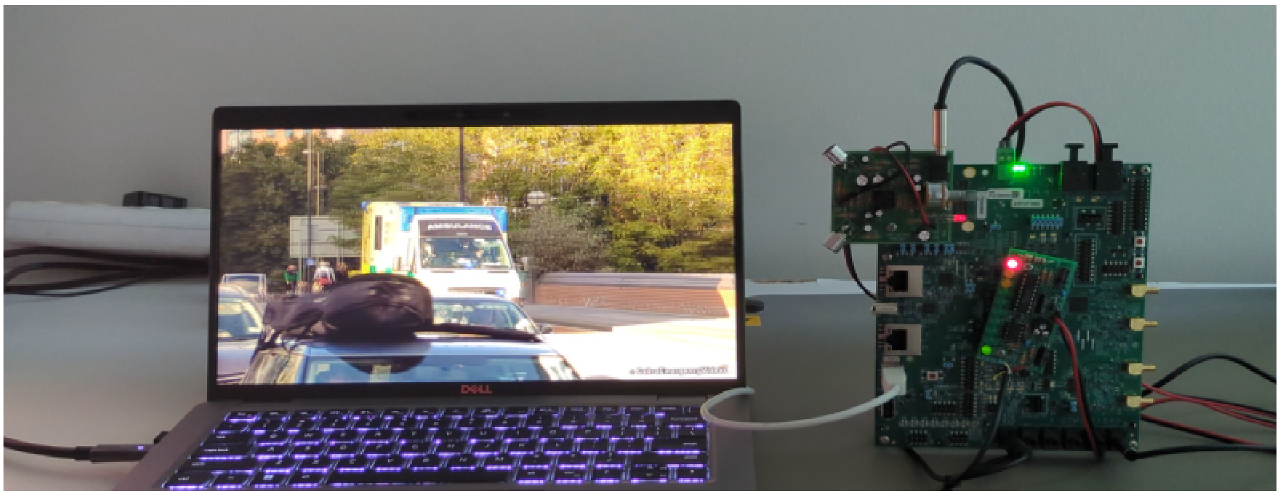


Figure 7.1: Hardware Demonstrator of the end-to-end solution.

drivers, and evaluations of optimization techniques like Quantization Aware Training (QAT) and Neural Architecture Search (NAS) to reduce network size. Overall, the project demonstrates strong adherence to the specified requirements, with ongoing improvements anticipated for high-noise scenarios.

7.1. Evaluation of Satisfying NXP requirements

Requirement Category	**Requirement Description**	**Status**
Functional Requirements		
Memory Constraints	Maximum size of Model and Buffers limited to 300KB on NXP Product Hardware with 512KB; up to 1MB if needed.	Satisfied. The first version of the model used only 270KB, and the later version with updated memory used 370KB.
Cycles	ML cycle loads must be less than 25%, ideally under 5% for real-time processing.	Satisfied. 0.007 seconds processing time for one second of audio
Performance	Reaction time are under 3 seconds in low noise (0dB SNR) and under 5 seconds in high noise (-6dB SNR).	Satisfied. All the SNR values till -24dB showed <3sec Turn-Around-Time.
	False positive rate: <1 per 10 hours of low noise audio (>1 sec); <6 per 10 hours of low noise audio (<1 sec).	The test audio was conducted for 6.8hrs of test audio and the total false positives were 24 secs for simple threshold, 42 secs for energy-based, 195 secs for window-based, 38 secs for majority voting, 1252 secs for adaptive, 23 secs for Savitzky-Golay, 33 secs for median filter.
	False positive rate: <1 per 5 hours of high noise audio (>1 sec); <6 per 5 hours of high noise audio (<1 sec).	-
Non-Functional Requirements Satisfied	Demonstrator on the SAF9100 audio processor	End-to-end real-time demonstrator on the NXP SAF9100 audio processor board with microphone and siren detection visualization.
Comparison of architectures	Comparison of memory, compute, and precision metrics of different network architectures: CNN, FC, and RNN.	Satisfied
Vigilance enhancement	Comparison of the breaking point of siren detection performance with human drivers.	Satisfied
Optimization Assessment	Investigation of optimization techniques like QAT, and NAS for reducing network size.	Satisfied

Table 7.1: Summary of customer requirements satisfied.

This work covers all the gaps discussed in Chapter 3, Data Availability and Quality are met by using an extensive dataset of audio of 278 hrs. The complexity of the model and the resource constraints discussed in the requirements of Chapter 2.2 in terms of model size (<300 KB), Cycle count <5%, overall turn around time of <3secs for high noise audio, etc. were met. The model was evaluated in real-world audio and showed a promising accuracy of more than 85% (varying according to the kind of audio) and this can be further improved by retraining on additional audio sounds such as engine sounds, trumpet bells or other siren-alike sounds. The breaking point SNR where the system no longer satisfies the 3 sec response time to siren was found to be in the range of -40dB. The entire processing happens in real time, from capturing raw audio, extracting features, model inference, and post-processing.

7.2. Future Work

Looking ahead, several avenues for future research and development can be pursued to further enhance the capabilities and performance of the siren detection system.

7.2.1. Deployment and Field Testing

Extensive field testing in diverse real-world environments is necessary to validate the robustness and reliability of the system. Collaborating with the automotive and public safety industries for pilot deployments can provide valuable feedback and drive iterative improvements to the system. For example, in the tests, it was found that the model assumed a certain kind of tram engine pick-up sound to be a siren as it was siren-like. The model can then be retrained with such data to improve its overall robustness.

7.2.2. Integration of Transformers

The potential of transformer-based architectures in audio processing is vast. Future research can explore the integration of transformer models for siren detection, which might offer improved performance over LSTMs by better handling long-range dependencies and capturing more intricate patterns in audio signals. A transformer-based architecture was attempted, a variable-size transformer could not be designed like the case for LSTM due to unsupported parameters by TFLite micro-library. Hence the idea was to deploy a transformer model-based architecture designed to handle varying lengths of audio inputs as can be seen in Figure 7.2. Figure 7.2 illustrates a possible architecture and the data flow for both 1-second and 3-second audio inputs. However, deploying a 2 model architecture on the hardware was complicated and gathering test data to evaluate such an architecture was also challenging. Overall, this is a good direction to explore in future work.

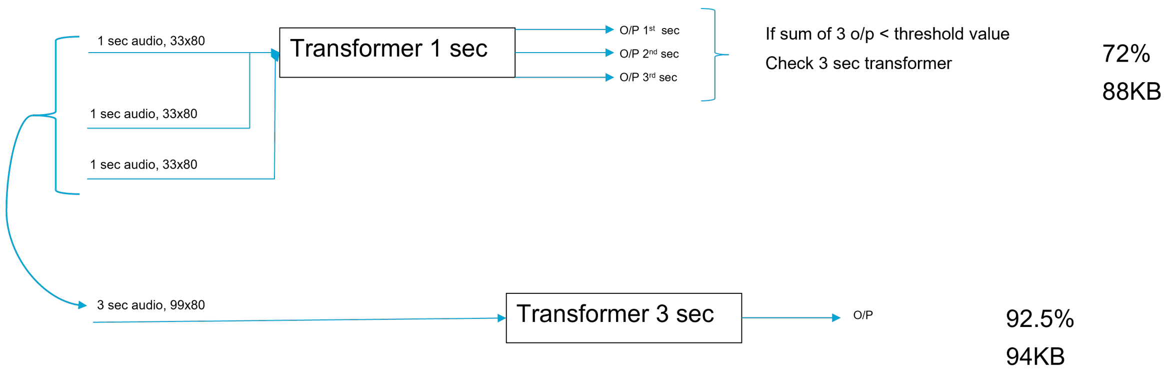


Figure 7.2: Transformer-based architecture for processing 1-second and 3-second audio inputs.

7.2.3. Multi-Modal Detection Systems

Integrating audio detection with other sensors (e.g., visual, radar, or vibration) can create a more comprehensive emergency vehicle detection system. Multi-modal approaches can leverage the strengths of different sensing modalities, improving overall reliability and accuracy.

7.2.4. Improvements in TFLite Micro

TFLite Micro currently provides a robust platform for deploying machine learning models on microcontrollers. However, further improvements can be made, like developing some custom operators which need some development time for example to enable flexible transformer architectures, Int16 support for some of the operators, etc. TFLite Micro can enable the deployment of larger and more sophisticated models without compromising performance.

7.2.5. Enhanced Quantization Techniques

While quantization significantly reduces model size and increases inference speed, exploring advanced quantization techniques like post-training quantization and quantization-aware training (QAT) tailored for dynamic input sizes can provide further improvements. These techniques can help maintain the model's accuracy while making it more suitable for deployment on low-power devices. For this a custom solution could be developed as QKeras and tfmot that were attempted in this research needed the input size to be fixed values.

7.2.6. Calibration

The calibration process involves adjusting system parameters to account for different conditions, such as ambient noise levels, various types of siren sounds, and the specific characteristics of the deployment environment.

During the initial setup, the system needs to be calibrated in a controlled environment to establish baseline parameters. This includes testing the system with different known siren sounds (specific to the region) and various noise levels to determine the best threshold values and sensitivity settings. Once the initial calibration is finished, the system should have mechanisms for dynamic adjustment to adapt to changing environmental conditions. Real-time monitoring of ambient noise levels and automatic adjustment of detection thresholds are crucial for maintaining high accuracy.

Regular maintenance and re-calibration are essential to account for any changes in the system or the environment. This can be scheduled periodically or triggered by significant changes in performance metrics, ensuring that the system continues to operate effectively over time.

By addressing these future work areas, the siren detection system can be significantly enhanced, paving the way for more advanced and reliable audio event detection solutions in various safety-critical applications.

References

[1] Michela Cantarini et al. “Acoustic Features for Deep Learning-Based Models for Emergency Siren Detection: An Evaluation Study”. In: *2021 12th International Symposium on Image and Signal Processing and Analysis (ISPA)*. 2021, pp. 47–53. DOI: 10.1109/ISPA52656.2021.9552140.

[2] Nikolaos Schizas et al. “TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review”. In: *Future Internet* 14.12 (2022). ISSN: 1999-5903. DOI: 10.3390/fi14120363. URL: https://www.mdpi.com/1999-5903/14/12/363.

[3] D Chirag Chinvar et al. “Ambulance Siren Detection using an MFCC based Support Vector Machine”. In: *2021 IEEE International Conference on Mobile Networks and Wireless Communications (ICMNWC)*. 2021, pp. 1–5. DOI: 10.1109/ICMNWC52512.2021.9688340.

[4] Gianmarco Cerutti et al. “Compact Recurrent Neural Networks for Acoustic Event Detection on Low-Energy Low-Complexity Platforms”. In: *IEEE Journal of Selected Topics in Signal Processing* 14.4 (2020), pp. 654–664. DOI: 10.1109/JSTSP.2020.2969775.

[5] Zaffar Haider Janjua et al. “IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge”. In: *Engineering Applications of Artificial Intelligence* 84 (2019), pp. 41–50. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2019.05.011. URL: https://www.sciencedirect.com/science/article/pii/S0952197619301113.

[6] Michela Cantarini et al. “Beware the Sirens: Prototyping an Emergency Vehicle Detection System for Smart Cars”. In: *Applied Intelligence and Informatics*. Ed. by Mufti Mahmud et al. Cham: Springer Nature Switzerland, 2022, pp. 437–451. ISBN: 978-3-031-24801-6.

[7] Islam Gomaa et al. “A Framework for Intelligent Fire Detection and Evacuation System”. In: *Fire Technology* 57 (2021), pp. 3179–3185.

[8] Himanshu Sharma, Ahteshamul Haque, and Frede Blaabjerg. “Machine Learning in Wireless Sensor Networks for Smart Cities: A Survey”. In: *Electronics* 10.9 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10091012. URL: https://www.mdpi.com/2079-9292/10/9/1012.

[9] Canva Pty Ltd. *Canva*. https://www.canva.com/. 2024.

[10] Gianmarco Cerutti et al. “Compact Recurrent Neural Networks for Acoustic Event Detection on Low-Energy Low-Complexity Platforms”. In: *IEEE Journal of Selected Topics in Signal Processing* 14.4 (2020), pp. 654–664. DOI: 10.1109/JSTSP.2020.2969775.

[11] Md Mohaimenuzzaman et al. “Environmental Sound Classification on the Edge: A Pipeline for Deep Acoustic Networks on Extremely Resource-Constrained Devices”. In: *Pattern Recognition* 133 (2023), p. 109025. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2022.109025. URL: https://www.sciencedirect.com/science/article/pii/S0031320322005052.

[12] Veronica Morfi and Dan Stowell. “Data-efficient weakly supervised learning for low-resource audio event detection using deep learning”. In: *ArXiv* abs/1807.06972 (2018). URL: https://api.semanticscholar.org/CorpusID:49867825.

[13] S. S. Stevens, J. Volkmann, and E. B. Newman. “A Scale for the Measurement of the Psychological Magnitude Pitch”. In: *Journal of the Acoustical Society of America* 8.3 (1937), pp. 185–190.

[14] P. Mahesha and D. S. Vinod. “LP-Hillbert transform based MFCC for effective discrimination of stuttering dysfluencies”. In: *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. 2017, pp. 2561–2565. DOI: 10.1109/WiSPNET.2017.8300225.

[15] Zrar Kh. Abdul and Abdulbasit K. Al-Talabani. “Mel Frequency Cepstral Coefficient and its Applications: A Review”. In: *IEEE Access* 10 (2022), pp. 122136–122158. DOI: 10.1109/ACCESS.2022.3223444.

[16] Asith Abeysinghe et al. “Mel frequency cepstral coefficient temporal feature integration for classifying squeak and rattle noise”. In: *The Journal of the Acoustical Society of America* 150 (July 2021), pp. 193–201. DOI: 10.1121/10.0005201.

[17] Sunil Kumar Kopparapu and M. Laxminarayana. “Choice of Mel filter bank in computing MFCC of a resampled speech”. In: May 2010, pp. 121–124. DOI: 10.1109/ISSPA.2010.5605491.

[18] Ahmed Krobba, Mohamed Debyeche, and Sid Ahmed Selouani. “Mixture linear prediction Gammatone Cepstral features for robust speaker verification under transmission channel noise”. In: *Multimedia Tools and Applications* 79 (July 2020). DOI: 10.1007/s11042-020-08748-2.

[19] Changwei Zhou et al. “Gammatone spectral latitude features extraction for pathological voice detection and classification”. In: *Applied Acoustics* 185 (2022), p. 108417. ISSN: 0003-682X. DOI: https://doi.org/10.1016/j.apacoust.2021.108417. URL: https://www.sciencedirect.com/science/article/pii/S0003682X21005119.

[20] Azza Moawad et al. “Spectrum Sensing by Cepstral Covariance Detection”. In: *IEEE Communications Letters* 26.6 (2022), pp. 1323–1327. DOI: 10.1109/LCOMM.2022.3157773.

[21] Roy D. Patterson et al. “Complex Sounds and Auditory Images”. In: Pergamon, 1992, pp. 429–446. DOI: 10.1016/B978-0-08-041847-6.50054-X.

[22] Cihun-Siyong Alex Gong et al. “Deep Learning with LPC and Wavelet Algorithms for Driving Fault Diagnosis”. In: *Sensors* 22.18 (2022). ISSN: 1424-8220. DOI: 10.3390/s22187072. URL: https://www.mdpi.com/1424-8220/22/18/7072.

[23] Amit Moondra and Poonam Chahal. “Improved Speaker Recognition for Degraded Human Voice using Modified-MFCC and LPC with CNN”. English. In: *International Journal of Advanced Computer Science and Applications* 14.4 (July 2023). Copyright - © 2023. This work is licensed under http://creativecommons.org/licenses/by/4.0/ (the “License”). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Last updated - 2023-11-27. URL: https://login.libauth.mskcc.org/login?url=https://www.proquest.com/scholarly-journals/improved-speaker-recognition-degraded-human-voice/docview/2819915906/se-2.

[24] Aankit Das et al. “A Hybrid Meta-Heuristic Feature Selection Method for Identification of Indian Spoken Languages From Audio Signals”. In: *IEEE Access* 8 (2020), pp. 181432–181449. DOI: 10.1109/ACCESS.2020.3028241.

[25] Yancai Xiao et al. “Low-Pass Filtering Empirical Wavelet Transform Machine Learning Based Fault Diagnosis for Combined Fault of Wind Turbines”. In: *Entropy* 23.8 (2021). ISSN: 1099-4300. DOI: 10.3390/e23080975. URL: https://www.mdpi.com/1099-4300/23/8/975.

[26] Shreya Chakravarty, Richa R. Khandelwal, and Kanchan M. Dhote. “Feature Extraction Techniques for Deep Learning based Speech Classification”. In: *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 2023, pp. 1–6. DOI: 10.1109/ICCCNT56998.2023.10307237.

[27] Hyejin Won et al. “Using various pre-trained models for audio feature extraction in automated audio captioning”. In: *Expert Systems with Applications* 231 (2023), p. 120664. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.120664. URL: https://www.sciencedirect.com/science/article/pii/S0957417423011661.

[28] Bhavi Dave and Kriti Srivastava. “Convolutional Neural Networks for Audio Classification: An Ensemble Approach”. In: *Proceedings of the 6th International Conference on Advance Computing and Intelligent Engineering*. Ed. by Bibudhendu Pati et al. Singapore: Springer Nature Singapore, 2023, pp. 253–262. ISBN: 978-981-19-2225-1.

[29] Kele Xu et al. “General audio tagging with ensembling convolutional neural networks and statistical features”. In: *The Journal of the Acoustical Society of America* 145.6 (June 2019), EL521–EL527. ISSN: 0001-4966. DOI: 10.1121/1.5111059. eprint: https://pubs.aip.org/asa/jasa/article-pdf/145/6/EL521/14075439/el521_1_online.pdf. URL: https://doi.org/10.1121/1.5111059.

[30] Xinhao Mei et al. “Audio Captioning Transformer”. In: *Detection and Classification of Acoustic Scenes and Events 2021*. 2021.

[31] Michele Scarpiniti et al. “Deep Recurrent Neural Networks for Audio Classification in Construction Sites”. In: *2020 28th European Signal Processing Conference (EUSIPCO)*. 2021, pp. 810–814. DOI: 10.23919/Eusipco47968.2020.9287802.

[32] Pablo Gimeno et al. “Multiclass audio segmentation based on recurrent neural networks for broadcast domain data”. In: *EURASIP Journal on Audio, Speech, and Music Processing* 2020.1 (2020), pp. 1–19.

[33] Chieh-Chi Kao et al. “R-CRNN: Region-based Convolutional Recurrent Neural Network for Audio Event Detection”. In: *arXiv preprint arXiv:1808.06627* (2018).

[34] Kun Fang et al. “A Fall Detection using Sound Technology Based on TinyML”. In: *2021 11th International Conference on Information Technology in Medicine and Education (ITME)*. 2021, pp. 222–225. DOI: 10.1109/ITME53901.2021.00053.

[35] H.R. Sabbella et al. “An Always-On tinyML Acoustic Classifier for Ecological Applications”. In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2022, pp. 2393–2396. DOI: 10.1109/ISCAS48785.2022.9937827.

[36] Giacomo Peruzzi, Alessandra Galli, and Alessandro Pozzebon. “A Novel Methodology to Remotely and Early Diagnose Sleep Bruxism by Leveraging on Audio Signals and Embedded Machine Learning”. In: *2022 IEEE International Symposium on Measurements Networking (MN)*. IEEE. 2022, pp. 1–6.

[37] https://www.cadence.com/en_US/home/tools/silicon-solutions/compute-ip/hifi-dsps/hifi-5.html.

[38] NXP. *One Chip Solution: Scalable Audio DSP Processing with AI/ML Capability*. https://www.nxp.com/products/audio-and-radio/audio-processors/one-chip-solution-scalable-audio-dsp-processing-with-ai-ml-capability:SAF9100. Accessed: [Insert the date of access here].

[39] Abraham Savitzky and M. J. E. Golay. “Smoothing and Differentiation of Data by Simplified Least Squares Procedures”. In: *Analytical Chemistry* 36.8 (1964), pp. 1627–1639. DOI: 10.1021/ac60214a047. URL: https://doi.org/10.1021/ac60214a047.

[40] Andrea Mattia Garavagno et al. “Running hardware-aware neural architecture search on embedded devices under 512MB of RAM”. In: *2024 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2024, pp. 1–2.

A

Source Code

A.1. Excerpt of CNN Hyperparameters

loss	precision	recall	accuracy	val_loss	val_precision	val_recall	val_accuracy	trial_id	score
0.951807	0.888405	0.887536	0.888562	0.261431	0.866748	0.775994	0.973854	58	0.973854
0.429042	0.905576	0.902797	0.90479	0.155321	0.858078	0.765414	0.972492	59	0.972492
0.246793	0.921987	0.915073	0.919211	0.134355	0.882537	0.720905	0.971437	13	0.971437
0.229661	0.922212	0.895297	0.910322	0.096505	0.808604	0.809194	0.970881	30	0.970881
0.228701	0.92096	0.912338	0.917417	0.10715	0.88706	0.707771	0.970881	46	0.970881
0.2504	0.910896	0.885661	0.899995	0.092204	0.822248	0.78475	0.970686	11	0.970686
0.226362	0.917684	0.906057	0.912812	0.095702	0.865017	0.727107	0.970575	14	0.970575
0.272813	0.920865	0.916852	0.919419	0.124499	0.784553	0.844947	0.97052	45	0.97052
4.234832	0.877371	0.877099	0.877843	1.189941	0.809138	0.801167	0.970464	41	0.970464
0.315829	0.914306	0.910714	0.913097	0.126469	0.811314	0.79533	0.970325	53	0.970325
0.916987	0.882674	0.881314	0.88265	0.260989	0.797537	0.803356	0.969492	37	0.969492
2.162869	0.859707	0.859819	0.860426	0.54855	0.865264	0.705217	0.969186	19	0.969186
0.197533	0.933105	0.91346	0.924351	0.112304	0.769564	0.843123	0.968825	25	0.968825
0.171743	0.94209	0.923336	0.933609	0.113123	0.807942	0.771981	0.968658	2	0.968658
0.201192	0.933127	0.911621	0.923513	0.113078	0.830731	0.737687	0.968575	22	0.968575
0.882877	0.846505	0.844069	0.846248	0.173136	0.818037	0.751186	0.968325	55	0.968325
3.939658	0.89193	0.89193	0.892448	1.368258	0.80747	0.765049	0.968213	32	0.968213
0.217572	0.922503	0.904206	0.914535	0.102334	0.753346	0.862459	0.968019	0	0.968019
0.169965	0.943622	0.92459	0.934988	0.122661	0.828216	0.728201	0.967797	23	0.967797
4.789254	0.857168	0.857383	0.857943	1.338289	0.806742	0.750821	0.967324	15	0.967324
0.265719	0.911133	0.89021	0.902164	0.116246	0.77292	0.799708	0.966852	33	0.966852
0.254221	0.907562	0.886114	0.89842	0.112306	0.754987	0.82853	0.966463	42	0.966463
0.17364	0.939672	0.921628	0.931559	0.120083	0.778955	0.777818	0.966268	28	0.966268
0.219897	0.923204	0.899656	0.91283	0.116667	0.822052	0.704487	0.965879	36	0.965879
0.223927	0.927876	0.914081	0.921891	0.129539	0.767064	0.7749	0.964935	8	0.964935
0.265922	0.902077	0.878066	0.891896	0.124568	0.711814	0.870485	0.963295	16	0.963295
0.311199	0.883102	0.851413	0.869981	0.128433	0.720697	0.814301	0.961823	29	0.961823
0.24559	0.913857	0.887786	0.90252	0.130644	0.703983	0.851149	0.961406	17	0.961406
0.189622	0.936784	0.923372	0.930864	0.1369	0.723588	0.7946	0.961239	12	0.961239
0.424711	0.865527	0.851998	0.860486	0.134141	0.751121	0.733309	0.961184	56	0.961184
0.231541	0.923351	0.901662	0.913822	0.132899	0.709687	0.82853	0.961128	4	0.961128
0.195552	0.934501	0.91064	0.922891	0.147335	0.691211	0.835618	0.961017	9	0.961017
0.18008	0.941987	0.924558	0.933396	0.125613	0.737566	0.756831	0.960435	1	0.960435
0.198906	0.933346	0.914545	0.924523	0.142131	0.678198	0.877697	0.959866	3	0.959866
0.263022	0.902699	0.883383	0.892796	0.159212	0.686625	0.817761	0.959866	34	0.959866
0.173877	0.939356	0.922301	0.932374	0.128244	0.714674	0.789317	0.959866	57	0.959866
0.267368	0.907384	0.878167	0.892379	0.165396	0.654515	0.831775	0.959756	5	0.959756
0.260633	0.909098	0.890076	0.901419	0.1766	0.66635	0.809908	0.959134	6	0.959134
0.300738	0.897876	0.863184	0.881616	0.186043	0.657722	0.805029	0.959079	24	0.959079
0.256927	0.899326	0.870116	0.88461	0.156426	0.718924	0.732319	0.958744	39	0.958744
0.221079	0.920675	0.894435	0.909444	0.151241	0.736931	0.676301	0.958321	10	0.958321
0.195292	0.93456	0.91258	0.923106	0.184485	0.675114	0.768889	0.957815	7	0.957815
0.346262	0.876019	0.854426	0.865435	0.211005	0.61559	0.778038	0.957682	20	0.957682
0.240675	0.909413	0.890867	0.899412	0.179789	0.672164	0.724306	0.957293	21	0.957293
0.258199	0.906889	0.882056	0.894252	0.175832	0.661491	0.764618	0.957293	35	0.957293
0.345318	0.876024	0.850204	0.864242	0.222736	0.589628	0.815578	0.956986	38	0.956986
0.239486	0.915588	0.896179	0.905314	0.222764	0.605284	0.724664	0.956719	40	0.956719
0.251669	0.914382	0.884674	0.90075	0.203641	0.598104	0.719222	0.95652	43	0.95652
0.227187	0.917664	0.906703	0.912455	0.237379	0.560015	0.802497	0.956155	44	0.956155
0.275167	0.903916	0.879234	0.892641	0.225214	0.587654	0.715842	0.956066	47	0.956066

Table A.1: Performance Metrics for Trials

A.2. Excerpt of FCNN Hyperparameter tuning

trial_id	learning_rate	optimizer	activation	filters_1	layers	loss	precision	recall	accuracy	val_loss	val_precision	val_recall	val_accuracy	best_step
3	1.00E-05	sgd	relu	224	4	0.1804	0.9466	0.9152	0.932	0.0848	0.8397	0.8453	0.9759	12
0	1.00E-05	rmsprop	relu	224	4	0.2587	0.9096	0.9122	0.9116	0.1235	0.8832	0.8496	0.9749	11
9	7.00E-05	adam	relu	224	4	0.1657	0.9501	0.9228	0.9375	0.0943	0.8068	0.8599	0.9737	10
11	1.00E-05	adam	relu	224	4	0.271	0.9094	0.8685	0.8914	0.0859	0.8956	0.7328	0.9732	3
28	0.0001	adam	relu	224	4	0.2066	0.9374	0.8991	0.918	0.1008	0.8273	0.8146	0.9729	5
14	0.0001	adam	relu	224	4	0.1948	0.9382	0.9063	0.9237	0.1005	0.7891	0.8687	0.9723	6
8	0.001	adam	relu	224	4	0.2059	0.9365	0.9031	0.9212	0.1068	0.7778	0.884	0.9719	3
2	0.001	adam	relu	224	4	0.1672	0.9506	0.9221	0.9373	0.1081	0.7723	0.8861	0.9714	7
15	1.00E-05	adam	relu	224	4	0.2762	0.9091	0.8644	0.8895	0.0926	0.8998	0.7016	0.9713	8
12	0.001	sgd	relu	224	4	0.2845	0.9086	0.8525	0.8839	0.1204	0.8308	0.7701	0.9705	8
30	0.0001	rmsprop	relu	224	4	0.1682	0.9465	0.9229	0.9357	0.1013	0.7709	0.8693	0.9704	7
26	0.001	rmsprop	relu	224	4	0.1989	0.9372	0.8993	0.9167	0.1184	0.7951	0.8394	0.9703	5
32	0.001	rmsprop	relu	224	4	0.2061	0.9371	0.8995	0.9182	0.1184	0.7952	0.8393	0.9702	6
27	0.001	sgd	relu	224	4	0.2115	0.9357	0.8967	0.9177	0.1193	0.7962	0.8389	0.97	6
19	7.00E-05	adam	relu	224	4	0.3088	0.8954	0.8483	0.8746	0.1128	0.7857	0.7839	0.9671	4
13	0.0001	adam	relu	224	4	0.3589	0.8938	0.8821	0.8892	0.1364	0.7933	0.7307	0.965	9
6	0.0001	adam	relu	224	4	0.2454	0.9204	0.8824	0.9035	0.1259	0.7259	0.8365	0.9635	11
18	0.0001	rmsprop	relu	224	4	0.3217	0.8903	0.8405	0.8691	0.1362	0.7337	0.8044	0.9629	5
5	0.001	sgd	relu	224	4	0.3827	0.8577	0.8037	0.8359	0.1448	0.7408	0.7839	0.9627	5
16	7.00E-05	adam	relu	224	4	0.197	0.9382	0.9089	0.9249	0.1307	0.7257	0.8204	0.9627	2
17	0.0001	adam	relu	224	4	0.2289	0.9242	0.8929	0.9102	0.1382	0.6992	0.8839	0.9622	3
7	0.001	adam	relu	224	4	0.2837	0.9067	0.857	0.8849	0.1652	0.6764	0.8423	0.9573	6
1	0.0001	rmsprop	relu	224	4	0.2636	0.9163	0.8705	0.896	0.1745	0.634	0.8825	0.9523	7
25	7.00E-05	rmsprop	relu	224	4	0.2604	0.9238	0.8622	0.8926	0.1483	0.6321	0.883	0.9523	4
22	1.00E-05	rmsprop	relu	224	4	0.2644	0.9235	0.8633	0.8931	0.1486	0.6323	0.8832	0.9521	4
24	0.001	sgd	relu	224	4	0.2363	0.9289	0.8774	0.8991	0.1734	0.6421	0.882	0.9514	7
23	7.00E-05	adam	relu	224	4	0.2687	0.9211	0.8612	0.8926	0.1747	0.6422	0.8811	0.9512	8
31	0.0001	adam	relu	224	4	0.2652	0.9253	0.8678	0.8975	0.1485	0.6237	0.882	0.9508	4
20	1.00E-05	adam	relu	224	4	0.2333	0.9301	0.8891	0.9059	0.1493	0.662	0.859	0.9473	6
21	0.0001	adam	relu	224	4	0.2488	0.9289	0.8705	0.8971	0.1784	0.6215	0.8818	0.9458	7
29	7.00E-05	adam	relu	224	4	0.2491	0.9294	0.8705	0.8969	0.1783	0.6217	0.8819	0.9456	8
10	1.00E-05	adam	relu	224	4	0.3037	0.8961	0.8832	0.8909	0.2014	0.5531	0.8147	0.9358	7
4	0.0001	rmsprop	relu	224	4	0.8037	0.5093	0.9229	0.5189	0.675	0.7309	0.0139	0.9245	20
38	0.001	sgd	leaky_relu	168	6	0.1859	0.9437	0.9123	0.9293	0.1184	0.7519	0.8649	0.968	5
39	1.00E-05	adam	leaky_relu	176	1	0.5541	0.8611	0.8559	0.8596	0.587	0.3161	0.8423	0.8492	11
40	0.001	adam	relu	40	7	0.2006	0.9394	0.9054	0.9239	0.1185	0.7957	0.8328	0.971	5
41	0.0001	sgd	tanh	96	2	0.3964	0.8589	0.7931	0.8322	0.3068	0.437	0.8387	0.9055	10
42	7.00E-05	rmsprop	tanh	168	3	0.263	0.9137	0.8721	0.8954	0.1081	0.8149	0.8161	0.9719	3
43	7.00E-05	adam	leaky_relu	240	9	0.2052	0.937	0.9028	0.9214	0.0903	0.8089	0.8869	0.9754	1
44	0.001	rmsprop	leaky_relu	88	8	0.235	0.9239	0.8945	0.9108	0.1204	0.746	0.8102	0.9645	4
45	0.001	sgd	leaky_relu	8	9	0.4247	0.842	0.7694	0.8134	0.2654	0.5249	0.8153	0.9298	4
46	7.00E-05	adam	leaky_relu	248	1	0.1941	0.9345	0.9235	0.9297	0.2095	0.5518	0.8394	0.9359	8
47	0.001	sgd	relu	184	8	0.2106	0.9347	0.8981	0.918	0.1008	0.8273	0.8146	0.9729	5
48	7.00E-05	sgd	relu	184	3	0.2587	0.9096	0.8827	0.898	0.1398	0.6686	0.7759	0.9537	8
49	0.0001	rmsprop	tanh	216	9	0.1682	0.9465	0.9229	0.9357	0.1013	0.7709	0.8693	0.9704	7
50	0.001	sgd	leaky_relu	48	5	0.2471	0.9195	0.8804	0.9021	0.1374	0.7077	0.8555	0.9621	6

Table A.2: FCNN architectures hyperparameter tuning

A.3. Excerpt of LSTM hyperparameter tuning

val_loss	val_precision	val_recall	val_accuracy	trial_id	score	best_step	learning_rate	optimizer	lstm_units
0.051	0.905	0.906	0.986	00013	0.962	9	0.0001	rmsprop	120, 120
0.057	0.914	0.870	0.984	00333	0.914	8	0.0001	rmsprop	120, 64
0.058	0.903	0.879	0.984	00431	0.903	10	0.0001	rmsprop	112, 40
0.059	0.925	0.830	0.982	00100	0.925	6	0.001	rmsprop	64, 44
0.062	0.894	0.882	0.983	00091	0.894	8	0.0001	rmsprop	128, 48
0.063	0.911	0.867	0.983	00424	0.911	4	0.0001	rmsprop	88, 88
0.063	0.915	0.868	0.984	00280	0.915	3	0.001	rmsprop	120, 72
0.064	0.956	0.815	0.983	00112	0.956	7	0.0001	rmsprop	112, 80
0.064	0.892	0.878	0.983	00208	0.892	7	0.00007	rmsprop	72, 120
0.064	0.916	0.845	0.982	00332	0.916	7	0.00007	rmsprop	88, 76
0.064	0.923	0.843	0.983	00432	0.923	3	0.00007	rmsprop	104, 120
0.064	0.935	0.816	0.982	00413	0.935	5	0.0001	rmsprop	88, 124
0.064	0.904	0.858	0.982	00278	0.904	3	0.001	rmsprop	88, 52
0.064	0.876	0.911	0.983	00312	0.876	9	0.0001	adam	80, 80
0.065	0.887	0.885	0.983	00209	0.887	10	0.00007	rmsprop	88, 60
0.065	0.909	0.871	0.984	00289	0.909	6	0.00007	rmsprop	96, 104
0.065	0.869	0.899	0.982	00394	0.869	7	0.0001	adam	128, 80
0.065	0.846	0.907	0.980	00027	0.846	11	0.0001	rmsprop	40, 116
0.066	0.899	0.887	0.984	00068	0.899	6	0.0001	rmsprop	120, 88
0.066	0.884	0.871	0.981	00002	0.884	5	0.001	adam	80, 88
0.066	0.893	0.874	0.982	00484	0.893	9	0.001	rmsprop	88, 36
0.066	0.876	0.900	0.983	00472	0.876	7	0.001	rmsprop	80, 32
0.067	0.893	0.873	0.982	00343	0.893	4	0.0001	rmsprop	96, 96
0.067	0.876	0.875	0.981	00235	0.876	7	0.00007	rmsprop	104, 128
0.067	0.865	0.887	0.981	00030	0.865	6	0.001	adam	64, 128
0.067	0.868	0.894	0.981	00166	0.868	5	0.0001	rmsprop	72, 108
0.069	0.874	0.883	0.981	00342	0.874	11	0.0001	rmsprop	48, 36
0.069	0.874	0.894	0.982	00102	0.874	4	0.0001	adam	96, 80
0.069	0.881	0.872	0.981	00340	0.881	3	0.001	rmsprop	120, 64
0.069	0.834	0.933	0.981	00109	0.834	14	0.001	adam	40, 116

Table A.3: Excerpt of best 75, for LSTM table

A.4. Architectures of the final models compared in Chapter 5 Section 3

Layer (type)	**Output Shape**	**Param #**
reshape (Reshape)	(None, 99, 80, 1)	0
depthwise_conv2d (DepthwiseConv2D)	(None, 99, 80, 224)	2240
max_pooling2d (MaxPooling2D)	(None, 49, 40, 224)	0
dropout (Dropout)	(None, 49, 40, 224)	0
flatten (Flatten)	(None, 439040)	0
dense (Dense)	(None, 1)	439041
Total params:	**441281 (1.68 MB)**	
Trainable params:	**441281 (1.68 MB)**	
Non-trainable params:	**0 (0.00 Byte)**	

Table A.4: Model Summary of 1 layer CNN

Layer (type)	**Output Shape**	**Param #**
reshape (Reshape)	(None, 99, 80, 1)	0
conv2d (Conv2D)	(None, 97, 78, 224)	2240
max_pooling2d (MaxPooling2D)	(None, 48, 39, 224)	0
conv2d_1 (Conv2D)	(None, 46, 37, 224)	451808
max_pooling2d_1 (MaxPooling2D)	(None, 23, 18, 224)	0
conv2d_2 (Conv2D)	(None, 21, 16, 224)	451808
max_pooling2d_2 (MaxPooling2D)	(None, 10, 8, 224)	0
flatten (Flatten)	(None, 17920)	0
dense (Dense)	(None, 128)	2293888
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129
Total params:	**3199873 (12.21 MB)**	
Trainable params:	**3199873 (12.21 MB)**	
Non-trainable params:	**0 (0.00 Byte)**	

Table A.5: Model Summary of 3 layer CNN

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 99, 224)	18144
dense_1 (Dense)	(None, 99, 224)	50400
dense_2 (Dense)	(None, 99, 224)	50400
dense_3 (Dense)	(None, 99, 224)	50400
flatten (Flatten)	(None, 22176)	0
dense_4 (Dense)	(None, 1)	22177
Total params:	**191521 (748.13 KB)**	
Trainable params:	**191521 (748.13 KB)**	
Non-trainable params:	**0 (0.00 Byte)**	

Table A.6: Model Summary of 4 layer FCNN

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 99, 224)	18144
dense_1 (Dense)	(None, 99, 224)	50400
dense_2 (Dense)	(None, 99, 224)	50400
dense_3 (Dense)	(None, 99, 224)	50400
dense_4 (Dense)	(None, 99, 224)	50400
dense_5 (Dense)	(None, 99, 224)	50400
dense_6 (Dense)	(None, 99, 224)	50400
dense_7 (Dense)	(None, 99, 224)	50400
dense_8 (Dense)	(None, 99, 224)	50400
dense_9 (Dense)	(None, 99, 224)	50400
flatten (Flatten)	(None, 22176)	0
dense_10 (Dense)	(None, 1)	22177
Total params:	**493921 (1.88 MB)**	
Trainable params:	**493921 (1.88 MB)**	
Non-trainable params:	**0 (0.00 Byte)**	

Table A.7: Model Summary of 9 layer FCNN

Layer (type)	**Output Shape**	**Param #**
input_1 (InputLayer)	[(None, None, 80)]	0
lstm (LSTM)	(None, None, 120)	96480
dropout (Dropout)	(None, None, 120)	0
lstm_1 (LSTM)	(None, None, 120)	115680
lstm_2 (LSTM)	(None, 32)	19584
output (Dense)	(None, 1)	33
Total params:	**231777 (905.38 KB)**	
Trainable params:	**231777 (905.38 KB)**	
Non-trainable params:	**0 (0.00 Byte)**	

Table A.8: Model Summary of Finalised LSTM architecture