IoT

MDPI

Article

A Deep Learning-Based Gunshot Detection IoT System with Enhanced Security Features and Testing Using Blank Guns

Tareq Khan

School of Engineering, Eastern Michigan University, Ypsilanti, MI 48197, USA; tareq.khan@emich.edu

Abstract: Although the U.S. makes up only 5% of the global population, it accounts for approximately 31% of public mass shootings. Gun violence and mass shootings not only result in loss of life and injury but also inflict lasting psychological trauma, cause property damage, and lead to significant economic losses. We recently developed and published an embedded system prototype for detecting gunshots in an indoor environment. The proposed device can be attached to the walls or ceilings of schools, offices, clubs, places of worship, etc., similar to smoke detectors or night lights, and they can notify the first responders as soon as a gunshot is fired. The proposed system will help to stop the shooter early and the injured people can be taken to the hospital quickly, thus more lives can be saved. In this project, a new custom dataset of blank gunshot sounds is recorded, and a deep learning model using both time and frequency domain features is trained to classify gunshot and non-gunshot sounds with 99% accuracy. The previously developed system suffered from several security and privacy vulnerabilities. In this research, those vulnerabilities are addressed by implementing secure Message Queuing Telemetry Transport (MQTT) communication protocols for IoT systems, better authentication methods, Wi-Fi provisioning without Bluetooth, and over-the-air (OTA) firmware update features. The prototype is implemented in a Raspberry Pi Zero 2W embedded system platform and successfully tested with blank gunshots and possible false alarms.

Keywords: blank guns; custom dataset; deep learning; frequency domain features; message queuing telemetry transport (MQTT); over-the-air (OTA) update; raspberry pi zero 2W; security; time domain features; wi-fi provisioning

Academic Editor: Amiya Nayak

Received: 25 November 2024
Revised: 30 December 2024
Accepted: 2 January 2025
Published: 3 January 2025

Citation: Khan, T. A Deep Learning-Based Gunshot Detection IoT System with Enhanced Security Features and Testing Using Blank Guns. *IoT* **2025**, *6*, 5. https://doi.org/10.3390/iot6010005

1. Introduction

In the USA in 2023, the total number of deaths due to gun violence was 18,854—among them, 1682 were children between the ages of 0 to 17 and the total injuries were 36,338 [1]. One way to reduce the loss from gun violence is to detect the incident early and notify the police as soon as possible. In this project, a novel gunshot detection IoT system is developed that automatically detects the indoor gunshot sound and then sends notifications to the first responders' smartphones as soon as the shooting happens—within a second. The proposed device can be attached to the walls or ceilings, similar to night lights or smoke detectors. The overall IoT system is shown in Figure 1. In this system, the gunshot detector device is connected to the Internet using local Wi-Fi. The devices and the smartphones of emergency responders are connected to the Message Queuing Telemetry Transport (MQTT) broker server as clients. The device detects gunshot sounds and sends notifications with location information to other client smartphones—those of security guards and the police—using secured MQTT protocol. The proposed system will reduce the duration between a gunshot

and the police arrival, and thus reduce the response time. Some significance of reducing the response time is mentioned below [2,3]:

- In the chaos of an active shooter event, it often takes bystanders an average of 5 min to call the police due to panic and confusion. This delay can prevent first responders from receiving accurate information in time to act effectively. Shortening this notification time ensures faster intervention, providing law enforcement with the critical details they need to mitigate threats and save lives.
- During an active shooter incident, the window for providing life-saving medical attention is narrow, with survival rates dropping by 7–10% for every minute without treatment. Unfortunately, with an average 5-min delay before 911 is contacted, critical minutes are often lost. Reducing response time can significantly improve survival chances, especially for victims with severe injuries, by ensuring that medical aid reaches them in time.
- Delays in emergency response not only affect survival but also increase medical costs. For each minute that passes without intervention, healthcare costs can rise by about 7%, as victims may require more intensive and prolonged care. Faster response times lower these costs by reducing the severity of injuries, leading to quicker recoveries and a better quality of life for survivors.

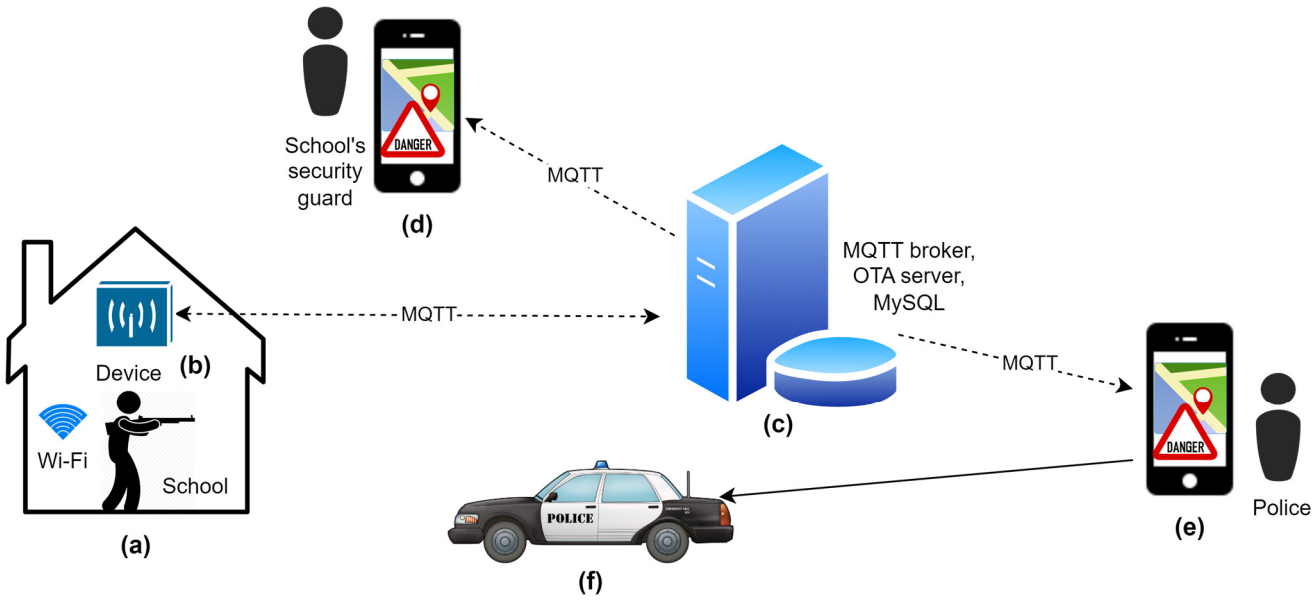


Figure 1. Crime scene (**a**) where the shooting happened. The gunshot detector device (**b**) is connected to the local Wi-Fi and the MQTT broker server (c) as a client. It detects gunshot sounds and sends notifications to smartphones of all MQTT clients such as the security guard (**d**) and police (**e**) using secured MQTT protocol. The police car (**f**) is dispatched.

The remainder of the paper is structured as follows: Section 2 reviews related work and provides a comparison with existing approaches. Section 3 outlines the materials and methods, including the custom dataset, deep learning model for gunshot detection, and the development of a secure prototype system. Section 4 presents the results of deep learning model training, validation, and testing, along with the performance evaluation of the prototype system. Section 5 covers the discussion and potential directions for future work. Lastly, Section 6 concludes the paper.

2. Related Works

The related works as commercial products and published literature are discussed below.

2.1. Commercial Products

There are several commercial products available on the market for gunshot detection and a comparison of those works with our work is shown in Table 1. ZeroEyes [4] provide software solution that integrates with existing security cameras to monitor live video feeds and detect firearms. The system analyzes over 36,000 images per second using an AI-based image classification system trained to identify guns in real-time. If a suspected firearm is detected, the image is sent to the ZeroEyes Operation Center (ZOC), where trained specialists review and confirm the presence of a weapon. Visual coverage is a significant limitation of systems like ZeroEyes compared to sound-based detection systems. Visual detection relies on security cameras having a clear line of sight to the firearm, which means its effectiveness is restricted by the camera placement, angle, lighting conditions, and potential obstructions in the environment. This approach is further slowed by the need for human confirmation of detected firearms before alerts are sent. The system was tested in real-time [5] and it took 25–30 s to receive a smartphone notification after the firearm was visible to the camera. Though they claim this technology can prevent shooting as this detection is pre-gunshot, its practical effectiveness in stopping an actual shooting is limited as firearms are often used within seconds of being drawn from a bag or pocket. It may take several minutes for the police to arrive after the notification is sent, meanwhile, the harm from the first gunshot is already done. Moreover, as camera images are sent to servers, it is a privacy concern because people can be monitored by the company all the time.

Table 1. Comparison with products on the market.

	ShotSpotter	ZeroEyes	AmberBox	This Work
Indoor/Outdoor	Outdoor	Indoor	Indoor	Indoor
Detection Method	Sound	Picture	Sound	Sound
Human in the loop	Yes	Yes	No	No
Response Time (s)	60	30	3.6	0.125

In contrast, sound travels in all directions and can be detected even if the source is not within any camera's field of view and in any lighting condition. A gunshot sound can be heard and detected without a line of sight and from corners, making sound-based systems more effective in different environments. This robust coverage gives audio detection systems an advantage in ensuring the gunshot is identified regardless of the shooter's location or the visual setup of the space. AmberBox [6] is an indoor gunshot detection system that identifies and tracks gunshots within 3.6 s by analyzing two distinct audio signatures: the muzzle blast and bullet shockwave. In addition to audio detection, it uses infrared sensors to detect the muzzle flash caused by gunpowder combustion. AmberBox utilizes machine learning to compare detected sounds against thousands of stored gunshot samples on each detector, enabling it to determine if a gunshot has occurred without needing a line of sight. Given its reliance on stored samples rather than dynamic real-time learning, AmberBox could be using K-Nearest Neighbors (KNN) or a similar algorithm that relies on stored examples to make real-time comparisons, rather than learning abstract patterns like state-of-the-art deep learning models do. At inference time, KNN compares the new input to every sample in the stored dataset to find the nearest neighbors, which can become computationally expensive and slow, especially if thousands of samples are stored and need to be compared.

Some cities deploy solutions from companies like SoundThinking (formerly ShotSpotter) [7] to detect and localize gunshots on a large scale. The work in [8] reports on the effect of using this system in high-crime areas. These systems rely on sensor modules installed

in outdoor areas across the city and are not designed for indoor incidents. Additionally, they are extremely costly to operate and maintain, requiring both automated processes and manual human analysis. The annual cost of this system can reach up to USD 90,000 per square mile of coverage. It is implemented by city authorities rather than individuals or private institutions, limiting its accessibility for personal use. To benefit from this system, one must reside in a city where it is deployed, which can be inconvenient.

Our proposed system is not designed for outdoor gunshot detection, rather it is mainly targeted at detecting indoor gunshot sounds such as inside schools, banks, shopping malls, stores, houses of worship, etc. For instance, there are 130 k K-12 schools, 75 k bank branches, and 384 k houses of worship in the USA where the system can be used. Unlike the work of [6], the proposed gunshot detector only stores the deep learning model in the device without storing thousands of sound samples, thus the inferencing time is much faster (less than a second). The proposed system can help to reduce the time between the first gunshot and contacting the emergency responders. Every second matters in such a gun violence situation. Mortality and health care costs may increase by 7–10% for each minute delay [2,3]. The proposed system will notify the police with exact map information within a second. Thus, police can arrive at the crime scene and catch the shooter early, injured people can be taken to the hospital quickly, and more lives can be saved. Unlike smart speakers such as Amazon Alexa or Google Home, the proposed gunshot detector processes and classifies sound directly on the device, without transmitting the audio to an external server. This ensures that users have no privacy concerns about being monitored without their consent.

2.2. Published Literature

A comparison of the proposed work with other works found in the literature is shown in Table 2. The authors in [9] propose a semi-supervised Non-negative Matrix Factorization (NMF) approach for gunshot detection, consisting of training and separation stages. Their results show a maximum true-positive (TP) rate of 50% when the signal-to-noise ratio (SNR) is 5 dB. However, no hardware implementation results are provided. In [10], the authors implement two parallel Gaussian Mixture Model (GMM) classifiers to discriminate between screams and noise, and gunshots and noise, using different audio features to train the classifiers. They report a precision of 93% for event detection at an SNR of 10 dB, though embedded system implementation and notification systems are not included. In [11], convolutional neural networks (CNNs), including VGG16, InceptionV3, and ResNet18, are trained with transfer learning for gunshot detection using Mel Frequency Cepstral Coefficients (MFCCs) from audio signals. ResNet18 achieves over 99% accuracy on their dataset, but hardware implementation and notification systems are not addressed. The work in [12] employs a custom CNN model for gunshot classification, using spectrograms generated from audio signals. The model achieves over 99% accuracy on their custom dataset and is implemented on a hardware system consisting of a USB microphone, Raspberry Pi, and SMS modem. When a gunshot is detected, the system sends an SMS alert to a predetermined list of phone numbers. The work lacks a map of the gunshot location, device control and status monitoring from smartphones, OTA updates, and security features. Moreover, SMS involves a per-message fee, which can make it costly.

In our previously published work [13], gunshot sounds are downloaded from online sources and a deep learning-based gunshot sound classifier is trained. A prototype of the gunshot detector device is implemented in the Jetson Nano embedded system platform. The central server with a database and smartphone apps for users and emergency responders has also been developed. The prototype was tested by playing gunshot sounds from a speaker.

Compared to these related works, this project involves recording a new custom dataset of blank gunshot sounds and training a deep learning model, using both time domain and frequency domain features, to differentiate between gunshot and non-gunshot sounds. The gunshot detection system is implemented consisting of servers, a detector device on a Raspberry Pi Zero 2W embedded system, and a smartphone app. The system is tested with blank gunshot sounds in real-time and also tested with potential false alarms such as fireworks, action movies, and balloon burst sounds.

Table 2. Comparison with the published literature.

	J. Morillas, et al. [9]	G. Valenzise, et al. [10]	J. Bajzik, et al. [11]	A. Morehead, et al. [12]	T. Khan [13]	This Work
Classifier	NMF	GMM	CNN	CNN	CNN	CNN
Dataset size	215	-	7000	<90,000	670,000	155,000
Accuracy	-	-	99%	99%	98%	99%
Precision	-	93%	-	-	98%	100%
Embedded system implementation	no	no	no	yes	yes	yes
Record gunshot with timestamp	no	no	no	no	yes	yes
Smartphone notification	no	no	no	yes (using SMS)	yes	yes
Plot on map	no	no	no	no	yes	yes
Data transmission and access security	no	no	no	no	no	yes
OTA update	no	no	no	no	no	yes
Device control and connection status monitoring from a smartphone	no	no	no	no	no	yes
Realtime testing with guns	no	no	no	yes	no	yes

Our earlier version of the system [13] had various security and privacy issues, which are resolved in this work by integrating secure MQTT communication protocols, improved authentication mechanisms, Wi-Fi provisioning without requiring Bluetooth, and over-the-air (OTA) firmware updates. To highlight the improvement in this work, a comparison with our previous work is shown in Table 3.

Table 3. Comparison with our previous work.

	Our Previous Work [13]	This Work
Dataset	Collected from online sources	Custom recording
Features for deep learning	Only frequency domain (MFCC)	Frequency (MFCC) and time domain
Gunshot testing	Playing gunshot sounds from a speaker	Using blank guns in real-time
False alarm testing	Playing non-gunshot sounds from a speaker	Real-time experiment with balloon pop, fireworks, and playing movie gunshot sounds from TV.
Notification protocol	FCM	MQTT

Table 3. *Cont.*

	Our Previous Work [13]	This Work
The device receives commands from the user	No	Yes
Users get notified of the device connection status	No	Yes
Data transmission security	No	Yes. By authentication and encryption
Security of recorded gunshot sound access from the device	No	Yes. By authentication and encryption
Wi-Fi provisioning	Needs Bluetooth	Bluetooth not needed
OTA update	No	Yes
Implementation platform and price	Jetson Nano (USD 99)	Raspberry Pi Zero 2W (USD 15)
Microphone	USB microphone with sound card	Tiny MEMS microphone without the need for a sound card
Prototype size	15.5 × 12.3 × 4 cm	7.6 × 5.1 × 5.1 cm
Mounting	On table	Directly plugged into AC wall outlet

3. Materials and Methods

The deep learning model for gunshot detection and the prototype system development are discussed below.

3.1. Gunshot Sound Detection Using Deep Learning

The generation of the custom dataset, time and frequency domain feature extraction, the deep learning model architecture, and the training of the model are discussed below.

3.1.1. Custom Dataset Generation

To classify gunshot sounds from other sounds, two classes in the dataset are required: gunshot sound and non-gunshot (or others) sound. They are briefly described below:

- **Gunshot sound:** In our work in [13], gunshot and non-gunshot sounds were downloaded from online datasets such as BGG [14], Free Firearm Sound Effects Library [15], Urbansound8k [16], etc. These sounds were recorded with different types of microphones, which are different from the microphone used in the gunshot detector prototype. The amplitudes of some of these sounds were normalized and they lost the loudness or volume information compared to background sounds. Due to this reason, the performance of the model suffered during the live tests and created frequent false alarms. To solve this problem, a new custom dataset is needed.

A custom recording device is designed and developed for this project and its hardware block diagram is shown in Figure 2. A miniature MEMS microphone [17] is interfaced with a Teensy 4.1 Development Board [18] using I2S protocol. The Teensy board contains an ARM Cortex-M7 processor running at 600 MHz, 7936K Flash, 1024K RAM, 2 I2S digital audio ports, USB, and other ports. Its USB port can operate in device or peripheral mode at 480 Mbit/s speed. The software enables bi-directional stereo audio streaming, and the board is recognized by the computer as a USB sound card. Using the Audio System Design Tool for Teensy Audio Library [19], firmware for the Teensy was generated to send I2S data from the microphone to the USB of the computer. The NCH WavePad version 17.44 [20] audio processing software was used in the computer for recording. Recordings were done at a 44.1 kHz sampling rate having a resolution of 32 bits in the mono channel mode.

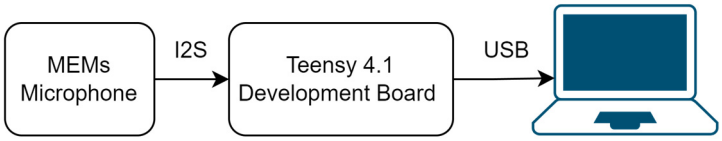


Figure 2. Block diagram of the custom recording device hardware.

In this project, the developed gunshot detector is trained and tested with blank gun sounds. Blanks or starter pistols generate exact gunshot sounds but do not throw any bullet. These guns are used for active shooter training and Hollywood movies. Thus, the gunshot sounds can be recorded and the detector can be tested safely in an indoor environment with blank guns. A Zoraki M906 [21] blank gun was used to develop the dataset. The recordings were done in Eastern Michigan University's Sill Hall building in 4 different locations: research lab, breakroom, building corridor, and faculty office hallway. All non-involved persons were evacuated and exterior doors to the building were locked. A Department of Public Safety (DPS) personnel was assigned for safety during the recording. The attendees wore safety eyeglasses, earplugs, and closed shoes. Three recording devices with laptops were placed at different places in each location. On each of the 4 locations, 25 shots were fired from random spots. In this way, $3 \times 4 \times 25 = 300$ gunshot sounds were recorded. Each of the gunshot sounds was then split into 1-s WAV files manually.

Data augmentation is a technique used to generate new training examples by making slight modifications to the existing data. This process enhances the model's ability to learn by providing it with more diverse examples, helping to improve its overall performance and robustness. Using NCH WavePad software, the following effects were added to augment the sound data: time shifting (this technique involves shifting the audio waveform in time, either forward or backward, without altering its content or duration); location effects such as auditorium, bathroom, and hanger; echo with different delays; different levels of phaser, equalizer, flanger, tremolo, vibrato, distortion, and chorus; and reverb for different types of building materials. More samples were created by overlaying non-gunshot sounds, as described below, with the gunshot sounds as gunshots may happen with the presence of background noise. After applying the data augmentations, a total of 77,667 WAV files were generated.

Other sounds: A custom dataset of non-gunshot sounds, defined as "other" sounds, was generated using the developed recording device. For this project, this dataset should contain the possible non-gunshot sounds that might be available in indoor environments such as in schools, offices, grocery stores, etc. Several crowd noises were recorded in the cafeteria, classrooms, and hallways. Moreover, a recording was done inside a lab where a group of volunteers sat together and made random sounds such as talking, laughing, clapping, screaming, playing ringtones, music, game sounds from smartphones, dropping objects, etc. Along with recorded sounds, some audio was downloaded from online sources such as 20–20,000 Hz audio sweep, a school bell tone, announcement, balloon pop, basketball bouncing, coffee shop ambiance, fire siren, footsteps, highway sounds, TV ambiance, rain, thunderstorm, party noise, power tools, vacuum, classroom music, radio ambiance, school cafeteria ambiance, school hallway ambiance between classes, sliding door opening and closing, smoke alarm, video game, whistles and horns, noisy classroom, kids screaming, etc. The amplitude of these downloaded sounds was reduced using WavePad software so that it approximately matches the amplitude levels of our custom recording device. All these sounds were split into 1 s WAV files and a total of 94,211 other or non-gunshot sound files were generated.

3.1.2. Time and Frequency Domain Feature Extraction

In our previous work in [13], only frequency domain features were used. However, we found that the loudness of the gunshot sound should be considered along with its frequency during classification, otherwise, the classifier produces false alarms when tested with low volume sound of gunshots that may come from TV or gaming devices. In this work, both time domain and frequency domain features are used as inputs to the classifier. They are briefly described below:

- **Time Domain Features:** The DC value from the one-second sound is first removed by subtracting its average from each sample. Each sample is also divided by the maximum 32-bit integer (i.e., $2^{31} - 1$) to normalize the samples so that the values range from -1 to 1. For each one-second sound, a feature vector of length 64 is generated for each time-domain feature. Among the 44,100 samples of the one-second sound, the first 44,096 samples were equally sliced to 64 windows, each window having $N = 689$ samples. For each of these 64 windows, 5 time-domain features: the average of absolute values ($F1$), maximum ($F2$), minimum ($F3$), standard deviation ($F4$), and differences between consecutive elements of the average vector ($F5$) are calculated. Let $x_i[j]$ represent the jth sample in the ith window, where $i \in \{1, 2 \ldots, 64\}$ and $j \in \{1, 2 \ldots, 689\}$. For each window i, the features are computed using (1)–(5). In (5), $F5_i$ is set to 0. A time domain plot of a gunshot sound is shown in Figure 3a, and the five calculated time domain features are shown in Figure 3b–f.

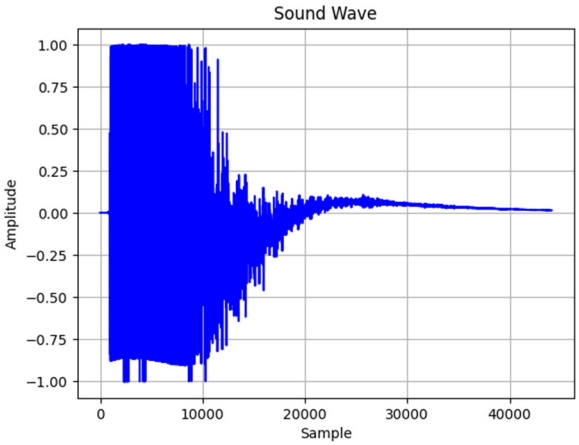
$$F1_i = \frac{1}{N}\sum_{j=1}^{N}|x_i[j]| \tag{1}$$

$$F2_i = \max_{j\in\{1,2,\ldots,N\}} x_i[j] \tag{2}$$

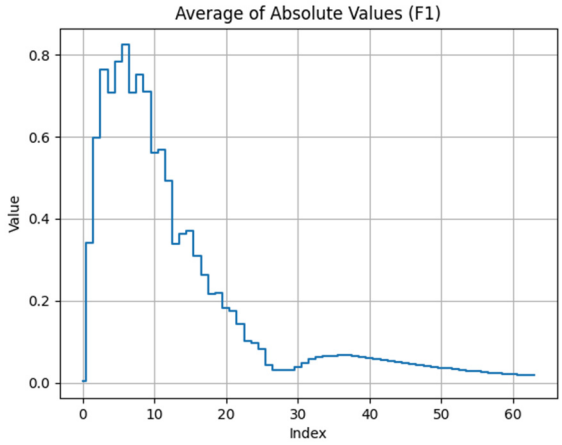
$$F3_i = \min_{j\in\{1,2,\ldots,N\}} x_i[j] \tag{3}$$

$$F4_i = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(x_i[j] - \mu_i)^2}, \text{ where } \mu_i = \frac{1}{N}\sum_{j=1}^{N} x_i[j] \tag{4}$$

$$F5_i = F1_i - F1_{i-1} \text{ when } i > 1 \tag{5}$$



(**a**)



(**b**)

Figure 3. *Cont.*

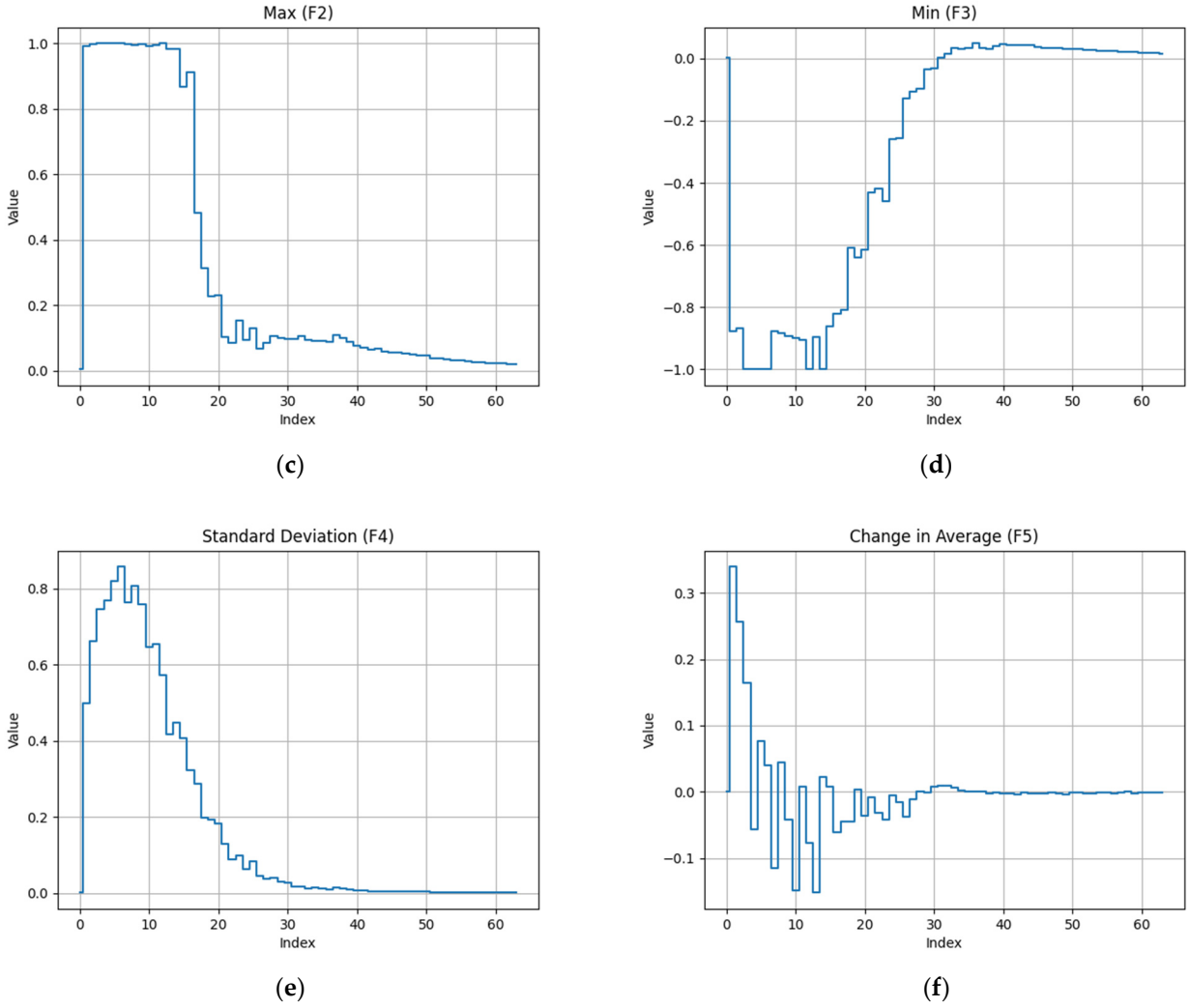


Figure 3. Time domain plot of a gunshot sound (**a**) and its calculated time domain features: average of absolute values (**b**), maximum (**c**), minimum (**d**), standard deviation (**e**), and differences between consecutive elements of the average vector (**f**).

- **Frequency Domain Features:** To extract frequency domain features, the DC value from the one-second sound is first removed by subtracting its average from each sample. Then Mel Frequency Cepstral Coefficients (MFCCs) [22,23] are generated by transforming the time-domain audio signal. The core idea behind MFCCs is to compress essential audio information into a compact set of coefficients, modeled after the auditory perception of the human ear. To calculate the MFCCs, the audio signal is first divided into windows of 20–40 milliseconds each. For every window, the power spectrum is computed by applying the Fast Fourier Transform (FFT). Then, triangular Mel filter banks are generated and applied to the power spectrum, resulting in a spectrogram. Since the human ear is more sensitive to small pitch variations at lower frequencies (below 1 kHz) than at higher ones, the first ten filters in the Mel filter bank are linearly spaced at frequencies of 100, 200, . . ., and up to 1000 Hz. Above 1 kHz, the filters follow the logarithmic Mel scale. After applying the filters, the logarithms of the filter bank energies are taken, and a discrete cosine transform (DCT) is applied to decorrelate the filter bank coefficients. The first DCT coefficient contains the average (or DC) value of the signal and it is not needed for classification. So, it is replaced with log energies of the signal. This process effectively captures the key characteristics of the sound, making it well suited for sound classification tasks.

In this project, the one-second sound is segmented into windows of 1024 samples (i.e., 23 milliseconds) each. Overlapping windows are implemented by calculating the stride (i.e., the offset for the next window in terms of samples) using (6) and (7) so that the total number of windows for the one-second sound matches with the feature length of the time domain features. To get TOTAL_WINDOW as 64, the OVERLAP_PERCENTAGE is set to 34, WINDOW as 1024, TOTAL_SAMPLES as 44,100, and STRIDE is calculated to be 676.

$$\text{STRIDE} = \text{round}\left(\text{WINDOW} \times \left(1 - \frac{\text{OVERLAP\_PERCENTAGE}}{100}\right)\right) \tag{6}$$

$$\text{TOTAL\_WINDOW} = \left\lfloor \frac{\text{TOTAL\_SAMPLES} - \text{WINDOW}}{\text{STRIDE}} \right\rfloor + 1 \tag{7}$$

A filter bank with 15 filters is applied, the FFT is computed with 1024 points, and the number of DCT coefficients is set to 13. This results in an MFCC representation consisting of 13 cepstral coefficients for each window. Then, the coefficients are normalized using max-min normalization. Here, the max and min values were set as constants of 50 and -50, respectively. They were calculated by finding the max and min of the MFCC coefficient values for the entire dataset. In Figure 4a, the MFCC coefficients are shown for the gunshot sound shown in Figure 3a.

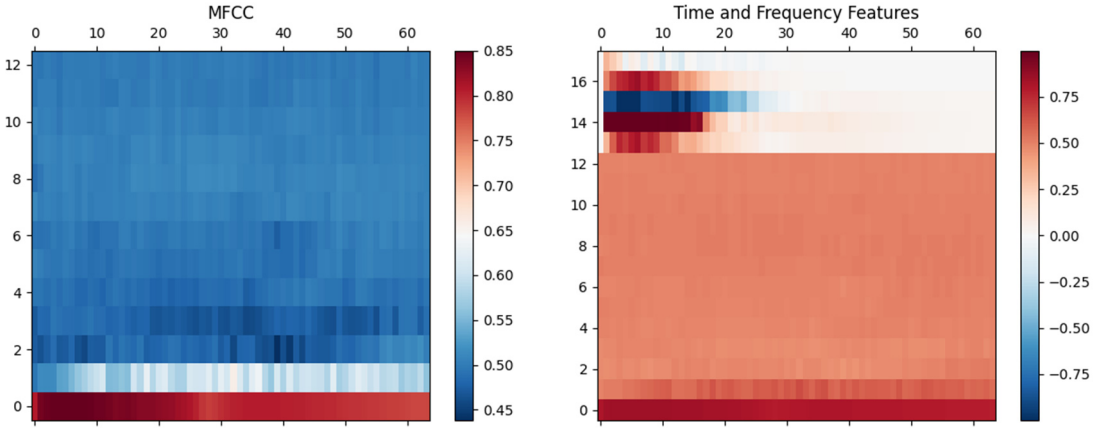


Figure 4. (**a**) MFCC coefficients of a gunshot sound as the frequency domain feature, (**b**) the frequency domain and the time domain features stacked together where the rows from 0 to 12 are the frequency domain features and the rows from 13 to 17 are the time domain features.

The time domain features and the frequency domain features are then stacked together and a 2D feature vector of 18×64 is generated as shown in Figure 4b. In Figure 4b, the rows from 0 to 12 are the frequency domain features and the rows from 13 to 17 are the five time domain features. After this transformation, the one-dimensional time-domain sound signal is converted into a two-dimensional matrix, similar to an image. By utilizing this image representation, image classification deep learning models, such as Convolutional Neural Networks (CNNs), can be applied for classification. A dataset of a total of 155,000 images is generated from the two classes of sound samples and used as input for the deep learning model during classification. The MFCC data and their corresponding class labels are randomly shuffled while maintaining the correct pairing between data and labels.

3.1.3. Architecture of the Deep Learning Model

Multiple attempts were made to create a model that effectively fits the dataset while maintaining adequate capacity (i.e., parameters) to avoid overfitting. Initially, a model with

a high number of parameters was used, and the batch size was progressively increased until memory constraints became a factor. Afterward, the model's capacity was gradually reduced to achieve the right trade-off between accuracy and capacity. Learning rate adjustments and decay settings were fine-tuned whenever the validation loss was not decreasing for long training periods. Ultimately, the final deep learning model, as shown in Figure 5, was designed to classify sounds as either gunshots or non-gunshots. The model's architecture and its optimizer are summarized below:

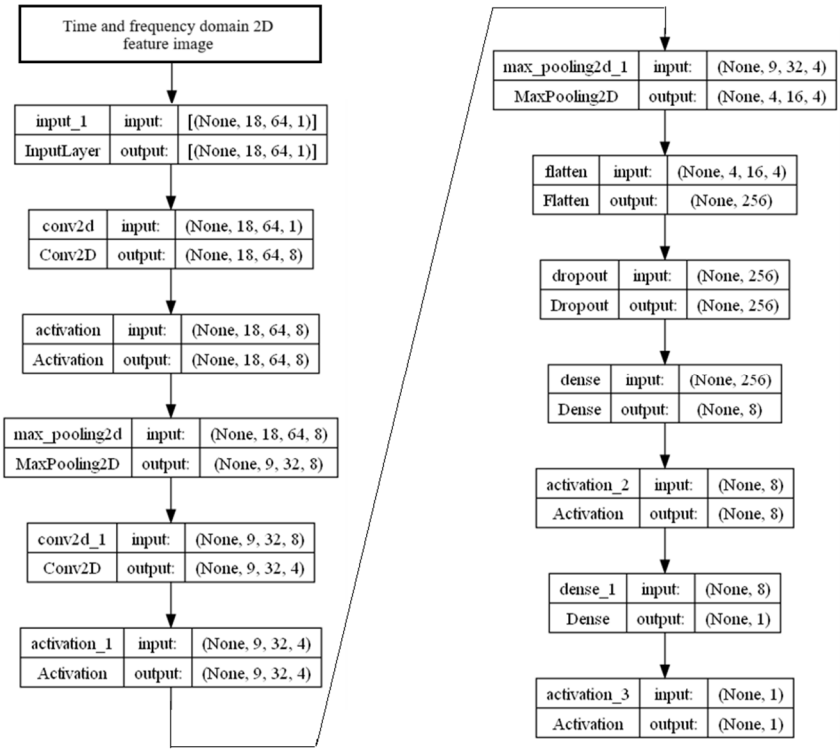


Figure 5. The architecture for the deep learning model.

- **Input Feature**: In this project, the one-dimensional time-domain audio signals are converted to two-dimensional 18 × 64 image-like representations, as discussed in Section 3.1.2. They serve as the input to the deep learning classifier.
- **Convolutional and Activation Layers**: The model consists of two convolutional layers [24] followed by ReLU activation functions [25]. The first convolutional layer uses 8 filters of size 3 × 3 with "same" padding, ensuring the output retains the input's spatial dimensions. The second convolutional layer uses 4 filters of size 2 × 2, also with "same" padding. Both layers are responsible for learning hierarchical spatial features from the input images, while the ReLU activations introduce nonlinearity by setting any negative values to zero, allowing the network to capture more complex patterns in the data.
- **Max Pooling Layers**: After each convolutional layer, a 2 × 2 max-pooling layer [26] is applied to down-sample the feature maps. The first max-pooling layer reduces the dimensions of the feature map from (18, 64, 8) to (9, 32, 8), while the second max-pooling layer further reduces it from (9, 32, 4) to (4, 16, 4). These layers help reduce the computational complexity of the model and retain the most relevant information by selecting the maximum value in each pooling window.
- **Flatten and Dropout Layers**: The 3D feature maps produced by the final max-pooling layer are flattened into a one-dimensional vector of size 256, allowing the data to be passed into fully connected layers. To prevent overfitting, a dropout layer with

a rate of 0.2 is applied, randomly setting 20% of the neurons to zero during each training iteration. This encourages the model to generalize better by not relying on specific neurons.

- **Fully Connected (Dense) Layers**: The flattened vector is passed through a dense (fully connected) layer [27,28] with 8 neurons, followed by a ReLU activation function. This layer learns the complex relationships between the features extracted by the convolutional layers. The final dense layer contains 1 neuron, which outputs a single value for binary classification.
- **Output Layer**: A Sigmoid activation function [29] is applied to the output of the final dense layer, producing a probability between 0 and 1. This allows the model to classify the input as either a gunshot or non-gunshot sound based on the computed probability.
- **Loss Function and Optimizer**: The model is compiled using the binary_crossentropy loss function, which measures the difference between predicted probabilities and the true labels (gunshot vs. non-gunshot). The RMSprop optimizer [30] is used with a specific learning rate and decay schedule to ensure efficient training.

3.1.4. Training the Deep Learning Model

From the gunshot and other class sound samples, 77,500 samples are randomly selected from each class and converted to 155,000 feature images. The images are then split into three distinct subsets: 70% of the images (108,500) were designated for training, 15% (23,250) were set aside for validation, and the remaining 15% (23,250) were reserved for testing. The test set was kept isolated until the model completed training and validation, allowing for an unbiased evaluation of model performance on unseen data.

The deep learning model, as shown in Figure 5, was implemented using Python and the Keras library. Keras, which serves as a high-level API for building neural networks on top of TensorFlow [31], was chosen for its flexibility and ease of use. The model was trained on a desktop computer equipped with a 12th Gen Intel Core i7 processor (6 cores) running at 2.10 GHz, 32 GB of RAM, and an NVIDIA GeForce RTX3070 GPU.

After training, the model was converted from a TensorFlow model to a LiteRT model [32], so that it can be used for inferencing in a low-resource embedded system.

3.2. Prototype Development with Security

The architecture of the gunshot detection system consists of the central server, gunshot detector devices, and smartphone app, as illustrated in Figure 1. Users attach the device to the walls or ceilings and configure its Wi-Fi settings through the smartphone app. Once the setup is complete, users can receive smartphone notifications from anywhere in the world, provided they have Internet access. The user and device data configuration using the smartphone and database schema of the proposed system was discussed in our previous work [13]. In this work, the focus is concentrated on some security issues such as MQTT communication protocol with improved authentication mechanisms, Wi-Fi provisioning without requiring Bluetooth, and over-the-air (OTA) firmware updates. A brief overview of the system's key modules is presented below.

3.2.1. Server Software

In the central server, as shown in Figure 1c, three server software runs MQTT broker, OTA server, and MySQL database server. The database has been described in our work in [13]. The MQTT broker and the OTA server are discussed below.

- **Message Queuing Telemetry Transport (MQTT) Broker:** In our previous work [13], there was no authentication to connect to the Transmission Control Protocol (TCP) server and it was open to cyber-attack. To solve this, a secure MQTT [33] transmission

protocol, designed for Internet of Things (IoT) applications, is used to communicate bidirectional data among the gunshot detector devices, the central server, and smartphone apps. The MQTT protocol offers an efficient and lightweight approach to messaging through a publish/subscribe model. According to the MQTT protocol, the transmitter client publishes a message to a topic in the broker server, and then the broker server sends the message to the receiving clients who subscribed to that particular topic.

In this project, the Mosquitto [34] is implemented in a Windows computer as the MQTT broker server. To increase security: authentication using username and password, and Transport Layer Security (TLS) certificates are configured. Username and password authentication ensure that only authorized users can access the broker. To create the password file and add a new user, the mosquito password utility [35] is used. One of its key security features is that it stores passwords in a hashed format rather than in plain text. Hashing algorithms such as SHA512, SHA256, and PBKDF2 are designed to be one-way functions, meaning it is computationally infeasible to reverse the hash to obtain the original password. This ensures that even if someone gains access to the password file, they cannot easily retrieve the actual passwords.

The purpose of using TLS certificates [36] in an MQTT broker and client device is to establish a secure, encrypted communication channel between them. TLS certificates provide authentication of both the broker and the client, ensuring that both parties can trust each other's identity before exchanging data. TLS ensures that data transmitted over the network are encrypted, protecting them from eavesdropping, tampering, or man-in-the-middle attacks. When TLS is used in the Mosquitto MQTT broker, common encryption algorithms include Advanced Encryption Standard (AES) for symmetric encryption, Rivest–Shamir–Adleman (RSA) or Elliptic Curve Cryptography (ECC) for key exchange and authentication, and SHA-256 for hashing to ensure data integrity, depending on the negotiated cipher suites. To configure TLS certificates [37] in the Mosquitto MQTT broker, we generate the necessary certificates: a CA certificate, a server certificate, and the corresponding private key using OpenSSL [38]. These certificates are used to encrypt communication between the broker and clients. The certificates are then copied to a secure location on the server. The Mosquitto configuration file is modified to enable TLS by specifying the paths to the CA certificate, server certificate, and private key, and setting the listener port to use TLS. When connecting, clients must also be configured to use TLS and use the same CA certificate to verify their identity.

To enable public access to the MQTT broker server from any location, it needs a fixed address and port number. The host computer's private IP is dynamically assigned by the router's dynamic host configuration protocol (DHCP) server, and it can change based on the devices connected to the local network. To resolve this, the private IP of the host computer is made static, and port forwarding is set up in the router [39]. The port forwarding mechanism directs incoming data packets from the Internet to the MQTT broker server. Additionally, the listener port is opened in the Windows Firewall settings [40]. A memorable and user-friendly name for the router's public IP is assigned using No-IP—a free dynamic domain name system (DDNS) service [41]. Although the router's public IP, assigned by the Internet service provider (ISP), does not change often, it may change after a few months or when the modem is restarted. To handle this, Dynamic DNS Update Client software [42] is installed on the host computer, which continuously checks for changes in the public IP and automatically updates the DNS at No-IP when necessary.

- **Over-the-Air (OTA) Server:** An OTA server is essential for managing firmware and security updates for IoT devices like gunshot detectors, without requiring physical access to the devices. Once the devices are installed, especially in hard-to-reach or

critical locations such as schools or offices, manually updating them can be costly and time-consuming. OTA updates allow system administrators to push new firmware or security patches remotely, ensuring that the devices remain secure and functional over time. This approach minimizes downtime, enhances device performance, and ensures quick deployment of updates in response to newly discovered vulnerabilities, all without having to remove or manually reconfigure each device.

In this project, the OTA server is written in Python and hosted on the same computer where the MQTT broker is hosted. To facilitate production-level deployment, the server uses the Waitress WSGI HTTP server [43] to handle requests. By using Waitress, the server can handle multiple requests simultaneously, making it suitable for environments where numerous devices may request firmware updates concurrently. The server is configured to listen on a particular port number and the port is opened in the Windows Firewall settings [40]. To access the server publicly, port forwarding is configured in the router [39].

The OTA update server implements HTTP Basic Auth [44] to ensure secure access to its endpoints, protecting sensitive data such as firmware files. It verifies user credentials stored in an external file (e.g., 'users.txt'). Usernames and hashed passwords are loaded from this file, allowing dynamic updates without modifying the server code. A separate Python script is used by the server administrators to add new users and hashed passwords; and append them to the file. Upon successful verification by the server, the client device is granted access to the server's resources, ensuring that only authorized devices can access version information or trigger firmware updates.

The server provides a dedicated endpoint for retrieving the current firmware version of a specified hardware version. The */get_fw_ver/<hw_ver>* endpoint serves as a mechanism to query the version of firmware that must be deployed for a particular device type. The server reads the firmware version from a text file (e.g., ver.txt) stored in a directory specific to each hardware version. This design enables efficient version control and ensures that the correct firmware version is served to requesting devices.

The core functionality of the OTA server is delivered through the */get_fw/<hw_ver>* endpoint, which provides the actual firmware update files. Upon receiving a request, the server first checks the corresponding ver.txt file to determine the latest firmware version for the specified hardware. It then locates the appropriate firmware package in .zip format (e.g., fw_v1.zip). The zip file contains 3 files: python code for gunshot detection and notification, python code for the server to access the recorded audio files, and the TensorFlow Lite deep learning model. If the Zip file is found, it is securely transmitted to the client using Flask's *send_file* function [45]. Error handling and logging are implemented to manage scenarios where files are missing or inaccessible.

3.2.2. Device for Gunshot Detection and Notification

The gunshot detection device monitors environmental sounds and classifies them as either gunshot or non-gunshot. Upon detecting a gunshot, the device sends a notification to smartphones over the Internet using the MQTT protocol and also saves the gunshot sound files locally on the device. Initial Wi-Fi and user configuration of the device, and controlling the device, such as enabling and disabling the device, is managed through the developed smartphone app. A brief overview of the device's hardware and firmware is provided below.

- **Hardware:** The hardware block diagram of the gunshot detection and notification device is shown in Figure 6. The Raspberry Pi (RPi) Zero 2W [46] is used as the main processing and communication unit. It has a 1 GHz quad-core 64-bit Arm Cortex-A53 CPU, 512 MB of SDRAM, 2.4 GHz 802.11 b/g/n Wi-Fi, Bluetooth Low Energy (BLE), onboard antenna, microSD card slot, and Hardware Attached on Top (HAT)

compatible 40-pin GPIO header. It also has a compact 65 mm × 30 mm form factor. A MEMS microphone breakout [17] is interfaced with the Raspberry Pi Zero 2W using the I2S interface for mono channel (e.g., left channel) input. The breakout board contains a compact, low-power microphone comprising of a high-performance SiSonic™ acoustic sensor, a serial analog-to-digital converter, and a signal conditioning interface that outputs audio in the standard 24-bit I2S format. The I2S interface facilitates integration with digital processors eliminating the need for an external audio codec or sound card. The Acoustic Overload Point (AOP) of this microphone is 120 dB. A sound level of 120 dB is equivalent to very loud noises, such as a rock concert, a jet engine from a short distance, or a gunshot. Thus, it can accurately capture sound levels up to 120 decibels without significant distortion. Three LEDs are connected to the GPIO ports of the Raspberry Pi in the active low configuration: a yellow LED to indicate listening mode, a green LED to indicate Internet connectivity, and a red LED to indicate gunshot event. Three current limiting resistors R1, R2, and R3 of 330 Ω are used for the LEDs. A push button switch is interfaced with a GPIO pin to reset the device manually by the user. For the power supply, a 100–240 v AC to 5 v DC converter module [47] is used. It can provide a maximum of 600 mA current and 3-watt power. A printed circuit board (PCB) containing the MEMS microphone connector, three LEDs with resistors, and the reset switch is developed, and connected to the 40-pin header of the Raspberry Pi as a HAT. A casing with a wall-outlet AC plug [48] is used to hold the electronics.

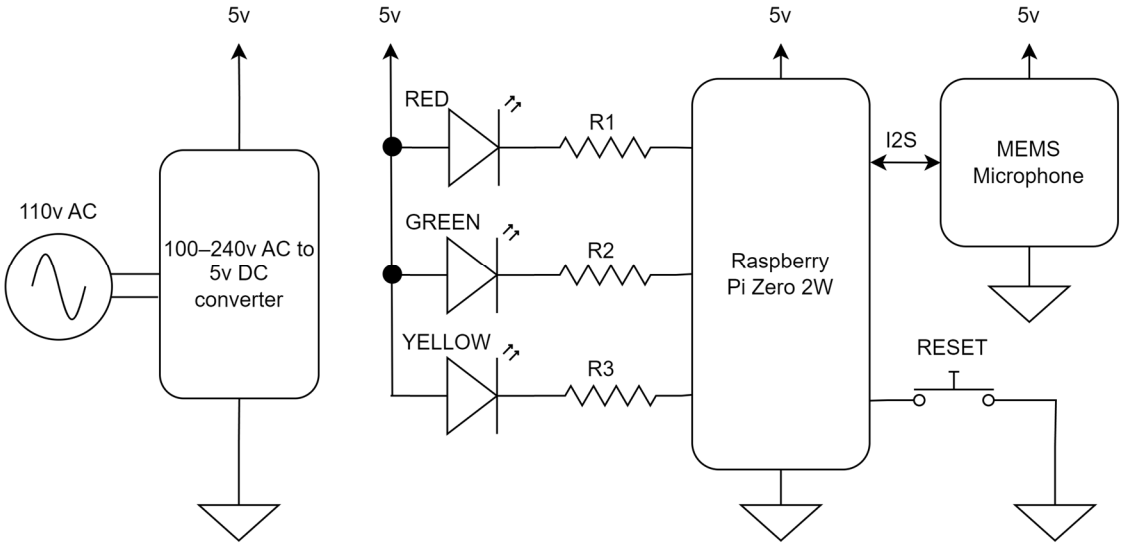


Figure 6. Hardware block diagram of the gunshot detection and notification device.

- **Firmware:** The Raspberry Pi Zero 2W is equipped with a 32 GB SD card, running Raspberry Pi OS 32-bit, which supports inferencing TensorFlow Lite models [49]. The application software is written in Python, with all necessary packages installed on the system. After boot, two Python programs operate concurrently in separate threads: one for initializing the device, Wi-Fi provisioning, detecting gunshots, and OTA update management, and the other for accessing the recorded gunshot sounds.

Initializing device: The device initialization process involves setting up the hardware and software components necessary for real-time gunshot detection. Upon boot, the Raspberry Pi Zero 2W configures its GPIO pins for various LED indicators and a reset button, ensuring proper signaling during operation. The system also loads the pre-trained TensorFlow Lite model into memory, preparing the neural network for real-time classification of audio data [49]. Additionally, the sound input subsystem is configured by initializing the

I2S microphone and setting up the audio stream for continuous capture at the 44.1 kHz sampling rate [50,51].

The MQTT client is also set up for communication with the MQTT broker, ensuring that detected gunshot events are transmitted in real-time [52]. To ensure communication security, the MQTT client is configured with the same CA certificate as discussed in Section 3.2.1, which provides encrypted communication over TLS. Upon successful connection to the MQTT broker, the device immediately sends a device status update to the user's smartphone indicating its connected status. In addition, the device subscribes to its specific MQTT topic, GSD_DEVICE_CMD/<DeviceID>, where <DeviceID> is the unique identifier for the device, allowing it to receive control commands from the user's smartphone. A "last will" message is configured within the MQTT client, allowing the device to notify the MQTT broker and the users if an unexpected disconnection occurs. To maintain connection stability, the MQTT client is configured with a keep-alive interval of 10 s. This setting ensures that the device periodically sends ping messages to the broker to verify that the connection remains active. If no communication is detected within the 10-s window, the MQTT broker can assume the device has disconnected and trigger the "last will" message.

A callback function, MQTT_on_message(), is a key component of the system's communication architecture, enabling real-time command from the user's smartphone to the device. This function is triggered whenever the device receives a message from the MQTT broker for the topic GSD_DEVICE_CMD/<DeviceID>. The smartphone app can send various commands to the device, such as enabling or disabling the gunshot detection functionality, requesting the device's local IP address, or initiating an OTA update. Upon receiving a command, it parses the payload and executes the appropriate action. For example, if the command is to enable the device, the system activates its detection mode by setting the isDeviceEnabled flag.

Upon boot, the init_ota() function determines whether all tasks of the OTA update have been completed. Lastly, any previous device status, such as enabled or disabled states, is read from the system files to determine whether the device should start in an active or inactive mode.

Wi-Fi provisioning: This function in the RPi device initiates the process of establishing a network connection, either by connecting to a pre-configured Wi-Fi network or setting up a hotspot for Wi-Fi provisioning. If the system detects that no active Wi-Fi connection is available, it automatically starts a hotspot [53], turns on all three LEDs, and waits for a smartphone app to provide Wi-Fi configuration information. The communication protocol between the RPi and the smartphone app is established through a socket connection [54], allowing the app to send Wi-Fi credentials securely to the device.

The smartphone app is responsible for scanning available Wi-Fi networks and allowing the user to select the desired network, and password if required. Once the user provides this information, the app opens a socket connection to the device hotspot and transmits the Wi-Fi configuration data in a structured format (e.g., security type, SSID, and password). The Python program running on the device receives these data through the socket, parses them, and attempts to connect to the specified Wi-Fi network. If successful, the device switches from hotspot mode to the configured network, turns off all the LEDs and continues the normal operation. The app verifies the successful Wi-Fi connection of the device by receiving the device serial number, <DeviceID>, which is needed by the smartphone to send commands to the device using MQTT.

Detecting gunshots: The pseudo-code of the main loop is shown in Figure 7. This loop governs the main operations, including audio recording, feature extraction, gunshot classification, and system state management.

```
while True:
  if isDeviceEnabled:
      audioData := RecordAudioSample()
      f := GenerateFeatures(audioData)
      for i from 0 to v-1:
          fc := Concatenate(fp[:, offset × (i + 1):], features[:, :offset × (i + 1)])
          isGunshot := ClassifySound(fc)
          if isGunshot:
              HandleGunshotEvent(audioData)
              break
      fp := f
  UpdateMQTTConnectionStatus()
  CheckResetButton()
  CheckForOTAUpdate()
```

Figure 7. Pseudocode for the main loop of the firmware.

At the beginning of each iteration of the loop, the device first checks if it is enabled by verifying the value of the isDeviceEnabled flag. If enabled, the system captures an audio sample using the RecordAudioSample() function. Here, the function turns on the yellow LED and waits until all the 1 s audio samples are read. It then turns off the yellow LED and leaves the function. The function does not terminate the recording process once it finishes reading the data. Instead, the audio stream continues running in the background, allowing continuous audio capture while doing feature extraction and classification. The average delay between this subsequent function call is 35.2 milliseconds, which is less than 1 s. This design ensures that no audio data are lost due to the delay in the remaining code of the loop. The audio data are then passed to the GenerateFeatures() function, which extracts both time-domain and frequency-domain features, as discussed in Section 3.1.2.

After generating the 2D features, f, the system processes them using overlapping windows, a technique that enhances the robustness of detection. Here, the number of overlaps, v, is set as 4 and *offset* as 16, calculated by dividing total feature columns = 64 by v = 4. Unlike the approach in our previous work in [13], where each iteration analyzes new 1 s audio data, this system creates overlap by combining the last columns of the previously generated feature matrix, fp, from the last audio sample with the beginning columns of the current feature matrix, f, from the new audio sample. For instance, the last 3/4th of fp columns are concatenated with the first 1/4th of the f column in the first iteration, then the last 2/4th of fp columns are concatenated with the first 2/4th of the f column in the second iteration, and so on. Finally, fp is set as f for the next cycle. This overlapping mechanism is essential for capturing transient audio events, such as gunshots, which may not be fully captured within a single window. By combining features from consecutive audio samples, the system increases the likelihood that the important characteristics of a gunshot are preserved across multiple windows, reducing the chances of missing critical audio patterns due to the boundaries.

Once the feature matrix is constructed, it is passed to the ClassifySound() function. This function employs the pre-trained convolutional neural network (CNN) model to classify the sound based on the extracted features. The CNN, optimized for real-time execution via TensorFlow Lite [49], analyzes the combined feature set and generates a probability score indicating the likelihood of the detected sound being a gunshot. If the

score exceeds the 0.5 probability threshold, the sound is classified as a gunshot. If a gunshot is detected, the system immediately triggers the HandleGunshotEvent() function, which manages all subsequent actions. These include publishing an MQTT message on the topic GSD_OBSERVER/<DeviceID> containing a timestamp, saving the audio data as a WAV file for forensic purposes, and turning on the red LED as a visual indicator.

In the main loop, the UpdateMQTTConnectionStatus() function continuously checks whether the device is connected to the MQTT broker and updates the status of the green LED accordingly. If the connection is lost, the device attempts to reconnect automatically. Finally, the CheckResetButton() function monitors the physical reset button connected to the Raspberry Pi's GPIO. If the button is pressed, the device measures the duration of the press to determine whether to perform a soft reboot or shut down the system. A short press triggers a reboot, allowing the system to restart. A long press, on the other hand, results in a complete shutdown.

OTA update management: The CheckForOTAUpdate () ensures that the system stays up-to-date with the latest firmware version without manual intervention. In each loop of the firmware, it checks the current time with the scheduled OTA update time. The scheduled OTA time is randomized based on the unique identity of the device to avoid overloading the server with many requests at the same time. If the current time matches the scheduled OTA time, the system initiates a series of operations to communicate with the OTA server that includes: verifying the availability of a new firmware version, downloading the update, installing it, restating, and deleting old files.

The first step in the OTA update process is to determine whether a new firmware version is available. When the update process is triggers, the device establishes a secure HTTP connection with the OTA server. This communication is facilitated through the Flask-based OTA server running on a designated IP and port as discussed in Section 3.2.1. It then sends a GET request to the server's/get_fw_ver/<hw_ver> endpoint, where <hw_ver> represents the hardware version of the device. This endpoint checks the ver.txt file on the server, which contains the current firmware version for the specified hardware version. Upon receiving the request, the OTA server verifies the identity of the device using HTTP Basic Authentication [44]. If authentication is successful, the server reads the ver.txt file, retrieves the current firmware version, and sends it back to the device in the HTTP response. The device then compares the received firmware version with its own. If the versions differ, indicating that a new update is available, the device proceeds with the update process.

Once the system confirms that a new firmware version is available, the device initiates the downloading and installation of the update. This function establishes another secure HTTP connection with the OTA server, targeting the /get_fw/<hw_ver> endpoint. The firmware files are packaged based on hardware version compatibility, preventing the installation of firmware on incompatible devices. The OTA server responds by sending the firmware file packaged as a Zip archive. The Zip file is streamed directly to the device, ensuring minimal latency. Upon receiving the Zip archive, the device uses Python's zip file module to extract the firmware files into the appropriate directory on the device.

After extracting the files, the device updates a shell script with the new filenames, which lists the Python files that will run after the system boots [55]. It also writes in an ota.dat file: the version number of its current firmware and the flag isUpdatingDone as False, indicating that its current firmware files still need to be deleted by the new firmware after the boot, as the currently running Python script cannot delete itself. Finally, the system triggers a reboot to apply the firmware changes, ensuring that the updated firmware is executed on the next boot. Upon boot, it checks the ota.dat file to determine whether all tasks of the update have been completed. If the flag *isUpdatingDone* is set to False, the function reads the version of the previous firmware that can now be deleted. The system

then proceeds to remove the old version's files, such as outdated Python scripts and model files, and sets *isUpdatingDone* as True. This function is crucial for freeing up space for future updates.

***Secure gunshot audio file access* via *local network*:** A secure web server using Flask on the RPi device is implemented to allow access to recorded gunshot audio files over the local Wi-Fi network. The server employs HTTP Basic Authentication with password hashing to ensure that only authorized users can access the files. SSL/TLS encryption ensures secure transmission of data between the client (smartphone or desktop) and the server, protecting sensitive information like passwords and audio files. The server also features a password change functionality. Users can update the administrator password through a web form, and the updated password is securely hashed and stored in a file.

Gunshot audio files, stored in a specific directory, are dynamically listed on the homepage, and users can click links to download the files. File access is handled by Flask's 'send_from_directory()' method, ensuring that only files in the designated directory are accessible.

3.2.3. Smartphone App

The smartphone app, developed for the Android platform, uses the MLWiFi library [56] to configure the RPi device's Wi-Fi settings. The app scans for available 2.4 GHz Wi-Fi networks and fills a combo box. The user then selects the SSID and enters its password in a textbox. The app then connects to the device's hotspot (SSID: "gsd_hotspot"). Upon establishing the connection via a socket, the app sends the selected network's SSID and password to the device. The device receives these credentials and attempts to connect to the specified network. Once the device is connected to the Wi-Fi, it sends the <DeviceID> to the smartphone app, and the device is added to a list. Users can add multiple devices to their list by configuring Wi-Fi, and the app automatically subscribes to MQTT topic GSD_OBSERVER/<DeviceID> associated with each device for real-time monitoring.

The app initializes the MQTT client at boot [57], ensuring it runs continuously as a background process, even when the app is not open. This MQTT client connects with the MQTT broker using authentication credential with SSL encryption; and it reconnects automatically if disconnected, ensuring that real-time communication with the device remains uninterrupted. The app can send MQTT messages related to device status such as connected or disconnected, gunshot notifications, and can send control commands, keeping the system responsive to user inputs and sensor events.

The app receives gunshot detection data through a call back function which is executed whenever MQTT massage arrives. When a gunshot event is detected, the app logs the event along with the timestamp and displays the information in real time. A phone notification is also raised to immediately alert the user. Additionally, when the user clicks on the logged event, the app opens Google Maps, showing the exact location where the gunshot was detected. This feature relies on the GPS data associated with the device. The detailed process of device configuration, including the input of GPS coordinates and storing them in the database, has been discussed in our previous work [13].

To access the recorded gunshot audio files stored on the device, the app sends an MQTT command to topic GSD_DEVICE_CMD/<DeviceID> requesting the device's latest local IP address, as this address may change depending on the network. Upon receiving the device's IP via MQTT, the app opens a browser to access the device's file server, allowing the user to play and download sound recordings directly from the device. To ensure security, access to these files is protected by HTTP Basic Authentication and encrypted using SSL/TLS. This provides a secure and dynamic way to access event-specific audio files stored on the device. The app allows users to send commands to the devices on the

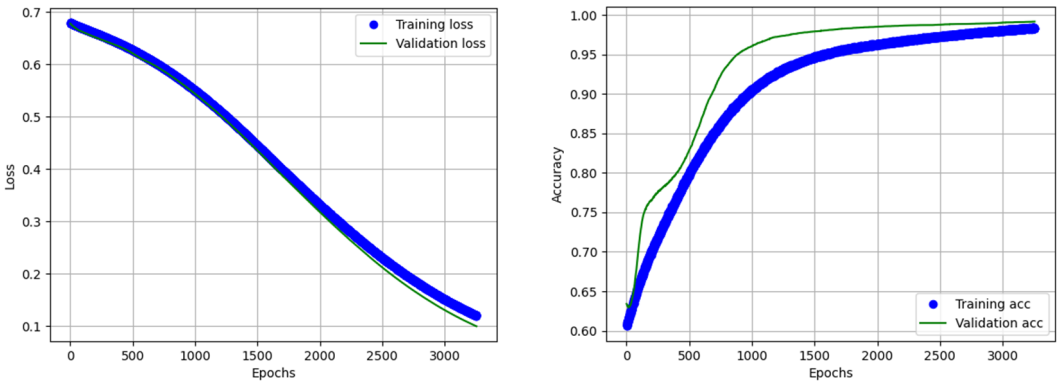
topic GSD_DEVICE_CMD/<DeviceID>, such as enabling or disabling the device, initiating over-the-air (OTA) updates, or requesting the current IP address. These commands are sent via MQTT, ensuring secure and efficient control over multiple devices.

4. Results

The results of the deep learning model and the performance of the prototype system are discussed below.

4.1. Deep Learning Model Results

The deep learning model, as described in Section 3.1.3, was trained and validated concurrently until either the validation loss dropped to 0.01 or less, or the model completed 50,000 epochs, whichever occurred first. The batch size for both training and validation was set to 4096. The learning rate was configured at 1×10^{-6}, with a learning rate decay of 1×10^{-7}. Figure 8a,b illustrate the loss versus epochs and accuracy versus epochs trends for both the training and validation datasets. These graphs clearly show a steady reduction in loss and a simultaneous rise in accuracy as the epochs progress. The model reached a validation loss of 0.01 by the 3255 epochs, concluding the training in 27 min and 41 s. Notably, the training and validation datasets reached an accuracy of approximately 98% and 99% respectively, after the 3255 epochs.



(**a**) (**b**)

Figure 8. (**a**) Loss over epochs for both training and validation datasets; (**b**) accuracy over epochs for both training and validation datasets.

Once training and validation were completed, the model, which consists of 2277 learned parameters (including filters, weights, and biases), was saved and was then tested on an unseen dataset of 23,250 samples. During this testing phase, the model recorded a loss of 0.0995 and an accuracy of 99%. Table 4 summarizes the loss and accuracy for the training, validation, and test datasets, demonstrating that the model's accuracy is consistent across all sets and its strong generalization capability. The confusion matrix for the test dataset is shown in Figure 9, while Table 5 provides the precision, recall, and F1 scores for the test dataset.

Table 4. The loss and accuracy of the training, validation, and test datasets.

	Training	Validation	Test
Loss	0.1205	0.1000	0.0995
Accuracy	0.9839	0.9916	0.9917

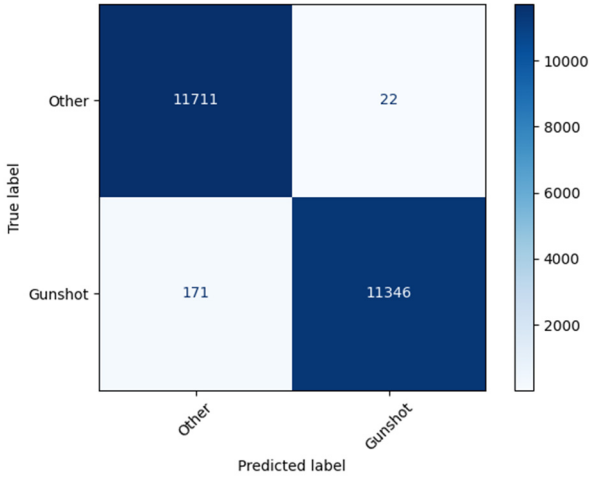


Figure 9. Confusion matrix of the test dataset.

Table 5. The precision, recall, and F1-scores of the test dataset.

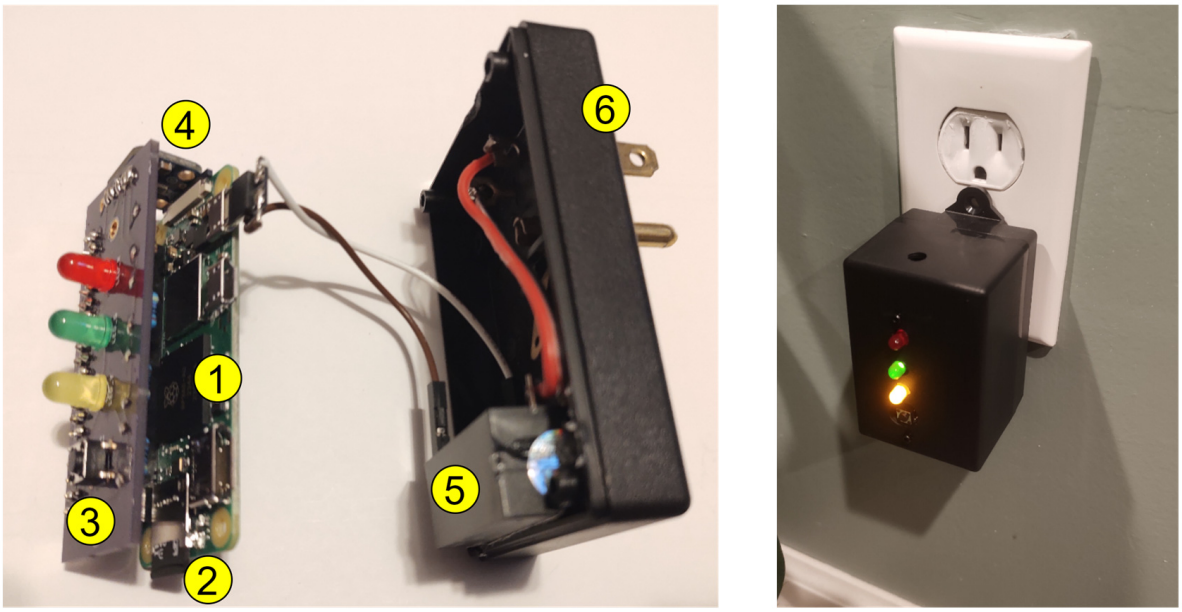
	Precision	Recall	F1-Score
Gunshot	1.00	0.99	0.99
Other	0.99	1.00	0.99

4.2. Prototype Testing Result

A working prototype of the proposed system, consisting of three gunshot detector devices, server, and smartphone applications, has been developed and successfully tested. Figure 10a displays the gunshot detector device with a custom-made PCB HAT and Figure 10b shows the device housed in a casing with dimensions of approximately 7.6 × 5.1 × 5.1 cm [48]. The device is programmed based on the methodology outlined in Section 3.2.2 and is configured to automatically execute programs upon boot. On the RPi Zero 2W, feature generation takes an average of 21.76 milliseconds, and a single inference using the TF-Lite deep learning model takes 3.7 milliseconds. Depending on internet latency, the MQTT broker may take from 10 ms to 100 ms to send the notification [58]. Thus, the maximum delay to receive the smartphone notification is 21.76 + 3.7 + 100 = 125.76 ms.

The device's current consumption was measured using a Keysight N6705C DC power analyzer [59] equipped with the N6781A source/measure unit (SMU) module [60], with the voltage set to 5 V. During normal operation, the current consumption ranged from approximately 200 mA to 400 mA, which is below the maximum 600 mA limit of the DC power supply module [47] used in the device.

The MQTT broker and OTA server, as discussed in Section 3.2.1, were running on an Internet-connected computer. After the gunshot detector devices were powered up, all three LEDs turned on to indicate that they were waiting for Wi-Fi provision from a smartphone. The device's Wi-Fi is then configured using the smartphone app, as discussed in Section 3.2.2. Firmware and Section 3.2.3. Some screenshots of the smartphone app for Wi-Fi provisioning are shown in Figure 11. Using the app, the Wi-Fi of the devices is configured and the devices are added to the app to receive notifications. The devices then went to the listening mode indicated by the blinking of the yellow LED, and it turned on the green LED to indicate a successful connection with the MQTT broker.



(**a**) (**b**)

Figure 10. (**a**) Photograph of the gunshot detector device: (1) Raspberry Pi Zero 2W; (2) SD card; (3) custom design PCB HAT containing three LEDs and one push button switch; (4) MEMs microphone breakout connected with the PCB HAT; (5) AC to DC power module; (6) bottom part of the casing with AC plugs. (**b**) Gunshot detector device in casing, connected with wall AC power outlet.

The gunshot detector was tested inside a lab environment by performing actual shootings using two types of blank guns: ZORAKI M906 semi-automatic blank pistol [21] and Ekol ASI fully-automatic blank machine gun [61]. The ZORAKI blank pistol can only make a single shot, whereas, the Ekol blank machine gun can make single or multiple shots in one trigger. Tests were conducted in the presence of security personnel of the university and wearing proper ear protection. The gunshot detector device was placed in the lab and in the building corridor, and shots were fired from different distances. In Table 6, the gunshot detection accuracy in single-shot mode at different distances is shown. Here, we see that it can detect gunshots with an accuracy of 100% up to 40-feet distances. After that, the accuracy reduces to 50% from a distance of 50 feet to 80 feet, and, finally, the accuracy becomes 0% when the distance is more than 80 feet. In Table 7, the gunshot detection accuracy with the Ekol machine gun in the research lab at different distances is shown. In this test, the machine gun was switched to automatic mode and it shot multiple ammos in one trigger. Here, we see that the detection accuracy is 100% for different numbers of ammos and at different distances. The sound pressure levels (SPL) were measured using a decibel meter [62] in the lab. The SPL was around 117 dBA in the lab for the gunshot, however, it reached over 130 dBA when more than 7 ammo were fired in a single trigger by the machine gun.

The proposed system was also tested for situations when a gunshot event happens in the presence of background noise. Background noise was created by playing sound effects of people talking in school hallways, thunderstorms, and nursery rhymes from a smart TV. The SPL was around 30–40 dBA in the absence of the background noise, and the SPL increased in the range of 60–70 dBA in the presence of the background noises. Using the Zokari pistol, gunshots were fired from different distances and the device detected the gunshots with 100% accuracy as shown in Table 8. No false alarm was generated due to the background sounds.

Table 6. Gunshot detection accuracy in single-shot mode at different distances.

Blank Gun Type	Number of Shots	Location	Distance (Feet)	Detection Accuracy
Zokari pistol	5	Research lab	16	100%
	5		32	100%
Ekol machine gun	4		16	100%
	6		32	100%
	2	Building corridor	10	100%
	2		20	100%
	2		30	100%
	2		40	100%
	2		50	50%
	2		60	50%
	2		70	50%
	2		80	50%
	2		90	0%
	2		100	0%

Table 7. Gunshot detection accuracy in multiple-shot mode at different distances.

Number of Ammos in a Shot	Distance (Feet)	Detection Accuracy
2	16	100%
3	32	100%
4	16	100%
5	32	100%
6	16	100%
7	32	100%
8	16	100%
9	32	100%

Table 8. Gunshot detection accuracy in the presence of background noise at different distances.

Background Noise	Distance (Feet)	Number of Shots	Detection Accuracy
Talking in school hallway	16	3	100%
	32	3	100%
Thunderstorm	16	3	100%
	32	3	100%
Nursery rhymes	16	3	100%
	32	3	100%

During testing, different sounds other than gunshots were generated such as talking, playing movies, clapping, laughing, etc. and they were successfully detected as non-gunshot sounds. To test the device for a false alarm, balloons were popped and gunshot sounds from TV were played in front of it, and the device was taken outside on July 4th night (the Independence Day of the USA) when the fireworks were happening. The device did not detect those sounds as gunshot sounds and did not create any false alarms.

The smartphone app received notifications in less than a second after the device detected the gunshot. Screenshots of the smartphone app showing a notification, event log with timestamp, and navigation direction on the gunshot location using Google Maps are shown in Figure 12a–c. Note the detailed process of user and device configuration,

including the input of GPS coordinates and storing them in the database, has been discussed in our previous work [13].

The OTA update was tested by configuring a new firmware version in the OTA server. The devices successfully updated their firmware from the OTA server at the scheduled time. The proposed devices were also tested by unexpectedly removing them from power. The MQTT broker successfully sent the last will messages to the smartphone indicating its disconnection status. When the devices were again powered on, the smartphone successfully received messages of their connected status. Commands from the smartphone app to the devices were sent, such as enable/disable a device, OTA now, and accessing recorded gunshot sounds that are stored on the devices, and they worked as expected. Authentication and password change features worked successfully to access the recorded gunshot sounds. The screenshots of the connection status update log, command buttons, and accessing gunshot sounds are shown in Figure 12d–f.

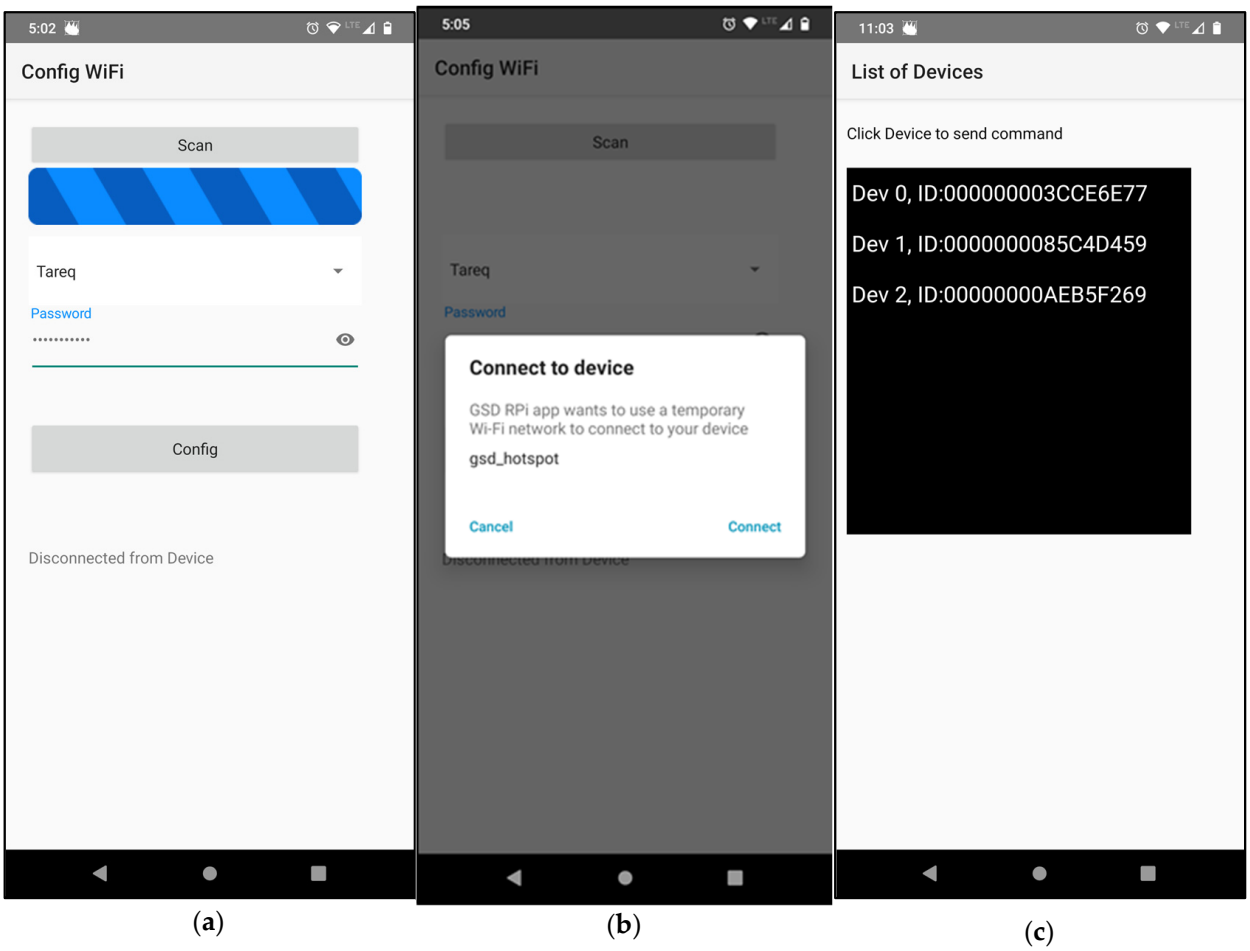


Figure 11. Screenshots of the smartphone app during Wi-Fi provisioning: (**a**) the app fills a combo box by scanning the nearby SSIDs, the user selects the SSID, and provides its password; (**b**) the app then confirms with the user to connect to the device's hotspot (SSID: "gsd_hotspot"), it then transmits the SSID and password via a socket to the device; (**c**) once the device is connected to the Wi-Fi, it sends its <DeviceID> to the smartphone app and the device is added to a list.

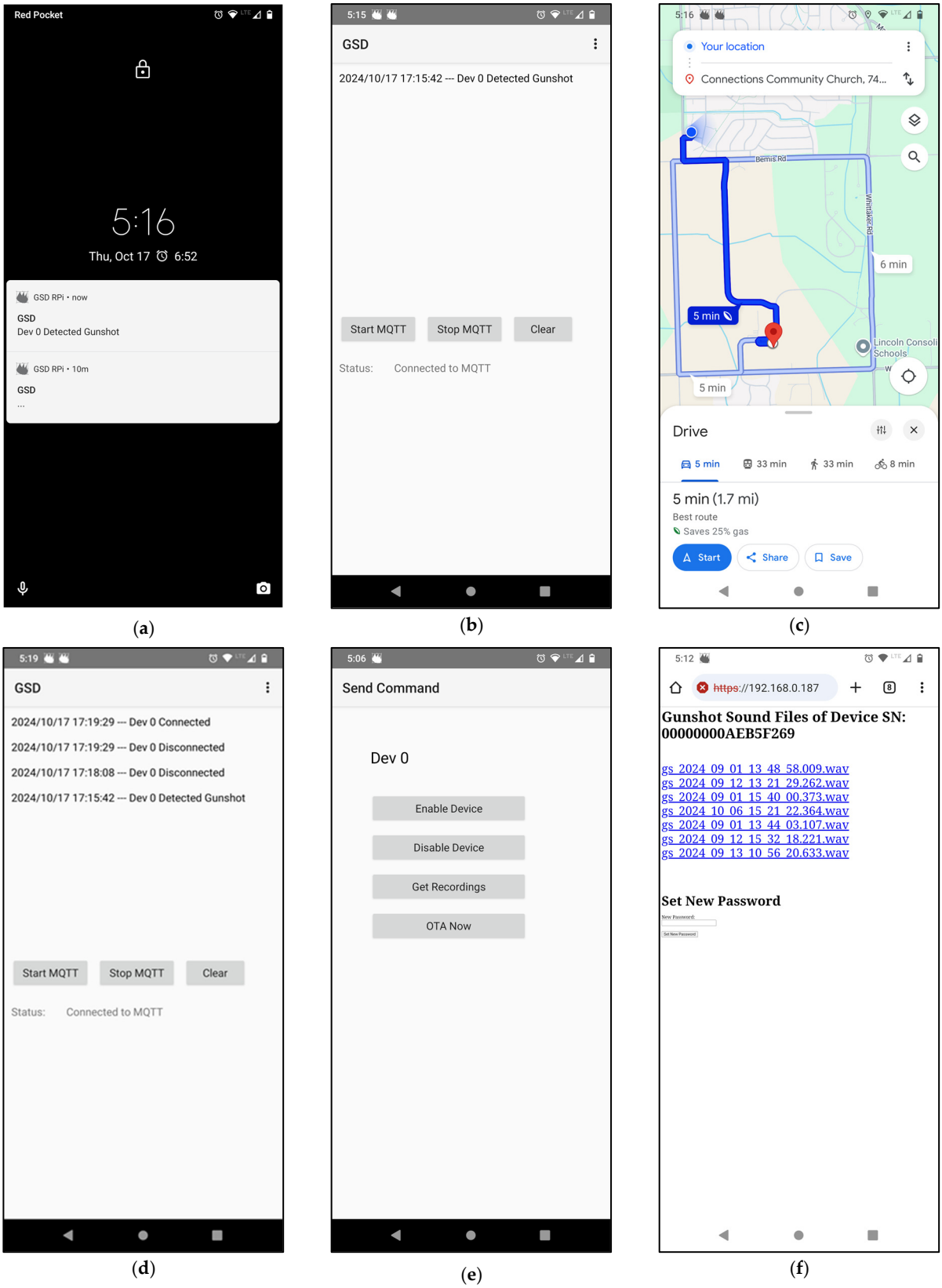


Figure 12. Screenshots of the smartphone app showing: (**a**) gunshot event notification; (**b**) event log with timestamp; (**c**) navigation direction on the gunshot location using Google Maps; (**d**) device connection status log; (**e**) command buttons to control device; (**f**) accessing gunshot sounds stored on the device.

5. Discussion

We recorded the same sound using the custom sound recording device, as discussed in Section 3.1.1. and in Figure 2, and the gunshot detector device, as discussed in Section 3.2.2. and in Figure 6. We then plotted the time domain signals of the two recordings and found that the amplitude of the sound recorded by the gunshot detector device is 1.44 times higher than the custom recording device that was used to develop the training dataset. Due to this mismatch, the gunshot detector device became too sensitive and was giving frequent false alarms. The problem was then solved by dividing the amplitude of the recorded signal by 1.44 and then generating the features for classification.

During testing, we found that if loud screaming is done by placing the mouth within around 30 cm distance of the microphone, the gunshot detector device classifies it falsely as a gunshot. However, if the distance is farther, the screaming is not classified as a gunshot. To solve this, we plan to add more screaming sounds to the training dataset of non-gunshot sounds.

It is observed from Tables 6 and 7 that the device was able to detect gunshot sounds from the machine gun, even though the deep learning model was only trained with the dataset made with blank pistol sounds. It shows the model is well generalized and is expected to detect gunshot sounds from different types of guns. Plans include making a dataset of gunshots with different types of guns and retraining the model, testing the system in shooting range with guns that throw bullets instead of blanks, and performing ethical hacking on the system to find further security variabilities.

Message Queuing Telemetry Transport (MQTT) [33], Firebase Cloud Messaging (FCM) [63], and Short Message Service (SMS) are three popular technologies for real-time notification delivery. MQTT is a lightweight, publish-subscribe messaging protocol designed for resource-constrained environments like IoT systems. It supports quick message delivery, multiple Quality of Service (QoS) levels for reliability, and robust security features such as SSL/TLS encryption. Additionally, MQTT is available as open source [29] and does not rely on third-party platforms, offering greater control and flexibility. In contrast, FCM, a cloud-based push notification service provided by Google, is aimed at mobile applications and offers features like device targeting and cross-platform compatibility. However, its dependency on Google's infrastructure and potential latency issues, especially during high traffic periods, can make it less suitable for critical applications like gunshot detection. SMS requires a GPRS modem in the embedded systems, adding hardware overhead. Additionally, SMS incurs a per-message fee and makes it expensive. For such scenarios, MQTT's low latency [58], lightweight nature, security features, and independence from external services make it a more reliable and efficient choice for ensuring timely notifications.

According to the Nyquist theorem [64], the sampling rate must be at least twice the highest frequency present in the signal to capture it accurately. Human hearing typically ranges from 20 Hz to 20,000 Hz, so a sampling rate of 44.1 kHz ensures that frequencies up to 22,050 Hz (just above human hearing) are captured without aliasing. The proposed system captures, records, and classifies gunshot and non-gunshot sounds, which might include sounds anywhere in the entire hearing frequency range. Thus, lowering the sampling rate may distort the signal and may lead to misclassification. If a low-resource microcontroller having limited RAM and speed is used, then it might be necessary to reduce the sampling rate and resolution, and compromise accuracy. However, the Raspberry Pi Zero 2W has sufficient RAM and processing speed to store 1 s of sound at a 44.1 kHz sampling rate with 32-bit resolution, which requires approximately 176.4 kB of memory for mono audio. With 512 MB of RAM and a 1 GHz quad-core processor, the device can handle these data efficiently without compromising accuracy or requiring a reduction in resolution. As

future work, we plan to implement the device in low resource microcontroller and perform experiments with lower sampling rate and resolution.

The study in [65] investigates the potential of using exclusively time-domain features for gunshot recognition, calculating a set of 11 features derived from time-domain signals. However, the results indicate a precision of only 69.3% for gunshot detection. The authors concluded that relying solely on time-domain features results in comparatively lower performance but acknowledged that some of these features have the potential to enhance recognition accuracy when combined with other conventional features. In our earlier work [13], we used only frequency-domain features. However, we observed that the loudness of the gunshot sound is critical for accurate classification. Neglecting amplitude-related features led to false alarms, particularly when the classifier encountered low-volume gunshot sounds from TVs or gaming devices. In real-world scenarios, amplitude-related features play a significant role in distinguishing real gunshots from potential false alarms. In this study, we utilize both time and frequency-domain features. Real-time testing with blank guns demonstrated that the proposed system achieves 100% accuracy in detecting gunshot sounds within a 40-foot range, as shown in Table 6. Moreover, the system successfully passed false alarm tests involving balloon pops, fireworks, and gunshot sounds from action movies.

6. Conclusions

In this project, a new custom dataset of gunshot sounds was recorded, and a deep learning model was trained to classify gunshot and non-gunshot sounds with 99% accuracy. The gunshot detection IoT system, consisting of the central server, gunshot detector devices, and a smartphone app, was developed and successfully tested with blank gunshots and possible false alarm situations. In this research, security vulnerabilities of the system were addressed by implementing secure MQTT communication protocols, authentication techniques, Wi-Fi provisioning without Bluetooth, and OTA firmware update features.

7. Patents

A provisional US patent (application# 63/707,567) has been filed based on the work reported in this manuscript.

Funding: This research was funded by the Provost's Research Support Award and the Phase II GameAbove Cybersecurity Research Award of Eastern Michigan University.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: The author would like to thank Mohamed Hagras for his technical help during the dataset creation and testing with blanks. Thanks go to Chief Matthew Lige from the Department of Public Safety (DPS) at Eastern Michigan University for providing police support during testing. Thanks also go to the volunteers who contributed to creating the non-gunshot dataset: Abdul-Azeez Madani, Abdulmalek Almansoob, Hussein Ahmad, J.T. Puszczewicz, Leo Da Silva Oliveira, Muhammad Ashraf, Ross Rhizal, and Sajad Abood.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Gun Violence Archive. Available online: https://www.gunviolencearchive.org/ (accessed on 24 September 2024).
2. Assumption, vs. Reality: Do Emergency Response Times Significantly Affect Active Shooter Outcomes? Available online: https://amberbox.com/blog/post/assumption-vs-reality-do-emergency-response-times-significantly-affect-active-shooter-outcomes (accessed on 24 September 2024).

3. Active Shooter Notification Time Costs Lives. Available online: https://guard911.com/active-shooter-notification-time-costs-lives/ (accessed on 24 September 2024).
4. ZeroEyes. Available online: https://zeroeyes.com/ (accessed on 30 September 2024).
5. Eastern Michigan University Adopts AI Gun Detection Technology. Available online: https://www.insightintodiversity.com/east-michigan-university-adopts-ai-gun-detection-technology/ (accessed on 1 October 2024).
6. AmberBox. Available online: https://amberbox.com/ (accessed on 1 October 2024).
7. SoundThinking. Available online: https://www.soundthinking.com/law-enforcement/gunshot-detection-technology (accessed on 24 December 2024).
8. Connealy, N.T.; Piza, E.L.; Arietti, R.A.; Mohler, G.O.; Carter, J.G. Staggered deployment of gunshot detection technology in Chicago, IL: A matched quasi-experiment of gun violence outcomes. *J. Exp. Criminol.* **2024**. [CrossRef]
9. Lopez-Morillas, J.; Canadas-Quesada, F.J.; Vera-Candeas, P.; Ruiz-Reyes, N.; Mata-Campos, R.; Montiel-Zafra, V. Gunshot detection and localization based on Non-negative Matrix Factorization and SRP-Phat. In Proceedings of the 2016 IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM), Rio de Janeiro, Brazil, 10–13 July 2016; pp. 1–5. [CrossRef]
10. Valenzise, G.; Gerosa, L.; Tagliasacchi, M.; Antonacci, F.; Sarti, A. Scream and gunshot detection and localization for audio-surveillance systems. In Proceedings of the 2007 IEEE Conference on Advanced Video and Signal Based Surveillance, London, UK, 5–7 September 2007; pp. 21–26. [CrossRef]
11. Bajzik, J.; Prinosil, J.; Koniar, D. Gunshot Detection Using Convolutional Neural Networks. In Proceedings of the 2020 24th International Conference Electronics, Palanga, Lithuania, 15–17 June 2020; pp. 1–5. [CrossRef]
12. Morehead, A.; Ogden, L.; Magee, G.; Hosler, R.; White, B.; Mohler, G. Low Cost Gunshot Detection using Deep Learning on the Raspberry Pi. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 3038–3044. [CrossRef]
13. Khan, T.H. Towards an indoor gunshot detection and notification system using deep learning. *Appl. Syst. Innov.* **2023**, *6*, 94. [CrossRef]
14. BGG Dataset (PUBG Gun Sound Dataset). Available online: https://github.com/junwoopark92/BG-Gun-Sound-Dataset (accessed on 1 August 2023).
15. Free Firearm Sound Effects Library. Available online: https://opengameart.org/content/the-free-firearm-sound-library (accessed on 1 August 2023).
16. Urbansound8k Dataset. Available online: https://urbansounddataset.weebly.com/urbansound8k.html (accessed on 1 August 2023).
17. Adafruit I2S MEMS Microphone Breakout—SPH0645LM4H. Available online: https://www.adafruit.com/product/3421 (accessed on 5 September 2024).
18. Teensy® 4.1 Development Board. Available online: https://www.pjrc.com/store/teensy41.html (accessed on 5 September 2024).
19. Audio System Design Tool for Teensy Audio Library. Available online: https://www.pjrc.com/teensy/gui/ (accessed on 5 September 2024).
20. NCH WavePad Audio Editing Software. Available online: https://www.nch.com.au/ (accessed on 5 September 2024).
21. ZORAKI M906 Semi-Auto Blank Pistol. Available online: https://blankgunarmory.com/zoraki-m906-semi-auto-blank-pistol-black-black/?1 (accessed on 5 September 2024).
22. Davis, S.; Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Trans. Acoust. Speech Signal Process.* **1980**, *28*, 357–366. [CrossRef]
23. Fayek, H. Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between. Available online: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html (accessed on 10 September 2024).
24. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
25. Nair, V.; Hinton, G.E. Rectified linear units improve restricted Boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, 21–24 June 2010; pp. 807–814.
26. Nagi, J.; Ducatelle, F.; Di Caro, G.A.; Ciresan, D.; Meier, U.; Giusti, A.; Nagi, F.; Schmidhuber, J.; Gambardella, L.M. Max-Pooling Convolutional Neural Networks for Vision-based Hand Gesture Recognition. In Proceedings of the 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA2011), Kuala Lumpur, Malaysia, 16–18 November 2011.
27. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, Sardinia, Italy, 13–15 May 2010; pp. 249–256.
28. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
29. Bishop, C.M. *Pattern Recognition and Machine Learning*; Springer: New York, NY, USA, 2006.
30. A Look at Gradient Descent and RMSprop Optimizers. Available online: https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b (accessed on 11 September 2024).

31. Keras: The Python Deep Learning Library. Available online: https://keras.io (accessed on 1 October 2024).
32. Convert TensorFlow Models. Available online: https://ai.google.dev/edge/litert/models/convert_tf (accessed on 11 September 2024).
33. MQTT: The Standard for IoT Messaging. Available online: https://mqtt.org/ (accessed on 2 October 2024).
34. Eclipse Mosquitto: An Open Source MQTT Broker. Available online: https://mosquitto.org/ (accessed on 2 October 2024).
35. Manage Password Files for Mosquito. Available online: https://mosquitto.org/man/mosquitto_passwd-1.html (accessed on 2 October 2024).
36. Paris, I.L.B.M.; Habaebi, M.H.; Zyoud, A.M. Implementation of SSL/TLS Security with MQTT Protocol in IoT Environment. *Wirel. Pers. Commun.* **2023**, *132*, 163–182. [CrossRef]
37. Configure SSL/TLS support for Mosquitto. Available online: https://mosquitto.org/man/mosquitto-tls-7.html (accessed on 2 October 2024).
38. OpenSSL. Available online: https://www.openssl.org/ (accessed on 2 October 2024).
39. How to Port Forward. Available online: https://www.noip.com/support/knowledgebase/general-port-forwarding-guide/ (accessed on 2 October 2024).
40. How Do I Open a Port on Windows Firewall? Available online: https://www.howtogeek.com/394735/how-do-i-open-a-port-on-windows-firewall/ (accessed on 2 October 2024).
41. No-IP. Available online: https://www.noip.com/ (accessed on 2 October 2024).
42. Dynamic DNS Update Client (DUC) for Windows. Available online: https://www.noip.com/download?page=win (accessed on 2 October 2024).
43. Waitress WSGI Server. Available online: https://docs.pylonsproject.org/projects/waitress/en/stable/ (accessed on 4 October 2024).
44. HTTP Basic Auth. Available online: https://flask-httpauth.readthedocs.io/en/latest/ (accessed on 4 October 2024).
45. Flask's Send File Function. Available online: https://flask.palletsprojects.com/en/latest/api/#flask.send_file (accessed on 4 October 2024).
46. Raspberry Pi Zero 2, W. Available online: https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/ (accessed on 7 October 2024).
47. HLK PM01 AC DC Converter 220 V to 5, V. Available online: https://www.amazon.com/EC-Buying-Step-Down-Intelligent-3-3V/dp/B09Z253MQ2/?th=1 (accessed on 7 October 2024).
48. PM2320 AC Wall Plug Enclosure. Available online: https://www.polycase.com/pm2320#PM2320T03XWT (accessed on 7 October 2024).
49. LiteRT: Quickstart for Linux-Based Devices with Python. Available online: https://ai.google.dev/edge/litert/microcontrollers/python#install_tensorflow_lite_for_python (accessed on 16 October 2024).
50. Recording Stereo Audio on a Raspberry, Pi. Available online: https://makersportal.com/blog/recording-stereo-audio-on-a-raspberry-pi (accessed on 16 October 2024).
51. PyAudio. Available online: https://github.com/CristiFati/pyaudio?tab=readme-ov-file (accessed on 16 October 2024).
52. Paho-Mqtt. Available online: https://pypi.org/project/paho-mqtt/ (accessed on 16 October 2024).
53. Nmcli—Command-Line Tool for Controlling NetworkManager. Available online: https://networkmanager.dev/docs/api/latest/nmcli.html (accessed on 16 October 2024).
54. Socket—Low-Level Networking Interface. Available online: https://docs.python.org/3/library/socket.html (accessed on 16 October 2024).
55. How to Run a Raspberry Pi Program on Startup. Available online: https://learn.sparkfun.com/tutorials/how-to-run-a-raspberry-pi-program-on-startup/all#method-2-autostart (accessed on 16 October 2024).
56. MLwifi library. Available online: https://www.b4x.com/android/forum/threads/mlwifi-library-updated-to-v4-00.125051/ (accessed on 17 October 2024).
57. jMQTT Library. Available online: https://www.b4x.com/android/help/jmqtt.html (accessed on 17 October 2024).
58. Gavrilov, A.; Bergaliyev, M.; Tinyakov, S.; Krinkin, K.; Popov, P. Using IoT Protocols in Real-Time Systems: Protocol Analysis and Evaluation of Data Transmission Characteristics. *J. Comput. Netw. Commun.* **2022**, *2022*, 7368691. [CrossRef]
59. N6705C DC Power Analyzer. Available online: https://www.keysight.com/us/en/product/N6705C/dc-power-analyzer-modular-600-w-4-slots.html (accessed on 18 October 2024).
60. N6781A Two-Quadrant SMU for Battery Drain Analysis. Available online: https://www.keysight.com/us/en/product/N6781A/2-quadrant-smu-battery-drain-analysis-20v-1a-6v-3a-20w.html (accessed on 18 October 2024).
61. Ekol Fully Automatic Front Firing Blank Gun. Available online: https://blankgunarmory.com/asi-fully-automatic-front-firing-blank-gun-9mm-matte-black/?1 (accessed on 18 October 2024).
62. Decibel Meter. Available online: https://www.amazon.com/gp/product/B09G6HLNVV/ (accessed on 18 October 2024).

63. Firebase Cloud Messaging. Available online: https://firebase.google.com/docs/cloud-messaging (accessed on 25 December 2024).
64. Landau, H.J. Sampling, data transmission, and the Nyquist rate. *Proc. IEEE* **1967**, *55*, 1701–1706. [CrossRef]
65. Hrabina, M.; Sigmund, M. Gunshot recognition using low level features in the time domain. In Proceedings of the 2018 28th International Conference Radioelektronika (RADIOELEKTRONIKA), Prague, Czech Republic, 19–20 April 2018; pp. 1–5. [CrossRef]