Faculty of Computer Science and Information Technology

Gunshot Audio Analysis for Firearm Type Detection Using Machine Learning Algorithms

KIUNG DE KAI

Bachelor in Computer Science with Honours (Information System)
2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE Gunshot Audio Analysis for Firearm Type Detection using Machine Learning Algorithm

ACADEMIC SESSION: 2024/2025 -2

KIUNG DE KAI

(CAPITAL LETTERS)

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

☐	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
☐	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
☑	UNRESTRICTED	

Validated by

(AUTHOR'S SIGNATURE)

(SUPERVISOR'S SIGNATURE)

Permanent Address

No.1474, Paris Heights 2, 96500
Bintangor, Sarawak,
Meradong, 96508, Sarawak

Date: 20TH JUNE 2025

Date: _____

Note * Thesis refers to PhD, Master, and Bachelor's Degree
** For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

Gunshot Audio Analysis for Firearm Type Detection using Machine Learning Algorithms

KIUNG DE KAI

This project is submitted in partial fulfilment of the requirements for the degree of Bachelor of Computer Science with Honours (Information System)

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

2025

Analisis Audio Tembakan untuk Pengesanan Jenis Senjata Api Menggunakan Algoritma Pembelajaran Mesin

KIUNG DE KAI

Projek ini merupakan salah satu keperluan untuk Ijazah Sarjana Muda Sains Komputer dengan Kepujian (Sistem Maklumat)

Fakulti Sains Komputer dan Teknologi Maklumat

UNIVERSITI MALAYSIA SARAWAK

2025

DECLARATION

I hereby declare that this project is my original work. I have not copied from any other student's work or from any other sources except where due reference or acknowledgement is not made explicitly in the text, nor has any part had been written for me by another person.

…….

(KIUNG DE KAI) 18 JUNE 2025

Matric No: 79790

ACKNOWLEDGEMENT

First of all, I would like to express my deep gratitude to my supervisor, Dr. Mohammad bin Hossin, who provided valuable help, guidance and advice throughout the development of this project. His expertise and support played an important role in helping me complete my work. At the same time, I would also like to thank my examiner, Associate Professor Dr. Lee Nung Kion, whose valuable comments and suggestions helped to improve the quality of my project.

I would also like to thank my final year project coordinator, Professor Dr. Wang Yin Chai, for his help, who provided me with the necessary information and guidance to complete my final year project. In addition, I would like to thank all my friends and classmates who supported me and provided me with the information I needed to complete this project.

Finally, I would like to express my special thanks to my parents and siblings for their unwavering support and encouragement. Their love and encouragement gave me the strength and motivation to complete this project. Without their help, this project would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CNN	Convolutional Neural Network
IDE	Integrated Development Environment
kNN	k-Nearest Neighbors
LSTM	Long Short-Term Memory
MFCC	Mel-frequency Cepstral Coefficients
RBF	Radial Basis Function
SNR	Signal to Noise Ratios
SUS	System Usability Scale
SVM	Support Vector Machine
UML	Unified Modeling Language
ZCR	Zero-Crossing Rate

ABSTRACT

Current gunshot detection systems primarily focus on identifying the presence of gunfire but lack the ability to classify the specific type of firearm. This thesis presents a machine learning-based system designed to classify firearm types based on gunshot audio recordings. A dataset consisting of 851 gunshot recordings across eight firearm classes was used. 108-dimensional feature vector comprising Mel-frequency cepstral coefficients (MFCCs), Chroma, Spectral Contrast, Zero-Crossing Rate (ZCR), Energy, and Spectral Bandwidth. The features were normalized and used to train three classification models: Support Vector Machine (SVM), k-Nearest Neighbors (kNN), and Convolutional Neural Network (CNN). Each model was optimized using grid search with 5-fold stratified cross-validation. Experimental results showed that the SVM achieved an accuracy of 93.26%, while both the kNN and CNN models achieved higher accuracies of 96.77%. A web-based application was developed using Django, allowing users to upload gunshot audio and select a preferred model for real-time firearm classification. Although the system performs effectively on the curated dataset, its accuracy decreases when applied to real-world audio samples, indicating the need for improved generalizability. This project demonstrates the potential of machine learning for firearm identification from audio signals and provides a functional prototype contributing to intelligent surveillance and public safety technologies.

Keywords: Audio pre-processing, firearm identification, gunshot audio classification, machine learning, public safety

ABSTRAK

Sistem pengesanan tembakan sedia ada kebanyakannya hanya mengesan kehadiran tembakan tanpa keupayaan untuk mengelaskan jenis senjata api yang digunakan. Tesis ini memperkenalkan satu sistem berasaskan pembelajaran mesin yang direka untuk mengelaskan jenis senjata api berdasarkan rakaman audio tembakan. Dataset yang digunakan mengandungi 851 rakaman tembakan merangkumi lapan kelas senjata api. Langkah pra-pemprosesan termasuk penyeragaman durasi rakaman, penambahan hingar Gaussian, dan pengekstrakan vektor ciri berdimensi 108 yang terdiri daripada pekali cepstral Mel-frequency (MFCC), Chroma, Kontras Spektrum, Kadar Persilangan Sifar (ZCR), Tenaga, dan Lebar Jalur Spektrum. Ciri-ciri ini dinormalisasi dan digunakan untuk melatih tiga model klasifikasi: Support Vector Machine (SVM), k-Nearest Neighbors (kNN), dan Convolutional Neural Network (CNN). Setiap model dioptimumkan menggunakan grid carian dengan dengan penstratifikasi validasi-silang 5-lipatan. Keputusan eksperimen menunjukkan bahawa model SVM mencapai ketepatan sebanyak 93.26%, manakala model kNN dan CNN masing-masing mencapai ketepatan yang lebih tinggi iaitu 96.77%. Aplikasi berasaskan web dibangunkan menggunakan rangka kerja Django yang membolehkan pengguna memuat naik audio tembakan dan memilih model untuk klasifikasi jenis senjata api secara masa nyata. Walaupun sistem ini menunjukkan prestasi yang baik pada dataset yang dikurasi, ketepatannya menurun apabila digunakan pada rakaman audio dari sumber sebenar, menunjukkan keperluan untuk meningkatkan generalisasi model. Projek ini menunjukkan potensi pembelajaran mesin dalam pengenalpastian senjata api melalui signal audio dan menyediakan prototaip tersedia yang berpotensi menyumbang kepada teknologi keselamatan awam dan pengawasan pintar.

Kata Kunci: *Pra-pemprosesan audio, pengenalpastian senjata api, pengelasan audio tembakan, pembelajaran mesin, keselamatan awam*

CHAPTER 1: INTRODUCTION

1.0 Introduction

The need for security with advanced features related to firearms has increased significantly in the last few years. In gunfire-related incidents, rapid identification of gunfire is crucial for public safety and law enforcement agencies to act efficiently in critical situations(Nijhawan et al., 2022). Most gunshot detection systems presently guarantee the presence of gunfire, though few classify the type of firearm(Chen et al., 2024). This can inherently reduce the capabilities of law enforcement to judge threats effectively and to make appropriate, timely decisions during response situations.

It is purported that sound-based analysis can be a bright and surefire method for identifying firearm types based on unique audio signatures. Firearms of different calibers and mechanisms produce unique acoustic patterns that can be captured and analyzed to infer the kind of weapon. This involves training models to identify these soft audio features and, using machine learning algorithms, classify the sounds into firearm categories with high accuracy. In such a way, this approach will allow one to develop a gunfire detection system that will add critical details about the weapon involved.

This project is intended to utilize machine learning for the detection of firearm types based on gunshot audio recordings. To this effect, the current project will entail the collection or access of gunshot sound data, extracting the necessary features of the audio from these recordings, and after that training a machine learning model that identifies firearm types.

Additionally, a Python-based web application will be developed allowing users to upload an audio file for classification-a convenient way to do fast firearm identification.

By creating a tool to detect and classify types of firearms based on gunshot sounds, this project aims to provide an existing gap in the current gunshot detection systems. Detection of firearms emanating from the sound, therefore, would play to the strengths of public safety by offering law enforcement and security agencies a novel means of gathering intelligence in real-time. The outcomes of this project will add to research on the classification of sounds, apart from providing insight into the practical application of machine learning for audio analysis in security domains.

1.1 Problem Statement

In many gunshot incidents, identification of the type of firearm by its unique audio signature is critical for public safety and appropriate law enforcement action(Teng et al., 2024). Most gunshot detection systems that exist identify where gunfire incidents have happened but generally fail in gunfire classification(Chen et al., 2024). This inability to classify firearms may hinder the law enforcement's ability to make appropriate threat evaluations and timely decisions in dangerous situations.

This study addresses the fact that effective firearm classification from gunshot sounds provides room for improvement in security today. That is, incomplete information might be the only thing available to law enforcers in responding to incidences with guns, which can hamper their effectiveness in action and public safety. This should warrant the requirement of the tool,

which may precisely identify firearm types with gunshot audio to help with rapid response and decision-making during critical situations.

The proposed system attempts to respond to this challenge by developing a machine learning-based system that can analyze the sounds from gunshots and classify those sounds according to the type of firearm fired. By applying advanced audio processing techniques along with machine learning algorithms, this system could become able to provide, in due time and with high precision, situational awareness information for law enforcement, leading to better public safety outcomes.

1.2 Project Scope

The scope of the project involves the development of a machine learning-based system to analyze gunshot audio and classify them according to the type of firearm used. The project focuses on the following major areas of concern:

1. Data Collection:

Collect diverse gunshot audio datasets that could represent the various types of firearms. These can include publicly available data such as dataset from Kaggle.

2. Audio Feature Extraction:

Techniques to be applied for feature extraction from the gunshot recording for further processing may include those on spectral features, temporal features, and other acoustic characteristics that best represent the signature audio of different firearms.

3. Development of the Machine Learning Model:

Develop and train machine learning models to classify the features extracted. This could mean being involved in testing various algorithms-both traditional machine learning and deep learning techniques-to determine which would work best for firearms classification.

4. Web Application Development:

Web Python-based application that can enable uploading audio files of gunshots and classify them in real time. It will have a proper interface for the user's interaction and the results to be provided in terms of firearm type detected.

5. Model Evaluation:

Evaluate the trained machine learning model using appropriate performance metrics such as accuracy, precision, recall, and F1-score. The evaluation will involve testing the model on unseen data to ensure its robustness and reliability in real-world scenarios.

6. Reporting and Documentation:

The report should be comprehensive, detailing the methodology that was used for the project, findings, and performance metrics. The documentation will shed light on how effective the system developed will be and point out areas that need future research and improvement.

1.3 Aims and Objectives

This project aims to classify different firearms using a gunshot sound with a machine learning based system. In order to achieve the aim, this project has setup three objectives. The objectives of this project are to:

Objectives:

1. build a machine learning model that can accurately classify firearm types based on gunshot audio recordings.

2. develop a Python-based web application that allows users to upload gunshot audio files for real-time firearm classification.

3. evaluate the model's accuracy.

1.4 Significance of Project

This project holds significant value for both public safety and advancements in machine learning for audio analysis. By enabling accurate firearm type detection based solely on gunshot audio, the project addresses a crucial gap in existing gunshot detection systems, which often lack the capability to classify specific firearms. The ability to identify firearm types rapidly and accurately can enhance situational awareness for law enforcement, aiding in more informed decision-making and potentially reducing response times during critical incidents. This tool can provide a novel layer of intelligence to security strategies, supporting real-time threat assessment and public safety measures.

1.5 Thesis Organization

This thesis organization consists of five chapters that are organized as follows.

In Chapter 1, Introduction, the background and context of the project are discussed. This includes the need for advanced gunshot detection systems, the importance of firearm classification for public safety, and the gap in existing technologies. The chapter also outlines the problem statement, objectives, scope, significance of the project etc. All these items will be discussed briefly to provide an overview of the project.

In Chapter 2, Literature Review, previous studies on gunshot detection, firearm classification, and audio analysis using machine learning are reviewed. Key preprocessing techniques, feature extraction methods, and classification models such as SVM, kNN, and CNN are discussed.

In Chapter 3, Methodology, the step-by-step process for developing the firearm classification system is described. This includes data pre-processing, feature extraction techniques, the development and evaluation of machine learning models, and the creation of a Python-based Web application for firearm classification.

In Chapter 4, Implementation, it will focus on the implementation of the proposed application. All the steps of implementation will be properly documented in this chapter.

In Chapter 5, Evaluation Results, the outcomes of the gunshot classification system are presented. This includes model evaluation using performance metrics such as accuracy, precision, recall, and F1-score, as well as confusion matrix and time efficiency analysis. The

chapter also covers real-world testing using external audio samples and system functionality testing through a web-based interface.

In Chapter 6, Conclusion and Future Works, the thesis is concluded by Final Year Project and presents the result of the study. Furthermore, this chapter also lays out of the project's achievements, limitations, and suggestions for future enhancements.

1.6 Project Schedule

The successful development of the project depends mainly on an accurate project schedule. Identifying project milestones is a guideline to determine whether the project can be completed on time. The project will take a total of two semesters to complete. A Gantt chart is chosen as a tool to arrange the various stages of the project schedule.

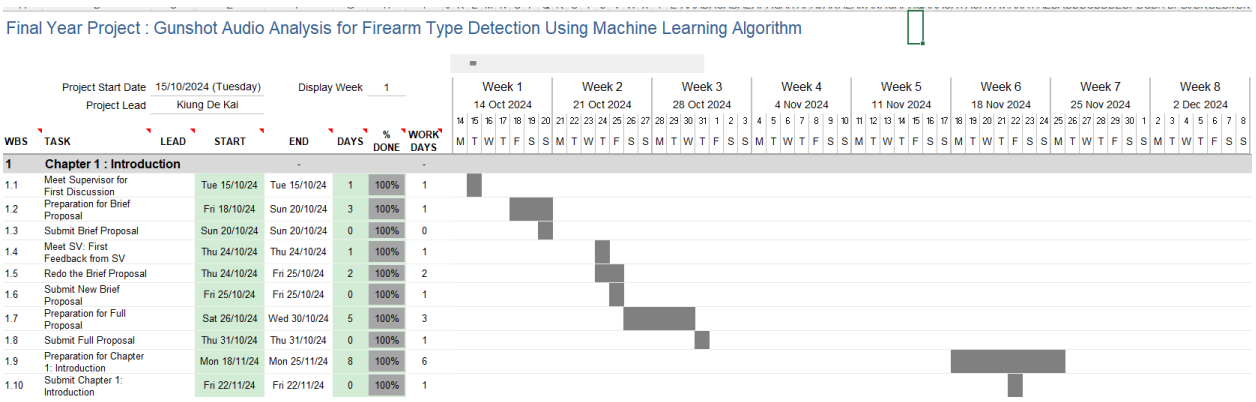

Figure 1.1: Gantt Chart for Chapter 1

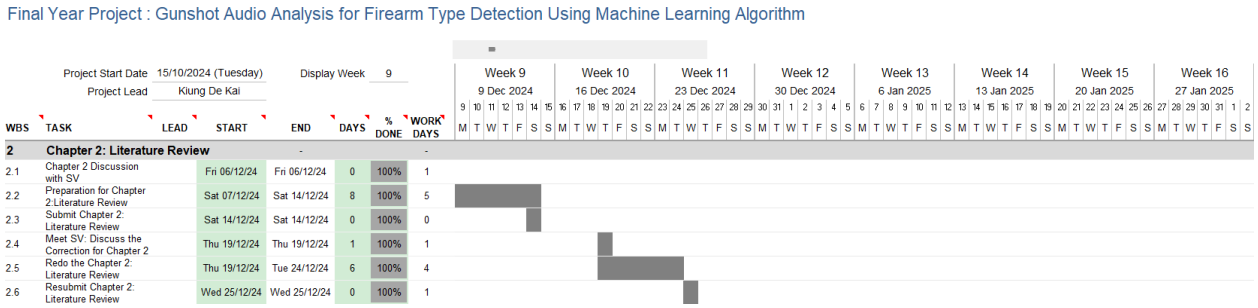

Figure 1.2: Gantt Chart for Chapter 2

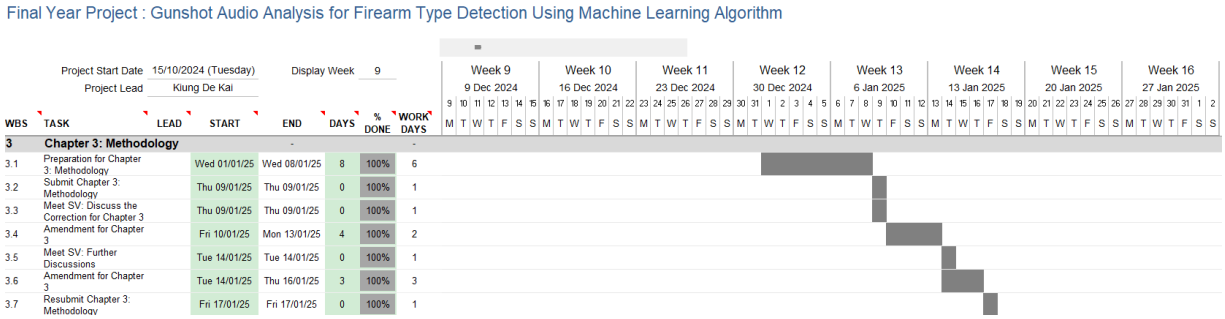

Figure 1.3: Gantt Chart for Chapter 3

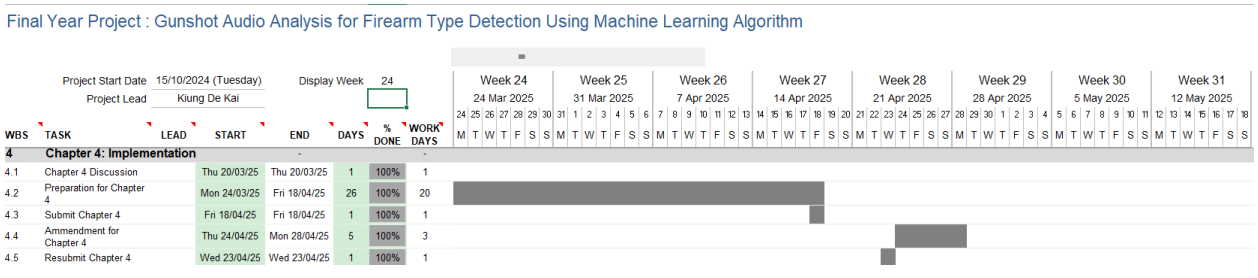

Figure 1.4: Gantt Chart for Chapter 4

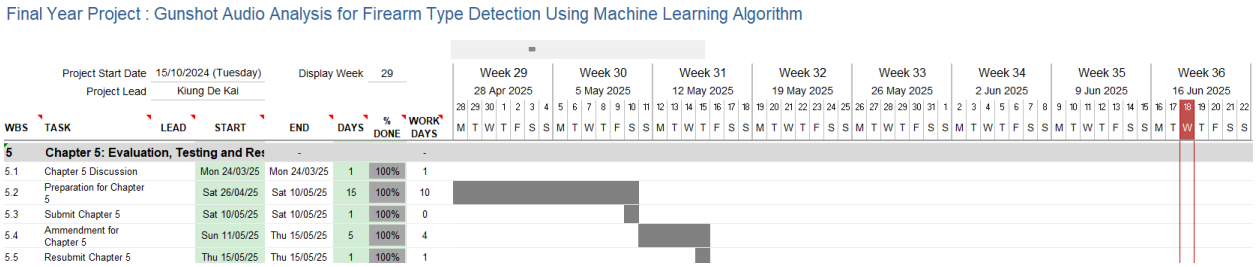

Figure 1.5: Gantt Chart for Chapter 5

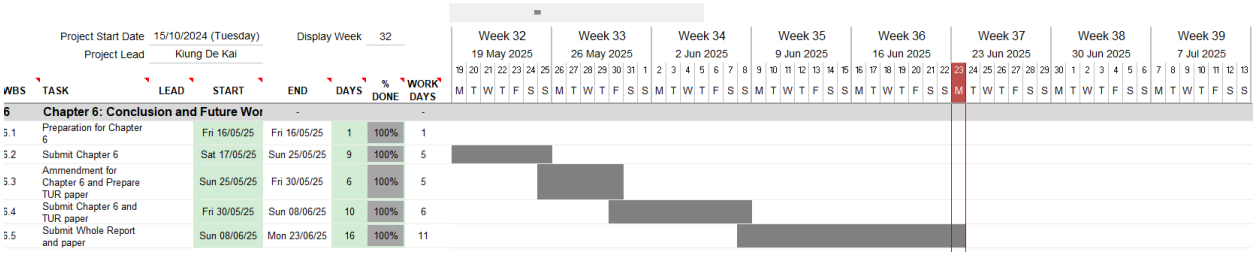

Figure 1.6: Gantt Chart for Chapter 6

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter provides a comprehensive review of the literature on gunshot type classification, emphasizing the role of machine learning in analyzing and categorizing firearm audio signals. It examines the effectiveness of supervised learning models, particularly Support Vector Machines (SVM), Convolutional Neural Networks (CNNs), and k-Nearest Neighbors (kNN), in identifying subtle acoustic patterns unique to specific firearm types. The chapter also explores the critical processes of audio preprocessing and feature extraction which enhance the accuracy of classification systems. By synthesizing insights from previous research, this chapter establishes a solid foundation for developing a robust system to classify gunshot sound types effectively.

2.2 Machine Learning in Audio Analysis

Machine learning is a subset of Artificial Intelligence (AI) that focuses on developing systems capable of learning from data to make predictions or decisions without being explicitly programmed. At its core, machine learning utilizes algorithms to identify patterns and relationships within data, enabling systems to generalize and perform tasks on unseen inputs. It can be broadly categorized into three types: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, which is most relevant to this study, models are trained on labeled datasets where inputs are paired with corresponding outputs, allowing the algorithm to learn mappings between features and targets. Examples include classification tasks, such as distinguishing between firearm types based on gunshot audio, and regression tasks, such as predicting continuous values (Nesar et al., 2024; Teng et al., 2024).

In the context of audio analysis, machine learning has proven transformative by enabling systems to handle complex acoustic patterns and large-scale datasets. Applications include speech recognition, where models identify spoken words, environmental sound classification, which involves detecting and categorizing natural or urban sounds, and music genre identification. These systems typically rely on preprocessing techniques to transform raw audio into structured features, such as Mel-Frequency Cepstral Coefficients (MFCCs) or spectrograms, which effectively summarize the frequency and temporal characteristics of the audio signal(Nesar et al., 2024).

Machine learning's ability to generalize and classify based on subtle acoustic features was critical to the project. The high-dimensional and transient nature of gunshot sounds makes manual identification challenging. Teng et al. (2024) observed that machine learning provides a scalable solution by enabling models to learn subtle acoustic patterns unique to specific firearm types through carefully curated and preprocessed datasets. These datasets often undergo feature extraction processes, such as Mel-Frequency Cepstral Coefficients (MFCCs) generation, which capture critical time-frequency information in gunshot sounds.

2.3 Audio Processing and Feature Extraction

Audio processing and feature extraction are critical components in gunshot detection and classification systems. These processes involve transforming raw audio data into meaningful representations that can be effectively analyzed by machine learning algorithms. This section reviews the techniques commonly used in preprocessing and feature extraction for gunshot audio analysis, based on the reviewed literature.

2.3.1 Techniques for preprocessing gunshot audio

Preprocessing is a critical step in preparing raw audio data for machine learning models. It aims to enhance the quality of the audio signals, facilitate effective feature extraction, and improve the accuracy of subsequent classification tasks. Preprocessing helps isolate the relevant components of gunshot audio, minimize noise and distortions, and standardize data formats, thereby increasing model robustness.

In a study by Nguyen and Nguyen (2025), a data augmentation technique was applied by introducing Gaussian noise into the original audio samples. Specifically, noise with a mean of 0 and a variance of 0.01 was added to simulate real-world noisy environments. This augmentation exposed the model to a broader range of conditions and helped prevent overfitting by diversifying the training set. Following augmentation, the expanded dataset provided a more substantial basis for training and evaluating the machine learning model, enhancing its generalization capabilities (Nguyen & Nguyen, 2025).

Additionally, after feature extraction, normalization was applied to standardize the extracted features. Standard scaling was applied to normalize the features, ensuring each had a mean of zero and a standard deviation of one. This step is critical because it eliminates scale-related biases and ensures that no single feature dominates the learning process (Nguyen & Nguyen, 2025).

Another important preprocessing technique is noise reduction, which aims to isolate gunshot sounds from environmental noise such as traffic, human voices, or animal sounds. Singh and Zhuang, (2022) emphasized the critical role of noise reduction in ensuring clear and high quality gunshot audio for classification. By leveraging robust feature extraction techniques like

Mel-Frequency Cepstral Coefficients (MFCCs) and their derivatives, they focused on isolating gunshot-specific acoustic characteristics. Effective noise reduction improves the signal-to-noise ratio and enhances the ability of machine learning models to accurately identify firearm types, particularly in noisy environments (Nguyen & Nguyen, 2025; Singh & Zhuang, 2022).

These preprocessing techniques collectively ensure that the raw gunshot audio is transformed into clean, consistent, and meaningful data suitable for machine learning model input. Together, these steps form the foundation for reliable feature extraction and accurate classification in gunshot detection systems.

2.3.2 Feature Extraction Methods

Feature extraction is a fundamental process in audio classification tasks, as it transforms raw audio signals into structured numerical representations that summarize the essential spectral and temporal characteristics of the sound. In the work of Nguyen and Nguyen (2025), a comprehensive set of audio features was employed to characterize gunshot sounds, aiming to capture a broad range of acoustic properties.

One of the primary features utilized was the Mel-Frequency Cepstral Coefficients (MFCCs), which are designed to model the human auditory system's perception of sound. Thirteen MFCC coefficients were extracted from each audio recording and summarized using three statistical measures: mean, standard deviation, and median, resulting in a total of 39 MFCC-related features per sample. MFCCs have demonstrated robustness in capturing the

spectral envelope of audio signals, making them effective for distinguishing gunshot sounds from other acoustic events, particularly in noisy environments.

In addition to MFCCs, Chroma features were extracted to represent the distribution of pitch classes over time(Urbano et al., 2014). Twelve chroma vectors were calculated and, similarly to MFCCs, summarized by mean, standard deviation, and median, contributing 36 features to the overall feature set. Chroma features are particularly useful for distinguishing harmonic content and pitch structures in audio signals.

Spectral Contrast was also included, measuring the difference between spectral peaks and valleys across seven frequency bands. This feature captures the contrast in energy levels between different parts of the spectrum and was summarized using three statistical measures for each band, yielding 21 features.

Furthermore, four additional spectral features were extracted: Spectral Centroid, Zero-Crossing Rate (ZCR), Energy, and Spectral Bandwidth. The spectral centroid indicates the center of mass of the frequency spectrum and correlates with the perceived brightness of the sound. The ZCR quantifies the rate at which the audio signal crosses the zero-amplitude axis, offering insights into the noisiness and transient properties of the signal. Energy measures the signal's strength by summing the squared amplitude values, while spectral bandwidth reflects the spread of the frequency distribution around the centroid. Each of these four features was summarized using the mean, standard deviation, and median, contributing a total of 12 features.

Collectively, the extracted features resulted in a 108-dimensional feature vector for each audio sample, comprising 39 MFCC features, 36 Chroma features, 21 Spectral Contrast features,

and 12 additional spectral features. Feature extraction was implemented using the Librosa library, an open-source Python package widely utilized for audio signal analysis (Nguyen & Nguyen, 2025).

2.4 Machine Learning Models for Gunshot Classification

Various machine learning algorithms have been applied to gunshot audio classification. This section will concentrate on three such techniques: Support Vector Machines (SVM), K-Nearest Neighbor (kNN) and Convolutional Neural Network (CNN).

2.4.1 Support Vector Machines (SVM)

Support Vector Machines (SVM) are one of the most widely used machine learning algorithms, known for their ability to handle high-dimensional data and classify complex datasets with clear boundaries. Initially developed for binary classification tasks, SVMs have been extended to support multi-class classification and are commonly applied in fields such as text categorization, image recognition, and audio classification(*Support Vector Machine (SVM) Algorithm - Javatpoint*, n.d.). SVMs operate by finding an optimal hyperplane that separates data points into distinct classes, maximizing the margin between them. This characteristic makes SVMs particularly effective in applications where distinguishing between similar but overlapping features is critical, such as in gunshot audio classification.

Support Vector Machines (SVMs) are particularly well-suited for audio classification tasks due to their robustness in handling small datasets and high-dimensional feature spaces. Gunshot audio signals often exhibit distinctive yet subtle acoustic characteristics, which make

SVMs an excellent choice for distinguishing gunshot sounds from background noise or other sound events. Vavrek et al. (2010) observed that when SVM is applied to features such as Mel-frequency cepstral coefficients (MFCC), it can effectively capture the spectral and temporal characteristics of gunshots. The study demonstrated that SVMs could achieve high classification accuracy even in noisy environments, showcasing their ability to process structured acoustic data efficiently.

Nesar et al. (2024) has utilized SVM to classify firearm types, such as pistols and rifles, using features extracted from gunshot sounds. The study employed a variety of audio features, including Mel-Frequency Cepstral Coefficients (MFCCs), spectral skewness, kurtosis, and harmonic ratio. These features effectively captured both temporal and spectral characteristics of gunshot sounds, providing a robust representation of their unique acoustic signatures.

A significant advantage of SVM, as observed by Nesar et al. (2024) is its robustness to noisy data. The study introduced Gaussian noise into the dataset to simulate real-world environments and evaluated the model's performance under varying signal-to-noise ratios (SNR). The SVM model maintained high accuracy, exceeding 90% at an SNR of 45 dB or higher. This robustness is a crucial consideration for this project, as real-world scenarios often involve noise interference that can obscure the clarity of gunshot audio signals. The ability of SVM to maintain reliable performance under such conditions underscores its suitability for this application.

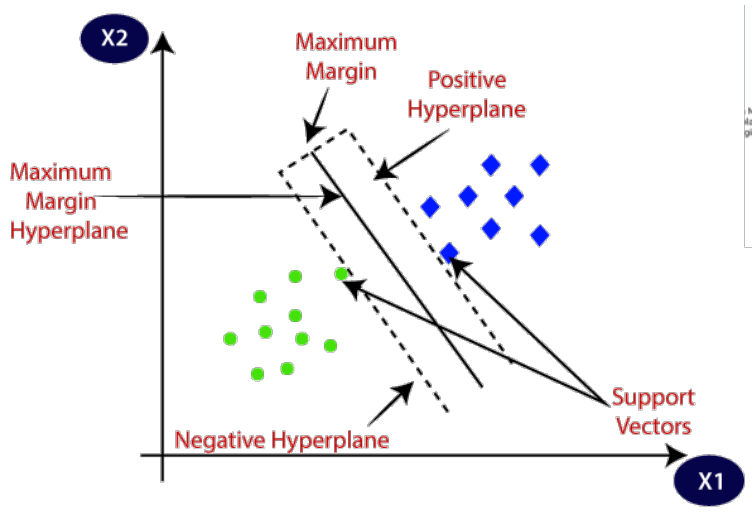

Figure 2.1: Simplified Architecture of SVM Model

The architecture of SVMs, as illustrated in Figure 2.1, is fundamental to their performance in tasks such as firearm type classification. SVM begins by extracting key features from the dataset, such as MFCCs, skewness, and harmonic ratios in this project. These features are then mapped into a higher-dimensional space using kernel functions (if the data is not linearly separable). SVM determines the optimal hyperplane that maximizes the margin between classes, with the closest data points which are usually known as supporting vectors to play a critical role in defining this boundary. The robustness of SVMs lies in their ability to handle complex datasets, ensure clear decision boundaries, and remain resilient to noise through their margin maximization approach.

In conclusion, Support Vector Machines provide a proven and interpretable approach to gunshot audio classification, as evidenced by the work of Nesar et al. (2024). Their effectiveness in feature-based classification, combined with robustness to noise and adaptability through hyperparameter tuning, makes them a reliable choice for achieving the objectives of this project.

By adopting the methodologies and findings of previous research, SVM can play a pivotal role in developing a robust system for firearm type detection based on gunshot audio signals.

2.4.2 k-Nearest Neighbor (kNN)

The k-Nearest Neighbor (kNN) algorithm is one of the simplest and most effective methods for classification tasks (*K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint*, n.d.). As a non-parametric, instance-based learning approach, it operates without making assumptions about the underlying data distribution. This characteristic makes kNN particularly suitable for applications requiring flexibility in handling complex and high-dimensional datasets. Its simplicity and effectiveness have rendered it a popular choice for various classification tasks, including gunshot audio classification for firearm type detection.

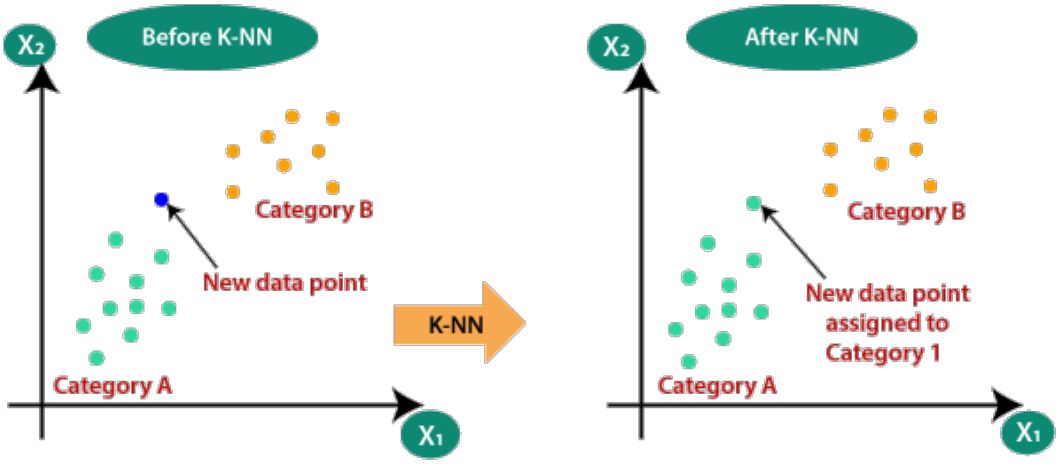

Figure 2.2: Simplified Architecture of kNN Model

Figure 2.2 provides a visual representation of the kNN architecture, demonstrating how the algorithm operates. In the "Before kNN" stage, data points are positioned within a

multidimensional feature space, with a new query point introduced for classification. The "After kNN" stage shows how the algorithm identifies the nearest neighbors based on a predefined distance metric, such as the Manhattan (City Block) distance, and assigns the query point to the class with the majority vote. This process highlights the proximity-based decision-making core of kNN, making it a suitable choice for handling complex tasks like firearm type detection based on gunshot audio.

The core principle of kNN lies in its reliance on proximity-based decision-making. For a given query instance, the algorithm identifies the "k" nearest neighbors from the training dataset based on a predefined distance metric, such as the Manhattan (City Block) distance(Tuncer et al., 2021). The class of the query instance is then determined by a majority vote among its neighbors.

For the gunshot audio analysis and firearm type detection, kNN is a strong candidate for the classification phase. Its computational simplicity ensures that it can be implemented in real-time applications, a critical requirement for enhancing public safety. Moreover, the integration of feature selection techniques, such as Iterative ReliefF (IRF), further optimizes kNN's performance by identifying the most informative features(Tuncer et al., 2021). IRF reduces the dimensionality of the dataset, improving both classification accuracy and computational efficiency strategy that aligns well with the objectives of this project.

However, the kNN algorithm is not without limitations. It is computationally intensive for large datasets, as it requires calculating the distance between the query instance and all training instances. Additionally, its sensitivity to the choice of distance metric, and feature scaling can impact performance.

2.4.3 Convolutional Neural Networks (CNNs)

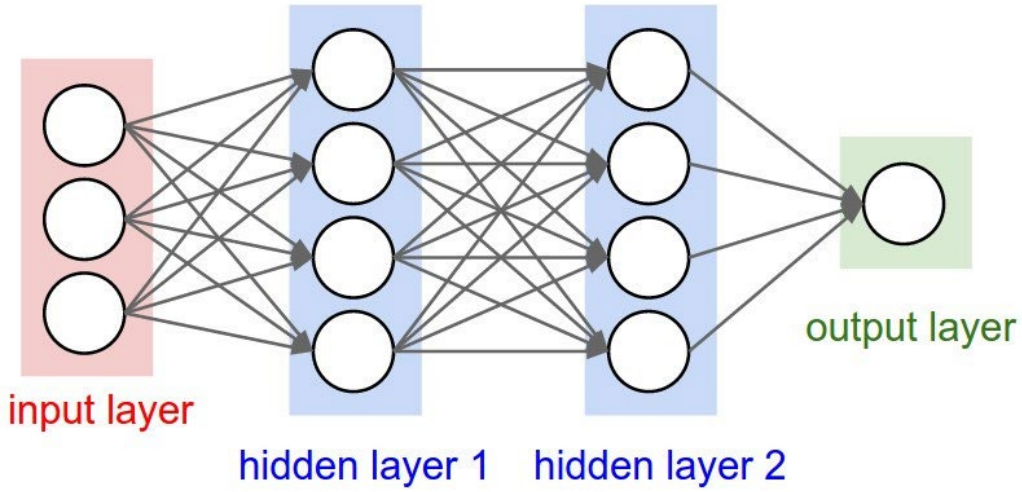

Figure 2.3: Simplified Architecture of CNN Model

Convolutional Neural Network (CNN) is a specialized type of neural network designed to process and analyze data with a grid-like structure, such as images, spectrograms, and acoustic signals. The architecture of a CNN typically comprises three main types of layers: the input layer, the hidden layers, and the output layer(*Introduction to Convolution Neural Network*, 2024). The input layer is responsible for receiving raw data, which, in the context of this project, consists of spectrograms derived from gunshot audio recordings. These spectrograms, representing the time-frequency domain of the audio signals, provide the CNN with a structured format to extract meaningful acoustic features. Figure 2.3 shows the simplified architecture of the CNN model, which demonstrates how data flows through the network to achieve classification.

Convolutional Neural Networks (CNNs) have emerged as a pivotal tool in the realm of gunshot audio analysis due to their capability to process intricate acoustic data. Shendre et al. (2024) employed a CNN model integrated with Short-Time Fourier Transform (STFT) for

feature extraction and trilateration for location prediction, showcasing CNNs' versatility in audio event classification and geolocation applications. Similarly, Irungu et al. (2023) has leveraged pre-trained CNN models using transfer learning, fine-tuning them to classify gunshots within urban sound environments. Their work underscored the effectiveness of transfer learning in optimizing CNN performance while mitigating computational demands.

The adoption of CNNs in these studies highlights several advantages, including automated feature extraction, robustness to noise, scalability, and seamless integration with complementary techniques like trilateration and attention mechanisms. However, challenges such as computational intensity and the need for extensive parameter tuning remain areas for further improvement.

In conclusion, CNNs have proven to be an indispensable asset in gunshot detection and classification, offering unparalleled accuracy and adaptability. Their application across diverse studies underscores their transformative impact on acoustic event detection and positions them as a cornerstone for advancing public safety technologies.

2.5 Comparison of Previous Research

Table 2.1 summarizes the key studies reviewed in this chapter, highlighting their methodologies and key findings relevant to gunshot audio analysis and firearm type detection using machine learning algorithms.

Table 2.1: Comparison of Previous Research

Study	Dataset Used	Methodology	Key Findings
Explosion Sound Classification Using Machine Learning Method by (Nguyen & Nguyen, 2025)	851 audio files - 8-gun types (AK-47, IMI Desert Eagle, AK-12, M16, M249, MG-42, MP5, Zastava M92) - Each file: 2s duration, 44100 Hz sampling rate - Augmented with Gaussian noise (total 1702 samples)	-Audio preprocessing (Gaussian noise addition) - Feature extraction: 7 features (MFCC, Chroma, Spectral Contrast, Spectral Centroid, ZCR, Energy, Spectral Bandwidth) - Feature vector (108 dimensions) - SVM model with RBF kernel - Grid search and 5-fold cross-validation for hyperparameter tuning (C=10, γ=0.01, one-vs-one strategy)	- Achieved 95.32% accuracy on validation set - Outperformed previous k-NN method (94.48%) - Real-world AK-47 gunfire classified with 44.12% confidence - Model prediction time: 70 ms per sample - Classified the gun type based on gunshot sound
Machine Learning Analysis on Gunshot Recognition (Nesar et al., 2024)	- Synthetic data generated using anechoic gunshot recordings from Glock-19 pistol, AR15 rifle, and 308R rifle. - 500 kHz sampling rate. - 36,300 samples total.	- Feature Extraction: 19 features including MFCCs, spectral features, skewness, kurtosis. - Models Used: AdaBoost Ensemble, SVM, Neural Networks. - Hyperparameter tuning via Bayesian Optimization. - Robustness check with Additive White Gaussian Noise (AWGN). - Feature selection via Wrapper-type Sequential Feature Selection.	- AdaBoost Ensemble achieved 99.9% accuracy. - Maintained >80% accuracy at 40 dB SNR. - Reduced feature set (9 out of 19) maintained high accuracy (~99%). - Classified the gun type based on gunshot sound
Gunshot Detection from Audio Excerpts of Urban Sounds Using Transfer Learning (Irungu et al., 2023)	UrbanSounds8K Dataset - 8,732 audio excerpts - 10 sound classes (Gunshot, Dog Barking, etc.) - Gunshot samples: 374 (~4.28%) - Duration: 0.06–4 seconds - Format: .wav files - Techniques: Random Over Sampling, SMOTE for class imbalance	- Preprocessing: Mel spectrogram generation - Transfer Learning: Pre-trained CNN models - Model: YAMNet (MobileNetV1-based) - Libraries: Librosa, Pandas, NumPy, Scikit-learn - Evaluation: Precision, Recall, F1-Score, Accuracy	- Precision (Gunshot): 100% - Sensitivity (Recall): 99% - F1-Score: 0.99 - Overall Accuracy: 99.4% - Highly effective for real-time gunshot detection in urban and military applications - Binary Classification: Gunshot (1) vs. Other Sounds (0)
An Automated Gunshot Audio Classification Method Based on Finger Pattern Feature Generator and Iterative ReliefF Feature Selector (Tuncer et al., 2021)	851 gunshot audio files from 8 different gun models (collected from YouTube; 2-second clips; 44.1 kHz sampling rate)	-Feature generation using Finger Pattern (finger-pat) descriptor, statistical moments, and Discrete Wavelet Transform (DWT). - Feature selection with Iterative ReliefF (IRF). - Classification using k-Nearest Neighbor (kNN) (Manhattan distance).	-Achieved 94.48% classification accuracy for gun model identification with robust performance (validated by 10-fold cross-validation). - Classified the gun type based on gunshot sound

2.6 Summary

This chapter reviewed key literature on the application of machine learning techniques for gunshot audio analysis and firearm type classification. It highlighted how machine learning, particularly supervised learning models such as Support Vector Machines (SVM), k-Nearest

Neighbors (kNN), and Convolutional Neural Networks (CNN), can significantly enhance the classification of complex acoustic signals. It discussed essential preprocessing steps, including noise reduction, normalization, and data augmentation, as well as key feature extraction methods, such as MFCCs and spectral contrast, which capture important spectral and temporal characteristics of gunshots. Despite these advancements, challenges remain in achieving reliable firearm classification under real-world conditions, emphasizing the need for continued research. This study builds on these findings and aims to develop a robust machine learning-based system for accurate firearm type detection from gunshot audio.

CHAPTER 3: METHODOLOGY

3.1 Introduction

This chapter discusses the methodology used for this research project. The methodology consists of five key stages: literature review, data preprocessing, machine learning model development, system development, and documentation. The system development follows the prototyping model, where iterative refinements are made based on feedback until a satisfactory product is achieved. The refined prototype is then developed into the final product, designed to meet the project's objectives for gunshot audio classification and firearm type detection.

3.2 Project Methodology

This section outlines the methodology for this research project. The methodology commences with a literature review, detailed in Chapter 2, which provides insights into the current state of gunshot audio analysis. Next, data preparation includes feature extraction processes to enhance dataset quality. Machine learning models (SVM, kNN, CNN) are then developed and evaluated using metrics like accuracy and confusion matrices. All three models are retained and integrated into the final system, allowing users to select their preferred model during inference. The system includes functionalities for audio upload, feature extraction, classification, and result display via a user-friendly interface. Finally, comprehensive documentation is prepared, covering system architecture, workflows, and user guides to ensure the usability and scalability of the solution. Figure 3.1 below illustrates the complete process of the project methodology.

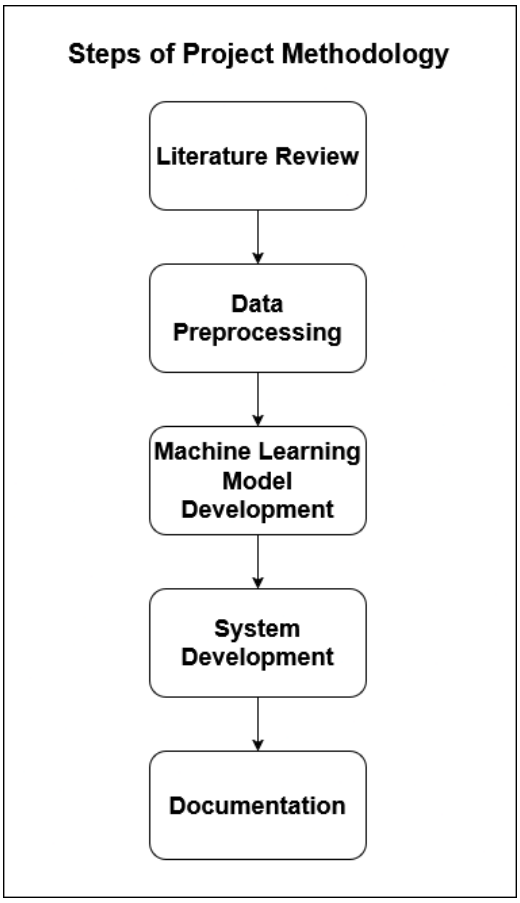

Figure 3.1: Steps of Project Methodology

3.3 Step 1: Literature Review

The literature review forms the basis for this project by exploring research on gunshot audio analysis and firearm classification. This review, presented in Chapter 2, analyzed existing studies on feature extraction methods such as Mel-Frequency Cepstral Coefficients (MFCCs), Chroma features, and Spectral Contrast, which are essential for capturing acoustic patterns. It also examined machine learning models, including Support Vector Machines (SVM), k-Nearest Neighbors (kNN), and Convolutional Neural Networks (CNN), highlighting their effectiveness in handling complex acoustic data and dataset variations. The findings from the literature review

guide the selection of methodologies and tools for this project, ensuring a robust approach to gunshot audio classification and firearm type detection.

3.4 Step 2: Data Preprocessing

The dataset used for this project was sourced from Kaggle's Gunshot Audio Dataset, a comprehensive collection of 851 audio files representing eight different firearm models(*Gunshot Audio Dataset*, n.d.). This dataset was contributed by Tuncer et al. (2021). This dataset was developed using audio recordings gathered from publicly available YouTube videos, ensuring a wide variety of environmental conditions and real-world applicability. Each audio file has a duration of two seconds, a sampling rate of 44,100 Hz, and is stored in **.wav* format for consistency. The dataset underwent thorough quality control, where audio files were carefully reviewed to ensure they were free of extraneous noise, redundant content, or repeated segments. This meticulous review process guarantees that only high-quality, relevant gunshot audio is included, providing a solid foundation for machine learning. Table 3.1 shows the distribution of the number of audio samples across the eight firearm types included in the dataset.

Table 3.1: Gunshot Dataset

No.	Gun type	Number of samples
1	AK-47	72
2	IMI Desert Eagle	100
3	AK-12	98
4	M16	200
5	M249	99
6	MG42	100
7	MP5	100
8	Zastava M92	82

The preprocessing phase is a critical step in preparing the gunshot audio dataset for effective machine learning classification. This project follows a preprocessing pipeline adapted from Nguyen and Nguyen (2025), using the same gunshot audio dataset to ensure consistency and comparability with established benchmarks. Their study represents one of the most recently published works in this domain, providing a reliable and up-to-date foundation for benchmarking and model evaluation.

First, all gunshot audio files were loaded at a fixed sampling rate of 44,100 Hz and were either truncated or zero-padded to a uniform of two second duration. This standardization ensures that each audio input has a consistent length and sampling configuration, facilitating batch processing and model training (Nguyen & Nguyen, 2025).

To enhance the robustness of the model against noise, Gaussian noise augmentation was applied to the audio samples. A zero-mean Gaussian noise with a variance of 0.01 was added to each original recording, effectively doubling the size of the dataset. This augmentation exposes the model to a broader range of audio conditions, improving generalization and mitigating overfitting to the training data (Nguyen & Nguyen, 2025).

Following augmentation, a comprehensive feature extraction process was conducted to transform raw audio signals into structured numerical representations. A 108-dimensional feature vector was extracted from each audio file, comprising:

- 39 MFCC-related features: 13 Mel-Frequency Cepstral Coefficients (MFCCs), summarized using the mean, standard deviation, and median for each coefficient.

- 36 Chroma features: 12 Chroma features, each summarized by mean, standard deviation, and median.
- 21 Spectral Contrast features: 7 spectral contrast values, summarized similarly.
- 12 additional spectral features: Spectral Centroid, Zero-Crossing Rate (ZCR), Spectral Bandwidth, and Energy, each summarized with mean, standard deviation, and median.

3.4.1 Feature Extraction Formulas

To accurately represent the audio characteristics, several key features are mathematically defined as follows:

- Mel-Frequency Cepstral Coefficients (MFCCs):

$$c_n = \sqrt{\frac{2}{K}} \sum_{k=1}^{K} (\log S_k) \cos\left(\frac{1}{K}(n(k-0.5)\pi)\right) \quad \text{Equation 3.1}$$

where:

- K is the number of bandpass filters,
- S_k is the spectral power at filter k.

- Spectral Centroid:

$$C_t = \frac{\sum_{n=1}^{N} M_t[n] \cdot n}{\sum_{n=1}^{N} M_t[n]} \quad \text{Equation 3.2}$$

where:

- *Mt [n]* is the amplitude of frequency n in the frequency spectrum corresponding to window *t*.

- Zero Crossing Rate (ZCR): ZCR of an audio signal *x* of length *N* is defined as the number of times the audio transitions from positive to negative.

$$ZCR = \frac{1}{2}\sum_{n=0}^{N-1} |sgn(x[n]) - sgn(x[n-1])| \quad \text{Equation 3.3}$$

- Energy, which reflects the signal's strength, is computed using the following formula:

$$E = \sum_{i=1}^{N} x[i]^2 \quad \text{Equation 3.4}$$

where:

- *x[i]* is the amplitude value of the *i-th* sample in the audio signal.
- *N* is the total number of samples in the signal.

- Spectral Bandwidth: The difference between high and low frequencies in a continuous band. This feature helps distinguish gunfire from other sounds based on the width of the spectrum.

$$\text{bandwidth}[t] = \left(\sum_{k} S[k,t] \cdot |\text{freq}[k,t] - \text{centroid}[t]|^{p} \right)^{1/p} \quad \text{Equation 3.5}$$

where:

- *S[k,t]* is the magnitude of the spectrogram at frequency portion k and frame *t;*
- *freq[k, t]* is the frequency value at portion k and frame *t*;
- *centroid[t*] is the spectral centroid at frame *t*;
- *p* is a parameter, usually set to 2 (Euclidean distance).

These features collectively capture critical information about the spectral and temporal characteristics of gunshot sounds, aligning closely with human auditory perception while being robust to noise and recording variations (Nguyen & Nguyen, 2025).

To ensure that all features contributed equally to the classification process, the extracted feature vectors were normalized using *StandardScaler*, resulting in zero mean and unit variance across all features. This step eliminates scale-related biases and improves the convergence of machine learning algorithms.

Additionally, gunshot class labels were encoded using a *LabelEncoder*, translating textual class names into integer-encoded representations suitable for model training and evaluation. One-hot encoding was subsequently applied to the labels for compatibility with neural network architectures that require categorical outputs.

Finally, the processed feature vectors and labels were saved in the form of *.*npy* files for efficient storage and reusability, and the trained scaler and label encoder were serialized using *joblib* for consistent preprocessing during inference.

In summary, the preprocessing phase involved data standardization, noise augmentation, comprehensive feature extraction, and normalization. This structured approach ensures that the dataset is optimized for machine learning models, laying a solid foundation for developing a robust gunshot audio classification system.

3.5 Step 3: Machine Learning Model Development

The machine learning phase of this project involves the development, hyperparameter tuning, training, and evaluation of models to classify gunshot audio into firearm types. Three primary models are utilized: Support Vector Machines (SVM), k-Nearest Neighbors (kNN), and Convolutional Neural Networks (CNN). Each model is chosen for its unique strengths in audio classification, ensuring a comprehensive approach to achieving high accuracy and robustness.

3.5.1 Model Selection

Support Vector Machines (SVM) are employed for their ability to handle high-dimensional data and distinguish between classes with clear boundaries. The k-Nearest Neighbors (kNN) algorithm is selected for its simplicity and effectiveness in classifying data based on distance metrics. Convolutional Neural Networks (CNNs) are included to explore deep learning capabilities, where the network learns hierarchical features directly from the 108-dimensional numerical inputs.

3.5.2 Data Splitting

Following the methodology proposed by Nguyen and Nguyen (2025), the preprocessed dataset is divided into 80% for training and 20% for testing using stratified sampling to ensure equal class distribution. This train-test split ensures an unbiased evaluation of model performance on unseen data.

3.5.3 Hyperparameter Tuning

To optimize the model configurations, Grid Search combined with 5-Fold Stratified Cross-Validation is applied to the training set. This tuning approach systematically evaluates multiple combinations of hyperparameters:

- **SVM**: Regularization parameter C, kernel type (*rbf, linear, poly*), kernel coefficient gamma, and decision function shape (*ovo, ovr*).
- **kNN**: Number of neighbors (*n_neighbors*), distance metric (*manhattan, euclidean, cosine*), and weighting strategy (*uniform, distance*).
- **CNN**: Optimizer (*rmsprop, adam*), dropout rate, number of convolutional layers, and number of filters.

The best hyperparameters are selected based on cross-validation accuracy. This ensures that the models generalize well without overfitting or underfitting.

3.5.4 Final Model Training and Evaluation

Once the best hyperparameters are identified, each model is retrained on the full training set using the optimized settings. The trained models are then evaluated on the 20% hold-out test

set. Evaluation metrics include accuracy, precision, recall, and F1-score, providing a comprehensive assessment of classification performance. Confusion matrices are also generated to visualize the model's ability to distinguish between different firearm classes.

This two-stage process, consisting of hyperparameter optimization and final model evaluation, closely follows the methodology of Nguyen and Nguyen (2025), ensuring a reliable and standardized framework for gunshot audio classification using machine learning.

In summary, the machine learning phase leverages SVM, kNN, and CNN models to build a robust classification system for firearm type detection. By combining traditional algorithms with deep learning techniques, the project ensures a comprehensive and effective approach to solving the classification problem.

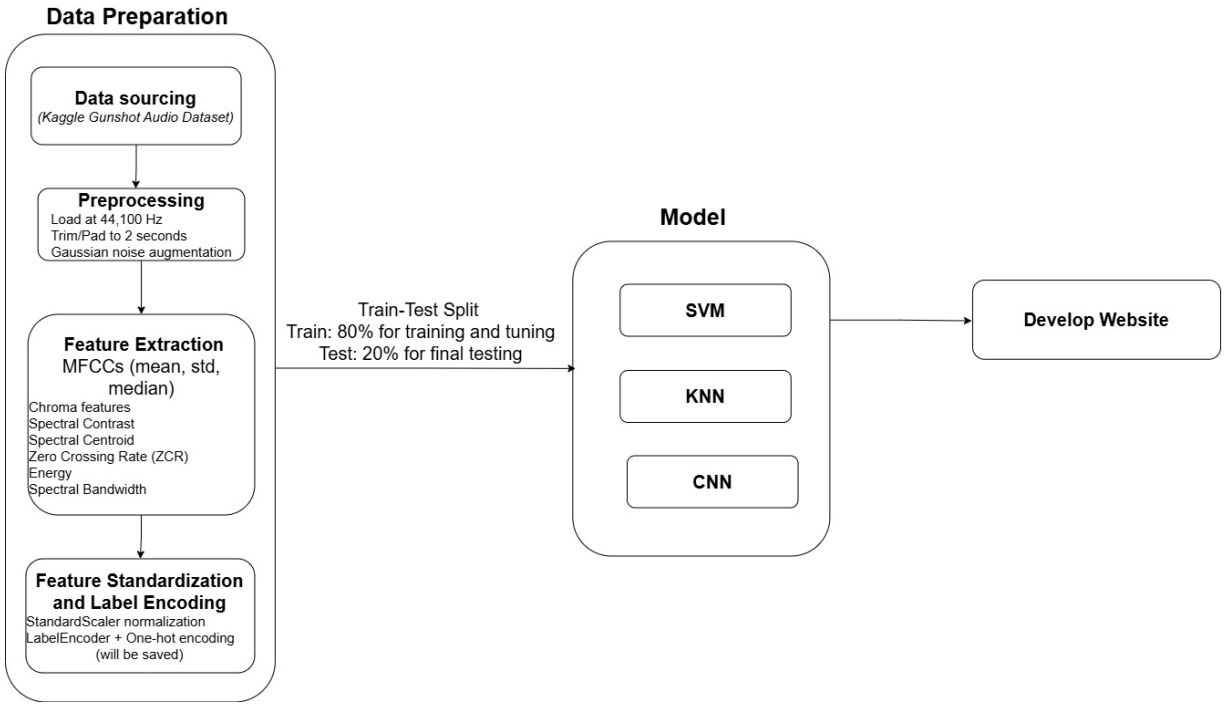

Figure 3.2: Overview of the Machine Learning Workflow for Gunshot Audio Classification

The machine learning workflow for firearm classification using gunshot audio is outlined in Figure 3.2. The process begins with data preparation, which includes sourcing gunshot audio recordings, applying preprocessing steps such as resampling and Gaussian noise augmentation, extracting relevant audio features like MFCCs, Chroma, Spectral Contrast, and others, followed by feature standardization and label encoding. Once the dataset is properly prepared, it is divided into a training set (80%) and a testing set (20%) to facilitate model development and evaluation.

To improve the performance of the models, hyperparameter tuning was carried out using Grid Search in combination with 5-Fold Cross-Validation. This tuning process was applied individually to the Support Vector Machine (SVM), k-Nearest Neighbor (kNN), and Convolutional Neural Network (CNN) models. The goal was to identify the optimal set of hyperparameters that yield the highest classification accuracy during testing. For each model, the version that achieved the best testing performance was selected and considered the final, optimized model. These best-performing models were subsequently used to power the backend of the developed web application, enabling users to classify gunshot audio files and identify firearm types in real time.

3.6 Step 4: System Development

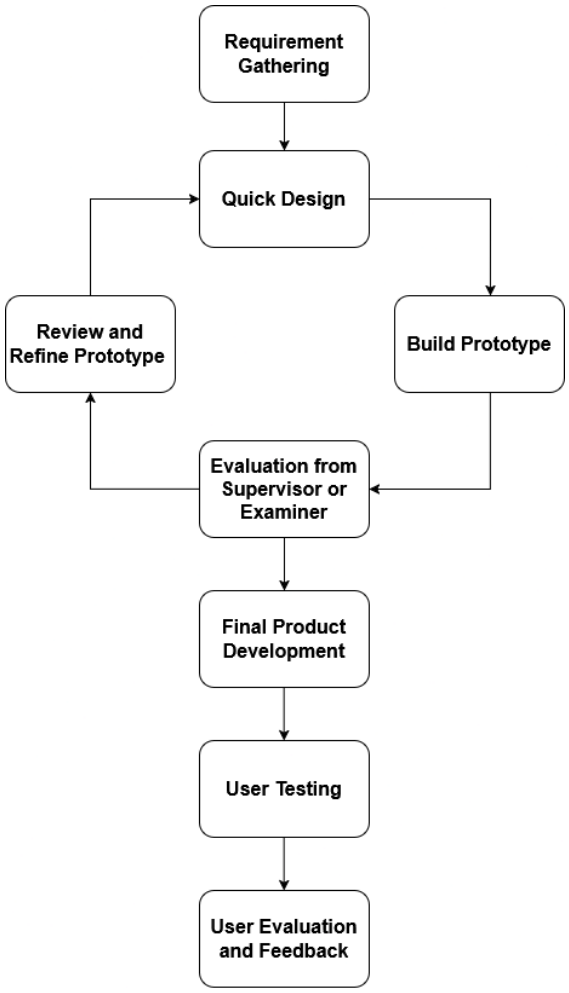

Figure 3.3: Overall Flow of the Prototyping Model

In this research, system development will adhere to the prototyping model as shown in Figure 3.3 which emphasizes iterative refinement and feedback-driven improvements to create an effective and user-centered solution(*Software Prototyping Model and Phases*, 2023). This model is well-suited for projects requiring rapid validation and adjustment, as it promotes continuous evaluation, refinement, and alignment with stakeholder expectations. By enabling

iterative development, this approach minimizes risks, ensures high-quality outcomes, and allows for adaptability to evolving requirements.

The process consists of seven key stages: Requirement Gathering, Quick Design, Build Prototype, Evaluation from Supervisor or Examiner, Review and Refine Prototype, Final Product Development, User Testing and User Evaluation and Feedback. The process begins with gathering the system's requirements, which serve as the foundation for a quick initial design. A prototype is then built and evaluated by supervisors or examiners, whose feedback guides further refinement during the Review and Refine Prototype phase. After development, User Testing is conducted to evaluate the system’s functionality, accuracy, and usability in real-world scenarios. This phase provides valuable insights into user experiences and helps identify any remaining issues or areas for improvement. Finally, User Evaluation and Feedback are gathered to assess the system's overall performance and usability, ensuring it meets practical needs and expectations. This iterative and feedback-driven approach ensures that the final system is robust, functional, and aligned with the project goals.

3.6.1 Phase 1 - Requirement Gathering

The requirement gathering phase is crucial to laying a solid foundation for the development of the gunshot audio analysis system. This phase focuses on identifying the functional, technical, and user requirements that the system must fulfill to achieve its objectives effectively. Extensive research was conducted to understand the challenges associated with firearm type detection from gunshot audio and to identify existing gaps in the field. Key insights from academic studies, user needs, and project goals were synthesized to define the system's requirements.

The primary functional requirements include the ability to upload gunshot audio files, process and analyze the audio data through advanced preprocessing techniques, extract meaningful features like MFCCs, and classify firearm types accurately using machine learning models such as SVM, kNN and CNNs.

User requirements focus on creating a web-based interface that is intuitive, user-friendly, and capable of displaying firearm classification results in a clear format. The interface should provide smooth interaction for uploading audio files and viewing results. The requirements were consolidated into a requirement specification document to guide the system design and development phases, ensuring alignment with user expectations and technical objectives.

3.6.2 Phase 2 - Prototyping

After the initial requirements have been examined and documented, a web-based application prototype will be designed and developed. The prototype design will focus primarily on the functionality and user-friendliness of the application, ensuring that the application can be run outside of the development environment. The requirements and design of the prototype will be reiterated based on the assessment requirements provided by the supervisor or examiner. This phase will end once the majority of the requirements for the application have been implemented, and critical bugs have been resolved to ensure stable and consistent performance.

3.6.2.1 Quick Design

The quick design phase establishes the foundation for the development of the gunshot audio analysis system by focusing on core functionality, system architecture, and initial

validation of components. The design incorporates essential elements, including an audio preprocessing pipeline with data augmentation, and feature extraction techniques such as Mel-Frequency Cepstral Coefficients (MFCCs). These features are critical for accurate firearm type classification using machine learning models such as SVM, kNN, and CNN. Preliminary implementations of these models are included to ensure the system achieves basic functionality.

The design is visually represented using UML (Unified Modeling Language) diagrams created with draw.io, which include the Use Case Diagram, Class Diagram, Sequence Diagram, and State Diagram. These diagrams outline the system's functionality, structure, workflow, and states, serving as a blueprint for implementation. They capture essential components and their interactions, ensuring clarity in design and alignment with the project's objectives and requirements.

Additionally, the design prioritizes the development of a web-based interface that enables users to upload audio files, process them, and view firearm classification results. Iterative refinements, guided by supervisor feedback, ensure the design remains aligned with the project's requirements while setting a solid foundation for prototype development and testing.

3.6.2.1.1 Use Case Diagram

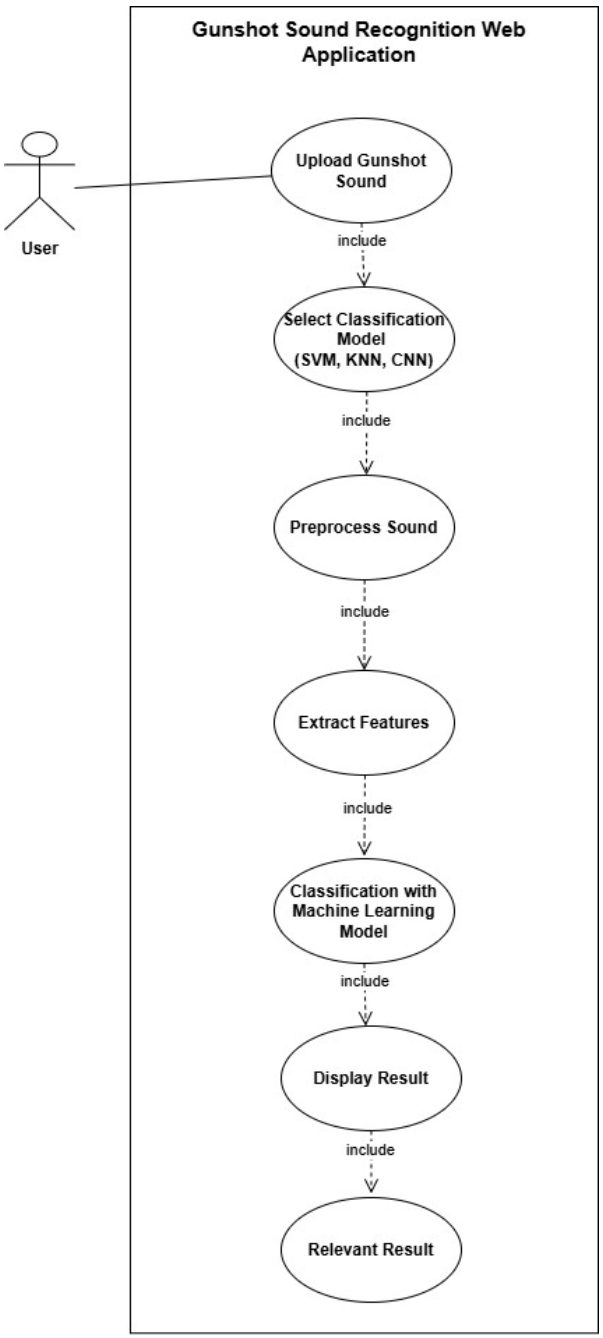

Figure 3.4: Use Case Diagram

Figure 3.4 shows the use case diagram for the Gunshot Sound Recognition Web Application. It illustrates how the actor (user) interacts with the system to perform the main functions. The process begins when the user uploads a gunshot sound file. The user then selects a classification model from the options provided (SVM, KNN, or CNN).

Once the model is selected, the system processes the audio file, extracts the relevant features, and performs classification using the chosen machine learning model. Finally, the system displays the result, including information relevant to the classification outcome.

3.6.2.1.2 Class Diagram

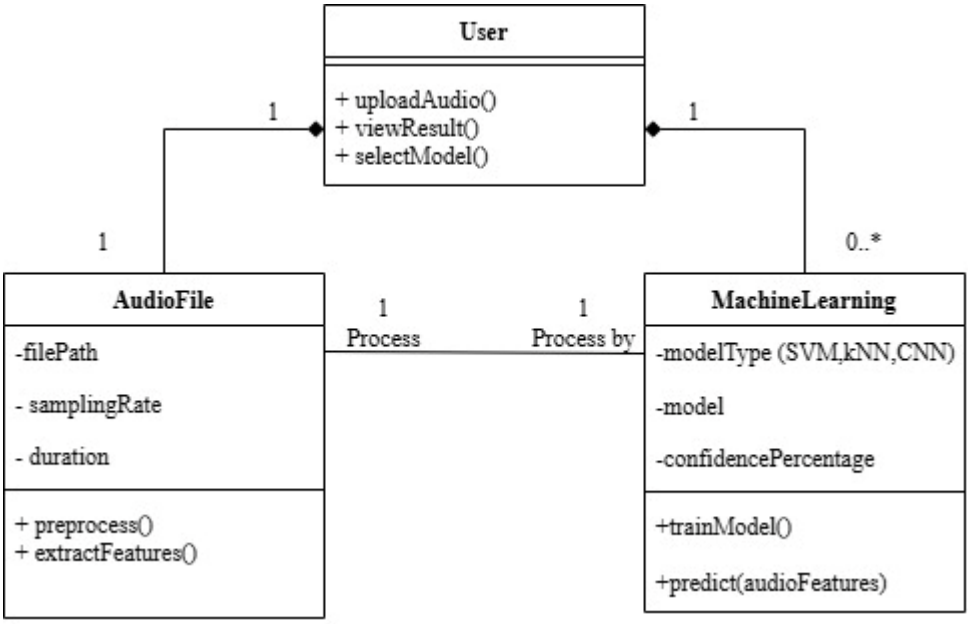

Figure 3.5: Class Diagram

The class diagram in Figure 3.5 illustrates the static structure of the Gunshot Sound Recognition Web Application by depicting the system's key components, their attributes and operations, and the relationships between them. This diagram offers a conceptual overview of

how the different parts of the system interact to support the core functionalities of uploading, processing, and classifying gunshot audio recordings.

The User class represents the individual who interacts with the application through a web interface. In practice, the user uploads a single 2-second gunshot recording and selects one of the available machine learning models (Support Vector Machine (SVM), k-Nearest Neighbors (kNN), or Convolutional Neural Network (CNN)) to perform classification. *viewResult()* to observe the output of the classification process. These operations reflect the user's role in initiating and completing the recognition process.

The *AudioFile* class manages the uploaded gunshot audio. Its attributes include *filePath*, *samplingRate*, and duration, which describe essential metadata about the recording. This class is responsible for two critical processes: *preprocess()*, which prepares the audio file by normalizing, and *extractFeatures()*, which generates relevant numerical features such as Mel-Frequency Cepstral Coefficients (MFCCs) and spectral values that are required for input into the machine learning models. Since the system is designed to handle only one audio file per upload session, the user interacts with exactly one instance of *AudioFile* at a time.

The *MachineLearning* class is responsible for classifying the extracted features using the user-selected model. It contains the *modelType* attribute to indicate the algorithm being used and the model attribute, which holds the trained machine learning classifier. The class also includes *confidencePercentage*, which stores the classification confidence level as a percentage value. Two main operations are defined for this class: *trainModel()* (used during system development) and *predict(audioFeatures)*, which performs classification on the processed input. real-time user interaction, prediction is triggered immediately after feature extraction.

The relationships among the classes are straightforward. A user interacts with one audio file per session and selects a classification model to initiate processing. The *AudioFile* class processes the uploaded file and passes its extracted features to the *MachineLearning* class. The *MachineLearning* class then performs classification and generates a prediction along with the confidence percentage. Notably, the output is rendered directly on the result page without being stored in any database or persistent storage.

Although the actual system is implemented using a function-based approach in Django and does not define these classes explicitly in the code, this class diagram serves as a conceptual model. It is intended to provide a clearer understanding of how the system components could be logically organized using object-oriented principles. This abstraction enhances the clarity of system design and aligns with best practices in modeling software architecture.

3.6.2.1.3 Sequence Diagram

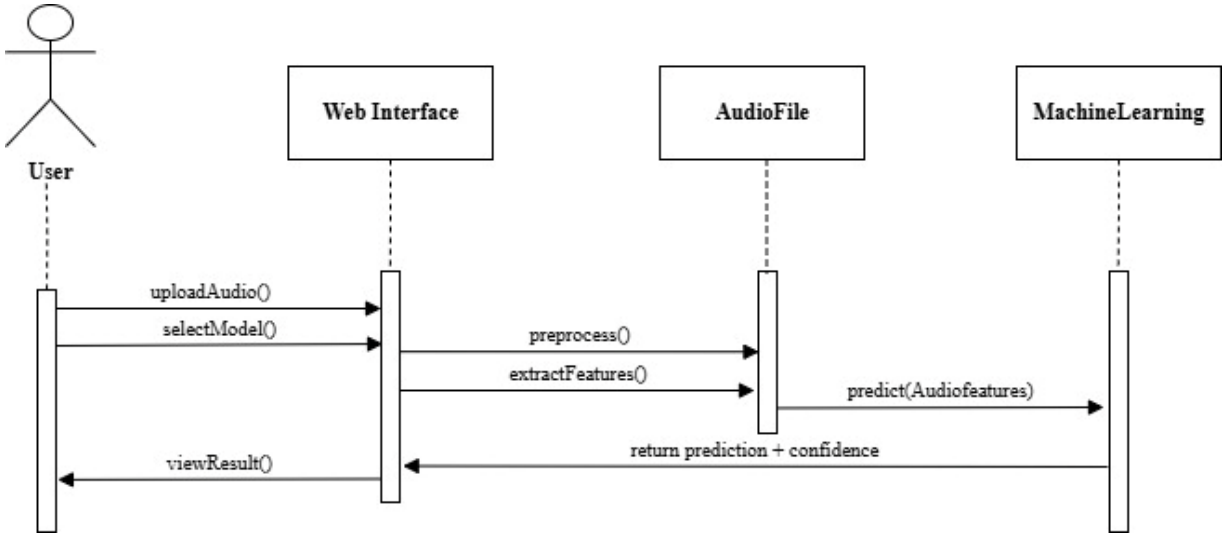

Figure 3.6: Sequence Diagram

A sequence diagram is used to illustrate the interactions between objects or components in a system and the sequence of data transfer between them. It is designed to model the dynamic behavior of an application and show how the components work together in a specific use case or process.

Figure 3.6 illustrates the sequence diagram for the Gunshot Audio Classification System, depicting the dynamic behavior and interaction between the system's key components during the process of audio upload, feature extraction, classification, and result rendering.

This diagram represents how the *User*, *Web Interface*, *AudioFile*, and *MachineLearning* components collaborate to process an audio file and return a classification result. The sequence begins when the user interacts with the web interface by uploading a gunshot audio file (*uploadAudio())* and selecting a preferred machine learning model (*selectModel()*), such as SVM, kNN, or CNN.

Once the audio and model selection are received by the *Web Interface*, it initiates audio processing by calling *preprocess()* on the *AudioFile* component. This step includes normalizing and preparing the audio signal for feature extraction. Following preprocessing, the *extractFeatures()* method is invoked to compute meaningful features from the audio, such as MFCCs (Mel-Frequency Cepstral Coefficients), which are essential for classification.

The extracted features are then passed to the *MachineLearning* component using the *predict(Audiofeatures)* call. Depending on the user's model choice, the appropriate trained classifier (SVM, kNN, or CNN) is applied to perform inference on the input features. The model

returns the predicted gunshot class along with a confidence score. This result is sent back through the *Web Interface* as *return prediction + confidence*.

Finally, the *Web Interface* renders the classification result back to the user using the *viewResult()* action, completing the interaction. The user is presented with the predicted gunshot type, the associated confidence percentage, and a representative image, all within the result page.

3.6.2.1.4 State Diagram

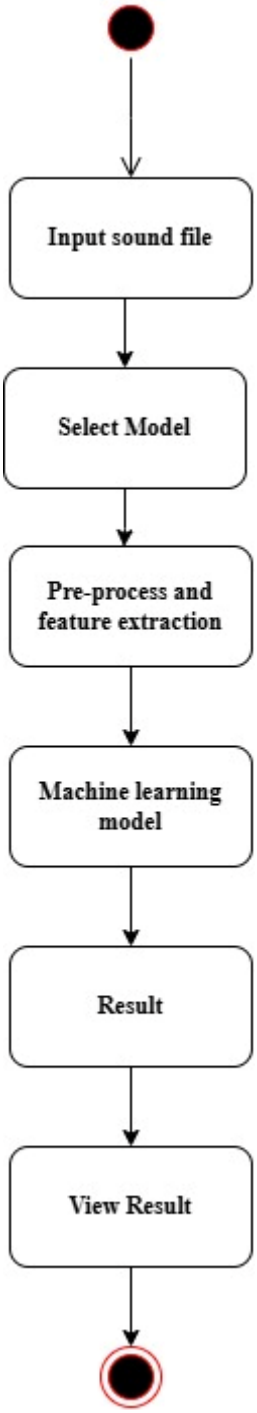

Figure 3.7: State Diagram

A state diagram is a representation of the various states and transitions that occur within an application, illustrating its dynamic behavior and how it responds to specific events or actions. State diagrams help model the progression of a process by outlining the sequence of states and the triggers that cause transitions between them.

Figure 3.7 presents the state diagram of the Gunshot Audio Classification System, which illustrates the different states the system passes through from the beginning to the end of the audio classification process. This diagram represents the sequential flow of control as the system processes an uploaded audio file and delivers a classification result to the user.

The process begins with the "Input sound file" state, where the user uploads a gunshot audio recording via the web interface. Once the file is received, the system moves to the "Select Model" state, in which the user chooses a machine learning model to use for classification such as SVM, kNN, or CNN. This selection determines how the system will interpret the extracted features from the audio input.

Next, the system enters the "Pre-process and feature extraction" state. At this point, the uploaded audio is normalized and transformed into numerical features, such as Mel-Frequency Cepstral Coefficients (MFCCs), which are essential for the classification task. The extracted features are then passed into the selected model in the "Machine learning model" state, where the prediction is computed.

Following this, the system transitions to the "Result" state, where the classification output and its associated confidence level are generated. Finally, in the "View Result" state, the

predicted gun type and confidence percentage are rendered and displayed to the user along with a representative image.

This state diagram effectively models the high-level behavior and lifecycle of the system, showing the logical progression through each operational stage.

3.6.2.1.5 User Interface

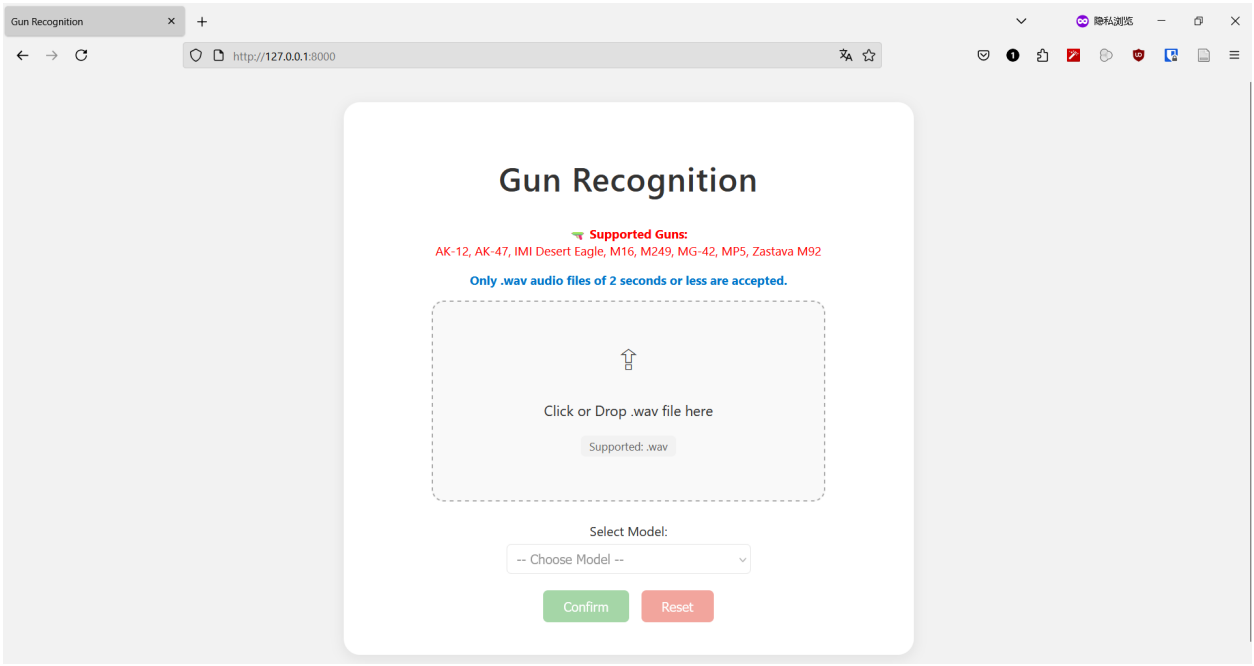

Figure 3.8: Main Page for File Upload and Model Selection

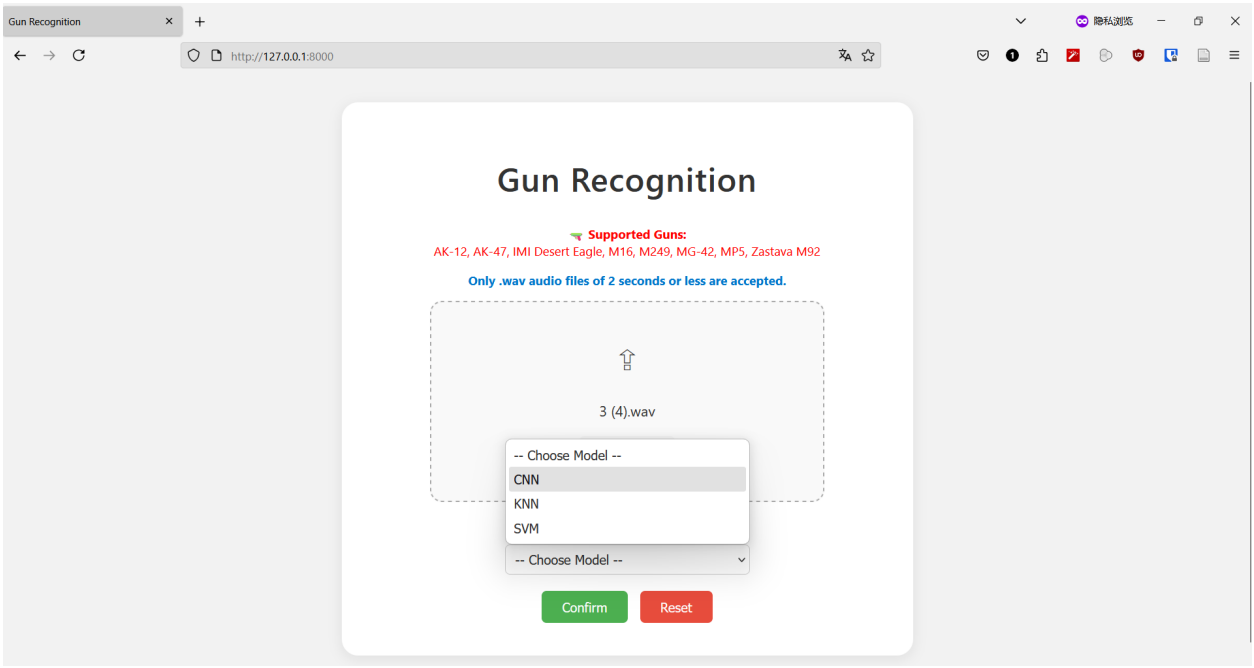

Figure 3.9: Model Selection and Uploaded File Preview

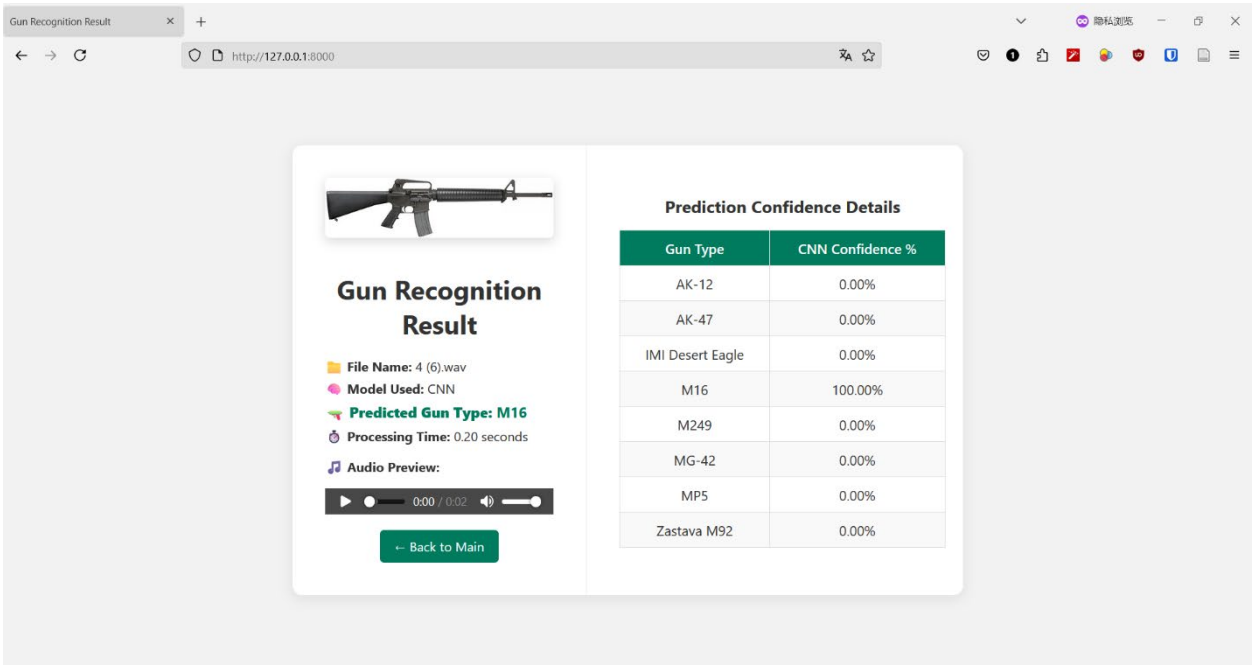

Figure 3.10: Classification Result Display

Before proceeding to the implementation phase, it is crucial to carefully design the user interface (UI) to ensure that the system is both user-friendly and intuitive. The design shown in Figure 3.8, 3.9, and 3.10 visually represents the structure and functionality of the proposed Gunshot Audio Classification System. These UI mockups provide developers and users with a clear preview of the final product. This preparatory step is vital for increasing the project's success rate by ensuring the system meets user requirements and is readily adopted by students, researchers, and public safety personnel.

The user interface of the proposed web-based Gunshot Audio Classification System is designed with simplicity, clarity, and usability in mind. A key feature of the system is the "Audio Upload and Result Display" workflow, which allows users to seamlessly upload a gunshot audio file and view the firearm classification result.

The main functional areas of the interface include:

1. Audio Upload Section:
    - Users are provided with a drag-and-drop zone to upload *.*wav* audio files or select files via a standard file dialog. Clear instructions and supported file types are displayed to guide users. A dropdown menu enables users to select their preferred classification model (CNN, KNN, or SVM) before submission. This flexible interaction design caters to a wide range of users, enhancing accessibility and ease of use.
2. Result Display Section:
    - Upon audio submission, the system processes the file and displays a detailed classification result. This includes the file name, model used for classification,

the predicted gun type (e.g., "M16"), and the processing time. Additionally, a confidence table shows the classification probabilities for each firearm type, allowing users to assess the certainty of the prediction. An audio player embedded in the interface allows users to preview the uploaded gunshot sound.

By designing the interface in Figma and emphasizing user-centered features, the proposed system ensures smooth, efficient, and accessible user experience. The interface is especially beneficial for public safety agencies, academic researchers, and students who require quick and accurate firearm classification from gunshot audio data.

3.6.2.2 Build Prototype

The application prototype will be developed iteratively after each quick design phase, with each version building upon the strengths and proven elements of its predecessors. Each subsequent prototype will incorporate refinement and additional features to achieve a more complete and functional system. Once a prototype iteration is developed, it will be submitted for evaluation by supervisors or examiners. If further improvements are needed, the cycle of designing, building, and testing will be repeated until the prototype reaches a satisfactory level of completion.

3.6.2.3 Evaluation from Supervisors/Examiners

During this phase, the prototype will be reviewed by supervisors or examiners to ensure it meets the project's requirements and performance standards. The evaluation will focus on aspects such as the system's ability to classify firearm sounds accurately, the effectiveness of its

audio preprocessing pipeline, and the usability of the web-based interface. Structured feedback will be collected through testing sessions and discussions to identify areas requiring further improvement. This feedback will guide the iterative development process, ensuring the prototype evolves toward meeting the project's goals.

3.6.2.4 Review and Refine Prototype

The review and refine phase focus on improving the prototype based on feedback from supervisors or examiners. Key improvements include making the machine learning models more accurate, enhancing the audio preprocessing steps, and fixing any bugs or issues. The user interface is also updated to make it easier to use and more intuitive, with improvements to features like file uploads, result displays, and system responsiveness.

This process is repeated until the prototype works reliably and meets all the project goals. Each version builds on the previous one, adding successful features and fixing problems. Once all the refinements are complete, the prototype will be ready to move forward as a fully functional web application.

3.6.3 Phase 3 - Final Product Development

The final product development stage involves transforming the refined prototype into a fully operational and deployable system. At this stage, all system components, including audio preprocessing, feature extraction, and machine learning models, will be finalized and subjected to comprehensive testing to ensure seamless integration and functionality. Additional enhancements may focus on optimizing the system for scalability and ensuring compatibility

across various operating systems and devices. Furthermore, the user interface will be refined to deliver a polished and professional user experience.

The completed system will be tested on various hardware configurations to confirm its reliability and robustness. Following this, the system and its source code will be uploaded to a GitHub repository. Users will be provided with the option of accessing the system: downloading a pre-packaged version from the source code. This phase ensures that the project delivers a reliable and scalable solution for gunshot audio analysis and firearm type detection, meeting all project objectives and user expectations.

3.6.4 Phase 4 - User Testing

The user testing phase focuses on evaluating the final product by allowing real users to interact with the system and provide feedback on its performance and usability. During this phase, users will upload gunshot audio files, test the system's classification capabilities, and assess the overall user experience. The goal is to identify any remaining issues, such as inaccuracies in firearm classification, inefficiencies in system responsiveness, or challenges with the interface. Feedback gathered from user testing will be documented systematically and analyzed to guide further improvements, ensuring the system meets user expectations and functions reliably in real-world scenarios.

3.6.5 Phase 5 - User Evaluation and Feedback

Following the completion of the system development and internal testing phases, user evaluation was carried out to assess the overall usability and effectiveness of the gunshot

classification system. This phase aimed to collect structured feedback from users regarding the system's interface, functionality, and user experience.

To achieve this, participants were invited to complete a Google Form containing the standardized System Usability Scale (SUS) questionnaire. SUS was originally created in 1986 by John Brooke to assess the usability of computing systems(*What Every Client Should Know about SUS Scores | Bentley University*, n.d.). It has since become one of the most widely adopted tools for measuring usability across a range of digital and physical systems.

The SUS consists of 10 fixed statements, alternating between positive and negative phrasing, each rated on a 5-point Likert scale ranging from "Strongly Disagree" (1) to "Strongly Agree" (5). This design minimizes response bias and provides a standardized method for capturing users' perceptions of usability(*What Every Client Should Know about SUS Scores | Bentley University*, n.d.).

SUS scoring involves converting user responses: scores for odd-numbered items are adjusted by subtracting 1, while scores for even-numbered items are subtracted from 5. The sum of all adjusted scores is then multiplied by 2.5 to scale the result to a maximum of 100. Individual participant scores were calculated and averaged to produce a single SUS score for the system(*What Every Client Should Know about SUS Scores | Bentley University*, n.d.).

Interpretation of the SUS score is based on established usability benchmarks. A score of 68 represents average usability, while scores above 80 indicate excellent usability. This evaluation provides a clear, quantifiable indication of how users perceive the system's ease of use compared to industry standards.

The SUS-based evaluation ensures that the system's usability is assessed using a reliable, objective, and globally recognized standard. The results from this evaluation will guide any future improvements to enhance the system's user satisfaction and effectiveness. The SUS questionnaire used in this evaluation is shown in Figure 3.11 and provided in full in Appendix A.

		Strong disagree 1	2	3	4	Strongly agree 5
1.	I think that I would like to use this gunshot classification system frequently.					
2.	I found the gunshot classification system unnecessarily complex.					
3.	I thought the gunshot classification system was easy to use.					
4.	I think that I would need the support of a technical person to be able to use this system.					

Figure 3.11: Questionnaire

3.7 Step 5: Documentation

Documentation is an essential component of the system development process, providing a comprehensive record of the project to support usability, maintenance, and scalability. For this project, detailed documentation will be created to cover all aspects of the system, including its architecture, workflows, and technical specifications. User manuals will provide clear instructions on how to interact with the system, such as uploading audio files and interpreting firearm classification results. Technical guides will outline the implementation details of the audio preprocessing pipeline, feature extraction techniques, and machine learning models, ensuring that future developers can maintain or enhance the system as needed. Additionally, the documentation will include performance reports summarizing testing outcomes, evaluation metrics, and refinements made during development. Maintenance guidelines will also be provided to support system updates or scalability in response to evolving requirements. This thorough documentation will serve as a valuable resource for users, stakeholders, and developers, ensuring the longevity and adaptability of the system.

3.8 Development and Testing Environment

This section outlines the hardware and software environment used during the development, training, and testing of the Web-based Gunshot Audio Recognition System.

Table 3.2: Hardware Specifications

No.	Hardware	Specifications
1.	CPU	Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
2.	RAM	16 GB
3.	GPU	Intel(R) Iris(R) Xe Graphics (Integrated GPU)
4.	Operating Systems	Windows 11 24H2
5.	Cloud Computing	Google Colab (used only for CNN hyperparameter tuning)

Google Colab was used exclusively for hyperparameter tuning of the Convolutional Neural Network (CNN) model, leveraging its GPU capabilities to speed up the tuning process. After identifying the best configuration, the final CNN model was retrained locally on the system's CPU. All other machine learning tasks, including data preprocessing, SVM and kNN model training, and web application development, were performed entirely on the local machine.

Table 3.3: Software Specifications

No.	Software	Requirement
1.	IDE	Microsoft Visual Studio Code
2.	Front End	HTML, CSS, integrated with Django Framework
3.	Machine Learning Language	Python
4.	Python Libraries	NumPy, Pandas, Scikit-learn, TensorFlow, Librosa, etc.
5.	Web Browser	Mozilla Firefox

This environment supports the full pipeline of machine learning model development, audio preprocessing, web application integration, and system testing.

3.9 Summary

This chapter discusses the comprehensive methodology employed to develop a firearm classification system based on gunshot audio analysis. It begins with data preparation, where audio recordings are preprocessed and converted into structured 108-dimensional feature vectors through techniques like MFCC, Chroma, and ZCR extraction. Machine learning models including Support Vector Machine (SVM), K-Nearest Neighbor (kNN), and Convolutional Neural Network (CNN) were selected for model development due to their proven effectiveness in high-dimensional classification tasks. Hyperparameter tuning was conducted using *GridSearchCV* to optimize each model's performance. The system development followed a

prototyping approach, allowing iterative improvements to the web-based application, which supports audio upload, model selection, and real-time classification. Model evaluation metrics such as accuracy and confusion matrix were used to validate performance, ensuring the system's robustness and reliability.

CHAPTER 4: IMPLEMENTATION

4.1 Introduction

This chapter explains the implementation of the Web-based Gunshot Audio Recognition System using machine learning techniques. The system allows users to upload gunshot audio files through a web interface, select a classification model, and receive the predicted firearm type. Implementation was conducted using the hardware and software environment listed in Table 3.2 and Table 3.3. The system integrates three machine learning models: Support Vector Machine (SVM), k-Nearest Neighbors (kNN), and Convolutional Neural Network (CNN). Hyperparameter tuning was performed using Grid Search with 5-Fold Cross-Validation to optimize model performance. The implementation process includes data preprocessing, model training, system integration, and user interaction, all of which are discussed in this chapter.

4.2 Machine Learning Implementation

This section details the development, tuning, and training of the machine learning models used for firearm classification, including preprocessing, feature extraction, and model optimization.

4.2.1 Pre-processing and Feature Extraction (preprocess.py)

In this project, audio pre-processing and feature extraction are key steps to prepare the gunshot audio data for machine learning classification. These processes are handled systematically in the preprocess.py script.

The pre-processing phase begins with loading the raw *.*wav* audio files using the *librosa* library at a standardized sampling rate of 44.1kHz. Each audio recording is either padded or truncated to a fixed duration of two seconds using the *pad_to_duration()* function to ensure consistent input length across all samples. To improve model robustness, Gaussian noise is added to the audio signals as an augmentation technique via the *add_gaussian_noise()* function. Each audio recording generates both original and Gaussian-noised variants, effectively doubling the size of the training dataset and improving generalization. This augmentation helps the model generalize better and reduces overfitting by simulating variations in real-world conditions, as shown in Figure 4.1.

```
# ---------------------- AUDIO PROCESSING ----------------------
def pad_to_duration(audio, sr, duration):
    """Pad or truncate the audio to match the specified duration."""
    target_len = int(sr * duration)
    if len(audio) < target_len:
        return np.pad(audio, (0, target_len - len(audio)))
    return audio[:target_len]

def add_gaussian_noise(audio, mean=0, var=0.01):
    """Add Gaussian noise to the audio signal using a fresh random generator each time."""
    rng = np.random.default_rng()  # Independent generator; not affected by np.random.seed()
    noise = rng.normal(mean, np.sqrt(var), len(audio))
    return audio + noise
```

Figure 4.1: Preprocessing Steps

Following pre-processing, feature extraction is performed using the *extract_features()* function, which generates a 108-dimensional feature vector for each audio file. These features include statistical summaries (mean, standard deviation, and median) computed from:

- **Mel-Frequency Cepstral Coefficients (MFCC)** - 13 coefficients capturing the timbral texture. (13 coefficients × 3 = 39)

- **Chroma Features** - reflecting the distribution of pitch classes. (12 semitones × 3= 36)
- **Spectral Contrast** - representing the difference between peaks and valleys of the spectrum. (7 bands × 3 = 21)
- **Spectral Centroid** - indicating the center of mass of the spectrum. (1 × 3 = 3)
- **Zero Crossing Rate (ZCR)** - measuring signal noisiness. (1 × 3 = 3)
- **Spectral Bandwidth** - reflecting frequency spread. (1 × 3 = 3)
- **Energy** - capturing the signal's intensity. (1 × 3 = 3)

*** Total: 39 + 36 + 21 + 9 + 3 = 108** (3 refer to combining statistical measures (Mean, Standard Deviation, Median))

Each of these features is summarized across time to create a compact and informative representation of the audio signal, as shown in Figure 4.2.

```python
# ---------------------- FEATURE EXTRACTION (108D) ----------------------
def extract_features(audio, sr):
    """
    Extract a 108-dimensional feature vector using:
    - MFCC (13 x 3)
    - Chroma (12 x 3)
    - Spectral contrast (7 x 3)
    - Spectral centroid, ZCR, bandwidth, energy (each x 3)
    """
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
    chroma = librosa.feature.chroma_stft(y=audio, sr=sr)
    contrast = librosa.feature.spectral_contrast(y=audio, sr=sr)
    centroid = librosa.feature.spectral_centroid(y=audio, sr=sr)
    zcr = librosa.feature.zero_crossing_rate(y=audio)
    energy = np.array([np.sum(audio ** 2)])
    bandwidth = librosa.feature.spectral_bandwidth(y=audio, sr=sr)

    def summarize(feature):
        """Compute mean, std, median for a given feature array."""
        return [np.mean(feature), np.std(feature), np.median(feature)]

    features = []
    for f in [mfcc, chroma, contrast]:
        for row in f:
            features.extend(summarize(row))

    for f in [centroid, zcr, bandwidth]:
        features.extend(summarize(f[0]))

    features.extend(summarize(energy))  # energy is 1D array

    return np.array(features[:108])  # ensure it's 108D
```

Figure 4.2: Feature Extraction

Once features are extracted, they are standardized using *StandardScaler* to normalize the feature range, ensuring that no single feature disproportionately influences the learning process. To maintain consistency during inference, the *StandardScaler* and *LabelEncoder* fitted during training are serialized with joblib and reused. Figure 4.3 demonstrates the standardization process applied to the 108-dimensional feature vectors, ensuring uniform feature scaling for optimal machine learning model performance. Processed features and their corresponding labels are then encoded using *LabelEncoder* and saved in a one-hot encoded format suitable for classification models, as shown in Figure 4.4. Additionally, both the scaler and label encoder objects are saved to disk with *joblib* for later use during model inference, as shown in Figure 4.5.

```
# Normalize 108D features (shared input for all models)
scaler = StandardScaler()
X_feat_scaled = scaler.fit_transform(X_feat)
```

Figure 4.3: Standardization of Extracted Features Using StandardScaler

```
# Process each file by class
for class_name in gun_classes:
    class_path = os.path.join(DATA_DIR, class_name)
    for file in os.listdir(class_path):
        if file.endswith(".wav"):
            file_path = os.path.join(class_path, file)
            results = process_file(file_path, class_name)
            total_files[class_name] += len(results)
            for stat_feat, label in results:
                feat_list.append(stat_feat)
                label_list.append(label)

# Label encoding + one-hot encoding
encoded_labels = label_encoder.transform(label_list)
categorical_labels = to_categorical(encoded_labels)
```

Figure 4.4: Encoding Class Labels using LabelEncoder and to_categorical

```
# Save encoders for later use
joblib.dump(scaler, os.path.join(MODEL_DIR, 'scaler.pkl'))
joblib.dump(label_encoder, os.path.join(MODEL_DIR, 'label_encoder.pkl'))
```

Figure 4.5: Saving StandardScaler and LabelEncoder Objects for Model Inference

The complete pre-processing pipeline is orchestrated through the *run_pipeline()* function, which processes all audio files in the dataset and generates the final feature matrices. Key parts of the implementation, such as the preprocessing steps and feature extraction, are illustrated in Figure 4.1 and Figure 4.2, showcasing the core functionality used to prepare the gunshot audio data.

4.2.2 SVM Model (svmtuner.py and svm.py)

The Support Vector Machine (SVM) algorithm was chosen as one of the core classifiers for firearm audio classification due to its proven ability to handle high-dimensional feature spaces effectively. SVM works by identifying an optimal hyperplane that best separates the classes, providing strong generalization performance even with limited training data.

To optimize the model's predictive capability, hyperparameter tuning was conducted using the svmtuner.py script. A comprehensive grid search strategy was applied, exploring a range of values for the regularization parameter (C), kernel types *(rbf, linear, poly)*, kernel coefficient *(gamma),* and decision function shape *(ovo, ovr).* This tuning process was guided by 5-fold Stratified Cross-Validation, ensuring a balanced evaluation across all firearm classes. Figure 4.6 illustrates the *GridSearchCV* configuration and parameter tuning process. The optimal combination of hyperparameters was selected based on the highest cross-validation accuracy achieved during the search.

```
# ---------------------- HYPERPARAMETER TUNING ----------------------
log("🔍 Starting SVM hyperparameter tuning with 5-fold cross-validation...")

param_grid = {
    'C': [1, 10, 100],
    'gamma': [0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear', 'poly'],
    'decision_function_shape': ['ovo', 'ovr']
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid = GridSearchCV(SVC(), param_grid, cv=cv, scoring='accuracy', verbose=2, n_jobs=-1)
grid.fit(X_train, y_train)

log(f"\n✅ Best Parameters: {grid.best_params_}")
log(f"📈 Cross-Validation Accuracy (on training data): {grid.best_score_ * 100:.2f}%")
```

Figure 4.6: Configuration of Hyperparameter Tuning for the SVM Model using GridSearchCV and 5-Fold Cross-Validation

After determining the best parameters, the svm.py script was used to retrain the final SVM model on the training dataset. The optimized model employed the *Radial Basis Function (RBF)* kernel with a regularization parameter of *C=10* and a kernel coefficient of *gamma=0.01*. Training and testing were performed using an 80%:20% split of the dataset. The model's performance was then evaluated through classification reports, confusion matrices, and accuracy metrics to comprehensively assess its effectiveness. Figure 4.7 depicts the final training process and the structure of the trained SVC model. Finally, the trained model was serialized and saved using *joblib* for integration into the Django-based web application. This structured and systematic approach ensured the final SVM model was robust, efficient, and capable of accurately classifying gunshot audio recordings.

```
# ---------------------- DEFINE & TRAIN FINAL MODEL ----------------------
log("🚀 Training final SVM model with best hyperparameters...")
start_time = time.time()
final_model = SVC(
    C=10,
    gamma=0.01,
    kernel='rbf',
    decision_function_shape='ovo',
    probability=True
)
final_model.fit(X_train, y_train)
log(f"⏱️ Model training time: {time.time() - start_time:.2f} seconds")

# ---------------------- SAVE MODEL ----------------------
joblib.dump(final_model, MODEL_PATH)
log(f"💾 SVM model saved to: {MODEL_PATH}")
```

Figure 4.7: Final Training and Saving of the Optimized SVM Model for Deployment

4.2.3 kNN Model (knntuner.py and knn.py)

The k-Nearest Neighbor (kNN) algorithm was selected as one of the classifiers for firearm audio classification tasks due to its simplicity and effectiveness in non-parametric learning scenarios. KNN operates by identifying the *k* closest data points in the feature space and assigning the most common class among them, making it particularly suitable for datasets with complex class boundaries.

To enhance the model's performance, hyperparameter tuning was carried out using the knntuner.py script. A *GridSearchCV* was applied to systematically search for the best combination of hyperparameters including the number of neighbors *(n_neighbors)*, distance weighting *(weights),* and distance metrics *(metric)*. The hyperparameter search grid included values for *n_neighbors* of 1, 3, 5, and 7; weights set to either *'uniform'* or *'distance'*; and distance metrics of *'manhattan'*, *'euclidean'*, and *'cosine'*. 5-fold Stratified Cross-Validation was used to ensure that the training folds were balanced across all gunshot classes, minimizing bias and variance. Figure 4.8 illustrates the *GridSearchCV* configuration and hyperparameter search space used during tuning. Once the optimal set of parameters was found, the best model was identified based on validation accuracy. The final configuration selected consisted of *n_neighbors=1*, *weights='uniform'*, and *metric='manhattan'*.

```python
# ---------------------- HYPERPARAMETER TUNING ----------------------
param_grid = {
    'n_neighbors': [1, 3, 5, 7],
    'weights': ['uniform', 'distance'],
    'metric': ['manhattan', 'euclidean', 'cosine']
}

log("🚀 Performing GridSearchCV on training data...")
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
grid = GridSearchCV(
    KNeighborsClassifier(),
    param_grid,
    cv=cv,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)
grid.fit(X_train, y_train)
```

Figure 4.8: Configuration of Hyperparameter Tuning for the kNN Model using GridSearchCV and 5-Fold Cross-Validation

After determining the best hyperparameters, the knn.py script was used to retrain the KNN model on the training set with the selected configuration. An 80:20 training-testing split was applied, and model performance was assessed through classification reports and confusion matrices. The retrained model was saved using *joblib* to facilitate its deployment within the Django web-based system. Figure 4.9 shows the final training procedure and the resulting kNN model configuration. This methodological process ensures that the kNN model is not only simple and fast but also robust and capable of accurately classifying firearm sounds with minimized risk of overfitting.

```
start_time = time.time()
knn_model = KNeighborsClassifier(
    n_neighbors=1,
    weights='uniform',
    metric='manhattan',
)
final_model = knn_model
final_model.fit(X_train, y_train)
joblib.dump(final_model, MODEL_PATH)
```

Figure 4.9: Final Training and Saving of the Optimized kNN Model for Deployment

4.2.4 CNN Model (cnntuner.py and cnn.py)

To address the firearm audio classification task, a Convolutional Neural Network (CNN) model was developed, leveraging its strength in capturing spatial hierarchies within structured data. CNNs are particularly effective for audio classification due to their ability to model local temporal dependencies in feature sequences derived from audio signals.

The development process began with hyperparameter tuning using the cnntuner.py script. A dynamic CNN architecture was designed, allowing for the flexible adjustment of the number of convolutional layers and filter sizes based on different configurations. Figure 4.10 illustrates the dynamic CNN model structure as implemented in the tuning phase.

```python
# ---------------------- BUILD DYNAMIC CNN MODEL ----------------------
def build_cnn_model(optimizer='rmsprop', dropout_rate=0.3, num_filters=32, num_conv_layers=3, **kwargs):
    model = Sequential()
    model.add(tf.keras.Input(shape=(X.shape[1], 1)))

    max_filters = 512  # Cap to avoid memory explosion

    for i in range(num_conv_layers):
        filters = min(num_filters * (2 ** i), max_filters)
        model.add(Conv1D(filters, 3, activation='relu', padding='same'))
        model.add(BatchNormalization())
        model.add(MaxPooling1D(2))
        model.add(Dropout(dropout_rate))

    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(len(label_encoder.classes_), activation='softmax'))

    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# ---------------------- WRAP WITH SCIKERAS WRAPPER ----------------------
wrapped_model = KerasClassifier(
    model=build_cnn_model,
    optimizer='rmsprop',
    dropout_rate=0.3,
    num_filters=32,
    num_conv_layers=3,  # Default, GridSearch will override
    verbose=0,
    target_type="int"
)
```

Figure 4.10: Dynamic CNN Architecture from cnntuner.py

To determine the optimal CNN architecture, an extensive hyperparameter search was conducted using *GridSearchCV* from the *scikit-learn* library, in conjunction with the *KerasClassifier* wrapper provided by *SciKeras*. This approach enables *Keras* models to be treated as scikit-learn estimators, allowing seamless integration with *GridSearchCV*. The search explored a range of hyperparameters, including optimizer types *(RMSprop, Adam),* dropout *rates (0.3, 0.5*), number of convolutional filters *(32, 64, 128),* depth of convolutional layers *(ranging from 2 to 6)*, and batch sizes *(16, 32).* To ensure a balanced evaluation across firearm

classes, 5-fold Stratified Cross-Validation was employed. Figure 4.11 illustrates the grid search process and its configuration.

```
# ---------------------- GRID SEARCH SETUP ----------------------
param_grid = {
    'optimizer': ['rmsprop', 'adam'],
    'dropout_rate': [0.3, 0.5],
    'num_filters': [32, 64, 128],
    'num_conv_layers': [2, 3, 4, 5, 6],  # <-- Dynamic 2 to 6 layers
    'fit__batch_size': [16, 32],
    'fit__epochs': [50]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid = GridSearchCV(estimator=wrapped_model,
                    param_grid=param_grid,
                    cv=cv,
                    scoring='accuracy',
                    verbose=2,
                    n_jobs=1)

# ---------------------- START GRID SEARCH ----------------------
log("🚀 Starting GridSearchCV with 5-Fold Cross-Validation...")
start_time = time.time()
grid_result = grid.fit(X_train, y_train)
log(f"⏱️ Total search time: {time.time() - start_time:.2f} seconds")
```

Figure 4.11: GridSearchCV Setup and Parameter Grid

Prior to model training, each audio sample is transformed into a 108-dimensional feature vector, capturing statistical descriptors such as MFCCs, chroma features, spectral contrast, spectral centroid, zero-crossing rate, bandwidth, and energy. To accommodate the input requirements of one-dimensional convolutional neural networks (1D CNNs), these feature vectors are reshaped into a two-dimensional format of (108, 1), where 108 represents the feature sequence and 1 denotes the single input channel. This structure enables 1D convolutional layers

to effectively capture local patterns and temporal dependencies within the feature set, enhancing the model's ability to learn relevant characteristics from the audio signals.

Upon identifying the best-performing configuration, the final CNN model was retrained using optimal hyperparameters. The selected architecture comprised three convolutional blocks, each incorporating batch normalization, max-pooling, and dropout layers to enhance regularization and stability. This was followed by a fully connected dense layer with 256 neurons and a *softmax* output layer for multi-class classification across firearm types.

The final model was trained using the cnn.py script, with the dataset partitioned into 80% for training and 20% for testing to evaluate its generalization ability. Early stopping and learning rate reduction callbacks were employed to mitigate overfitting and improve training convergence.

Model evaluation was performed by generating a classification report and a normalized confusion matrix, offering insights into the classification performance across all firearm classes. Figure 4.12 presents the confusion matrix results. Finally, the trained model was serialized and saved in the *cnn_best_model.keras* format using the *model.save()* function, preparing it for seamless integration into the Django-based web application.

```python
# ---------------------- CONFUSION MATRIX ----------------------
cm = confusion_matrix(y_val, y_pred, normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(cm,
            annot=True,
            fmt=".2f",
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_,
            cmap='Blues')
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45)
plt.tight_layout()
conf_matrix_path = os.path.join(LOG_DIR, "cnn_confusion_matrix.png")
plt.savefig(conf_matrix_path)
plt.show()
plt.close()
log(f"🖼️ Confusion matrix saved to: {conf_matrix_path}")

# ---------------------- SAVE MODEL ----------------------
model.save(MODEL_PATH)
log(f"💾 CNN model saved to: {MODEL_PATH}")
```

Figure 4.12: Confusion Matrix of CNN Model After Training

This methodical approach, encompassing dynamic model construction, systematic hyperparameter optimization, and comprehensive evaluation, ensured the CNN model was robust, accurate, and well-suited for firearm audio classification tasks.

4.2.5 Model Inference Integration (views.py)

The views.py file functions as the central backend logic of the Django-based web application, managing user interactions, handling audio file uploads, executing machine learning inferences, and returning classification results to the frontend interface. Upon server initialization, several critical components are loaded into memory to optimize performance: the pre-trained Convolutional Neural Network (CNN) model saved in *Keras .keras* format, along

with the serialized Support Vector Machine (SVM) and K-Nearest Neighbor (KNN) models stored using *joblib*. Additionally, a *LabelEncoder* is loaded to map numeric predictions back to firearm class labels, and a *StandardScaler* is employed to ensure that input features are normalized consistently with the original training data. These resources are loaded once at startup to ensure efficient and consistent inference operations throughout the application's lifecycle, as shown in Figure 4.13.

```
# ----- Load artifacts -----
BASE_MODEL_DIR = os.path.abspath(os.path.join(settings.BASE_DIR, '..', 'gunshot_project', 'models'))

label_encoder = joblib.load(os.path.join(BASE_MODEL_DIR, 'label_encoder.pkl'))
scaler = joblib.load(os.path.join(BASE_MODEL_DIR, 'scaler.pkl'))

svm_model = joblib.load(os.path.join(BASE_MODEL_DIR, 'svm_best_model.pkl'))
knn_model = joblib.load(os.path.join(BASE_MODEL_DIR, 'knn_best_model.pkl'))
cnn_model = load_model(os.path.join(BASE_MODEL_DIR, 'cnn_best_model.keras'))
```

Figure 4.13: Loading Pre-trained Models and Preprocessing Artifacts

When a user uploads an audio file via the web interface, the *upload_audio* view function is invoked. This function handles the incoming POST request, saves the uploaded audio file into the server's media directory, and passes the file path to the feature extraction pipeline. The file is then processed through the *extract_features_from_file()* function, which performs audio preprocessing using the *librosa* library. The function ensures that the audio is padded or trimmed to a fixed duration, followed by the extraction of statistical audio features. Specifically, a 108-dimensional feature vector is generated, comprising Mel-Frequency Cepstral Coefficients (MFCCs), chroma features, spectral contrast, spectral centroid, zero-crossing rate, bandwidth, and energy statistics. This comprehensive feature set captures the essential characteristics of the gunshot audio signals.

Once extracted, the feature vector is normalized using the preloaded *StandardScaler* to match the scaling applied during model training. For CNN inference, the feature vector is reshaped into a (108, 1) format to conform to the expected input dimensions for 1D convolutional layers, while for SVM and kNN, it remains as a flat one-dimensional array.

The system supports inference through three different models:

- **Support Vector Machine (SVM):** A robust classifier for high-dimensional feature spaces, providing probability-based predictions.
- **k-Nearest Neighbor (kNN):** A non-parametric, instance-based learner optimized with hyperparameter tuning for firearm classification.
- **Convolutional Neural Network (CNN):** A deep learning model capable of automatically learning complex feature patterns from the extracted audio vectors.

The *get_model_confidences()* function is responsible for generating class probability scores for each model, as shown in Figure 4.14. For SVM and kNN models, prediction probabilities are computed using *predict_proba*, while for the CNN model, the feature vector is reshaped and passed through the neural network to obtain a *softmax* probability distribution over all firearm classes. The final predicted class is determined by selecting the label with the highest probability.

```python
# Prediction based on model choice
confidence_table = []

if model_choice in ['svm', 'knn', 'cnn']:
    models = {
        'svm': svm_model,
        'knn': knn_model,
        'cnn': cnn_model
    }
    model = models.get(model_choice)
    probs = get_model_confidences(model, features_scaled, model_choice)
    final_index = np.argmax(probs)

    # Prepare confidence table
    for idx, gun_type in enumerate(gun_classes):
        confidence_table.append({
            'gun_type': gun_type,
            model_choice: f"{probs[idx]*100:.2f}%"
        })
```

Figure 4. 14: Model Prediction and Confidence Computation

Additionally, the system compiles a detailed confidence table that presents the probability scores for each firearm class, enhancing transparency and allowing users to assess the model's confidence in its predictions. A firearm-specific image is also selected based on the predicted class to provide a visual summary of the result.

The classification outcome, confidence percentages, processing time, and other relevant metadata are rendered through the result.html template, offering users a comprehensive and intuitive report, as shown in Figure 4.15. To optimize system performance, all machine learning models and preprocessing artifacts such as the Convolutional Neural Network (CNN), Support Vector Machine (SVM), K-Nearest Neighbor (KNN) models, the *StandardScaler*, and the *LabelEncoder* are preloaded into memory during server startup. This design ensures that the models are readily available for inference, avoiding the overhead of reloading models on every request and enabling low-latency, real-time classification. This structured and efficient inference pipeline ensures that users can seamlessly upload firearm audio recordings and receive accurate, real-time classification results within an interactive and user-friendly web application.

```
return render(request, 'result.html', {
    'result': result,
    'file_name': audio.name,
    'model_choice': model_choice,
    'model_display_name': model_display_names.get(model_choice, model_choice),
    'elapsed': elapsed,
    'file_url': settings.MEDIA_URL + audio.name,
    'gun_image_url': gun_image_url,
    'confidence_table': confidence_table,
})
```

Figure 4.15: Rendering Classification Results

4.3 Web Application Implementation

This section explains the development of the web-based system, including environment setup, user interface design, system modules, and deployment.

4.3.1 Environment setup

The development of any software project requires an environment setup that provides the tools and frameworks required to develop and deploy the system. In this section, we will discuss the environment setup of the Django framework for developing a Web-based Gunshot Audio Recognition System. In addition, we will explore Visual Studio Code as it is the integrated development environment (IDE) used in this project. The following subsections will detail the installation and setup process of these essential tools.

4.3.1.1 Django Framework Setup

The environment setup for the Django Framework was accomplished by following the installation process provided in the official Django documentation(*Download Django*, n.d.). The setup was initiated by installing Python 3.11.9, which serves as the primary programming language for both the machine learning models and the web framework. After Python was

successfully installed, a virtual environment (venv) was created to manage project-specific dependencies and avoid conflicts with system-wide packages.

Once the virtual environment was activated, Django 5.2 was installed via the Python package manager pip. Additional libraries required for machine learning such as NumPy, Pandas, Librosa, Scikit-learn, TensorFlow, and Keras were also installed within this virtual environment to ensure isolated and organized project dependencies. The python library used in the project will save as the name requirement.txt and attached under Appendix D.

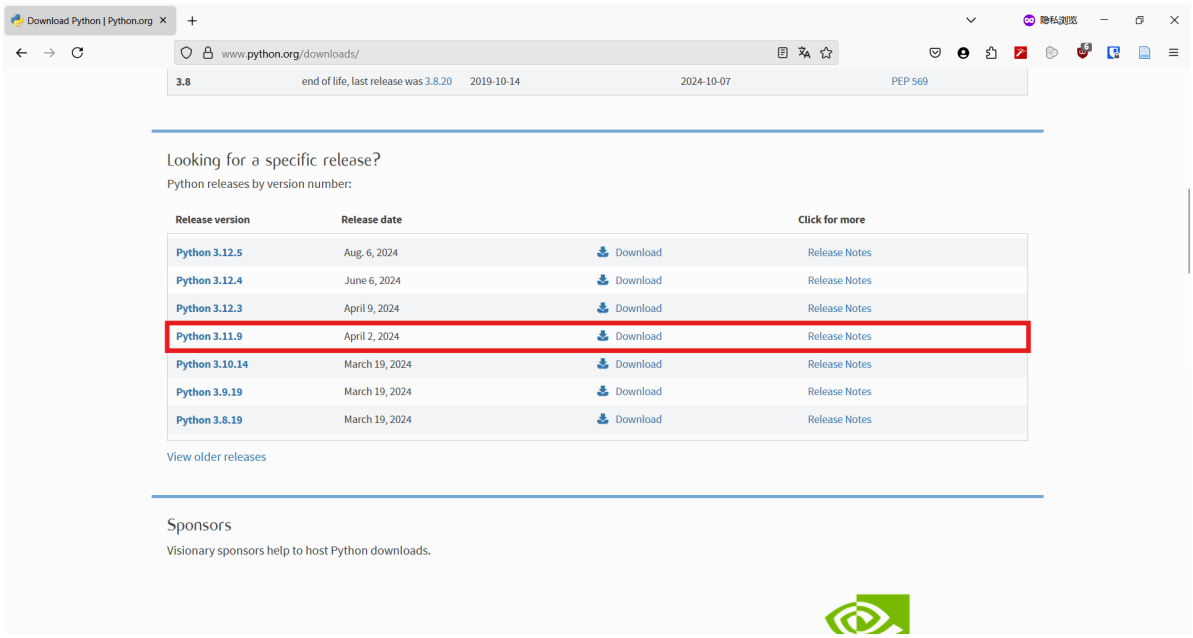

Figure 4.16: The Webpage for Downloading Python

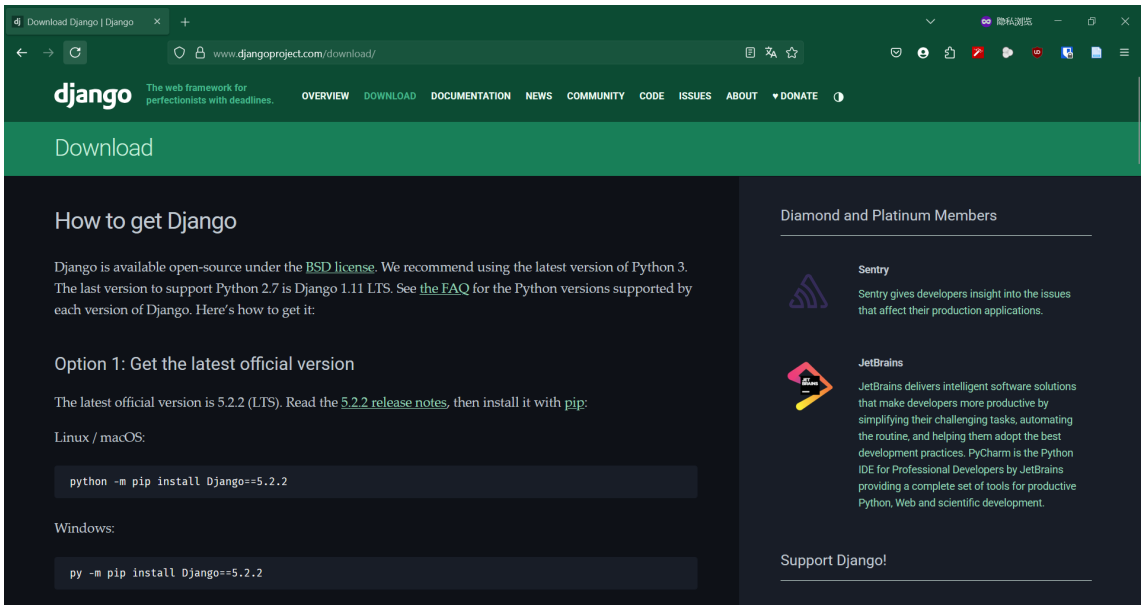

Figure 4.17: Django Framework Installation Command

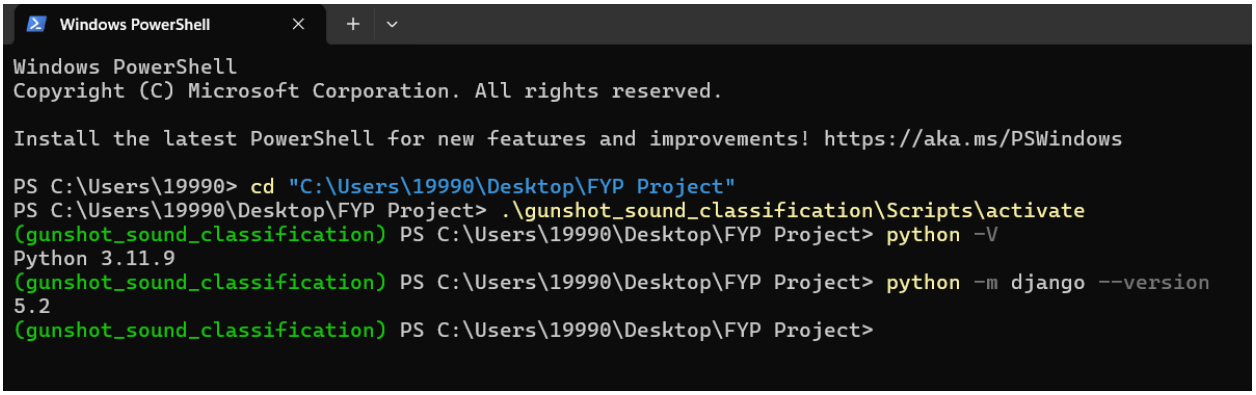

Figure 4.18: Python Version and Django Version

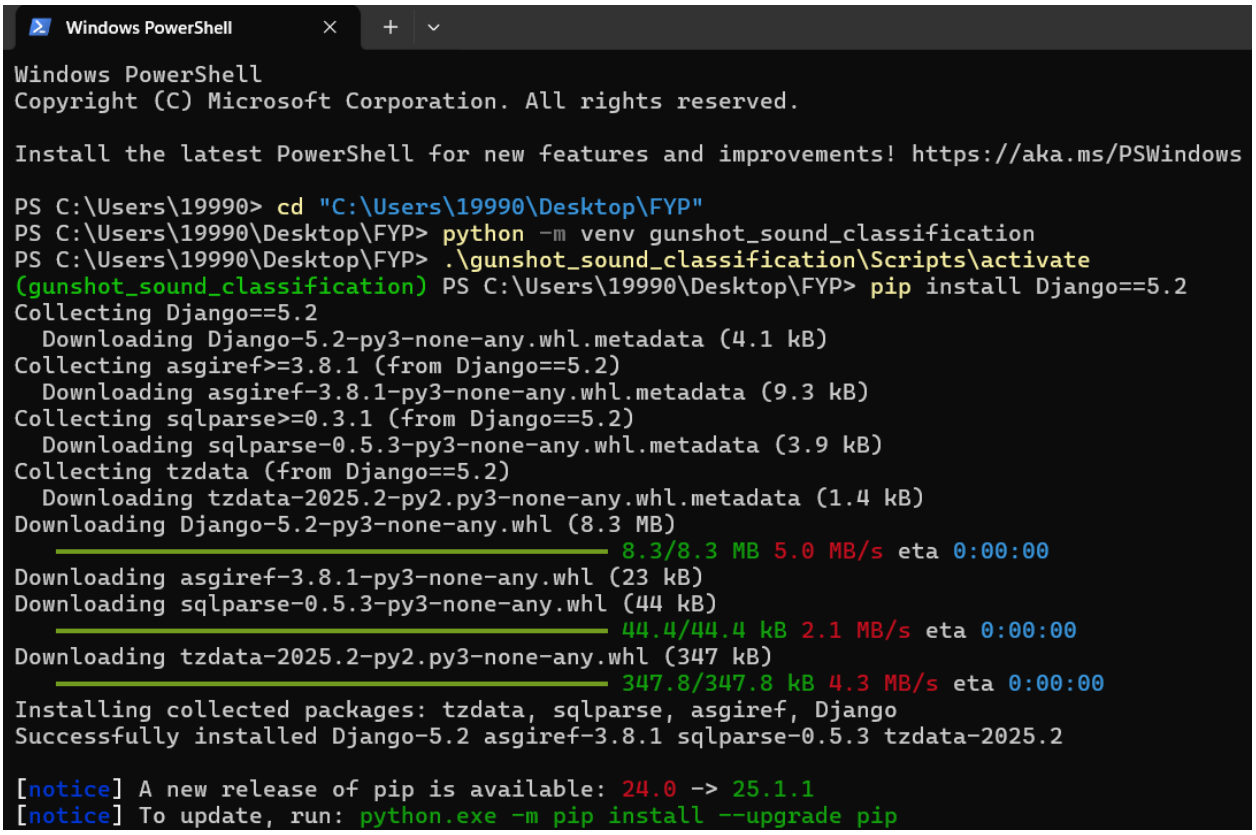

Figure 4.19: Creating A Test Django Framework Project

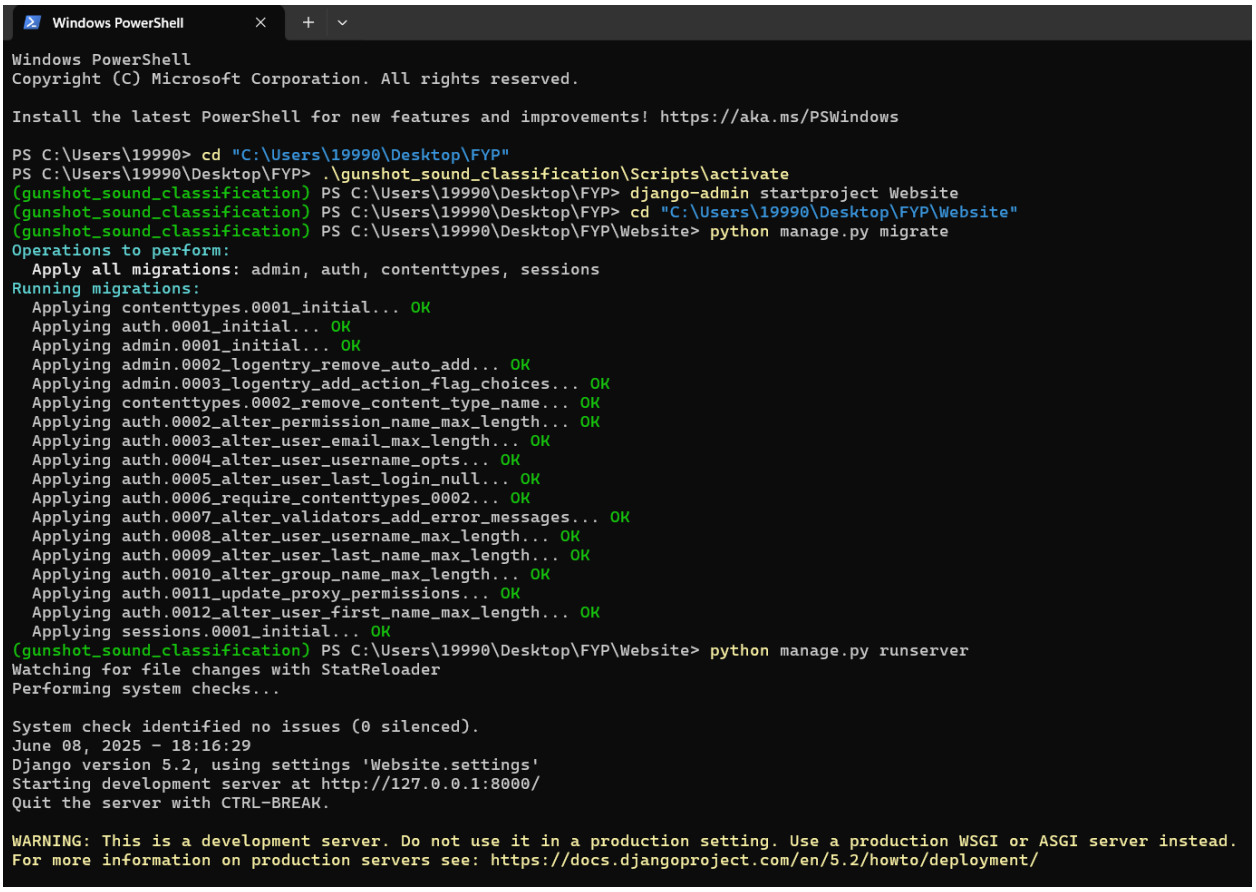

Figure 4.20: Running the Test Django Framework

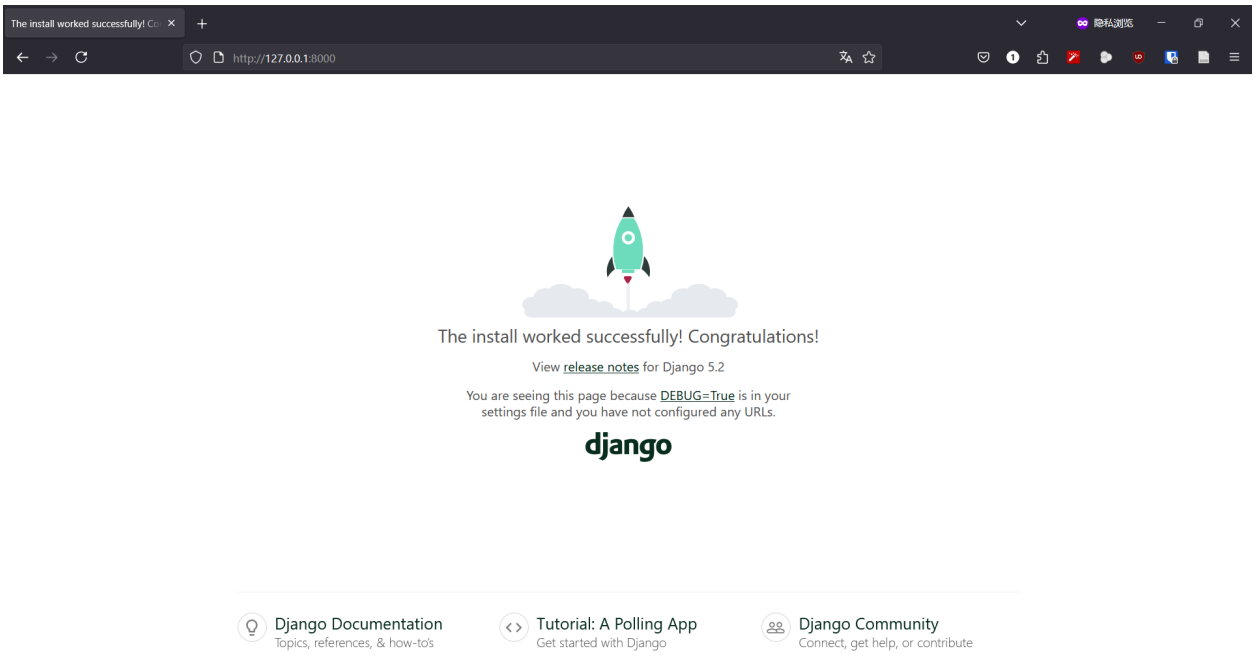

Figure 4.21: The Successful Installation of the Test Django Framework

This setup was essential for the development and deployment of the Web-Based Gunshot Audio Recognition System. The Visual Studio Code environment, combined with the Django framework, provided a robust platform to build the application's backend. It enabled the system to efficiently handle audio uploads, preprocess the audio, perform gunshot classification using machine learning models, and return the predicted firearm type to the user through a user-friendly web interface.

4.3.1.2 Visual Studio Code Setup

For this project, Microsoft Visual Studio Code (VS Code) was used as the main integrated development environment (IDE). VS Code is known for its lightweight nature, cross-

platform support, and robust extension marketplace, making it suitable for Python and Django web development tasks.

The IDE was installed on a Windows 11 24H2 system powered by an Intel Core i7-1165G7 processor with 16GB of RAM. After installation, essential extensions like Python, and Django were added to support coding, debugging, and project management activities.

Customization of the workspace included setting the default Python interpreter, enabling auto-formatting, linting, and Git integration to streamline the development workflow. These configurations allowed smooth development of the web-based gunshot audio recognition system, facilitating efficient model training, web interface creation, and application testing. Screenshots showing the Visual Studio Code environment, including the workspace setup is provided in Figure 4.22.

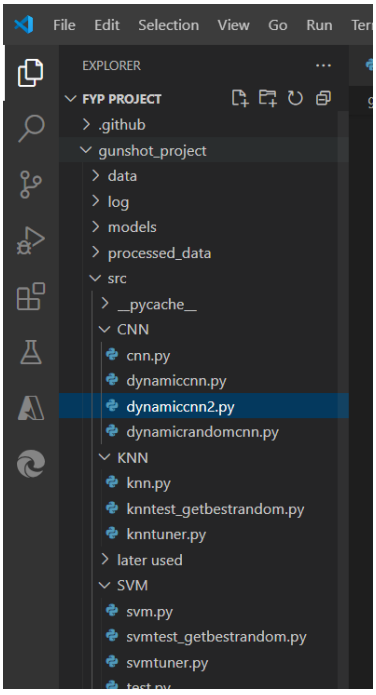

Figure 4.22: Screenshot of Django Project in Visual Studio Code

4.3.2 System Modules

The Web-Based Gunshot Audio Recognition System was developed based on the design methodology outlined in Chapter 3. Built using the Django framework, the system brings together multiple components to create an efficient and user-friendly application. This section describes the core modules implemented in the system, which include the development of the graphical user interface (GUI), the preprocessing and feature extraction pipelines, and the training of machine learning models. The models employed in this system include Support Vector Machine (SVM), k-Nearest Neighbors (kNN), and Convolutional Neural Network (CNN). These modules are seamlessly integrated within the Django environment to ensure smooth interaction between the front-end interface and the machine learning backend.

4.3.3 Graphical User Interface – GUI (main.html and result.html)

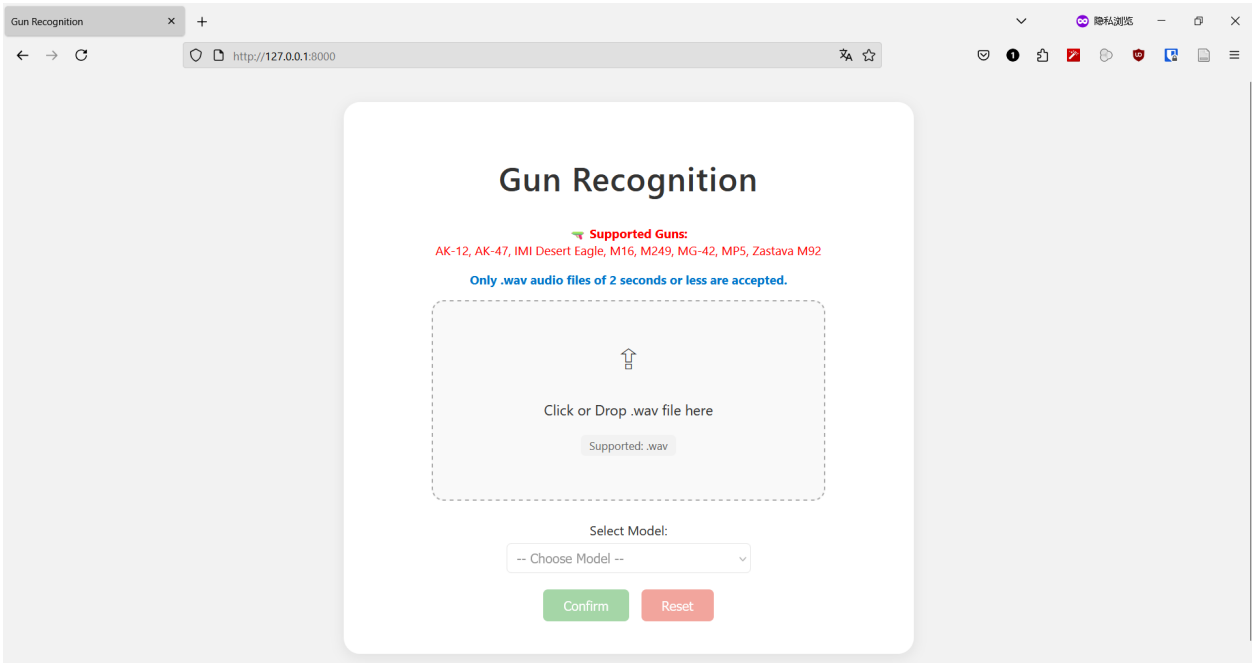

Figure 4.23: Main Page for File Upload and Model Selection

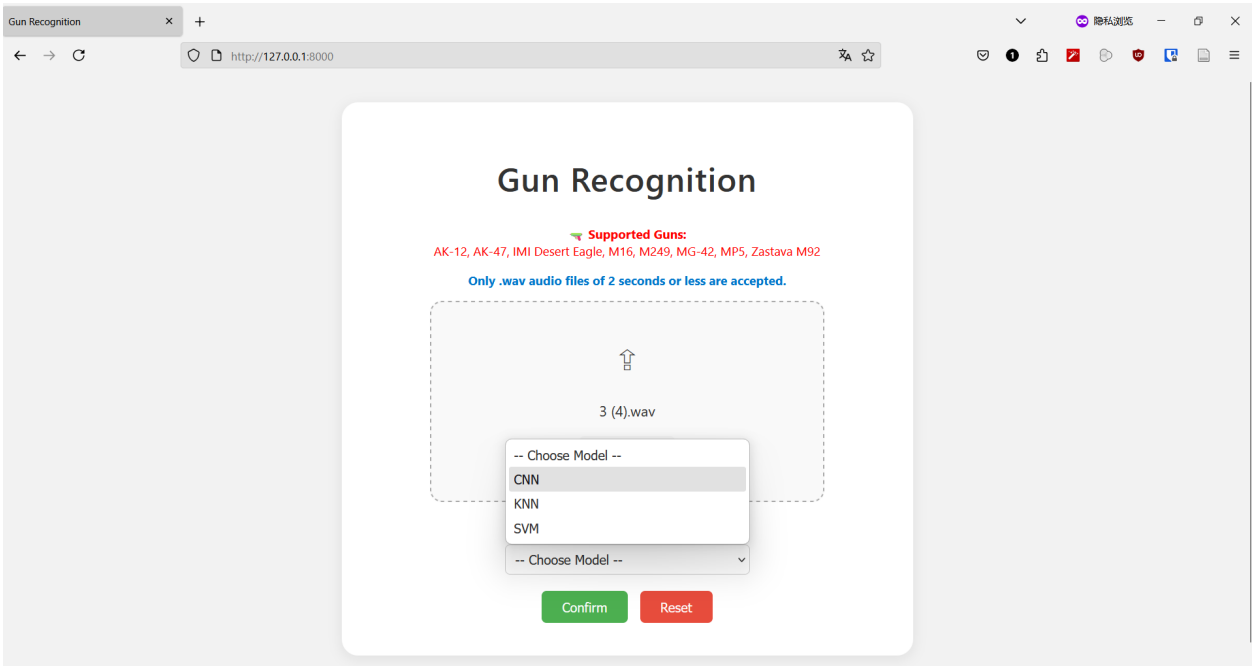

Figure 4.24: Model Selection and Uploaded File Preview

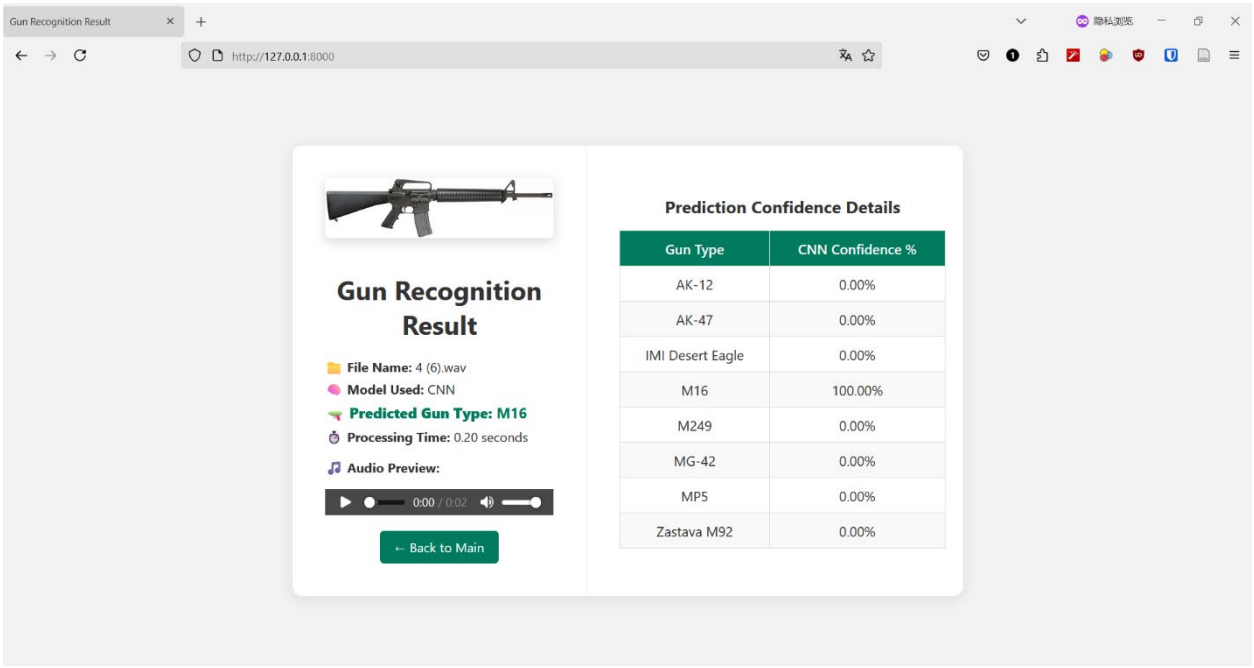

Figure 4.25: Classification Result Display

The Gun Recognition Web Application is an intelligent, web-based system designed to classify gunshot sounds using advanced machine learning models. Developed using the Django framework, a powerful Python web framework, the application delivers a streamlined and intuitive user experience, allowing users to easily upload gunshot audio files for classification.

The interface offers drag-and-drop functionality as well as traditional file browsing for uploading two seconds **.wav* audio files, as shown in Figure 4.23. Users can select from multiple machine learning models (Support Vector Machine (SVM), k-Nearest Neighbors (kNN), or Convolutional Neural Network (CNN)) to perform the classification based on their preference, as shown in Figure 4.24. This flexible model selection ensures users can choose between different algorithmic approaches depending on their needs for speed or accuracy.

After processing, the system predicts the firearm type and displays detailed confidence percentages for each model, offering insights into the prediction certainty, as shown in Figure 4.25. Additionally, a preview of the uploaded audio is available, enhancing user interactivity and transparency.

The web application is optimized for desktop browsers, ensuring reliable performance and a smooth interface. Designed primarily for personal laptop deployment, the system delivers efficient gunshot sound classification without requiring external hosting, making it suitable for academic research and prototyping purposes.

4.3.4 Deployment

This section describes the deployment strategy adopted for the Web-Based Gunshot Audio Recognition System. This system is deployed locally on a personal laptop to facilitate development, testing, and demonstration without external dependencies or hosting costs.

The web application was developed using the Django framework and Visual Studio Code as the primary Integrated Development Environment (IDE). Once the system development and machine learning model training were completed, the application was deployed locally using Django's built-in development server.

To maintain an organized development workflow and enable version control, Git was utilized throughout the project. Upon completion of the Final Year Project, the full source code, including the trained machine learning models (SVM, kNN, CNN) and Django web components, will be uploaded to a GitHub repository for backup, collaboration, and future reference.

GitHub Repository: https://github.com/VincentK0311/Gunshot-Audio-Classifier.git

The deployment process is outlined as follows:

1. Dependency Management

A **requirements.txt** file is maintained to document all the Python libraries and packages required for the system. This ensures that the project environment can be easily replicated by executing, as shown in Figure 4.26:

```
cd "your path\Gunshot-Audio-Classifier-main"
python -m venv gunshot_sound_classification
.\gunshot_sound_classification\Scripts\activate
pip install -r requirements.txt
```

Figure 4.26: Installing Required Python Libraries from requirements.txt

This approach allows others to recreate the project environment with minimal effort.

2. Local Server Deployment

The Django web application is run locally using Django's development server with the following commands:

```
cd "your path\Gunshot-Audio-Classifier-main"
.\gunshot_sound_classification\Scripts\activate
cd Website
python manage.py migrate
python manage.py runserver
```

Figure 4.27: Running the Django Development Server

Once executed, the system becomes accessible at: http://127.0.0.1:8000/

This setup enables a smooth demonstration and user interaction directly from the local machine.

2. Version Control and Source Code Management

Git was used to track progress and manage file versions throughout development. After project completion, the full source code, including Web application (Django), Preprocessing scripts, Trained models *(*.pkl, *.keras*) were uploaded to a public GitHub repository. This ensures backup, transparency, and easy access for lecturers or future developers.

3. Machine Learning Model Management

The trained machine learning models (SVM, KNN, CNN) are stored locally within the models' directory of the project. During the application's runtime, these models are loaded to perform gunshot sound classification.

4. Running Python Scripts (Optional)

If you want to manually run any standalone Python script such as svm.py, knn.py, or cnn.py, you can use the following:

```
cd "your path\Gunshot-Audio-Classifier-main"
.\gunshot_sound_classification\Scripts\activate
cd gunshot_project\src
python filename.py
```

Figure 4.28: Running the Python Code

Replace filename.py with the name of the script you wish to run. This is useful for testing specific model behavior, debugging, or retraining purposes.

By employing this approach, the system is fully functional in a local environment, allowing seamless demonstration without dependency on an internet connection. Uploading the project to GitHub ensures future maintainability and facilitates potential enhancements.

4.4 Summary

This chapter explained the actual implementation of the firearm classification system. The feature extraction pipeline was realized through the preprocess.py script, converting raw audio files into normalized 108-dimensional vectors suitable for machine learning models. Each classifier (SVM, KNN, and CNN) was implemented and trained, with hyperparameter tuning applied to enhance model performance. The CNN model was dynamically constructed, adjusting the number of convolutional layers based on optimal configurations. A Django-based web application was developed to provide a user-friendly platform where users can upload firearm audio files, select a classification model, and view prediction results along with a confidence table. The backend, handled through views.py, integrates the trained models to perform real-time inference. This systematic and integrated approach ensures a seamless pipeline from data input to model prediction and result display.

CHAPTER 5: EXPERIMENTAL RESULTS

5.1 Introduction

This chapter presents the evaluation, testing, and results of the developed gunshot classification system. It begins by outlining the importance of systematic evaluation in assessing model performance and system functionality. The focus is on analyzing the effectiveness of the SVM, kNN, and CNN models through various evaluation metrics, testing procedures, and real-world validation. The chapter also includes system testing to ensure reliable end-to-end functionality of the web-based application.

5.2 Model Evaluation

This section presents the evaluation of the three classification models (SVM, KNN, and CNN) used for firearm type classification based on gunshot audio. The evaluation includes hyperparameter tuning, performance metrics such as accuracy, precision, recall, and F1-score, confusion matrix analysis, time efficiency, and real-world testing. Each model was assessed using a stratified 80:20 train-test split, with hyperparameter tuning performed on the training data using grid search and 5-fold cross-validation. The evaluation aims to compare model robustness, generalization ability, and practical usability within the deployed system.

5.2.1 Hyperparameter Tuning

To ensure optimal performance of the classification models used in this study (Support Vector Machine (SVM), k-Nearest Neighbors (kNN), and Convolutional Neural Network (CNN)), comprehensive hyperparameter tuning process was conducted for each model using

grid search with stratified 5-fold cross-validation. The dataset was initially split into 80% training and 20% testing using stratified sampling to preserve class distribution. Hyperparameter tuning was performed only on the training portion (80%), where the 5-fold cross-validation was applied internally to evaluate different parameter combinations without exposure to the test set.

Once the best-performing hyperparameters were identified through this process, the models were retrained using the entire 80% training set with the selected parameters. They were then evaluated on the unseen 20% test set to obtain the final performance metrics such as accuracy, precision, recall, and F1-score. This procedure ensures that the reported test accuracy is an unbiased estimate of how well the model generalizes to new data and avoids information leakage from using the test set during model tuning.

SVM Hyperparameter Tuning

Grid search with stratified 5-fold cross-validation was used to find optimal hyperparameters, with results shown in Table 5.1. The data is divided into 5 subsets and shuffled before division.

Table 5.1: Results of Best Hyperparameter Search for SVM

Parameter / Values	C	γ (gamma)	Multi-class strategy	Kernel
Test	1, 10, 100	0.1, 0.01, 0.001	ovo, ovr	RBF, Linear, Poly
Best	**10**	**0.01**	**ovo**	**RBF**

For the SVM model, a grid search was applied to four key hyperparameters: the regularization parameter C, the kernel coefficient γ, the multi-class strategy, and the kernel type. Specifically, the values tested included C = (*1,10,100*), γ = (*0.1,0.01,0.001*), multi-class strategies = (*ovo,*

ovr), and kernel types = (*rbf, linear, poly*). This resulted in a total of 54 unique parameter combinations. Each combination was evaluated using stratified 5-fold cross-validation, leading to 270 evaluation runs. As shown in Table 5.1, the best-performing configuration used a Radial Basis Function (RBF) kernel with C=*10*, γ=*0.01*, and a one-vs-one (*ovo*) decision strategy.

KNN Hyperparameter Tuning

Table 5.2: Results of Best Hyperparameter Search for KNN

Parameter / Values	n_neighbors	weights	metric
Test	1, 3, 5, 7	uniform, distance	manhattan, euclidean, cosine
Best	**1**	**uniform**	**manhattan**

The kNN model was tuned using a grid search over the number of neighbors (k), weight functions, and distance metrics. The tested combinations were k = (*1,3,5,7*), weights = (uniform, distance), and distance metrics = (*manhattan, euclidean, cosine*). This produced a total of 24 unique combinations. Each combination was evaluated using stratified 5-fold cross-validation, leading to 120 evaluation runs. As summarized in Table 5.2, the best results were achieved using k=1, with uniform weights and the Manhattan distance metric.

CNN Hyperparameter Tuning

Table 5.3: Results of Best Hyperparameter Search for CNN

Parameter / Values	dropout_rate	fit_batch_size	fit_epochs	num_conv_layers	num_filters	optimizer
Test	0.3, 0.5	16, 32	50	2, 3, 4, 5, 6	32, 64, 128	rmsprop, adam
Best	**0.3**	**32**	**50**	**3**	**128**	**rmsprop**

For the CNN model, tuning was performed using the *KerasClassifier* wrapper and a dynamically constructed model with varying numbers of convolutional layers and filters. The hyperparameters tuned included the optimizer (*rmsprop, adam*), dropout rate (*0.3, 0.5*), number of filters (*32, 64, 128*), number of convolutional layers (*2 to 6*), batch size (*16, 32*), and number of epochs (fixed at *50*). This resulted in a large search space, covering 120 configurations. Each configuration was evaluated using stratified 5-fold cross-validation, leading to a total of 600 evaluation runs. As shown in Table 5.3, the best configuration was found with a dropout rate of *0.3*, batch size of *32*, *3* convolutional layers, 128 filters, and the *rmsprop* optimizer.

The hyperparameter tuning process successfully identified optimal configurations for each model. These parameters were used to retrain the models on the full training set and evaluate their performance on the test set in the subsequent sections. All tuning was conducted using *GridSearchCV* with stratified 5-fold cross-validation on 80% of the data, ensuring fairness, robustness, and reproducibility.

5.2.2 Classification Performance Evaluation

This section provides a comparative analysis of the classification performance of three machine learning models: Support Vector Machine (SVM), k-Nearest Neighbors (KNN), and

Convolutional Neural Network (CNN), based on evaluation using a 20 percent unseen test set. The evaluation metrics used include accuracy, precision, recall, and F1-score, which together provide a comprehensive assessment of each model's classification effectiveness. Both macro-averaged metrics and class-wise performance are considered to evaluate generalization capability across all firearm types.

Table 5.4: SVM - Evaluation Metrics for Each Gun Type

Gun type	Precision	Recall	F1-score	Support
AK-12	1.00	1.00	1.00	39
AK-47	0.97	1.00	0.98	29
IMI Desert Eagle	0.89	0.80	0.84	40
M16	0.85	0.99	0.91	80
M249	0.93	0.93	0.93	40
MG-42	1.00	0.90	0.95	40
MP5	1.00	0.82	0.90	40
Zastava M92	0.97	1.00	0.99	33

Table 5.5: KNN - Evaluation Metrics for Each Gun Type

Gun type	Precision	Recall	F1-score	Support
AK-12	0.97	1.00	0.99	39
AK-47	1.00	1.00	1.00	29
IMI Desert Eagle	0.93	0.95	0.94	40
M16	0.98	1.00	0.99	80
M249	0.95	0.95	0.95	40
MG-42	0.95	0.97	0.96	40
MP5	0.97	0.88	0.92	40
Zastava M92	1.00	0.97	0.98	33

Table 5.6: CNN - Evaluation Metrics for Each Gun Type

Gun type	Precision	Recall	F1-score	Support
AK-12	0.95	1.00	0.97	39
AK-47	1.00	0.97	0.98	29
IMI Desert Eagle	0.95	0.97	0.96	40
M16	0.97	0.97	0.97	80
M249	0.93	0.97	0.95	40
MG-42	1.00	0.97	0.99	40
MP5	0.95	0.90	0.92	40
Zastava M92	1.00	0.97	0.98	33

Table 5.7: Comparison of Model Results

Model	*Accuracy*	*Unweighted Avg Precision (Macro Avg)*	*Unweighted Avg Recall (Macro Avg)*	*Unweighted Avg F1-Score (Macro Avg)*
KNN model from (Tuncer et al., 2021)	0.9448	0.9392	0.9491	0.9441
SVM model from (Nguyen & Nguyen, 2025)	0.9532	0.9562	0.9550	0.9548
SVM	0.9326	0.95	0.93	0.94
KNN	0.9677	0.97	0.96	0.97
CNN	0.9677	0.97	0.97	0.97

As shown in Table 5.7, the CNN and kNN models achieved the highest overall accuracy of 96.77%, while the SVM model attained an accuracy of 93.26%. In terms of macro-averaged precision, recall, and F1-score, CNN and kNN both scored consistently at 0.97 across all three metrics. The SVM model, although slightly lower in overall accuracy, still achieved competitive results, with a precision of 0.95, a recall of 0.93, and an F1-score of 0.94. Compared to previous works, such as the SVM model by Nguyen and Nguyen (2025) and the kNN model from Tuncer et al. (2021), all three models in this study either matched or outperformed these benchmarks, indicating the effectiveness of the applied preprocessing and tuning strategy.

To further understand the strengths and limitations of each model, Table 5.4, 5.5, and 5.6 detail class-wise performance for each firearm type. The SVM model (Table 5.4) exhibited perfect classification for AK-12 and AK-47 (F1-score = 1.00 and 0.98, respectively), as well as for Zastava M92 (F1-score = 0.99). However, its performance on IMI Desert Eagle and MP5 was comparatively lower, with F1-scores of 0.84 and 0.90, respectively. This suggests that SVM struggled to differentiate these classes due to overlapping audio characteristics.

The kNN model (Table 5.5) demonstrated improved class-level balance. It achieved perfect precision and recall for AK-47, along with high F1-scores for M16 (0.99), M249 (0.95),

and MG-42 (0.96). Nevertheless, MP5 remained a challenge, with a slightly lower recall of 0.88 and F1-score of 0.92, although this was an improvement over the SVM result for the same class.

CNN (Table 5.6) showed the most stable and consistent performance across all firearm classes. It achieved F1-scores of 0.97 or higher for AK-12, AK-47, M16, M249, MG-42, and Zastava M92. Notably, it also achieved an F1-score of 0.96 for IMI Desert Eagle, outperforming both SVM and kNN in that class. This consistency suggests that CNN's ability to extract deep temporal and spectral patterns enables better generalization, even for challenging gunshot classes.

Overall, CNN slightly outperformed the other models in both overall metrics and per-class stability, followed closely by kNN. While SVM achieved high performance for some classes, it was more affected by inter-class similarity in challenging firearm types like Desert Eagle and MP5. These results confirm the robustness of CNN for audio-based firearm classification and validate the comparative advantage of data-driven deep learning approaches in handling complex acoustic patterns.

To ensure the reliability and consistency of the reported results, each model (SVM, kNN, and CNN) was trained and tested 20 times using randomized stratified splits. For each run, both training and testing accuracy were recorded. This repeated evaluation allowed for the identification of both the highest and average performance values, offering a more robust understanding of each model's stability and generalization capability.

Specifically, the SVM model achieved the highest testing accuracy of 93.26%, with an average of 87.85%. The CNN model recorded the highest testing accuracy of 96.77%, with an

average of 95.37%, while the KNN model also reached 96.77% at best, with an average of 92.24%. All training accuracy was consistently high, with CNN and kNN models showing 100% training accuracy across all runs. Detailed comparison of best and average results is further discussed in Section 5.4 and documented in Appendix C.

5.2.3 Confusion Matrix Evaluation

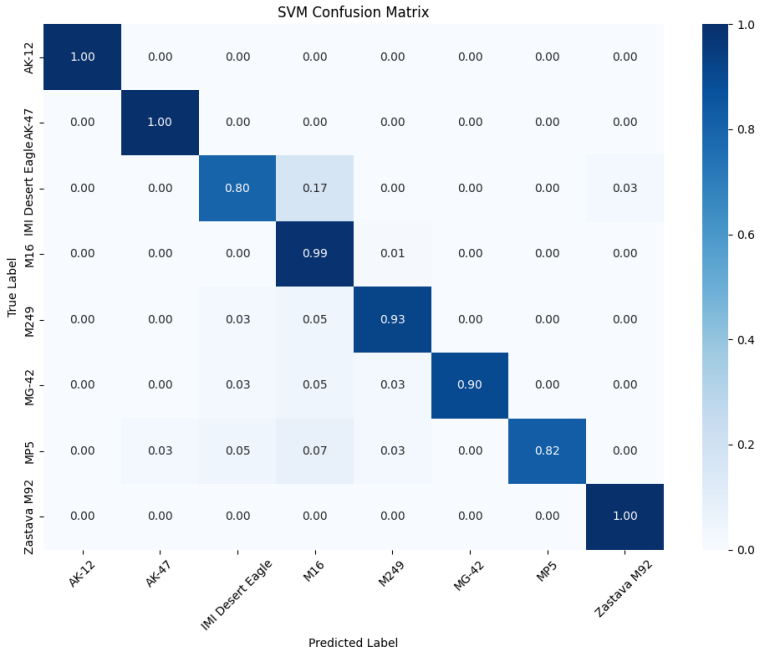

Figure 5.1: SVM Confusion Matrix

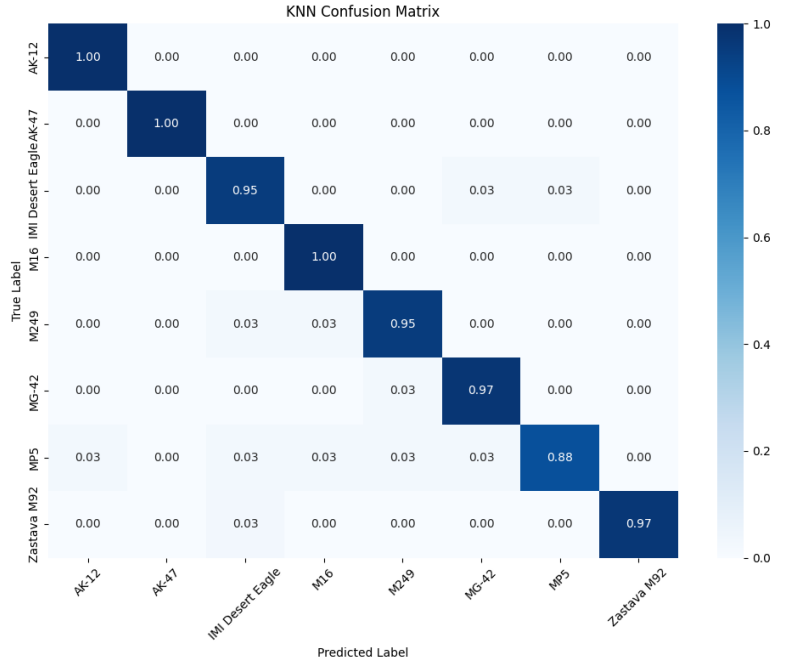

Figure 5.2: KNN Confusion Matrix

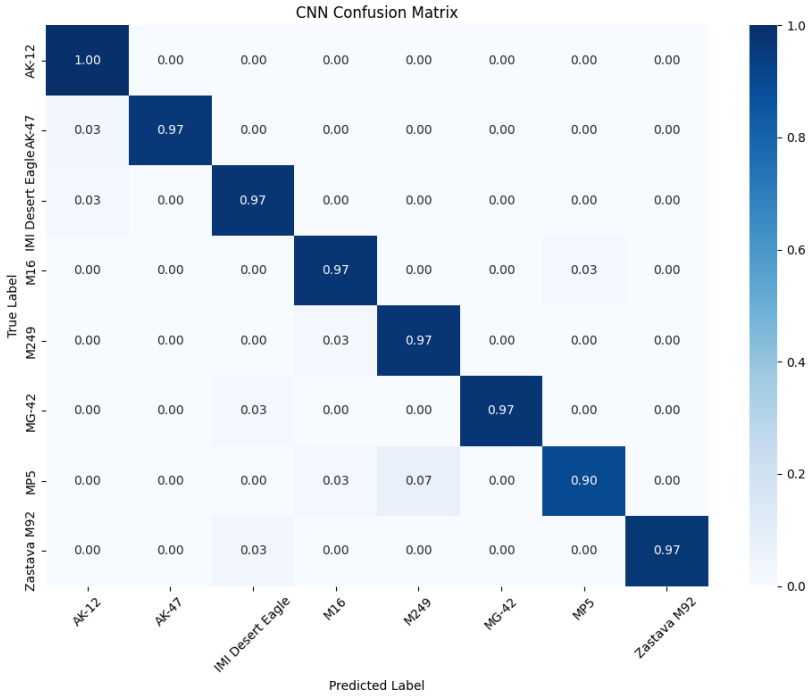

Figure 5.3: CNN Confusion Matrix

Confusion matrices were used to visualize class-wise prediction performance for SVM, kNN, and CNN models. As shown in Figure 5.1, 5.2 and 5.3, all models performed well, but differences in misclassification patterns were observed.

The SVM model correctly classified AK-12, AK-47, and Zastava M92 with 100% accuracy but struggled with IMI Desert Eagle and MP5, which showed confusion with M16 and MG-42.

The KNN model showed more balanced predictions, especially for IMI Desert Eagle and M16, and slightly improved recall for MP5 (88%).

The CNN model had the most consistent performance, with all classes scoring above 90%. Misclassifications were minimal, and MP5 was classified with 90% accuracy, better than SVM and comparable to KNN.

Overall, the confusion matrices support CNN as the most stable model, followed by KNN, while SVM showed more confusion in acoustic similar classes.

5.2.4 CNN Training History: Loss and Accuracy Analysis

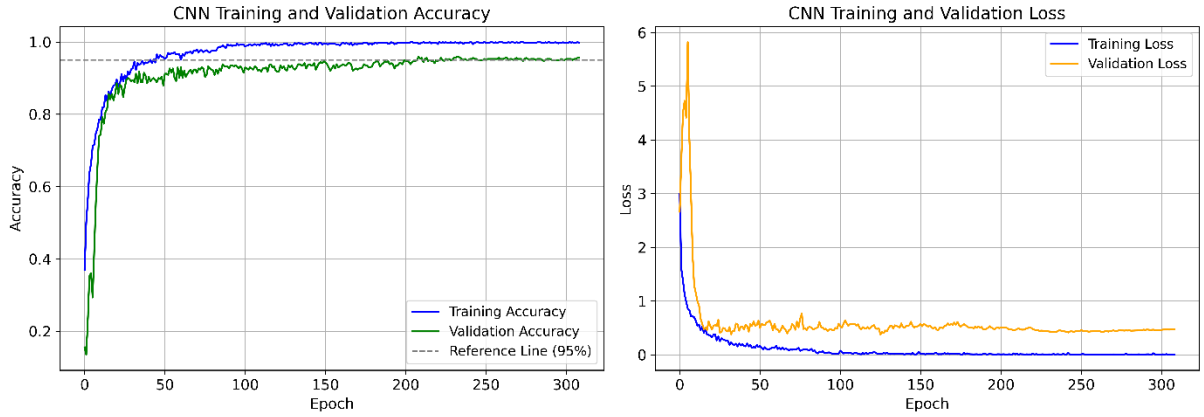

Figure 5.4: CNN Training and Validation Accuracy and Loss Graph

Figure 5.4 shows the training history of a convolutional neural network (CNN) model, showing the training and validation accuracy and corresponding loss values over 300 epochs.

From the graph, the training accuracy increases steadily and eventually approaches 100%, while the validation accuracy converges to about 95.89%. A reference line of 95% accuracy is also plotted in the figure to emphasize the high-performance benchmark achieved by the model. The close proximity between the training and validation accuracy curves indicates that the model has good generalization ability to unknown data and there is no sign of overfitting.

In addition, the training loss shows a smooth and continuous downward trend, indicating that the objective function is effectively minimized. Although the validation loss initially decreases rapidly, it then stabilizes and fluctuates slightly, which is common in deep learning models and does not indicate severe overfitting. The relatively consistent behavior of the accuracy and loss curves confirms the stability of the model during training.

It is worth noting that the highest test accuracy recorded in this study is 96.77%, which was achieved in a separate training run. However, the corresponding historical objects required to generate the training and validation graphs were not retained during this training process. Therefore, Figure 5.4 shows the results of another separate training run, which achieves a slightly lower, but still high, but validation accuracy of 95.89%. This run was chosen to visualize the dynamics and overall convergence trend of the model.

5.2.5 Time Efficiency Evaluation

Table 5.8: Model Execution Time

Model	Total Tuning Time (Grid Search + 5-Fold CV)	Training and Testing Time per Run
SVM	19.34 seconds	0.45 seconds
KNN	3.85 seconds	0 second
CNN	3 hours 15 minutes 4 seconds	16 minutes 52 second

Table 5.8 summarizes the time efficiency of each model in terms of total tuning duration and average training-testing time per run. The SVM model was the fastest to tune, completing grid search and 5-fold cross-validation in just 19.34 seconds, while KNN was even faster at 3.85 seconds due to its non-parametric nature. In contrast, the CNN model required significantly more time for tuning, taking over three hours, which reflects the higher computational cost associated with deep learning methods.

For individual runs, KNN required no training time, as it is instance-based, while SVM completed training and testing in 0.45 seconds. The CNN, although slower, required 16 minutes

and 52 seconds per run. Additionally, preprocessing and feature extraction took 2 minutes and 40 seconds, which was consistent across all models.

5.2.6 Real-World Testing

Table 5.9: Real-World Testing Results using YouTube Gunshot Audio Samples

Gun	Model	Predict (Confidence %)	Status
AK47 (*AK-47 FULL AUTO shooting*, n.d.)	SVM	M249 (23.43%)	False
	KNN	AK12 (100%)	False
	CNN	M249 (99.16%)	False
AK12(Jaeger Z999, 2023)	SVM	MG-42 (66.07%)	False
	KNN	MP5 (100%)	False
	CNN	MG-42 (99.80%)	False

To evaluate the robustness of the models in real-world settings, we obtained gunshot audio clips from YouTube videos and tested them with a deployed web system. These samples were not part of the original dataset and contain real-world environmental conditions such as background noise, reverberation, and microphone quality differences.

As shown in Table 5.9, all three models (SVM, KNN, and CNN) misclassified the gun types in YouTube samples. For instance, the SVM model predicted an AK-47 recording as M249 with only 23.43% confidence. Even the CNN model, which performed best during controlled evaluation, also misclassified the same clip as M249 with a high confidence of 99.16%. Similarly, all models failed to correctly classify an AK-12 sample; the CNN model misclassified it as MG-42 with 99.80% confidence.

These mispredictions highlight the sensitivity of the models to audio recorded in uncontrolled and noisy conditions. This difference may be due to the recording environment, compression artifacts, and acoustic variations that were not present in the training data.

While the system performed well on clean and preprocessed samples, this test highlights that further improvements are needed in processing real audio. Future work may focus on data augmentation using noisy samples, background noise filtering, or retraining with more diverse datasets to enhance the model's generalization ability.

To further demonstrate the effectiveness of the model on common data, Table 5.10 shows sample predictions using audio files from the preprocessed dataset. All three models, SVM, KNN, and CNN, successfully classified the given samples (AK-47 and AK-12) with high confidence percentages ranging from 96.79% to 100%. These correct predictions show that the models are able to accurately handle data with similar characteristics to the training distribution. The results also reinforce the previous evaluation metrics and contrast with the incorrect predictions observed in the actual test shown in Table 5.9.

Table 5.10: Sample Predictions from Preprocessed Dataset

Gun	Model	Predict (Confidence %)	Status
AK-47 (1 (3).wav)	SVM	AK-47 (96.79%)	True
	KNN	AK-47 (100%)	True
	CNN	AK-47 (100%)	True
AK-12 (3 (20).wav)	SVM	AK-12 (98.03%)	True
	KNN	AK-12 (100%)	True
	CNN	AK-12 (100%)	True

5.3 Testing

5.3.1 Web System Testing and Validation

To ensure the web-based system behaves as expected from an end-user perspective, we conducted a series of functional and user acceptance test cases, as shown in Table 5.11. These tests validate core features such as uploading a file, selecting a model, extracting features, displaying predictions, and handling various user input scenarios.

Each model (SVM, KNN, CNN) was tested in the ensemble and prediction pipeline. The system correctly handles uploading a valid **.wav* file, processes it with the selected model, and displays the prediction along with the confidence percentage, processing time, and associated gun image. A key validation feature involves checking audio duration on the client side using JavaScript: files up to 2.5 seconds in length are accepted, but only the first 2 seconds are used for feature extraction during preprocessing. This constraint maintains consistency with the training data based on Nguyen and Nguyen (2025). Files longer than 2.5 seconds are automatically rejected with an alert, ensuring input compatibility. Front-end operations such as drag-and-drop upload, reset functionality, file validation, and handling of insufficient input alerts also work as expected.

In addition, we tested error handling for scenarios such as uploading an unsupported file format or submitting a form without selecting a model. All expected behaviors were triggered, validating the robustness of the user interface and the reliability of the back-end processing.

The results show that the end-to-end system, from voice input to result output, runs smoothly and accurately, providing a user-friendly and reliable platform for predicting gun type based on gunshot sounds.

Table 5.11: Web System Test Cases

Test Case Description	Input	Expected Output	Actual Result	Status
Upload valid .wav audio file	AK-47.wav	File saved, no errors	As Expected	Pass
Select valid model and submit	Model: CNN	Gun type predicted, result page displayed	As Expected	Pass
Display prediction confidence percentages	AK-47.wav, Model: SVM	Table with probabilities shown	As Expected	Pass
Unsupported file type upload	MP5.mp3	Error message displayed, form reset	Error handled: “Only .wav files are supported.”	Pass

No model selected	AK-47.wav	Alert message shown	Alert triggered	Pass
Reset form	Any input	All fields cleared, form disabled	UI reset worked	Pass
Audio playback functionality	Uploaded .wav	Audio player plays uploaded clip	Playback works	Pass
Correct gun image displayed	M16.wav	M16 image shown on result page	Correct image	Pass
Display processing time	Any .wav file	Time shown (e.g., "2 seconds")	Time shown	Pass
Upload audio longer than 2.5 seconds	Any.wav (3s)	Alert triggered, file rejected, form reset	Alert displayed, file rejected	Pass

5.3.2 Usability Evaluation - System Usability Scale (SUS)

This section evaluates the usability of the developed gunshot classification system using the System Usability Scale (SUS), a standardized tool introduced by Brooke (1996) and widely adopted for assessing user experience. To gather structured feedback, a group of final-year students completed the SUS questionnaire, allowing the system's intuitiveness, accessibility, and overall user satisfaction to be measured against established usability benchmarks(*What Every Client Should Know about SUS Scores | Bentley University*, n.d.).

5.3.2.1 Respondent Answers

A Google Form was created based on the standardized System Usability Scale (SUS) questionnaire, as shown in Figure 3.11 and detailed in Appendix A. This section presents the responses from 28 participants who evaluated the system.

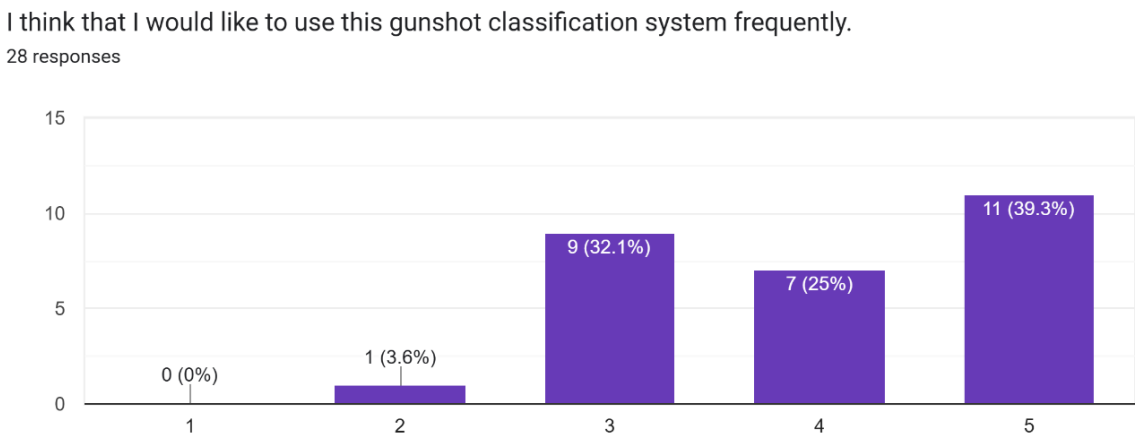

Figure 5.5: SUS Question 1 Result

Figure 5.5 shows the responses to the statement "I think that I would like to use this gunshot classification system frequently." The majority of participants responded positively, with 25% selecting 4 and 39.3% selecting 5. This indicates that most users found the system engaging and would be willing to use it regularly, reflecting a high level of user satisfaction and acceptance.

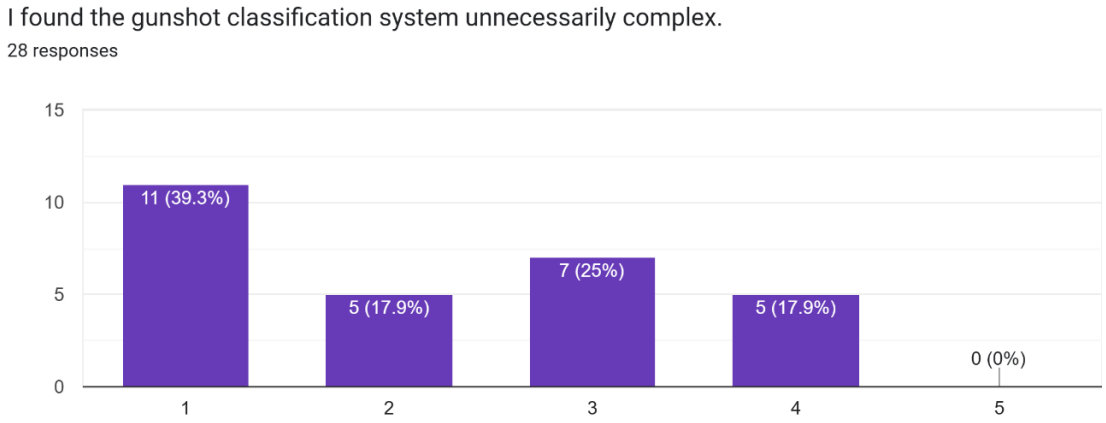

Figure 5.6: SUS Question 2 Result

Figure 5.6 illustrates responses to the statement "I found the gunshot classification system unnecessarily complex." The majority of participants disagreed with this statement, with 39.3% selecting 1 (strongly disagree) and 17.9% selecting 2. This suggests that users generally found the system straightforward and easy to navigate, with minimal perceived complexity in its design and functionality.

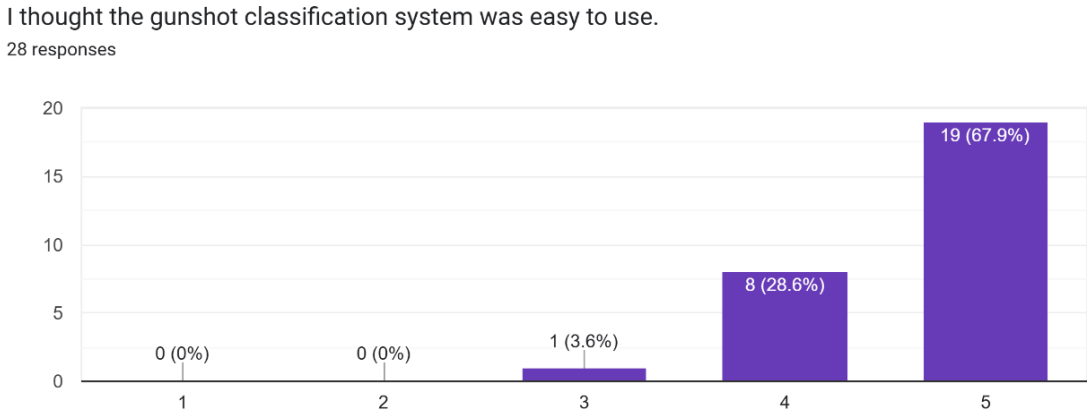

Figure 5.7: SUS Question 3 Result

Figure 5.7 illustrates the responses to the statement "I thought the gunshot classification system was easy to use." 67.9% of participants selected 5 (strongly agree), while 28.6% selected 4, indicating that users generally perceived the system as easy to operate. Very few users rated below 4, suggesting the interface was intuitive and accessible.

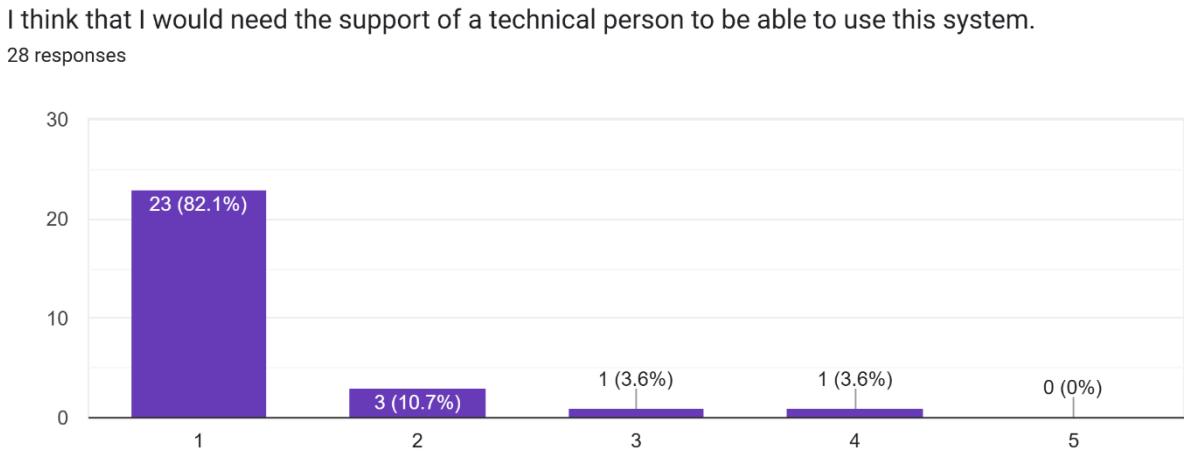

Figure 5.8: SUS Question 4 Result

Figure 5.8 presents the responses to the statement "I think that I would need the support of a technical person to be able to use this system." Most participants (82.1%) selected 1 (strongly disagree), while only a small number selected higher values on the scale. These results suggest that users found the system straightforward and did not feel the need for technical assistance to operate it.

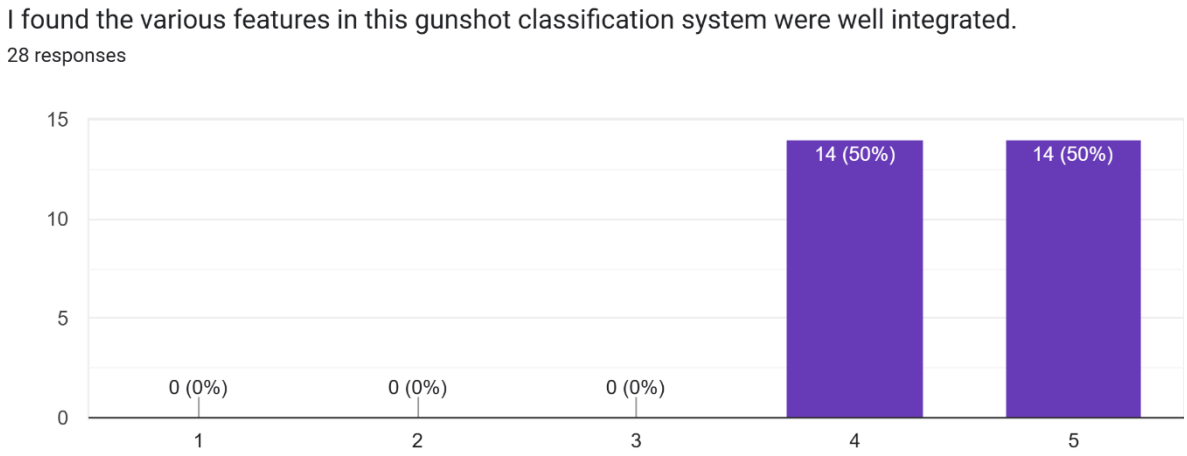

Figure 5.9: SUS Question 5 Result

Figure 5.9 presents the responses to the statement "I found the various features in this gunshot classification system were well integrated." Half of the participants (50%) selected 4, and the other half (50%) selected 5, showing strong agreement. These responses suggest that users felt the system's features worked well together and provided a consistent experience.

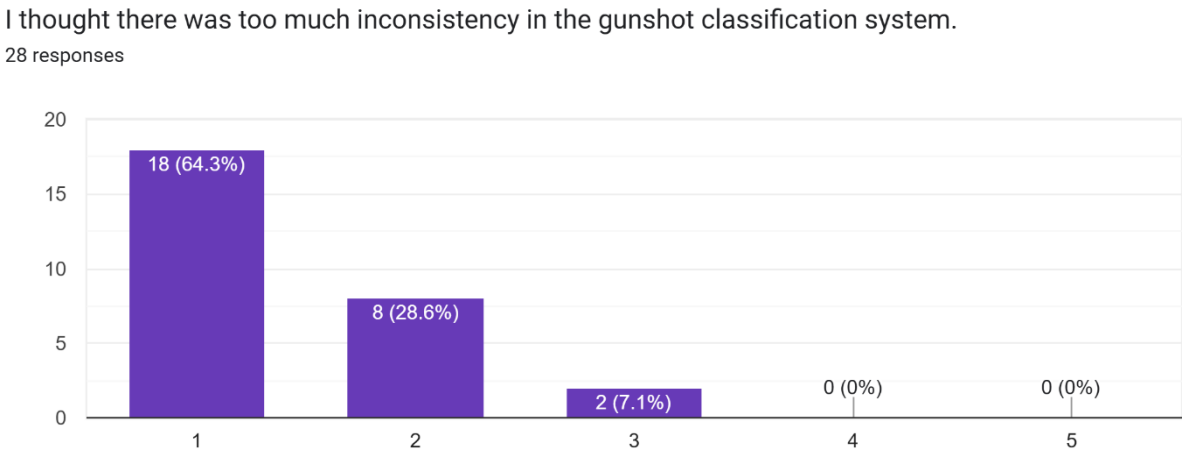

Figure 5.10: SUS Question 6 Result

Figure 5.10 presents the responses to the statement "I thought there was too much inconsistency in the gunshot classification system." Most participants (64.3%) selected 1 (strongly disagree), and 28.6% selected 2, suggesting that users generally felt the system was consistent and reliable in its design and operation.

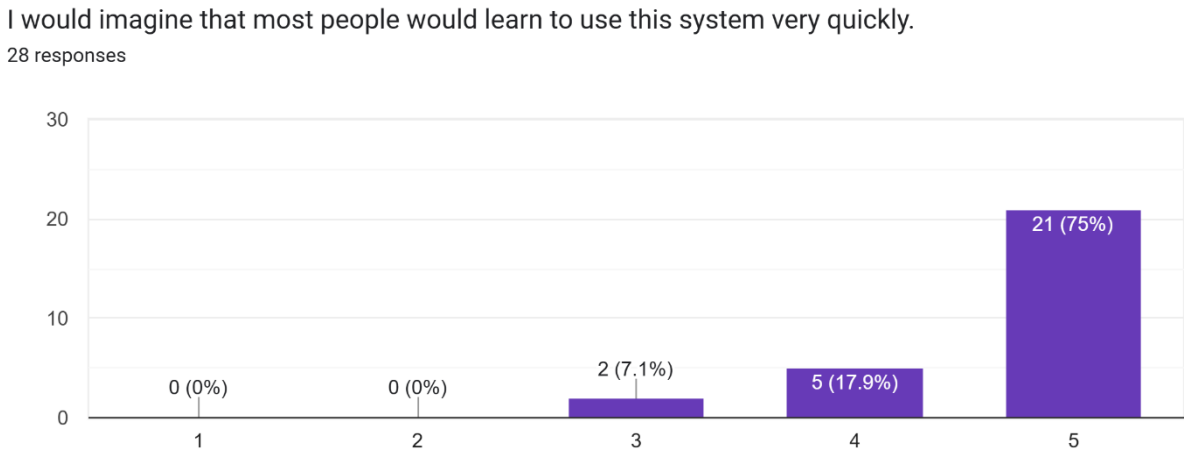

Figure 5.11: SUS Question 7 Result

Figure 5.11 shows the responses to the statement "I would imagine that most people would learn to use this system very quickly." A large portion of participants (75%) selected 5 (strongly agree), and another 17.9% selected 4. These results indicate that users believed the system is easy to learn and can be quickly understood by others.

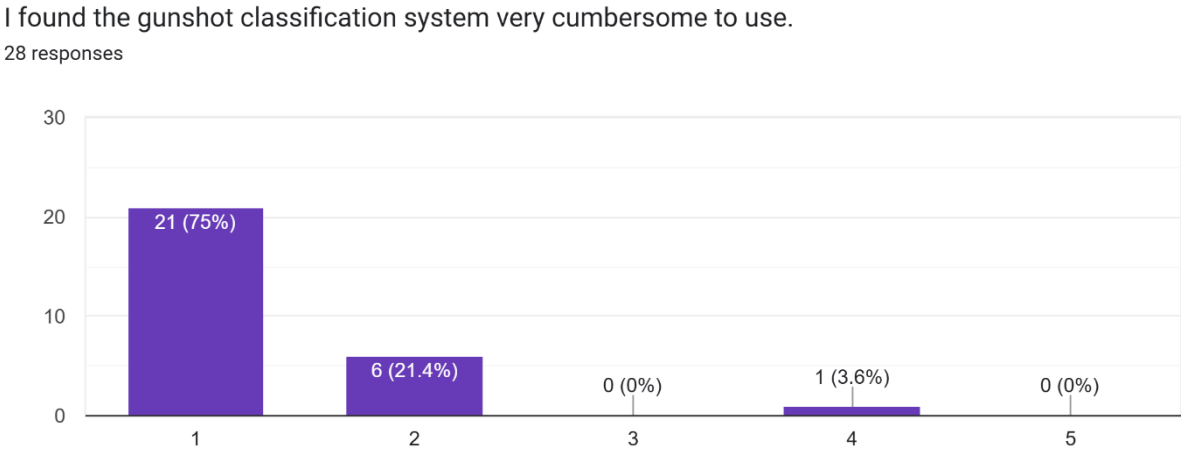

Figure 5.12: SUS Question 8 Result

Figure 5.12 presents the responses to the statement "I found the gunshot classification system very cumbersome to use." Most users selected 1 (75%) or 2 (21.4%), indicating that the system was not seen as difficult or awkward to use. This suggests the system is generally smooth and comfortable to operate.

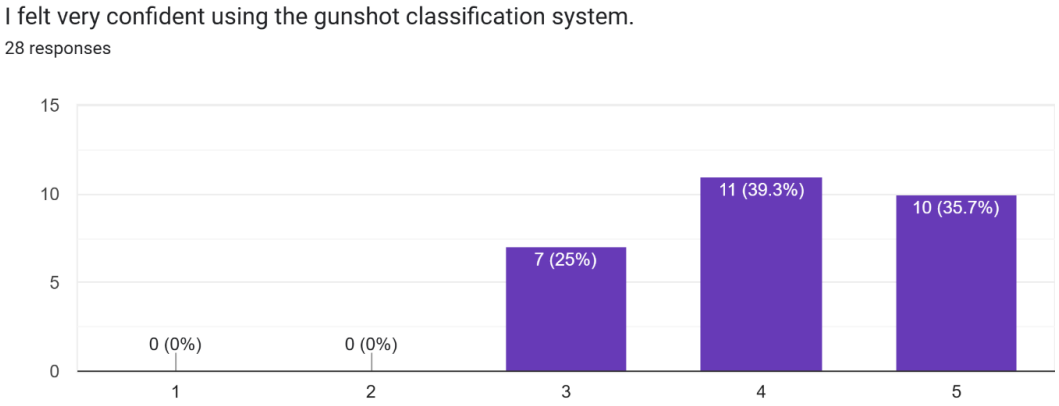

Figure 5.13: SUS Question 9 Result

Figure 5.13 presents the responses to the statement “I felt very confident using the gunshot classification system.” A large portion of users chose 4 (39.3%) and 5 (35.7%), while 25% selected 3. This suggests that most participants felt confident while interacting with the system.

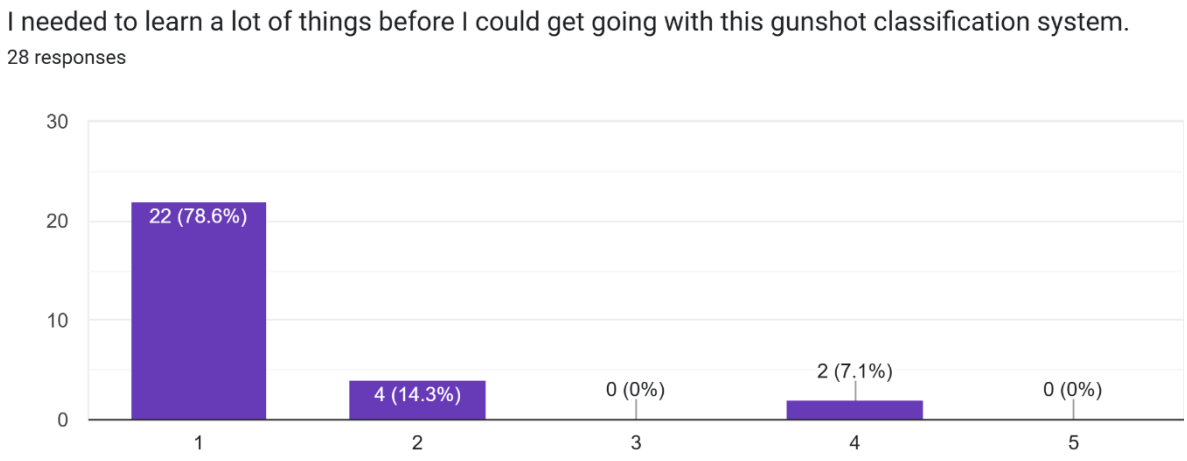

Figure 5.14: SUS Question 10 Result

Figure 5.14 shows the responses to the statement “I needed to learn a lot of things before I could get going with this gunshot classification system.” Most participants (78.6%) selected 1 (strongly disagree), indicating that the system was easy to start using without much prior knowledge.

5.3.2.2 SUS Answers

This section presents responses from 28 participants who completed the System Usability Scale (SUS) questionnaire. Each of the 10 standard statements in the SUS is rated on a 5-point Likert scale to capture user perceptions of the system's ease of use, complexity, trustworthiness, and learning curve.

To calculate each participant's SUS score, the following steps were applied:

- For odd-numbered questions (Q1, Q3, Q5, Q7, Q9), the adjusted score is calculated using the formula:

$$Score = User\ Rating - 1 \quad \text{Equation 5.1}$$

- For even-numbered questions (Q2, Q4, Q6, Q8, Q10), the adjusted score is calculated using the formula:

$$Score = 5 - User\ Rating \quad \text{Equation 5.2}$$

- The adjusted scores across all 10 items are then summed and multiplied by 2.5 to convert the total to a scale of 0 to 100:

$$SUS\ Score = (\sum Score) \times 2.5 \quad \text{Equation 5.3}$$

Table 5.12 summarizes each participant's raw responses and their corresponding SUS score. The individual scores range from 55.0 to 100, indicating varied but generally positive perceptions of the system's usability.

Table 5.12: SUS Score for Each Participant

P/Q	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUS Score
P1	4	3	5	1	4	1	5	1	3	1	77.5
P2	5	1	5	1	5	1	5	1	5	1	100
P3	3	1	5	1	4	1	3	1	4	2	82.5
P4	3	2	4	2	5	2	5	2	5	4	75
P5	5	1	5	2	5	2	5	1	5	2	92.5
P6	3	1	4	1	5	1	5	1	3	1	87.5
P7	5	1	5	1	5	1	5	1	5	1	100
P8	3	2	4	3	5	1	3	1	4	2	75
P9	3	4	5	1	5	1	5	1	4	1	85
P10	4	3	5	1	4	1	5	1	3	1	85
P11	5	1	5	1	4	1	5	1	4	1	95
P12	3	2	5	1	5	1	5	1	5	1	92.5
P13	4	1	5	1	4	2	5	1	3	1	87.5
P14	3	2	4	1	5	1	5	1	3	1	85
P15	2	4	5	1	4	1	5	1	4	1	80
P16	4	3	5	1	4	2	4	2	3	1	77.5
P17	5	3	5	1	5	2	5	1	4	1	90
P18	3	1	4	1	5	2	5	1	4	1	87.5
P19	5	1	4	1	5	1	5	1	5	1	97.5
P20	4	4	5	1	4	1	4	1	3	1	80
P21	5	1	5	1	5	1	5	1	5	1	100
P22	5	3	5	1	5	2	5	1	5	1	92.5
P23	5	3	5	1	4	1	4	2	5	1	87.5
P24	5	4	5	1	4	1	5	2	4	1	85
P25	5	4	3	1	4	2	5	2	4	1	77.5
P26	4	1	5	1	4	1	5	1	5	1	95
P27	4	3	4	4	4	3	4	4	4	4	55
P28	3	2	4	2	4	3	4	2	4	2	70
Total											2395

To determine the overall usability of the system, the average SUS score across all participants was calculated using the formula:

$$\text{Average SUS Score} = \frac{\sum_{i=1}^{n} Individual\ SUS\ Score_i}{\text{n}} \qquad \text{Equation 5.4}$$

Where:

- $\sum$Individual SUS Score$_i$ is the sum of all individual SUS scores.
- n is the total number of participants.

To calculate the overall SUS score for this system:

$$\text{Average SUS Score} = \frac{2395}{28} \approx 85.54$$

The System Usability Scale (SUS) evaluation was conducted with 28 participants, and their individual scores are presented in Table 5.12. The average SUS score calculated was 85.54, placing the system in the "Excellent" usability category. This suggests that most users found the system intuitive, efficient, and easy to navigate. Among the participants, 8 individuals scored above 90, indicating a high level of satisfaction with the user experience. Meanwhile, the remaining users mostly scored in the range of 75 to 90, further supporting strong usability perceptions. Only one participant scored below 70, suggesting a slightly lower but still acceptable experience. These findings demonstrate a generally positive reception and validate the system's design and user interface as effective and user-friendly.

5.4 Results

This section summarizes the overall findings from the evaluation and testing of the three classification models (SVM, KNN, and CNN) used for firearm type identification based on gunshot audio.

During model evaluation, the CNN and KNN models achieved the highest classification performance, with an overall accuracy of 96.77%, and macro-averaged precision, recall, and F1-score all reaching 0.97. The SVM model showed competitive performance with 93.26% accuracy, but lower recall in some firearm classes, especially IMI Desert Eagle and MP5, as observed in the classification report and confusion matrix.

Time efficiency evaluation revealed that KNN was the fastest in terms of training and testing time, taking only 3.85 seconds for tuning and 0 seconds for prediction (as it is a lazy learner). The CNN model, while have the accurate like kNN, had the highest computational cost, requiring over 3 hours for hyperparameter tuning and approximately 17 minutes for each training and testing cycle. Preprocessing and feature extraction were consistent across all models, taking 2 minutes and 40 seconds per run.

Real-world testing using YouTube gunshot audio samples showed that all models failed to correctly classify the recordings. This performance drop can be attributed to background noise, compression artifacts, and acoustic variation does not present in the training data, indicating a lack of generalization to uncontrolled environments. In contrast, all models demonstrated perfect classification on preprocessed dataset samples, as shown in Table 5.10.

Functionality testing of the web-based system confirmed that audio upload, model selection, prediction, and result visualization work reliably. All test cases passed, showing a user-friendly and operational end-to-end system.

Overall, the results confirm that CNN provides the most balanced and accurate performance for gunshot classification on clean data, while KNN offers excellent speed with slightly lower robustness. A detailed comparison of each model's best and average training and testing accuracy across 20 evaluation runs is provided in Appendix C. Real-world performance, however, highlights the need for further enhancement in noise handling and generalization capability.

5.5 Summary

This chapter presented a comprehensive evaluation of the proposed gun classification system using SVM, kNN, and CNN models. The evaluation is performed using classification metrics, confusion matrices, time efficiency analysis, real-world testing, and system functional verification. Among all models, CNN has the highest accuracy and consistency, while kNN performs well. All models perform well on preprocessed dataset samples, but have problems processing noisy real-world audio, exposing limitations in their generalization capabilities. The web-based system successfully passes all functional test cases, demonstrating its reliability and usability.

CHAPTER 6: CONCLUSION AND FUTURE WORKS

6.1 Achievements

This project successfully implemented a gunshot audio classification system capable of identifying firearm types from short audio recordings using machine learning models, integrated within a web application. The system utilizes established methods for audio feature extraction and classification, combined with a simple and functional user interface.

The dataset used was obtained from Kaggle, consisting of 851 gunshot audio samples from eight different firearm types. The audio files were already standardized to two second durations, making them suitable for consistent feature extraction and model training.

All preprocessing and feature extraction methods were fully adapted from a single referenced research article. As described in the article, Gaussian noise augmentation was used to simulate realistic background interference and improve model generalization. The feature extraction process included Mel-Frequency Cepstral Coefficients (MFCCs), Chroma, Spectral Contrast, Zero Crossing Rate (ZCR), and other spectral descriptors. These features were combined into 108-dimensional vectors, which were normalized using a pre-fitted scaler to ensure consistent input for machine learning models.

Three commonly used machine learning models were implemented: Support Vector Machine (SVM), k-Nearest Neighbors (kNN), and Convolutional Neural Network (CNN). All models were tuned using grid search and validated through 5-fold stratified cross-validation. The CNN and KNN models achieved the best performance, each reaching a testing accuracy of 96.77%, indicating strong classification capability for the given dataset.

The system was deployed using the Django web framework, allowing users to upload an audio file, select one of the trained models, and receive instant predictions. The result includes the predicted firearm type, confidence score, and a corresponding firearm image. The system runs entirely in memory during each session and does not store user data or results.

In summary, the project achieved its intended objectives by integrating public data, literature-based methods, and standard machine learning models into a complete and functional audio classification system. The final system reflects the practical application of audio processing and machine learning concepts and serves as a working prototype that could be extended with additional features, such as a larger dataset, improved model variety, or enhanced user interaction.

6.2 Limitations

Despite the promising results and success of the gun classification system, the project faces some limitations that must be acknowledged. These limitations highlight areas for improvement and lay the foundation for future improvements.

First, classification accuracy is highly dependent on the quality, quantity, and diversity of the dataset used. While the dataset is real and publicly available, containing 851 gunshot recordings from eight types of firearms, it may not adequately capture the full range of sound variations encountered in the real world. Real gunshots vary significantly based on factors such as the type of environment (urban vs. rural), distance from the recording source, the directionality of the gun, the echo effects of buildings or natural features, and background noise

(such as wind, vehicles, or crowds). Because the dataset consists primarily of clean, curated clips extracted from online sources, the trained models may have difficulty generalizing outside of these controlled conditions.

Second, the system's predictive performance is limited by the preprocessed dataset used during training and internal testing. The models perform well when predicting gun types based on audio samples that follow the same preprocessing pipeline and acoustic properties as the training data. However, when new raw gunshot recordings were collected from external sources such as YouTube and tested without retraining, the system produced inaccurate or false classifications. This suggests that the model has limited generalization capabilities when exposed to truly unseen data with different noise levels, recording quality, or gun characteristics. This behavior is a common problem in real-world deployments, where models overfit to the training conditions but fail to adapt to unfamiliar input distributions.

Furthermore, the web application currently allows users to upload audio clips up to 2.5 seconds in duration; however, due to the constraints of the preprocessing pipeline adapted from Nguyen and Nguyen (2025), only the first 2 seconds of the audio are processed and used for classification. This approach ensures consistency with the training dataset but may result in information loss if key gunshot features occur beyond the 2-second mark. Consequently, users uploading longer recordings may unknowingly submit incomplete data for analysis, potentially affecting classification accuracy.

Third, computational resource limitations pose challenges for experimenting with more complex architectures or larger training. While SVM and KNN models were trained locally, CNN training was performed on Google Colab due to the lack of a dedicated GPU on the

development machine. This limited the depth and diversity of CNN architecture explored, as training time and session quotas limited the ability to experiment with deeper networks, longer epochs, or advanced regularization techniques. Additionally, hyperparameter tuning was limited to grid search over a narrow range, which leaves room for optimization via more advanced techniques such as random search or Bayesian optimization.

Additionally, the current system is inherently static, as it can only classify a predefined set of gun types. If new gun types need to be identified, the system needs to manually collect new samples, reprocess the data, retrain the model, and redeploy. The current lack of incremental learning or dynamic model update mechanisms limits the scalability and adaptability of the system in an evolving threat environment as new gun types may emerge.

Finally, practical deployment considerations such as latency, environmental robustness, and integration with real-time recording hardware were not considered in this project. The system was only tested using uploaded .wav files through a web interface. Real-time gunfire detection from microphone streams or deployment in mobile/edge environments has not yet been achieved, so the system's real-time performance, responsiveness, and accuracy in uncontrolled environments remain untested.

While the system achieves high accuracy on a range of experimental datasets, its current form is limited by the representativeness of the datasets, generalization to unknown data, computational resources, model adaptability, and limited real-world testing. These limitations form the basis for important future work on how to transform the prototype system into a deployable and reliable gun detection system.

6.3 Future Works

To enhance the system's accuracy, robustness, and applicability in real-world scenarios, several directions for future work are recommended. One of the most important areas is the expansion and diversification of the dataset. Increasing the amount and variety of gunshot audio recordings, collected under different environmental conditions, distances, and using a broader range of firearm types, can significantly improve the model's ability to generalize. This may involve gathering data through controlled field recordings, exploring publicly available law enforcement datasets if accessible, or generating synthetic gunshot sounds using acoustic realistic simulations.

Another essential area of improvement involves increasing the model's ability to generalize to previously unseen audio. While the current models perform effectively on preprocessed and familiar data, they tend to struggle with new or noisy input. Future development should focus on applying domain adaptation techniques, noise-robust training strategies, and effective regularization methods to improve performance across different audio sources. Enhancing data augmentation techniques to simulate a wider range of environmental conditions, such as background noise, varying echo patterns, and diverse recording qualities, can further support this goal.

The exploration of more advanced model architecture also presents a promising path for future work. Implementing deep learning models such as one-dimensional convolutional neural networks (1D-CNNs), long short-term memory (LSTM) networks, or transformer-based architectures may provide improved capabilities in capturing the temporal and spectral features

of gunshot audio. Additionally, ensemble learning approaches that combine the outputs of multiple classifiers could be employed to enhance overall accuracy and reliability.

Finally, conducting real-world testing and obtaining user validation is critical for ensuring the system's effectiveness in practical applications. Collaborations with law enforcement agencies, emergency responders, or security professionals will facilitate testing in real-life environments, such as outdoor shooting ranges or urban areas. These evaluations can provide essential insights into the system's performance under realistic conditions and help refine its functionality. Gathering feedback from end users will also be valuable for guiding future improvements and ensuring the system is both practical and reliable in operational use.

REFERENCES

AK-47 FULL AUTO shooting: Awesome sound #ak47 MAG DUMP #fullauto #burst. (n.d.). [Video recording]. Retrieved June 11, 2025, from https://www.youtube.com/shorts/LMQIWLg1bnY

Chen, Z., Zheng, H., Wu, L., Huang, J., & Yang, Y. (2024). Deep Transfer Learning Based Intelligent Gunshot Detection and Firearm Recognition Using Tri-Axial Acceleration. *IEEE Internet of Things Journal*, 1–1. IEEE Internet of Things Journal. https://doi.org/10.1109/JIOT.2024.3489963

Download Django. (n.d.). Django Project. Retrieved June 8, 2025, from https://www.djangoproject.com/download/

Gunshot audio dataset. (n.d.). Retrieved January 17, 2025, from https://www.kaggle.com/datasets/emrahaydemr/gunshot-audio-dataset

Introduction to Convolution Neural Network. (2024, October 10). GeeksforGeeks. https://www.geeksforgeeks.org/introduction-convolution-neural-network/

Irungu, J., Ancel, J., Mahmoud, W., & Denis, M. (2023). *Gunshot detection from audio excerpts of urban sounds using transfer learning*. 045003. https://doi.org/10.1121/2.0001783

Jaeger Z999 (Director). (2023, April 1). *Russian Military Service Rifle AK12* [Video recording]. https://www.youtube.com/watch?v=zKZn0U5TDiw

K-Nearest Neighbor(KNN) Algorithm for Machine Learning—Javatpoint. (n.d.). Www.Javatpoint.Com. Retrieved January 16, 2025, from https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

Nesar, S. B., Whitaker, B. M., & Maher, R. C. (2024). Machine Learning Analysis on Gunshot Recognition. *2024 Intermountain Engineering, Technology and Computing (IETC)*, 249–254. https://doi.org/10.1109/IETC61393.2024.10564263

Nguyen, V. T., & Nguyen, D. T. (2025). Explosion sound classification using machine learning method based on audio features. *Journal of Military Science and Technology*, *102*, 133–140. https://doi.org/10.54939/1859-1043.j.mst.102.2025.133-140

Nijhawan, R., Ansari, S. A., Kumar, S., Alassery, F., & El-kenawy, S. M. (2022). Gun identification from gunshot audios for secure public places using transformer learning. *Scientific Reports*, *12*(1), 13300. https://doi.org/10.1038/s41598-022-17497-1

Shendre, S. P., Priya, M. Y., Sridhar, S., Kamble, S. S., & Khanna, M. (2024). Anylogic and Python-Based Gunshot Detection System Using CNN and Trilateration. *2024 IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS)*, 1–7. https://doi.org/10.1109/ICITEICS61368.2024.10625443

Singh, R. B., & Zhuang, H. (2022). Measurements, Analysis, Classification, and Detection of Gunshot and Gunshot-like Sounds. *Sensors*, *22*(23), 9170. https://doi.org/10.3390/s22239170

Software Prototyping Model and Phases. (2023, March 30). GeeksforGeeks. https://www.geeksforgeeks.org/software-prototyping-model-and-phases/

Support Vector Machine (SVM) Algorithm—Javatpoint. (n.d.). Www.Javatpoint.Com. Retrieved January 16, 2025, from https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

Teng, Y., Zhang, K., Lv, X., Miao, Q., Zang, T., Yu, A., Hui, A., & Wu, H. (2024). Gunshots detection, identification, and classification: Applications to forensic science. *Science & Justice*, *64*(6), 625–636. https://doi.org/10.1016/j.scijus.2024.09.007

Tuncer, T., Doğan, Ş., Akbal, E., & Aydemir, E. (2021). *AN AUTOMATED GUNSHOT AUDIO CLASSIFICATION METHOD BASED ON FINGER PATTERN FEATURE GENERATOR AND ITERATIVE RELIEFF FEATURE SELECTOR*.

Urbano, J., Bogdanov, D., Herrera, P., Gomez, E., & Serra, X. (2014). *WHAT IS THE EFFECT OF AUDIO QUALITY ON THE ROBUSTNESS OF MFCCs AND CHROMA FEATURES?*

Vavrek, J., Pleva, M., & Juhár, J. (2010). *Acoustic events detection with Support Vector Machines*.

What Every Client Should Know about SUS Scores | Bentley University. (n.d.). Retrieved June 21, 2025, from https://www.bentley.edu/centers/user-experience-center/what-every-client-should-know-about-sus-scores

APPENDICES

Appendix A

		Strong disagree 1	2	3	4	Strongly agree 5
1.	I think that I would like to use this gunshot classification system frequently.					
2.	I found the gunshot classification system unnecessarily complex.					
3.	I thought the gunshot classification system was easy to use.					
4.	I think that I would need the support of a technical person to be able to use this system.					
5.	I found the various features in this gunshot classification system were well integrated.					
6.	I thought there was too much inconsistency in the gunshot classification system.					
7.	I would imagine that most people would learn to use this system very quickly.					
8.	I found the gunshot classification system very cumbersome to use.					
9.	I felt very confident using the gunshot classification system.					
10.	I needed to learn a lot of things before I could get going with this gunshot classification system.					

Appendix B

时间戳记	Email:	I think th	I found t	I though	I think th	I found t	I though	I would i	I found t	I felt ver	I needed
6/17/2025 2:49:41	jameslau@	4	3	5	1	4	1	5	1	3	1
6/17/2025 8:18:28	78061@sis	5	1	5	1	5	1	5	1	5	1
6/17/2025 8:46:34	79242@sis	3	1	5	1	4	1	3	1	4	2
6/17/2025 8:47:21	79084@sis	3	2	4	2	5	2	5	2	5	4
6/17/2025 9:55:03	79642@sis	5	1	5	2	5	2	5	1	5	2
6/17/2025 10:27:29	malsonlee2	3	1	4	1	5	1	5	1	3	1
6/17/2025 10:28:12	milkbubbly	5	1	5	1	5	1	5	1	5	1
6/17/2025 10:29:59	meatan69@	3	2	4	3	5	1	3	1	4	2
6/17/2025 11:02:58	81800@sis	3	4	5	1	5	1	5	1	4	1
6/17/2025 11:06:38	79880@sis	4	3	5	1	4	1	5	1	3	1
6/17/2025 11:10:03	104778@s	5	1	5	1	4	1	5	1	4	1
6/17/2025 11:16:23	78363@sis	3	2	5	1	5	1	5	1	5	1
6/17/2025 11:25:24	79189@sis	4	1	5	1	4	2	5	1	3	1
6/17/2025 11:28:05	80003@sis	3	2	4	1	5	1	5	1	3	1
6/17/2025 11:31:06	77868@sis	2	4	5	1	4	1	5	1	4	1
6/17/2025 11:33:45	80121@sis	4	3	5	1	4	2	4	2	3	1
6/17/2025 11:35:20	80034@sis	5	3	5	1	5	2	5	1	4	1
6/17/2025 11:37:16	99284@sis	3	1	4	1	5	2	5	1	4	1
6/17/2025 11:40:18	78196@sis	5	1	4	1	5	1	5	1	5	1
6/17/2025 11:42:53	79802@sis	4	4	5	1	4	1	4	1	3	1
6/17/2025 11:47:37	81688@sis	5	1	5	1	5	1	5	1	5	1
6/17/2025 11:48:31	81466@sis	5	3	5	1	5	2	5	1	5	1
6/17/2025 11:49:52	79256@sis	5	3	5	1	4	1	4	2	5	1
6/17/2025 11:50:59	78236@sis	5	4	5	1	4	1	5	2	4	1
6/17/2025 11:51:47	79858@sis	5	4	3	1	4	2	5	2	4	1
6/17/2025 11:57:37	79861@sis	4	1	5	1	4	1	5	1	5	1
6/17/2025 12:54:40	yewyiting@	4	3	4	4	4	3	4	4	4	4

Appendix C

Testing for SVM (test 20 times)

No	Training Accuracy (%)	Testing Accuracy (%)
1	95.22	88.56
2	95.66	87.39
3	95.22	85.04
4	96.11	85.04
5	95.15	88.27
6	95.59	88.56
7	95.59	88.56
8	95.66	85.63
9	95.30	90.03
10	95.81	87.68
11	95.89	88.56
12	95.96	86.51
13	95.66	89.44
14	96.11	87.10
15	95.59	86.80
16	95.44	87.68
17	**95.59**	**93.26 (highest)**
18	96.18	89.15
19	95.30	89.74
20	95.66	86.51
Average	95.63	87.85

Testing for KNN (test 20 times)

No	Training Accuracy (%)	Testing Accuracy (%)
1	100	90.91
2	100	91.79
3	100	92.38
4	100	93.84
5	100	92.38
6	100	91.50
7	100	91.79
8	100	89.44
9	100	91.20
10	100	91.79
11	100	92.96
12	100	92.96
13	100	91.20
14	100	93.26
15	100	92.96
16	100	92.67
17	**100**	**96.77 (highest)**
18	100	91.50
19	100	93.26
20	100	90.32
Average	100	92.24

Testing for CNN (test 20 times)

No	Training Accuracy (%)	Testing Accuracy (%)
1	100	96.19
2	**100**	**96.77 (highest)**
3	100	95.31
4	100	95.31
5	100	95.60
6	100	95.01
7	100	95.89
8	100	94.72
9	100	94.43
10	100	95.60
11	100	95.01
12	100	95.89
13	100	95.60
14	100	95.31
15	100	95.01
16	100	94.43
17	100	95.31
18	100	95.31
19	100	96.19
20	100	94.43
Average	100	95.37

Appendix D

Source code of the project

requirement.txt

```
absl-py==2.2.2
asgiref==3.8.1
asttokens==3.0.0
astunparse==1.6.3
audioread==3.0.1
blinker==1.9.0
cachetools==5.5.2
certifi==2025.1.31
cffi==1.17.1
charset-normalizer==3.4.1
colorama==0.4.6
comm==0.2.2
contourpy==1.3.1
cycler==0.12.1
debugpy==1.8.13
decorator==5.2.1
Django==5.2
executing==2.2.0
flatbuffers==25.2.10
fonttools==4.57.0
gast==0.4.0
google-auth==2.40.3
google-auth-oauthlib==1.0.0
google-pasta==0.2.0
grpcio==1.71.0
h5py==3.13.0
idna==3.10
imbalanced-learn==0.13.0
imblearn==0.0
ipykernel==6.29.5
ipython==9.1.0
ipython_pygments_lexers==1.1.1
jedi==0.19.2
Jinja2==3.1.6
joblib==1.5.1
jupyter_client==8.6.3
jupyter_core==5.7.2
```

```
keras==2.14.0
kiwisolver==1.4.8
lazy_loader==0.4
libclang==18.1.1
librosa==0.11.0
llvmlite==0.44.0
Markdown==3.7
markdown-it-py==3.0.0
MarkupSafe==3.0.2
matplotlib==3.10.1
matplotlib-inline==0.1.7
mdurl==0.1.2
ml-dtypes==0.2.0
msgpack==1.1.0
namex==0.0.8
nest-asyncio==1.6.0
numba==0.61.0
numpy==1.24.4
oauthlib==3.2.2
opt_einsum==3.4.0
packaging==24.2
pandas==2.2.3
parso==0.8.4
pillow==11.1.0
platformdirs==4.3.7
pooch==1.8.2
prompt_toolkit==3.0.50
protobuf==4.25.8
psutil==7.0.0
pure_eval==0.2.3
pyasn1==0.6.1
pyasn1_modules==0.4.2
pycparser==2.22
Pygments==2.19.1
pyparsing==3.2.3
python-dateutil==2.9.0.post0
pytz==2025.2
pywin32==310
pyzmq==26.4.0
requests==2.32.3
requests-oauthlib==2.0.0
rich==14.0.0
rsa==4.9.1
```

```
scikeras==0.11.0
scikit-learn==1.3.2
scipy==1.15.3
seaborn==0.13.2
six==1.17.0
sklearn-compat==0.1.3
soundfile==0.13.1
soxr==0.5.0.post1
sqlparse==0.5.3
stack-data==0.6.3
tensorboard==2.14.1
tensorboard-data-server==0.7.2
tensorflow==2.14.0
tensorflow-estimator==2.14.0
tensorflow-intel==2.14.0
tensorflow-io-gcs-filesystem==0.31.0
termcolor==3.0.1
threadpoolctl==3.6.0
tornado==6.4.2
tqdm==4.67.1
traitlets==5.14.3
typing_extensions==4.13.1
tzdata==2025.2
urllib3==2.3.0
wcwidth==0.2.13
Werkzeug==3.1.3
wrapt==1.14.1
```

preprocess.py

```
import os
import numpy as np
import librosa
import glob
import joblib
import random
from datetime import datetime
from sklearn.preprocessing import LabelEncoder, StandardScaler
from tensorflow.keras.utils import to_categorical
from collections import defaultdict
import time

# ---------------------- CONFIGURATION ----------------------
# on PC Set the base directory for your project
DATA_DIR = "C:\\Users\\19990\\Desktop\\FYP
Project\\gunshot_project\\data\\Gunshot audio dataset"
PROCESSED_DATA_DIR = "C:\\Users\\19990\\Desktop\\FYP
Project\\gunshot_project\\processed_data"
MODEL_DIR = "C:\\Users\\19990\\Desktop\\FYP Project\\gunshot_project\\models"
LOG_DIR = "C:\\Users\\19990\\Desktop\\FYP Project\\gunshot_project\\log"
SRC_DIR = SRC_DIR = "C:\\Users\\19990\\Desktop\\FYP
Project\\gunshot_project\\src"#used for inference in Django app (views.py)

# Ensure directories exist
os.makedirs(PROCESSED_DATA_DIR, exist_ok=True)
os.makedirs(MODEL_DIR, exist_ok=True)
os.makedirs(LOG_DIR, exist_ok=True)

# Augmentation control
AUGMENTATION_ENABLED = True # Turn off if you want only original files

# Audio settings
SAMPLE_RATE = 44100     # Audio sampling rate
SEGMENT_DURATION = 2.0  # Segment length in seconds

log_file_path = os.path.join(LOG_DIR,
f"preprocessing_log_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt")

def log_message(message):
    """Prints and saves log messages to file."""
    print(message)
    with open(log_file_path, 'a', encoding='utf-8') as f:
```

```
        f.write(f"{datetime.now()} - {message}\n")

# ---------------------- AUDIO PROCESSING ----------------------
def pad_to_duration(audio, sr, duration):
    """Pad or truncate the audio to match the specified duration."""
    target_len = int(sr * duration)
    if len(audio) < target_len:
        return np.pad(audio, (0, target_len - len(audio)))
    return audio[:target_len]

def add_gaussian_noise(audio, mean=0, var=0.01):
    """Add Gaussian noise to the audio signal using a fresh random generator
each time."""
    rng = np.random.default_rng()  # Independent generator; not affected by
np.random.seed()
    noise = rng.normal(mean, np.sqrt(var), len(audio))
    return audio + noise

# ---------------------- FEATURE EXTRACTION (108D) ----------------------
def extract_features(audio, sr):
    """
    Extract a 108-dimensional feature vector using:
    - MFCC (13 x 3)
    - Chroma (12 x 3)
    - Spectral contrast (7 x 3)
    - Spectral centroid, ZCR, bandwidth, energy (each x 3)
    """
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
    chroma = librosa.feature.chroma_stft(y=audio, sr=sr)
    contrast = librosa.feature.spectral_contrast(y=audio, sr=sr)
    centroid = librosa.feature.spectral_centroid(y=audio, sr=sr)
    zcr = librosa.feature.zero_crossing_rate(y=audio)
    energy = np.array([np.sum(audio ** 2)])
    bandwidth = librosa.feature.spectral_bandwidth(y=audio, sr=sr)

    def summarize(feature):
        """Compute mean, std, median for a given feature array."""
        return [np.mean(feature), np.std(feature), np.median(feature)]

    features = []
    for f in [mfcc, chroma, contrast]:
        for row in f:
            features.extend(summarize(row))
```

```
    for f in [centroid, zcr, bandwidth]:
        features.extend(summarize(f[0]))

    features.extend(summarize(energy))  # energy is 1D array

    return np.array(features[:108])  # ensure it's 108D

# ---------------------- FILE PROCESSING ----------------------
def get_gun_classes(data_dir):
    """Return list of gun class subdirectories."""
    return sorted([f for f in os.listdir(data_dir) if
os.path.isdir(os.path.join(data_dir, f))])

def process_file(file_path, class_name):
    """
    Load and process an audio file:
    - Pad/truncate to duration
    - Optionally augment
    - Extract features
    - Save .npy feature file
    """
    results = []
    try:
        audio, sr = librosa.load(file_path, sr=SAMPLE_RATE)
        audio = pad_to_duration(audio, sr, SEGMENT_DURATION)

        base_name = os.path.splitext(os.path.basename(file_path))[0]
        class_output_dir = os.path.join(PROCESSED_DATA_DIR, class_name)
        os.makedirs(class_output_dir, exist_ok=True)

        # Handle original and noisy variants
        variants = [(audio, "")]
        if AUGMENTATION_ENABLED:
            variants.append((add_gaussian_noise(audio), "_noisy"))

        for variant_audio, suffix in variants:
            stat_feat = extract_features(variant_audio, sr)

            # Save feature
            np.save(os.path.join(class_output_dir,
f"{base_name}{suffix}_feat.npy"), stat_feat)
```

```
            results.append((stat_feat, class_name))

        log_message(f"[✓] {class_name} - {base_name}.wav processed")
        return results

    except Exception as e:
        log_message(f"[!] Failed to process {os.path.basename(file_path)}:
{e}")
        return []

# ---------------------- MAIN PIPELINE ----------------------
def run_pipeline():
    """Main preprocessing pipeline that:
    - Reads and processes audio files
    - Extracts 108D features
    - Applies label encoding and normalization
    - Saves encoders and returns processed dataset
    """
    start_time = time.time()
    log_message("🚀 Starting article-compliant preprocessing pipeline...")

    gun_classes = get_gun_classes(DATA_DIR)
    feat_list, label_list = [], []
    total_files = defaultdict(int)

    label_encoder = LabelEncoder()
    label_encoder.fit(gun_classes)

    # Process each file by class
    for class_name in gun_classes:
        class_path = os.path.join(DATA_DIR, class_name)
        for file in os.listdir(class_path):
            if file.endswith(".wav"):
                file_path = os.path.join(class_path, file)
                results = process_file(file_path, class_name)
                total_files[class_name] += len(results)
                for stat_feat, label in results:
                    feat_list.append(stat_feat)
                    label_list.append(label)

     # Label encoding + one-hot encoding
    encoded_labels = label_encoder.transform(label_list)
    categorical_labels = to_categorical(encoded_labels)
```

```
    # Convert to numpy arrays
    X_feat = np.array(feat_list, dtype=np.float32)
    y = categorical_labels

    # Normalize 108D features (shared input for all models)
    scaler = StandardScaler()
    X_feat_scaled = scaler.fit_transform(X_feat)

    # Save encoders for later use
    joblib.dump(scaler, os.path.join(MODEL_DIR, 'scaler.pkl'))
    joblib.dump(label_encoder, os.path.join(MODEL_DIR, 'label_encoder.pkl'))

    # Logging dataset stats
    log_message("📊 File counts per class:")
    for c, count in total_files.items():
        log_message(f"   {c}: {count} samples")

    log_message(f" Classes: {label_encoder.classes_}")
    log_message(f"📊 Feature shape (ALL models): {X_feat_scaled.shape}")
    log_message(f"🎯 Labels shape: {y.shape}")
    log_message(f"⏱️ Total Time: {time.time() - start_time:.2f} sec")
    log_message("☑️ Preprocessing finished.")

    return X_feat_scaled, y, label_encoder

# ---------------------- LOAD DATASET ----------------------
def load_dataset(data_dir=PROCESSED_DATA_DIR):
    """
    Loads existing *_feat.npy or *_mfcc.npy files. If missing, triggers
preprocessing.
    """
    # Check for either type of saved feature file
    feat_files = glob.glob(os.path.join(data_dir, '**', '*_feat.npy'),
recursive=True)
    mfcc_files = glob.glob(os.path.join(data_dir, '**', '*_mfcc.npy'),
recursive=True)

    if not feat_files and not mfcc_files:
        print("⚠️ No .npy feature files found. Running preprocessing...")
        return run_pipeline()

    print("📥 Found existing preprocessed features. Loading...")
```

```
    feat_list, label_list = [], []
    gun_classes = get_gun_classes(data_dir)

    label_encoder_path = os.path.join(MODEL_DIR, 'label_encoder.pkl')
    if not os.path.exists(label_encoder_path):
        raise FileNotFoundError("❌ Label encoder not found. Please rerun the
preprocessing pipeline.")

    label_encoder = joblib.load(label_encoder_path)

    for class_name in gun_classes:
        class_dir = os.path.join(data_dir, class_name)
        #for file in os.listdir(class_dir):
        for file in sorted(os.listdir(class_dir)):
            if file.endswith("_feat.npy") or file.endswith("_mfcc.npy"):
                try:
                    feat = np.load(os.path.join(class_dir, file))
                    feat_list.append(feat)
                    label_list.append(class_name)
                except Exception as e:
                    log_message(f"[!] Failed to load {file}: {e}")

    encoded_labels = label_encoder.transform(label_list)
    categorical_labels = to_categorical(encoded_labels)

    scaler_path = os.path.join(MODEL_DIR, 'scaler.pkl')
    scaler = joblib.load(scaler_path)

    X = scaler.transform(np.array(feat_list, dtype=np.float32))
    y = categorical_labels

    print("☑ Dataset loaded successfully.")
    print(f"📊 Features shape: {X.shape}")
    print(f"🏷 Labels shape: {y.shape}")
    return X, y, label_encoder

# ----- FOR INFERENCE (e.g., Django App) -----
def extract_features_from_file(file_path, scaler_path, encoder_path):
    """
    Extracts 108D features from an audio file using the same preprocessing
    and feature extraction steps used during training.
    """
```

```
    audio, sr = librosa.load(file_path, sr=SAMPLE_RATE)
    audio = pad_to_duration(audio, sr, SEGMENT_DURATION)

    # Extract features
    stat_feat = extract_features(audio, sr)

    # Load scaler and label encoder (if needed for inverse transform or
confidence display)
    scaler = joblib.load(scaler_path)
    label_encoder = joblib.load(encoder_path)

    # Normalize the feature
    stat_feat_scaled = scaler.transform([stat_feat])  # Keep batch shape (1,
108)

    return stat_feat_scaled, label_encoder


# ---------------------- MAIN ENTRY ----------------------
if __name__ == "__main__":
    run_pipeline()
```

svm.py

```
import os
import joblib
import time
import numpy as np
from datetime import datetime
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from preprocess import load_dataset, MODEL_DIR, LOG_DIR
import matplotlib.pyplot as plt
import seaborn as sns

# ---------------------- CONFIGURATION ----------------------
MODEL_PATH = os.path.join(MODEL_DIR, "svm_best_model.pkl")

# ---------------------- LOGGING ----------------------
log_file = os.path.join(LOG_DIR,
f"svm_log_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt")
def log(message):
    print(message)
    with open(log_file, 'a', encoding='utf-8') as f:
        f.write(f"{message}\n")

# ---------------------- LOAD DATA ----------------------
log("🔄 Loading preprocessed dataset...")
start_time = time.time()
X, y, label_encoder = load_dataset()
y_class = np.argmax(y, axis=1)
log(f"⏱️ Data loading time: {time.time() - start_time:.2f} seconds")

# ---------------------- TRAIN/VALIDATION SPLIT ----------------------
X_train, X_val, y_train, y_val = train_test_split(
    X, y_class, test_size=0.2, stratify=y_class)

# ---------------------- DEFINE & TRAIN FINAL MODEL ----------------------
log("🚀 Training final SVM model with best hyperparameters...")
start_time = time.time()
final_model = SVC(
    C=10,
    gamma=0.01,
    kernel='rbf',
```

```
    decision_function_shape='ovo',
    probability=True
)
final_model.fit(X_train, y_train)
log(f"⏱️ Model training time: {time.time() - start_time:.2f} seconds")

# ---------------------- FINAL EVALUATION ----------------------
start_time = time.time()
train_acc = accuracy_score(y_train, final_model.predict(X_train))
val_acc = accuracy_score(y_val, final_model.predict(X_val))
y_pred = final_model.predict(X_val)

log("\n📊 Final Accuracy Summary:")
log(f"☑️ Training Accuracy: {train_acc * 100:.2f}%")
log(f"☑️ Testing Accuracy: {val_acc * 100:.2f}%")

log("\n📋 Classification Report:")
report = classification_report(y_val, y_pred,
target_names=label_encoder.classes_)
log(report)
log(f"⏱️ Evaluation time: {time.time() - start_time:.2f} seconds")
# ---------------------- CONFUSION MATRIX ----------------------
cm = confusion_matrix(y_val, y_pred, normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(cm,
            annot=True,
            fmt=".2f",
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_,
            cmap='Blues')
plt.title("SVM Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45)
plt.tight_layout()
conf_matrix_path = os.path.join(LOG_DIR, "svm_confusion_matrix.png")
plt.savefig(conf_matrix_path)
plt.show()
log(f"🖼️ Confusion matrix saved to: {conf_matrix_path}")

# ---------------------- SAVE MODEL ----------------------
joblib.dump(final_model, MODEL_PATH)
log(f"💾 SVM model saved to: {MODEL_PATH}")
```

knn.py

```
import os
import joblib
import time
import numpy as np
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

from preprocess import load_dataset, MODEL_DIR, LOG_DIR

# ---------------------- CONFIGURATION ----------------------
MODEL_PATH = os.path.join(MODEL_DIR, "knn_best_model.pkl")
log_file = os.path.join(LOG_DIR,
f"knn_log_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt")

def log(message):
    print(message)
    with open(log_file, 'a', encoding='utf-8') as f:
        f.write(f"{message}\n")

# ---------------------- LOAD DATA ----------------------
log("🔄 Loading preprocessed dataset...")
start_time = time.time()
X, y, label_encoder = load_dataset()
y_class = np.argmax(y, axis=1)
log(f"⏱️ Data loading time: {time.time() - start_time:.2f} seconds")

# ---------------------- TRAIN/VALIDATION SPLIT ----------------------
X_train, X_val, y_train, y_val = train_test_split(
    X, y_class, test_size=0.2, stratify=y_class
)

# ---------------------- TRAIN FINAL KNN MODEL ----------------------
log("🚀 Training final KNN model with best hyperparameters...")

start_time = time.time()
knn_model = KNeighborsClassifier(
    n_neighbors=1,
```

```
    weights='uniform',
    metric='manhattan',  # Using Manhattan distance as per the best
configuration
)

final_model = knn_model
final_model.fit(X_train, y_train)
log(f"⏱️ Model training time: {time.time() - start_time:.2f} seconds")

# ---------------------- EVALUATION ----------------------
start_time = time.time()
train_acc = accuracy_score(y_train, final_model.predict(X_train))
val_acc = accuracy_score(y_val, final_model.predict(X_val))
y_pred = final_model.predict(X_val)

log("\n📊 Final Accuracy Summary:")
log(f"☑️ Training Accuracy: {train_acc * 100:.2f}%")
log(f"☑️ Testing Accuracy: {val_acc * 100:.2f}%")

log("\n📋 Classification Report:")
report = classification_report(y_val, y_pred,
target_names=label_encoder.classes_)
log(report)
log(f"⏱️ Evaluation time: {time.time() - start_time:.2f} seconds")

# ---------------------- CONFUSION MATRIX ----------------------
cm = confusion_matrix(y_val, y_pred, normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(cm,
            annot=True,
            fmt=".2f",
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_,
            cmap='Blues')
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45)
plt.tight_layout()
conf_matrix_path = os.path.join(LOG_DIR, "knn_confusion_matrix.png")
plt.savefig(conf_matrix_path)
plt.show()
log(f"🖼️ Confusion matrix saved to: {conf_matrix_path}")
```

```
# ---------------------- SAVE MODEL ----------------------
joblib.dump(final_model, MODEL_PATH)
log(f"💾 KNN model saved to: {MODEL_PATH}")
log("☑ KNN training complete with final configuration.")
```

cnn.py

```
import os
import numpy as np
import time
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
Dropout, BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import tensorflow as tf

from preprocess import load_dataset, MODEL_DIR, LOG_DIR

# ---------------------- CONFIGURATION ----------------------
MODEL_PATH = os.path.join(MODEL_DIR, "cnn_best_model.keras")

# ---------------------- LOGGING ----------------------
log_file = os.path.join(LOG_DIR,
f"cnn_log_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt")
def log(message):
    print(message)
    with open(log_file, 'a', encoding='utf-8') as f:
        f.write(f"{message}\n")

# ---------------------- LOAD DATA ----------------------
log("🔄 Loading preprocessed dataset...")
start_time = time.time()
X, y, label_encoder = load_dataset()
X = X.reshape(X.shape[0], X.shape[1], 1)
y_class = np.argmax(y, axis=1)
log(f"⏱️ Data loading time: {time.time() - start_time:.2f} seconds")

# ---------------------- TRAIN/VALIDATION SPLIT ----------------------
X_train, X_val, y_train, y_val = train_test_split(
    X, y_class, test_size=0.2, random_state=47, stratify=y_class)
```

```
# ---------------------- BUILD CNN MODEL ----------------------
def build_cnn_model(input_shape, num_filters=128, dropout_rate=0.3,
optimizer='rmsprop'):
    model = Sequential([
        tf.keras.Input(shape=input_shape),

        # 1st Convolutional Block
        Conv1D(num_filters, 3, activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling1D(2),
        Dropout(dropout_rate),

        # 2nd Convolutional Block
        Conv1D(num_filters * 2, 3, activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling1D(2),
        Dropout(dropout_rate),

        # 3rd Convolutional Block
        Conv1D(num_filters * 4, 3, activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling1D(2),
        Dropout(dropout_rate),

        # Fully Connected Layers
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(len(label_encoder.classes_), activation='softmax')

    ])

    model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

model = build_cnn_model(input_shape=(X.shape[1], 1))

# ---------------------- CALLBACKS ----------------------
early_stop = EarlyStopping(monitor='val_accuracy', patience=100,
restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(
```

```
    monitor='val_loss',
    factor=0.5,
    patience=50,
    min_lr=1e-6,
    verbose=1
)

# ---------------------- TRAIN MODEL ----------------------
log("🚀 Training CNN model...")
start_time = time.time()

history = model.fit(X_train, to_categorical(y_train),
                    validation_data=(X_val, to_categorical(y_val)),
                    epochs=500,
                    batch_size=32,
                    verbose=1,
                    callbacks=[early_stop, reduce_lr])

log(f"⏱️ Model training time: {time.time() - start_time:.2f} seconds")

# ---------------------- TRAINING HISTORY PLOTS ----------------------
log("📊 Plotting training history (combined figure)...")

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Plot 1: Training and Validation Accuracy
axes[0].plot(history.history['accuracy'], label='Training Accuracy',
color='blue')
axes[0].plot(history.history['val_accuracy'], label='Validation Accuracy',
color='green')
axes[0].axhline(y=0.95, color='gray', linestyle='--', linewidth=1.2,
label='Reference Line (95%)')
axes[0].set_title("CNN Training and Validation Accuracy", fontsize=14)
axes[0].set_xlabel("Epoch", fontsize=12)
axes[0].set_ylabel("Accuracy", fontsize=12)
axes[0].tick_params(labelsize=12)
axes[0].legend(fontsize=11)
axes[0].grid(True)

# Plot 2: Training and Validation Loss
axes[1].plot(history.history['loss'], label='Training Loss', color='blue')
axes[1].plot(history.history['val_loss'], label='Validation Loss',
color='orange')
```

```
axes[1].set_title("CNN Training and Validation Loss", fontsize=14)
axes[1].set_xlabel("Epoch", fontsize=12)
axes[1].set_ylabel("Loss", fontsize=12)
axes[1].tick_params(labelsize=12)
axes[1].legend(fontsize=11)
axes[1].grid(True)

plt.tight_layout()
combined_plot_path = os.path.join(LOG_DIR,
"cnn_training_history_combined.png")
plt.savefig(combined_plot_path, dpi=600)  # Save with higher DPI for report
quality
plt.show()
plt.close()

log(f"🖼️ Combined training history plot saved to: {combined_plot_path}")

# ---------------------- FINAL EVALUATION ----------------------
log("\n📊 Final Accuracy Summary:")
start_time = time.time()
train_acc = accuracy_score(y_train, np.argmax(model.predict(X_train), axis=1))
val_acc = accuracy_score(y_val, np.argmax(model.predict(X_val), axis=1))
y_pred = np.argmax(model.predict(X_val), axis=1)

log(f"☑️ Training Accuracy: {train_acc * 100:.2f}%")
log(f"☑️ Testing Accuracy: {val_acc * 100:.2f}%")

log("\n📋 Classification Report:")
report = classification_report(y_val, y_pred,
target_names=label_encoder.classes_)
log(report)
log(f"⏱️ Evaluation time: {time.time() - start_time:.2f} seconds")

# ---------------------- CONFUSION MATRIX ----------------------
cm = confusion_matrix(y_val, y_pred, normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(cm,
            annot=True,
            fmt=".2f",
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_,
            cmap='Blues')
plt.title("CNN Confusion Matrix")
```

```
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45)
plt.tight_layout()
conf_matrix_path = os.path.join(LOG_DIR, "cnn_confusion_matrix.png")
plt.savefig(conf_matrix_path)
plt.show()
plt.close()
log(f"🖼️ Confusion matrix saved to: {conf_matrix_path}")

# ---------------------- SAVE MODEL ----------------------
model.save(MODEL_PATH)
log(f"💾 CNN model saved to: {MODEL_PATH}")
```

svmtuner.py

```
import os
import joblib
import numpy as np
import time
from datetime import datetime
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from preprocess import load_dataset, MODEL_DIR, LOG_DIR
import matplotlib.pyplot as plt
import seaborn as sns

# ---------------------- CONFIGURATION ----------------------
MODEL_PATH = os.path.join(MODEL_DIR, "tuner_svm_best_model.pkl")

# ---------------------- LOGGING ----------------------
log_file = os.path.join(LOG_DIR,
f"svm_tunner_log_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt")
def log(message):
    print(message)
    with open(log_file, 'a', encoding='utf-8') as f:
        f.write(f"{message}\n")

# ---------------------- LOAD DATA ----------------------
log("🔄 Loading preprocessed dataset...")
X, y, label_encoder = load_dataset()
y_class = np.argmax(y, axis=1)

# ---------------------- TRAIN/VALIDATION SPLIT ----------------------
X_train, X_val, y_train, y_val = train_test_split(
    X, y_class, test_size=0.2, random_state=42, stratify=y_class)

# ---------------------- HYPERPARAMETER TUNING ----------------------
log("🔍 Starting SVM hyperparameter tuning with 5-fold cross-validation...")
start_time = time.time()
param_grid = {
    'C': [1, 10, 100],
    'gamma': [0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear', 'poly'],
    'decision_function_shape': ['ovo', 'ovr']
```

```
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid = GridSearchCV(SVC(), param_grid, cv=cv, scoring='accuracy', verbose=2,
n_jobs=-1)
grid.fit(X_train, y_train)
elapsed_time = time.time() - start_time
log(f"⏱️ Total search time: {elapsed_time:.2f} seconds")
log(f"\n☑️ Best Parameters: {grid.best_params_}")
log(f"📈 Cross-Validation Accuracy (on training data): {grid.best_score_ *
100:.2f}%")

# ---------------------- RETRAIN FINAL MODEL ON FULL TRAINING SET ------------
----------
best_params = grid.best_params_

final_model = SVC(
    C=best_params['C'],
    gamma=best_params['gamma'],
    kernel=best_params['kernel'],
    decision_function_shape=best_params['decision_function_shape']
)
final_model.fit(X_train, y_train)

# ---------------------- FINAL EVALUATION ----------------------
train_acc = accuracy_score(y_train, final_model.predict(X_train))
val_acc = accuracy_score(y_val, final_model.predict(X_val))
y_pred = final_model.predict(X_val)

log("\n📊 Final Accuracy Summary:")
log(f"☑️ Cross-Validation Accuracy (on training data): {grid.best_score_ *
100:.2f}%")
log(f"☑️ Training Accuracy (on full training set): {train_acc * 100:.2f}%")
log(f"☑️ Validation Accuracy (on hold-out set): {val_acc * 100:.2f}%")

log("\n📋 Classification Report:")
report = classification_report(y_val, y_pred,
target_names=label_encoder.classes_)
log(report)
# ---------------------- SAVE MODEL ----------------------
joblib.dump(final_model, MODEL_PATH)
log(f"💾 Best SVM model saved to: {MODEL_PATH}")
# ---------------------- END ----------------------
```

knntuner.py

```
import os
import joblib
import time
import numpy as np
from datetime import datetime
from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold
from sklearn.metrics import classification_report, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

from preprocess import load_dataset, MODEL_DIR, LOG_DIR

# ---------------------- CONFIGURATION ----------------------
MODEL_PATH = os.path.join(MODEL_DIR, "knn_best_tuned_model.pkl")
log_file = os.path.join(LOG_DIR,
f"knn_grid_search_log_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt")

def log(message):
    print(message)
    with open(log_file, 'a', encoding='utf-8') as f:
        f.write(f"{message}\n")

# ---------------------- LOAD DATA ----------------------
log("🔄 Loading preprocessed dataset...")
X, y, label_encoder = load_dataset()
y_class = np.argmax(y, axis=1)

# ---------------------- TRAIN/VALIDATION SPLIT ----------------------
seed = 42
X_train, X_val, y_train, y_val = train_test_split(
    X, y_class, test_size=0.2, random_state=seed, stratify=y_class
)

# ---------------------- HYPERPARAMETER TUNING ----------------------
param_grid = {
    'n_neighbors': [1, 3, 5, 7],
    'weights': ['uniform', 'distance'],
    'metric': ['manhattan', 'euclidean', 'cosine']
}
```

```
log("🚀 Performing GridSearchCV on training data...")
start_time = time.time()
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
grid = GridSearchCV(
    KNeighborsClassifier(),
    param_grid,
    cv=cv,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)
grid.fit(X_train, y_train)
elapsed_time = time.time() - start_time
log(f"⏱️ Total search time: {elapsed_time:.2f} seconds")

# ---------------------- EVALUATION ----------------------
best_model = grid.best_estimator_
val_preds = best_model.predict(X_val)
train_acc = accuracy_score(y_train, best_model.predict(X_train))
val_acc = accuracy_score(y_val, val_preds)

log(f"\n☑️ Best Parameters: {grid.best_params_}")
log(f"📈 Training Accuracy: {train_acc * 100:.2f}%")
log(f"📈 Testing Accuracy: {val_acc * 100:.2f}%")

log("\n📋 Classification Report:")
report = classification_report(y_val, val_preds,
target_names=label_encoder.classes_)
log(report)

# ---------------------- SAVE MODEL ----------------------
joblib.dump(best_model, MODEL_PATH)
log(f"💾 Best tuned model saved to: {MODEL_PATH}")
log("☑️ Hyperparameter tuning and evaluation complete.")
```

cnntuner5.py

```
import os
import numpy as np
import time
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import StratifiedKFold, GridSearchCV,
train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from scikeras.wrappers import KerasClassifier
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
Dropout, BatchNormalization
from preprocess import load_dataset, MODEL_DIR, LOG_DIR

# ---------------------- CONFIGURATION ----------------------
MODEL_PATH = os.path.join(MODEL_DIR, "cnn_best_model.keras")

# ---------------------- LOGGING ----------------------
log_file = os.path.join(LOG_DIR,
f"cnntuner_log_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt")
def log(message):
    print(message)
    with open(log_file, 'a', encoding='utf-8') as f:
        f.write(f"{message}\n")

# ---------------------- LOAD DATA ----------------------
log("🔄 Loading preprocessed dataset...")
start_time = time.time()
X, y, label_encoder = load_dataset()
X = X.reshape(X.shape[0], X.shape[1], 1)

# Extract integer class labels for StratifiedKFold
y_labels = np.argmax(y, axis=1)  # integer class labels (0, 1, 2, ...,
n_classes-1)

# ---------------------- TRAIN/VALIDATION SPLIT ----------------------
log("🔀 Splitting data into 80% train and 20% validation sets
(stratified)...")
X_train, X_val, y_train, y_val = train_test_split(
```

```
    X, y_labels, test_size=0.2, random_state=42, stratify=y_labels  # <-- stratify to preserve class distribution
)

log(f"⏱️ Data loading time: {time.time() - start_time:.2f} seconds")

# ---------------------- BUILD DYNAMIC CNN MODEL ----------------------
def build_cnn_model(optimizer='rmsprop', dropout_rate=0.3, num_filters=32, num_conv_layers=3, **kwargs):
    model = Sequential()
    model.add(tf.keras.Input(shape=(X.shape[1], 1)))

    max_filters = 512  # Cap to avoid memory explosion

    for i in range(num_conv_layers):
        filters = min(num_filters * (2 ** i), max_filters)
        model.add(Conv1D(filters, 3, activation='relu', padding='same'))
        model.add(BatchNormalization())
        model.add(MaxPooling1D(2))
        model.add(Dropout(dropout_rate))

    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(len(label_encoder.classes_), activation='softmax'))

    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# ---------------------- WRAP WITH SCIKERAS WRAPPER ----------------------
wrapped_model = KerasClassifier(
    model=build_cnn_model,
    optimizer='rmsprop',
    dropout_rate=0.3,
    num_filters=32,
    num_conv_layers=3,  # Default, GridSearch will override
    verbose=0,
    target_type="int"
)

# ---------------------- GRID SEARCH SETUP ----------------------
param_grid = {
```

```
    'optimizer': ['rmsprop', 'adam'],
    'dropout_rate': [0.3, 0.5],
    'num_filters': [32, 64, 128],
    'num_conv_layers': [2, 3, 4, 5, 6],  # <-- Dynamic 2 to 6 layers
    'fit__batch_size': [16, 32],
    'fit__epochs': [50]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid = GridSearchCV(estimator=wrapped_model,
                    param_grid=param_grid,
                    cv=cv,
                    scoring='accuracy',
                    verbose=2,
                    n_jobs=1)

# ---------------------- START GRID SEARCH ----------------------
log("🚀 Starting GridSearchCV with 5-Fold Cross-Validation...")
start_time = time.time()
grid_result = grid.fit(X_train, y_train)
log(f"⏱️ Total search time: {time.time() - start_time:.2f} seconds")

# ---------------------- GRID SEARCH RESULTS ----------------------
log(f"Best Parameters: {grid_result.best_params_}")
log(f"Best Cross-Validation Accuracy: {grid_result.best_score_ * 100:.2f}%")

# ---------------------- FINAL EVALUATION ON ALL DATA ----------------------
log("\n📊 Final Evaluation on Validation Set with Best Model:")
best_model = grid_result.best_estimator_.model_

y_pred = np.argmax(best_model.predict(X_val), axis=1)  # <-- Predict on
validation set
y_true = y_val  # <-- Validation true labels

accuracy = accuracy_score(y_true, y_pred)
log(f"☑️ Accuracy on entire dataset: {accuracy * 100:.2f}%")

log("\n📋 Classification Report:")
report = classification_report(y_true, y_pred,
target_names=label_encoder.classes_)
log(report)
```

```
# ---------------------- SAVE MODEL ----------------------
grid_result.best_estimator_.model_.save(MODEL_PATH)
log(f"💾 CNN best model saved to: {MODEL_PATH}")
```

views.py

```
import os
import sys
import time
import numpy as np
import joblib
from django.conf import settings
from django.shortcuts import render
from .forms import AudioUploadForm
from tensorflow.keras.models import load_model

# ----- Setup src path -----
SRC_PATH = os.path.abspath(os.path.join(os.path.dirname(__file__), '..', '..',
'gunshot_project', 'src'))
sys.path.append(SRC_PATH)

# ----- Import feature extraction -----
# Ensure preprocess.py is in the same directory as this script or adjust the
import accordingly
from preprocess import SRC_DIR, extract_features_from_file

# ----- Load artifacts -----
BASE_MODEL_DIR = os.path.abspath(os.path.join(settings.BASE_DIR, '..',
'gunshot_project', 'models'))

label_encoder = joblib.load(os.path.join(BASE_MODEL_DIR, 'label_encoder.pkl'))
scaler = joblib.load(os.path.join(BASE_MODEL_DIR, 'scaler.pkl'))

svm_model = joblib.load(os.path.join(BASE_MODEL_DIR, 'svm_best_model.pkl'))
knn_model = joblib.load(os.path.join(BASE_MODEL_DIR, 'knn_best_model.pkl'))
cnn_model = load_model(os.path.join(BASE_MODEL_DIR, 'cnn_best_model.keras'))

# Display name mapping
model_display_names = {
    'cnn': 'CNN',
    'knn': 'KNN',
    'svm': 'SVM',
}

# ----- Get Probabilities for Confidence Table -----
def get_model_confidences(model, input_data, model_type):
    if model_type == 'cnn':
        input_reshaped = np.reshape(input_data, (1, 108, 1))
```

```
        probs = model.predict(input_reshaped, verbose=0)[0]
    else:
        probs = model.predict_proba([input_data])[0]
    return probs  # numpy array of probabilities

# ----- Main View -----
def upload_audio(request):
    if request.method == 'POST':
        form = AudioUploadForm(request.POST, request.FILES)
        if form.is_valid():
            start_time = time.time()
            audio = form.cleaned_data['audio_file']
            model_choice = form.cleaned_data['model_choice']

            try:
                # Save the file
                os.makedirs(settings.MEDIA_ROOT, exist_ok=True)
                save_path = os.path.join(settings.MEDIA_ROOT, audio.name)
                with open(save_path, 'wb+') as f:
                    for chunk in audio.chunks():
                        f.write(chunk)

                print("📁 Audio file saved to:", save_path)
                print("🧠 Model selected:", model_choice)

                # Extract features
                features_scaled, _ = extract_features_from_file(
                    save_path,
                    os.path.join(BASE_MODEL_DIR, 'scaler.pkl'),
                    os.path.join(BASE_MODEL_DIR, 'label_encoder.pkl')
                )
                features_scaled = features_scaled[0]
                print("📊 Feature shape:", features_scaled.shape)

                gun_classes = label_encoder.classes_

                # Prediction based on model choice
                confidence_table = []

                if model_choice in ['svm', 'knn', 'cnn']:
                    models = {
                        'svm': svm_model,
                        'knn': knn_model,
```

```
                'cnn': cnn_model
            }
            model = models.get(model_choice)
            probs = get_model_confidences(model, features_scaled,
model_choice)
            final_index = np.argmax(probs)

            # Prepare confidence table
            for idx, gun_type in enumerate(gun_classes):
                confidence_table.append({
                    'gun_type': gun_type,
                    model_choice: f"{probs[idx]*100:.2f}%"
                })

        else:
            raise ValueError("Invalid model choice.")

        # Decode result
        result = label_encoder.inverse_transform([final_index])[0]

        # Map class names to image filenames
        gun_image_map = {
            "AK-12": "ak_12.jpg",
            "AK-47": "ak_47.jpg",
            "IMI Desert Eagle": "imi_desert_eagle.jpg",
            "M16": "m16.jpg",
            "M249": "m249.jpg",
            "MG-42": "mg_42.jpg",
            "MP5": "mp5.jpg",
            "Zastava M92": "zastava_m92.jpg",
        }

        # Get image path based on prediction
        image_file = gun_image_map.get(result, "default.jpg")
        gun_image_url = f"gun_images/{image_file}"

        elapsed = f"{time.time() - start_time:.2f} seconds"

        return render(request, 'result.html', {
            'result': result,
            'file_name': audio.name,
            'model_choice': model_choice,
```

```
                    'model_display_name':
model_display_names.get(model_choice, model_choice),  # 🟢 Pretty name
                    'elapsed': elapsed,
                    'file_url': settings.MEDIA_URL + audio.name,
                    'gun_image_url': gun_image_url,
                    'confidence_table': confidence_table,
                })

            except Exception as e:
                print("❌ Error occurred during processing:", e)
                return render(request, 'main.html', {
                    'form': form,
                    'result': f"⚠️ Error during processing: {str(e)}",
                })

    else:
        form = AudioUploadForm()

    return render(request, 'main.html', {
        'form': form,
    })
```

main.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Gun Recognition</title>
    <link rel="stylesheet" href="{% static 'CSS/main.css' %}">
    <script src="{% static 'JS/main.js' %}" defer></script>
</head>
<body>
<div class="container">
    <div class="overlay">
        {% if result %}
    <div class="error-message" style="color: red; font-weight: bold; margin-
top: 10px;">
        {{ result }}
    </div>
{% endif %}


        <h1>Gun Recognition</h1>
        <div class="info-text">
            <p>
                🔫 <strong>Supported Guns:</strong><br>
                AK-12, AK-47, IMI Desert Eagle, M16, M249, MG-42, MP5, Zastava
M92
            </p>
            <p style="color: #0077cc; font-weight: bold;">
                Only .wav audio files of 2 seconds or less are accepted.
            </p>
        </div>
        <form method="post" enctype="multipart/form-data" id="uploadForm">
            {% csrf_token %}

            <!-- Drop + Click Upload -->
            <div class="upload-box">
                <div class="drop-area" id="drop-area">
                    <input id="fileInput" name="audio_file" type="file"
accept=".wav">
                    <span class="upload-icon">&#8682;</span>
                    <p id="fileLabel">Click or Drop .wav file here</p>
                    <p class="hint">Supported: .wav</p>
```

```
                </div>
            </div>

            <!-- Model Select -->
            <div class="model-choice">
                <label for="model">Select Model:</label>
                <select name="model_choice" id="model">
                    <option value="">-- Choose Model --</option>
                    <option value="cnn">CNN</option>
                    <option value="knn">KNN</option>
                    <option value="svm">SVM</option>
                </select>
            </div>

            <!-- Buttons -->
            <div class="buttons">
                <button type="submit" class="confirm"
disabled>Confirm</button>
                <button type="button" class="reset" id="resetBtn"
disabled>Reset</button>
            </div>
        </form>

        <!-- Loading Spinner -->
        <div class="spinner" id="spinner">
            <div class="loader"></div>
            <p>Processing...</p>
        </div>
    </div>
</div>
</body>
</html>
```

main.css

```
/* Base styling */
html, body {
    height: 100%;
    margin: 0;
    padding: 0;
    background-color: #f2f2f2; /* Light background like result.html */
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    color: #333;
}

.container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100%;
    padding: 10px;
}

.overlay {
    background: #fff; /* White background */
    padding: 40px 50px;
    border-radius: 20px;
    width: 100%;
    max-width: 600px;
    text-align: center;
    box-shadow: 0 4px 16px rgba(0, 0, 0, 0.08); /* subtle shadow */
}

h1 {
    margin-bottom: 30px;
    font-size: 2.5em;
    font-weight: 600;
    letter-spacing: 1px;
}

.upload-box {
    margin-bottom: 25px;
}

.drop-area {
    position: relative;
    display: flex;
```

```
    flex-direction: column;
    justify-content: center;
    align-items: center;
    border: 2px dashed #aaa;
    border-radius: 15px;
    padding: 40px;
    cursor: pointer;
    transition: all 0.3s ease;
    background-color: #f9f9f9; /* lighter background */
    max-width: 400px;
    margin: 0 auto;
}

.drop-area input[type="file"] {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    opacity: 0;
    cursor: pointer;
    z-index: 10;
}

.drop-area:hover {
    border-color: #4CAF50;
}

.drop-area.dragover {
    border-color: #4CAF50;
    background-color: rgba(76, 175, 80, 0.1);
    transform: scale(1.02);
    box-shadow: 0 0 15px rgba(76, 175, 80, 0.3);
}

.drop-area.valid {
    animation: pulse-border 1s ease-in-out 1;
}

@keyframes pulse-border {
    0% { box-shadow: 0 0 0px rgba(76, 175, 80, 0.5); }
    50% { box-shadow: 0 0 15px rgba(76, 175, 80, 0.4); }
    100% { box-shadow: 0 0 0px rgba(76, 175, 80, 0.5); }
```

```
}

.upload-icon {
    font-size: 2.5em;
    margin-bottom: 10px;
}

#fileLabel {
    font-size: 1.1em;
    font-weight: 500;
    margin-bottom: 5px;
}

.hint {
    font-size: 0.85em;
    color: #666;
    background-color: #f2f2f2;
    padding: 4px 10px;
    border-radius: 6px;
}

.model-choice {
    margin-top: 20px;
    margin-bottom: 10px;
}

.model-choice label {
    display: block;
    margin-bottom: 6px;
    font-weight: 500;
}

select#model {
    padding: 8px 12px;
    border-radius: 6px;
    border: 1px solid #ccc;
    font-size: 1em;
    width: 100%;
    max-width: 300px;
    background-color: #fff;
    color: #333;
}
```

```css
.buttons {
    display: flex;
    justify-content: center;
    gap: 15px;
    margin-top: 20px;
    flex-wrap: wrap;
}

.confirm, .reset {
    padding: 10px 25px;
    border: none;
    border-radius: 6px;
    font-size: 1em;
    cursor: pointer;
    transition: background-color 0.2s ease;
}

.confirm {
    background-color: #4CAF50;
    color: white;
}

.reset {
    background-color: #e74c3c;
    color: white;
}

.confirm:hover {
    background-color: #45a049;
}

.reset:hover {
    background-color: #c0392b;
}

.confirm:disabled,
.reset:disabled,
select:disabled {
    opacity: 0.5;
    cursor: not-allowed;
}

.spinner {
```

```
    display: none;
    flex-direction: column;
    align-items: center;
    margin-top: 30px;
}

.loader {
    border: 5px solid #f3f3f3;
    border-top: 5px solid #4CAF50;
    border-radius: 50%;
    width: 40px;
    height: 40px;
    animation: spin 1s linear infinite;
    margin-bottom: 10px;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

@media (max-width: 600px) {
    .overlay {
        width: 90%;
        padding: 30px 20px;
    }

    h1 {
        font-size: 1.8em;
    }

    .drop-area {
        padding: 30px 20px;
    }

    .buttons {
        flex-direction: column;
    }

    .buttons button {
        width: 100%;
    }
}
```

```css
.info-text {
    text-align: center;
    font-size: 0.95em;
    color: red;
}
```

main.js

```
document.addEventListener('DOMContentLoaded', () => {
    const dropArea = document.getElementById('drop-area');
    const fileInput = document.getElementById('fileInput');
    const fileLabel = document.getElementById('fileLabel');
    const resetBtn = document.getElementById('resetBtn');
    const confirmBtn = document.querySelector('.confirm');
    const modelSelect = document.getElementById('model');
    const spinner = document.getElementById('spinner');
    const form = document.getElementById('uploadForm');

    // Click on drop area triggers file dialog
    dropArea.addEventListener('click', (e) => {
        if (e.target !== fileInput) {
            fileInput.click();
        }
    });

    // Highlight drop area on drag
    dropArea.addEventListener('dragover', e => {
        e.preventDefault();
        dropArea.classList.add('dragover');
    });

    dropArea.addEventListener('dragleave', () => {
        dropArea.classList.remove('dragover');
    });

    // File dropped into area
    dropArea.addEventListener('drop', e => {
        e.preventDefault();
        dropArea.classList.remove('dragover');

        const files = e.dataTransfer.files;
        if (files.length !== 1 || !files[0].name.endsWith('.wav')) {
            alert('Please drop a single .wav file.');
            return;
        }

        const dt = new DataTransfer();
        dt.items.add(files[0]);
        fileInput.files = dt.files;
```

```
        checkAudioDuration(files[0]);
    });

    // File selected through browse
    fileInput.addEventListener('change', () => {
        const file = fileInput.files[0];
        if (!file || !file.name.endsWith('.wav')) {
            alert('Only .wav files are supported.');
            resetForm();
            return;
        }
        checkAudioDuration(file);
    });

    // Check audio duration (must be <= 2 seconds)
    function checkAudioDuration(file) {
        const audioEl = document.createElement('audio');
        audioEl.preload = 'metadata';
        audioEl.src = URL.createObjectURL(file);
        audioEl.onloadedmetadata = function() {
            URL.revokeObjectURL(audioEl.src);
            if (audioEl.duration > 2.5) { // allow up to 2.5 seconds for
safety
                alert('Audio must be 2 seconds or shorter.');
                resetForm();
            } else {
                updateUIAfterFileSelect(file.name);
            }
        };
    }

    // Reset button
    resetBtn.addEventListener('click', resetForm);

    // Show loading spinner on submit
    form.addEventListener('submit', () => {
        spinner.style.display = 'flex';
    });

    // Check model selection before submit
    confirmBtn.addEventListener('click', e => {
        if (!fileInput.files.length) {
            e.preventDefault();
```

```
            alert('Please upload your audio file first.');
        } else if (!modelSelect.value) {
            e.preventDefault();
            alert('Please choose the model first.');
        }
    });

    // Enable form after valid file upload
    function updateUIAfterFileSelect(fileName) {
        fileLabel.textContent = fileName;
        modelSelect.disabled = false;
        confirmBtn.disabled = false;
        resetBtn.disabled = false;
    }

    // Reset all
    function resetForm() {
        form.reset();
        fileLabel.textContent = 'Click or Drop .wav file here';
        modelSelect.disabled = true;
        confirmBtn.disabled = true;
        resetBtn.disabled = true;
        spinner.style.display = 'none';
    }

    // Initial disable
    resetForm();
});
```

result.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Gun Recognition Result</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{% static 'CSS/result.css' %}">
</head>
<body>
<div class="container">
    <div class="result-container"> <!-- Wrap the two panels -->

        <div class="left-panel">
            {% if gun_image_url %}
            <div class="image-panel">
                <img src="{% static gun_image_url %}"
alt="{{ result|slugify }} image">
            </div>
            {% endif %}

            <div class="info-panel">
                <h2>Gun Recognition Result</h2>

                <div class="info-group">
                    <p><strong>🗀 File Name:</strong> {{ file_name }}</p>
                    <p><strong>🤖 Model Used:</strong>
{{ model_display_name }}</p>
                    <p class="highlight"><strong>🔫 Predicted Gun
Type:</strong> {{ result }}</p>
                    <p><strong>⏱ Processing Time:</strong> {{ elapsed }}</p>
                </div>

                <div class="audio-group">
                    <p><strong>🎵 Audio Preview:</strong></p>
                    <audio controls>
                        <source src="{{ file_url }}" type="audio/wav">
                        Your browser does not support audio playback.
                    </audio>
                </div>

                <div class="actions">
```

```
                    <a href="{% url 'upload_audio' %}" class="button">← Back
to Main</a>
                </div>
            </div>
        </div> <!-- End of Left Panel -->

        {% if confidence_table %}
        <div class="right-panel">
            <div class="confidence-table">
                <h3>Prediction Confidence Details</h3>
                <div class="table-wrapper">
                    <table>
                        <thead>
                            <tr>
                                <th>Gun Type</th>
                                {% if confidence_table.0.svm %}<th>SVM
Confidence %</th>{% endif %}
                                {% if confidence_table.0.knn %}<th>KNN
Confidence %</th>{% endif %}
                                {% if confidence_table.0.cnn %}<th>CNN
Confidence %</th>{% endif %}
                            </tr>
                        </thead>
                        <tbody>
                            {% for row in confidence_table %}
                            <tr>
                                <td>{{ row.gun_type }}</td>
                                {% if row.svm %}<td>{{ row.svm }}</td>{%
endif %}
                                {% if row.knn %}<td>{{ row.knn }}</td>{%
endif %}
                                {% if row.cnn %}<td>{{ row.cnn }}</td>{%
endif %}
                            </tr>
                            {% endfor %}
                        </tbody>
                    </table>
                </div>
            </div>
        </div> <!-- End of Right Panel -->
        {% endif %}

    </div> <!-- End of result-container -->
```

```
</div> <!-- End of container -->
</body>
</html>
```

result.css

```css
/* Base Styling */
body {
    margin: 0;
    padding: 0;
    font-family: "Segoe UI", Tahoma, sans-serif;
    background-color: #f2f2f2;
    color: #333;
    height: 100vh;
    display: flex; /* Center horizontally */
    justify-content: center;
    align-items: center;
}

.container {
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 30px;
    height: auto;
}

/* Left panel */
.left-panel {
    flex: 1;
    padding: 25px;
    border-right: 1px solid #eee;
}

.image-panel {
    padding-bottom: 15px;
    text-align: center;
}

.image-panel img {
    width: 280px; /* fixed width */
    max-height: 280px;
    height: auto; /* maintain aspect ratio */
    border-radius: 6px;
    object-fit: contain;
    display: block;
    margin: 0 auto;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15); /* 🌑 nice soft shadow */
```

```
}

.info-panel h2 {
    font-size: 2em;
    margin-bottom: 20px;
    font-weight: 700;
    text-align: center;
}

.info-group p {
    margin: 6px 0;
    font-size: 1em;
}

.info-group .highlight {
    color: #007b5e;
    font-weight: bold;
    font-size: 1.1em;
}

.audio-group {
    margin-top: 15px;
}

audio {
    width: 100%;
    height: 32px;
}

.actions {
    margin-top: 15px;
    text-align: center;
}

.button {
    background-color: #007b5e;
    color: #fff;
    padding: 9px 18px;
    font-size: 1em;
    text-decoration: none;
    border-radius: 5px;
    display: inline-block;
}
```

```css
.button:hover {
    background-color: #005b47;
}

/* Right panel */
.right-panel {
    flex: 1;
    padding: 25px;
}

.confidence-table {
    margin-top: 20px;
    text-align: center;
}

.confidence-table h3 {
    text-align: center;
    margin-bottom: 15px;
    font-size: 1.3em;
}

/* Scrollable Table */
.table-wrapper {
    overflow-x: auto;
}

.confidence-table table {
    width: auto;         /* ⬅ Smaller width inside the panel */
    margin: 0 auto;     /* ⬅ Center the table */
    min-width: 400px;  /* ⬅ Optional: reduce min-width if you want */
    border-collapse: collapse;
    font-size: 1.05em; /* ⬅ Slightly smaller font */
}

.confidence-table th, .confidence-table td {
    border: 1px solid #ddd;
    padding: 10px;
    text-align: center;
}

.confidence-table th {
    background-color: #007b5e;
```

```
    color: white;
    font-weight: 600;
}

.confidence-table tr:nth-child(even) {
    background-color: #f9f9f9;
}

.confidence-table tr:hover {
    background-color: #f1f1f1;
}

.result-container {
    display: flex;
    flex-direction: row;
    background: #fff;
    border-radius: 12px;
    box-shadow: 0 4px 16px rgba(0, 0, 0, 0.08);
    overflow: hidden;
    max-width: 2000px;
    width: 90%;
}

.left-panel {
    flex: 6; /* 🟢 Left panel will take 60% */
    padding: 40px;
    border-right: 1px solid #eee;
}

.right-panel {
    flex: 4; /* 🟢 Right panel will take 40% */
    padding: 40px;
}
```