**Deep Inspect**

**(Deepfake Audio Detection and Data Collection)**



**By:**

**Talal Manshoor**

**18317**

**Noor ul Islam Huzaifa**

**18680**

**Supervised by:**

**Prof Ubaid Ullah Aleem**

**Riphah College of Computing Riphah International University, Faisalabad Campus Spring 2023**

**A Project Documentation Submitted to**

**Riphah College of Computing**

**Riphah International University, Faisalabad Campus**

**As a Partial Fulfillment of the Requirement for the Award**

**of the Degree of**

**Bachelors of Science in Computer Science**

**Riphah College of Computing**

**Riphah International University, Faisalabad Campus**

**Final Approval**

This is to certify that we have read the report submitted by ***Talal Manshoor (18317) and Noor ul Islam Huzaifa (18680)*** for the partial fulfillment of the requirements for the degree of the Bachelor of Science in Computer Science BSCS. It is our judgment that this report is of sufficient standard to warrant its acceptance by Riphah International University, Faisalabad Campus for the degree of Bachelor of Science in Computer Science BSCS.

**Committee:**

**1**

Prob Ubaid Ullah Aleem

(Supervisor)

**2**

Principal

Riphah College of Computing

**Declaration**

We hereby declare that this document **Deep Inspect** **(Deepfake Audio detection and Data Collection)** neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this project with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers especially our supervisor **Prof Ubaid Ullah Aleem**. If any part of the system is proved to be copied out from any source or found to be reproduction of any project from anywhere else, we shall stand by the consequences.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Talal Manshoor**

**18317**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Noor Ul Islam Huzaifa**

**18680**

**Dedication**

This work is dedicated to our mentors, parents, and especially to our supervisor, Prof. Ubaid Ullah Aleem, who have always been there to guide us through thick and thin. Their unwavering support and encouragement have made these innovations possible. Without them, none of this would have been achievable.

To our mentors, whose wisdom and guidance have helped us reach this point in our life and guided us in every aspect.

To our parents, who have been our pillars of strength and our constant source of inspiration.

Your faith in us has given us the confidence to pursue our dreams with determination.

To Prof. Ubaid Ullah Aleem, whose exceptional supervision and commitment have been instrumental in our journey, thank you for your patience, knowledge, and encouragement. Your dedication to our success has been a driving force behind our achievements.

We are profoundly grateful to each one of you for your unconditional support, without which this work would not have been possible.

**Acknowledgment**

With the grace of Allah Almighty, the guidance of our respectable instructors, and the support of our parents, we present you today, our Final Year Project. Without the motivation and encouragement of our kind Sir, Mr. Ubaid Ullah, we would not have been able to reach this point. He had been a guiding light in this complex journey. We could not have imagined having a better advisor and mentor for our BSCS Project.

In addition to hard work, this project was completed thanks to the proper guidance of our mentors, their timely availability, the assistance of our fellows, and the unwavering support of our parents. We would also like to thank the rest of the committee members for their perceptive comments and encouragement. We are highly thankful to everyone who has guided us on this journey. Your support and encouragement have been invaluable, and we are deeply grateful for the collective effort that has brought us to this milestone.

This experience has not only enhanced our technical skills but also taught us the importance of collaboration, perseverance, and the strength of a supportive community. Our work has enabled us to reach the diversity of knowledge. Working on this project has opened our hearts to new ideas and innovations. We encountered numerous technical and logistical challenges and dealt with unexpected setbacks that required quick problem solving. Trying to solve problems on our own and finding solutions to different obstacles has helped us recognize the challenges the future holds for us. We committed countless hours to research and development. Every challenge was an opportunity to learn and improve. The difficulties we faced strengthened our resolve and teamwork. We gained extensive knowledge and expertise in our respective fields. This project pushed us beyond our comfort zones and helped us mature professionally. We are grateful for the opportunity to grow and achieve new milestones.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Talal Manshoor**

**18317**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Noor Ul Islam Huzaifa**

**18680**

**Abstract**

In today's data-driven world, the rise of deepfake technology poses grave threats across domains like cybersecurity, politics, and privacy. While progress has been made in detecting deepfake videos, identifying synthetic deepfake audio remains a formidable challenge, especially in scenarios demanding low latency and high accuracy.

This project tackles the complexities of deep fake audio detection head-on, proposing an innovative approach powered by a Support Vector Machine (SVM) classifier. The SVM is trained on a meticulously crafted combination of traditional acoustic features like Mel-Frequency Cepstral Coefficients (MFCCs), Linear Prediction Cepstral Coefficients (LPCCs), and specialized features including Voicing Onset Time, Onset Periodicity, and Burst Time. These features are extracted from the diverse Fake Or Real dataset, comprising both deepfake and authentic audio samples.

Through a comprehensive analysis, encompassing implementation details, performance evaluation, and comparative studies, this thesis demonstrates the remarkable efficacy of the proposed method. The results are compelling, achieving an impressive accuracy in detecting deepfake audio, underscoring its potential as a powerful safeguard against the misuse of this technology in audio-based communications.

**List of abbreviation**

|  |  |
| --- | --- |
|  |  |
| AI | Artificial Intelligence |
| ANN | Artificial Neural Networks |
| CNN | Conventional Neural Networks |
| CPU | Central Processing Unit |
| CSS | Cascading Stylesheet |
| DL | Deep Learning |
| DT | Decision Tree |
| GB | Gradient Boosting |
| HTML | Hyper Text Markup Language |
| JS | JavaScript |
| LR | Logistic Regression |
| LSTM | Long Short - Term Memory |
| MB | Mega Byte |
| ML | Machine Learning |
| NB | Naïve Bayes |
| NBM | Naïve Bayes Multinomial |
| NCPR | National Computer Preference Register |
| NLP | Natural Language Processing |
| NLTK | Natural Language Toolkit |
| OS | Operating System |
| QR | Quick Response |
| SGB | Stochastic Gradient Boosting |
| SMS | Short Message Service |
| SPIM | Spam on Instant Messaging |
| SSL | Secure Socket Layer |
| SVM | Support Vector Machine |
| TF-IDF | Term Frequency – Inverse Document Frequency |
| UI | User Interface |
| UML | Unified Modeling Language |

Table of Contents

[Chapter 1 4](#_Toc167430870)

[1. Introduction 4](#_Toc167430871)

[1.1 Background 4](#_Toc167430872)

[1.2 Motivations and Challenges 4](#_Toc167430873)

[1.3 Goals and Objectives 5](#_Toc167430874)

[1.4 Solution Overview 5](#_Toc167430875)

[Chapter 2 7](#_Toc167430876)

[2. Literature Review 7](#_Toc167430877)

[2.1 Introduction 7](#_Toc167430878)

[2.1 Architectures Overview 8](#_Toc167430879)

[2.3 Summary 16](#_Toc167430880)

[Chapter 3 17](#_Toc167430881)

[3. Requirements Analysis 17](#_Toc167430882)

[3.1 Introduction 17](#_Toc167430883)

[3.2 Problem Scenarios 17](#_Toc167430884)

[3.3 Functional Requirements 18](#_Toc167430885)

[3.3.1 Chrome or Alternative Extension Browser 18](#_Toc167430886)

[3.3.2 Recording Functionality 18](#_Toc167430887)

[3.3.3 File Upload 18](#_Toc167430888)

[3.3.4 Audio Processing 18](#_Toc167430889)

[3.3.5 API Integration 18](#_Toc167430890)

[3.3.6 User Interface (UI) 19](#_Toc167430891)

[3.3.7 Error Handling 19](#_Toc167430892)

[3.3.8 Security 19](#_Toc167430893)

[3.3.9 Audio Processing Functions 19](#_Toc167430894)

[3.3.10 Prediction Functions 19](#_Toc167430895)

[3.3.11 Batch Analysis 19](#_Toc167430896)

[3.3.12 API Endpoints 19](#_Toc167430897)

[3.3.13 Logging and Error Handling 20](#_Toc167430898)

[3.3.14 File Management 20](#_Toc167430899)

[3.4 Non-Functional Requirements 20](#_Toc167430900)

[3.4.1 Look and Feel 20](#_Toc167430901)

[3.4.2 Usability 20](#_Toc167430902)

[3.4.3 Accessibility 20](#_Toc167430903)

[3.4.4 Performance Requirements 20](#_Toc167430904)

[3.4.5 Operational and Environmental Requirement 21](#_Toc167430905)

[3.4.6 Maintainability and Support Requirement 21](#_Toc167430906)

[3.4.7 Security Requirement 21](#_Toc167430907)

[3.4.8 Legal Requirements 21](#_Toc167430908)

[Chapter 4 22](#_Toc167430909)

[4. System Design 22](#_Toc167430910)

[4.1 Introduction 22](#_Toc167430911)

[4.2 Architect Diagram 22](#_Toc167430912)

[4.3 Data Flow Diagram 22](#_Toc167430913)

[4.4 Use Case Diagram 23](#_Toc167430914)

[4.5 Activity Diagram 24](#_Toc167430915)

[4.6 Model Training Diagram 25](#_Toc167430916)

[4.7 Model Inference Diagram 26](#_Toc167430917)

[Implementation 29](#_Toc167430918)

[5.1 Flow Control 29](#_Toc167430919)

[5.2 Components, Libraries, Web Services 29](#_Toc167430920)

[5.3 Code Layer 30](#_Toc167430921)

[5.3.1 Presentation Layer 31](#_Toc167430922)

[5.3.2 Data Management Layer 34](#_Toc167430923)

[6.1 Introduction 38](#_Toc167430924)

[6.2 List of Test Scenarios 38](#_Toc167430925)

[6.3 Performance and Evaluation 42](#_Toc167430926)

[4.7.1 6.3.1 Functional Testing 42](#_Toc167430927)

[4.7.2 6.3.2 Non-Functional Testing 44](#_Toc167430928)

[7.1 Introduction 46](#_Toc167430929)

[7.2 Achievements and Improvements 46](#_Toc167430930)

[7.3 Critical Review 47](#_Toc167430931)

[7.4 Future Recommendations 47](#_Toc167430932)

**Table of Figures**

[Figure 1 Features Rating 14](#_Toc167417044)

[Figure 2 Architect Diagram 21](#_Toc167417045)

# Chapter 1

# Introduction

## 1.1 Background

In recent years, the rapid advancement of deep learning and generative models has given rise to a technology known as "deepfakes." Deepfakes refer to synthetic media, such as audio, video, or images, that are generated using artificial intelligence techniques to manipulate or create content that can convincingly mimic real individuals. While deepfakes have legitimate applications in entertainment, animation, and accessibility, their potential for misuse has raised significant concerns.

One particularly concerning aspect of deepfake technology is the ability to generate highly realistic synthetic audio that can impersonate individuals' voices with remarkable accuracy. Malicious actors can exploit deepfake audio for various nefarious purposes, including spreading misinformation, committing fraud, impersonating public figures or celebrities, or even engaging in identity theft and extortion.

The consequences of deepfake audio misuse can be far-reaching and severe, undermining trust in audio-based communication channels, compromising personal privacy, and potentially leading to financial losses or reputational damage. As such, the development of effective deepfake audio detection methods have become a critical area of research and a top priority for organizations and individuals alike.

## 1.2 Motivations and Challenges

The primary motivation behind this research is the increasing prevalence of deepfake audio and the urgent need to mitigate the risks associated with its potential misuse. While various detection algorithms have been proposed, real-time detection of deepfake audio presents several challenges that must be addressed.

Firstly, real-time detection requires processing and analyzing audio streams as they are being generated or received, imposing stringent computational constraints and low latency requirements. Traditional detection methods often struggle to meet these demands, making them unsuitable for real-time applications.

Secondly, the diversity of deepfake audio generation techniques, such as voice conversion, speech synthesis, and voice cloning, each with unique characteristics and artifacts, presents a significant challenge. Developing a single detection method that can effectively identify all types of deepfake audio is a daunting task.

Thirdly, the availability of high-quality deepfake audio datasets for training and evaluating detection models is limited, as generating and labeling such data can be labor-intensive and time-consuming. This lack of representative data can hinder the development and performance of robust detection models.

## 1.3 Goals and Objectives

This project aims to address the challenges in low latency deepfake audio detection by proposing and evaluating a novel approach and in the usage of such models for general public by providing a chrome extension. The specific objectives are:

* To conduct a comprehensive literature review and identify the limitations of current low latency deepfake audio detection methods.
* To propose a novel method for low latency deepfake audio detection that leverages Support Vector Machines (SVMs) and a combination of traditional acoustic features and specialized features, such as Voicing Onset Time, Onset Periodicity, and Burst Time.
* To implement and evaluate the proposed solution using the Fake Or Real dataset, assessing its performance in terms of accuracy, latency, and computational efficiency in such scenarios.
* To compare the proposed approach with existing techniques and highlight its advantages and trade-offs.
* To have an easy to use extension to use this model in the real world.
* To make this model deployable on servers and clouds to utilize this model.

## 1.4 Solution Overview

### Solution Overview

The proposed solution in this thesis involves the development of a low latency deepfake audio detection system based on a Support Vector Machine (SVM) classifier. This system integrates a Chrome extension for audio recording and uploading, a Python backend for data processing, and a comprehensive AI/ML model for detection.

The Chrome extension provides a user-friendly interface for recording or uploading WAV files, facilitating seamless communication with the backend for data transmission. The Python backend is responsible for managing file uploads and extracting essential audio features from the collected files. It processes the data from these uploaded audio files, which are crucial for both training the model and making predictions.

For data collection, the system utilizes the Fake Or Real dataset, comprising both deepfake and real audio samples. Traditional acoustic features such as Mel-Frequency Cepstral Coefficients (MFCCs) and Linear Prediction Cepstral Coefficients (LPCCs) are extracted. Additionally, specialized features like Voicing Onset Time, Onset Periodicity, and Burst Time are used to capture the subtle anomalies present in deepfake audio, which enhances the detection capabilities.

The feature extraction process is integral to training the SVM classifier. By combining various audio features, the SVM leverages its discriminative power to effectively distinguish between real and deepfake audio samples. The system is optimized to maintain low latency, ensuring that it is computationally efficient and suitable for real-time applications.

In terms of performance, the system is evaluated using standard accuracy metrics to measure its detection effectiveness. A detailed computational complexity analysis is conducted to ensure that the system can operate efficiently in real-time. Furthermore, a comparative study with existing deepfake detection techniques highlights the improvements in both accuracy and efficiency offered by this approach.

The successful development and evaluation of this low latency deepfake audio detection system, Deep Inspect, have the potential to make significant contributions to the field of cybersecurity. By providing a practical and efficient solution for detecting deepfake audio, this project aims to mitigate the risks associated with the misuse of deepfake technology in audio-based communication channels.

# Chapter 2

# Literature Review

This chapter explains about the classification of Deepfakes and how the idea of Deepfake and its detection came into being. It highlights different Deepfakes audio and its classification models, developed through the ages, and the problems tackled by others. The developments till Deepfake in instant voice messaging and calls, which is the main emphasis of this research, are given.

## 2.1 Introduction

The advent of deepfake technology, particularly in the realm of audio, has revolutionized the creation of synthetic media. Deepfake audio leverages advanced neural networks and generative models to produce highly realistic voice recordings that mimic the speech patterns, tone, and intonation of real individuals. While this technology has legitimate applications in entertainment, accessibility, and localization, it also poses significant threats when used maliciously. The potential misuse of deepfake audio includes spreading misinformation, committing fraud, impersonating public figures, and engaging in identity theft, all of which can have severe societal and financial consequences.

The rise of deepfake audio has introduced significant concerns regarding the integrity and trustworthiness of audio-based communication. As generative models such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) continue to evolve, the quality of deepfake audio improves, making it increasingly difficult to distinguish between real and synthetic audio. This advancement necessitates the development of robust detection methods to ensure the authenticity of audio communications.

Deploying deepfake detection algorithms in low latency environments adds a layer of complexity. Low latency detection requires algorithms to process and analyze audio streams quickly and accurately, ensuring minimal latency and maintaining the integrity of communication channels. Additionally, the variability in audio quality due to factors such as background noise, recording equipment, and transmission quality further complicates the detection process. Detection algorithms must be robust enough to handle these variations while maintaining high accuracy.

The development and benchmarking of deepfake audio detection algorithms rely heavily on the availability of consistent and comprehensive datasets. However, the variability in datasets can hinder the training and evaluation of these algorithms, making it challenging to achieve reliable and generalizable performance. Consistent datasets that encompass a wide range of speakers, languages, and recording conditions are crucial for training models that can generalize well across different scenarios.

## Architectures Overview

Deep fake audio can be generated through various sophisticated techniques, each presenting unique challenges for detection. Voice conversion techniques, for instance, modify the speaker's identity while maintaining the original linguistic content. This allows the audio to sound like it is coming from a different person without altering the words spoken. Speech synthesis techniques, including text-to-speech (TTS) and voice cloning, generate speech from text or limited training data, creating entirely new audio segments. Additionally, audio manipulation techniques such as splicing and inpainting involve altering or combining existing audio samples to produce a convincing fake. Each of these methods exploits different aspects of the audio signal, resulting in distinct artifacts and distortions, which complicate the detection efforts[1], [2].

Traditional feature extraction methods, like Mel-frequency cepstral coefficients (MFCCs), often fall short in capturing the complex patterns inherent in deep fake audio. Although deep learning models can learn more discriminative features, they are prone to encoding biases from the training data, which limits their generalizability. Developing a comprehensive set of features that can effectively distinguish deep fake audio across various techniques and scenarios is a significant challenge that remains unresolved[1], [2].

Deep fake audio detection methods often struggle to generalize to unseen techniques, different audio domains, or varying environmental conditions such as noise and reverberation. Adversarial attacks, where fake audio is specifically crafted to evade detection, further threaten the robustness of current methods. Achieving both generalization and robustness necessitates extensive training data, diverse evaluation scenarios, and advanced techniques such as domain adaptation and adversarial training. These measures are essential to enhance the reliability and effectiveness of deep fake audio detection systems in real-world applications[1], [2].

Another critical challenge lies in the lack of robust datasets. Current datasets used to train deepfake detection models often suffer from limitations in diversity, size, and real-world applicability. Concerns around privacy and ethical considerations further complicate the collection of large-scale speech data. This lack of standardized datasets and evaluation protocols makes it difficult to compare and benchmark different detection methods effectively, hindering progress in the field. Finally, traditional signal processing techniques, while historically valuable, often fall short in the face of sophisticated deepfake generation methods. Techniques like spectral analysis and pitch-based features struggle to capture the increasingly complex patterns embedded in deepfakes. These limitations necessitate the development of more robust and nuanced detection methods[3], [4].

In the fight against deepfake audio, machine learning methods have emerged as a powerful weapon. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, play a crucial role. These models excel at learning distinctive features from audio signals. Imagine them sifting through the intricate patterns of human speech, identifying subtle inconsistencies that might betray a deepfake[5].

The quest for deepfake detection goes beyond mere feature extraction. Transformer-based models, like Conformers, are making waves by their ability to capture not only local details but also the broader context within an audio sample. Think of it as a detective meticulously examining not just fingerprints at a crime scene, but also the surrounding environment for clues. Attention mechanisms within these models further enhance their capabilities. Imagine the detective focusing on specific aspects of the scene based on their relevance, leading to a more insightful analysis. Additionally, self-supervised learning techniques are being explored to empower models to learn even without explicitly labeled data, akin to a detective honing their skills through observation and experience[5], [6].

But the fight against deepfakes isn't a solo mission. Hybrid approaches are gaining traction, combining the strengths of traditional signal processing techniques with the advanced capabilities of deep learning models. Imagine a seasoned detective collaborating with a cutting-edge forensics lab. For instance, hand-crafted features like Mel-frequency cepstral coefficients (MFCCs) can be used as initial inputs to deep learning models. These features, akin to a detective's initial observations, can guide the model's analysis and potentially improve both performance and interpretability. Hybrid approaches can also leverage traditional techniques for pre-processing audio data (like noise reduction) or post-processing the model's output (like refining classifications). This collaborative approach allows for a more comprehensive analysis[5], [6], [7].

Ensemble methods are proving valuable. Imagine a team of diverse detectives working together. Ensemble methods combine multiple detection models, each with its own strengths and weaknesses. These models can be of different types, or even trained on various subsets of data or features. Techniques like majority voting, stacking, and blending then combine the outputs of these models, mitigating the weaknesses of any single one. This creates a more robust and diverse detection approach, akin to a team with a wider range of expertise[7].

However, the battle against deepfakes is not without its challenges. Large neural networks and Transformers, while powerful, have a tendency to overfit. Imagine a detective fixated on a single piece of evidence, neglecting the bigger picture. This overfitting can limit the model's ability to generalize to unseen deepfake audio samples. To combat this, researchers are employing techniques like regularization (preventing overemphasis on specific features), data augmentation (artificially expanding the training data), and transfer learning (leveraging knowledge from related tasks). These methods are akin to the detective considering all available evidence, utilizing diverse training scenarios, and applying their knowledge from past cases. Yet, achieving robust performance across a vast spectrum of deepfake techniques and scenarios remains an ongoing challenge, demanding continuous research and innovation[5], [6], [7].

Deep learning has emerged as a powerful tool in the fight against deepfakes, but it's not without its complexities. One major challenge lies in interpretability. These models often function like intricate black boxes, adept at detecting deepfakes but leaving us in the dark about their reasoning. Imagine a judge delivering a verdict without any explanation such a lack of transparency can hinder trust and adoption of these deep learning models in real-world applications. Researchers are actively exploring techniques like saliency maps and attention visualization to shed light on the decision-making process within these models. However, interpreting the inner workings of complex deep learning models remains an ongoing area of research[7], [8].

Another hurdle is data dependency. Deep learning models are data-driven beasts, their performance heavily reliant on the quality and quantity of information they're trained on. Here's the challenge: acquiring diverse and representative datasets for deepfake detection is no easy feat. Emerging deepfake techniques constantly evolve, and underrepresented languages and accents further complicate the data collection process. Data augmentation techniques can be employed to artificially inflate the diversity of training data, but their effectiveness has limitations. It's akin to feeding a picky eater a plate of cleverly disguised vegetables they might consume more, but true variety and nutrient richness remain elusive[7], [8].

Computational requirements pose another obstacle. Large neural networks and Transformers, while powerful, are computationally demanding. This can be a roadblock for low latency deployment, particularly in environments with limited resources. Researchers are developing techniques like model compression and efficient architectures to streamline these models, but it's a delicate balancing act. Reducing computational demands often comes at the expense of performance, and finding the optimal sweet spot requires careful consideration[8], [9].

The lack of standardization adds another layer of complexity. Without a common set of benchmark datasets, it's difficult to compare and evaluate different deepfake detection methods fairly. Imagine judging a gymnastics competition where each athlete performs on a different apparatus it's nearly impossible to determine the true champion. Standardized datasets with well-defined evaluation protocols and metrics are crucial for establishing a level playing field, enabling researchers to objectively compare methods and accelerate progress in the field of deepfake detection[8], [9].

Deepfake detection, while a powerful tool, presents a complex ethical dilemma: the data it relies on. Using real speech data for training models raises privacy concerns. Imagine training a detective on real surveillance footage without proper anonymization – ethical red flags abound! To navigate this challenge, robust data anonymization techniques and strict ethical guidelines are essential. This ensures data privacy is protected while enabling effective development of deepfake detection methods[10].

However, anonymization isn't a silver bullet. Another approach gaining traction is synthetic data generation. This involves creating artificial speech data that mimics real audio, eliminating privacy concerns. Think of it as training detectives on meticulously crafted simulations, safeguarding real people's identities. While promising, synthetic data generation techniques need careful evaluation to ensure they are both effective and representative of real-world deepfakes. Feeding our detectives scenarios completely unlike real-world crimes would limit their effectiveness[11].

The battle against deepfakes is further complicated by the need for diverse training data. Imagine a detective trained only on cases involving a specific type of criminal – their ability to handle diverse situations would be hampered. Similarly, deepfake detection models require a rich tapestry of data encompassing various speaker demographics, recording conditions, and audio domains (e.g., phone calls, podcasts). Furthermore, datasets should include a spectrum of deepfake techniques, from traditional methods to the latest neural network-based approaches. This ensures the detection models can recognize a wide range of deepfakes, not just the most common ones. Continuous updating and expansion of datasets are crucial to keep pace with the ever-evolving landscape of deepfake audio techniques[10], [11].

The fight against deepfakes is a dynamic one. As deepfake creators devise new methods, existing datasets may become obsolete. This necessitates continuous efforts to create and update datasets, ensuring detection models are evaluated against the most recent and sophisticated techniques. Imagine a detective's training materials never reflecting new criminal tactics their effectiveness would quickly dwindle. Collaboration between researchers, industry partners, and stakeholders is essential for maintaining dataset relevance. Only through collective effort can we ensure deepfake detection methods stay ahead of the curve[10], [11].

Standardized evaluation metrics and protocols are vital for a fair fight. Imagine judging a competition without clear criteria – it's impossible to determine the true winner. Common metrics like accuracy and AUROC are valuable, but they may not capture the nuances of real-world deepfake detection. Developing specialized metrics that consider factors like low latency performance and generalizability, along with well-defined evaluation protocols, ensures fair and reliable comparisons of different deepfake detection methods. By establishing clear evaluation benchmarks, we can ensure the best possible tools are employed in the fight against deepfakes[10], [11], [12].

Deepfake detection, while valiant in its efforts, faces a significant hurdle: data. Annotating and curating high-quality datasets for deepfake audio detection is a monumental task. Imagine meticulously labeling countless audio samples as real or deepfake it's a labor-intensive and error-prone process. To overcome this challenge, researchers are exploring innovative strategies. Crowdsourcing, where tasks are distributed among a large online workforce, holds promise for accelerating data annotation. However, ensuring accuracy and consistency requires careful design and quality control measures. Another approach involves semi-supervised learning, where a small amount of labeled data is used to guide the model in labeling a larger unlabeled dataset. Imagine a detective with limited witness testimonies effectively training new recruits by leveraging their existing knowledge. However, both crowdsourcing and semi-supervised learning necessitate continuous data curation and validation. Just as a detective's training materials need to be fact-checked, deepfake detection datasets require ongoing monitoring to address potential errors and biases that can creep in during the annotation process[13], [14].

Looking beyond the limitations of single modalities like audio, researchers are exploring the potential of multimodal deepfake detection. Imagine a detective not only listening to a suspect's testimony but also analyzing their body language and facial expressions. Similarly, multimodal deepfake detection leverages not just audio, but also visual information (like lip movements) and textual cues (like inconsistencies in transcripts) to improve detection accuracy and robustness. By capturing inconsistencies across these modalities, these approaches provide additional red flags for identifying deepfakes. However, developing effective fusion strategies and architectures that seamlessly combine information from various sources remains an active area of research. It's akin to devising a robust interrogation strategy that effectively integrates all available evidence[14].

Researchers are pushing the boundaries of data dependency by exploring few-shot and zero-shot learning techniques. Imagine a detective who can identify a criminal after seeing only a few blurry photographs, or even without any prior reference. Few-shot learning aims to achieve similar feats in deepfake detection, requiring minimal labeled data. Zero-shot learning takes this a step further, attempting detection without any labeled data at all. Approaches like meta-learning and self-supervised learning show promise in this domain. While still under development, these techniques could significantly reduce data dependency, making deepfake detection more practical for real-world deployment scenarios where obtaining large labeled datasets might be challenging[15], [16].

Deepfake detection, while advancing rapidly, is locked in an ongoing battle with potential adversaries. As detection methods become more sophisticated, deepfake creators may devise techniques to bypass them. Imagine a cunning criminal constantly innovating new methods to evade capture. To stay ahead, researchers are actively exploring adversarial defenses. One approach, adversarial training, involves exposing deepfake detection models to deliberately manipulated audio samples, akin to training detectives with ever-evolving criminal tactics. By continuously challenging the models with these adversarial examples, they learn to become more robust in identifying deepfakes. Another promising technique is model assembling, where multiple detection models with diverse strengths and weaknesses are combined. Think of a team of detectives with different specialities working together – their collective expertise increases the chances of apprehending the culprit. Developing resilient detection methods through adversarial defenses is crucial for maintaining the upper hand in the fight against deepfakes[17].

Data scarcity and privacy concerns pose another challenge. Federated learning offers a glimmer of hope. Imagine detectives in different cities collaborating on a case without sharing confidential information. Federated learning allows deepfake detection models to be trained collaboratively across multiple devices or servers without directly sharing the underlying data. This approach addresses data scarcity concerns, particularly for underrepresented languages or accents, while simultaneously safeguarding privacy. By leveraging federated learning and privacy-preserving techniques, researchers can develop robust and generalizable detection methods applicable to a wider range of scenarios[18].

The fight against deepfakes extends beyond the lab and into the real world. Deployment considerations are paramount. Imagine equipping detectives with cumbersome, outdated equipment that hinders their effectiveness. Deepfake detection models need to be optimized for efficient inference, meaning they can run quickly and accurately on resource-constrained devices. Edge computing, where processing occurs closer to data sources, offers a potential solution. Furthermore, continuous monitoring and adaptation strategies are essential. Just as detectives need to adjust their tactics based on new information, deepfake detection models require ongoing fine-tuning to maintain effectiveness in the face of evolving deepfake techniques. By prioritizing efficient deployment, privacy-preserving techniques, and continuous adaptation, researchers can ensure their deepfake detection methods are practical and impactful in real-world scenarios[19].

We have compared some features of Deepfake models and give them score ratings explained in chart below:

Figure 1 Features Rating

A comparison of some models is given below in table:

Table 1 Models comparison

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Name | Time Complexity | Space Complexity | Compute Cost | Strengths |
| SVM | Support Vector Machine | Medium | Medium | Low | Good for binary classification, interpretable |
| Random Forest | Random Forest Classifier | High | High | Medium | Robust to overfitting, handles mixed data types |
| XGBoost | Extreme Gradient Boosting | High | High | High | Highly accurate, efficient for large datasets |
| CNN based | Convolutional Neural Network | High | High | High | Captures temporal and spectral features, requires large datasets |
| RNN based | Recurrent Neural Network | High | High | High | Captures sequential information, effective for long audio sequences |
| Transformer based | Transformer-based models (e.g., Wav2Vec2) | High | High | High | State-of-the-art performance, handles long audio sequences |

## 2.3 Summary

Deep fake audio detection is a rapidly evolving field with significant implications for trust, security, and privacy in audio-based communication. While substantial progress has been made, challenges persist in generalization, robustness, computational complexity, and dataset availability. Addressing these challenges requires interdisciplinary research, collaborative efforts, and the development of standardized datasets. Continuous exploration of emerging trends and future directions is essential for advancing the field and ensuring robust, trustworthy, and ethically responsible solutions for safeguarding audio-based communication integrity.

# Chapter 3

# Requirements Analysis

Now we will understand the Requirement Analysis for our Deepfake Platform and Model as it is necessary to understand the requirements for a better approach to achieve research gap and solve research problem.

## Introduction

This chapter delves into the requirement analysis for a deepfake audio detection system designed for rapid inference. The system aims to address the growing concern of deepfake audio content by providing a robust solution capable of operating in dynamic, real-world scenarios.

## Problem Scenarios

Deepfake audio content poses significant threats to personal and public security, as well as to the integrity of communication channels, making it a critical area of concern. One of the most alarming scenarios is the potential for spoofing identity. In this context, malicious actors can use deepfake technology to create audio clips that convincingly mimic the voices of individuals in sensitive positions, such as government officials, executives, or other authority figures. This can be exploited to gain unauthorized access to secure systems, confidential information, or to spread false information with potentially devastating consequences. The precision of deepfake audio can make it extremely difficult for recipients to distinguish between genuine and counterfeit communications, thereby compromising security protocols and trust.

Another significant threat posed by deepfake audio is the dissemination of misinformation. In an era where information spreads rapidly through digital media, counterfeit audio clips can be used to manipulate public opinion or sway the outcomes of political events. For instance, fabricated audio of a public figure making controversial statements can be released to damage their reputation or influence voters. The viral nature of social media amplifies the reach and impact of such misinformation, making it a powerful tool for those looking to manipulate public perception and discourse. This not only undermines the trust in media and public figures but also destabilizes social and political environments.

Deepfake audio poses severe risks in the realm of financial fraud. Fraudsters can use this technology to mimic the voices of CEOs, financial officers, or other authority figures within organizations to authorize fraudulent transactions. By creating highly convincing audio commands, they can deceive employees or automated systems into transferring funds, sharing sensitive financial information, or altering financial records. This type of fraud can result in significant financial losses for individuals and organizations, as well as legal and reputational damage. The sophistication of deepfake audio makes it a formidable tool for criminals, necessitating advanced detection and prevention measures to protect financial integrity and security.

## Functional Requirements

These are the requirements that assure the system's roles. The functional requirements that the application must meet are listed below:

### Chrome or Alternative Extension Browser

In order to work the system needs a browser. It works as an extension and stream audio from current tab to backend server using buffering. So, a browser is a must.

### Recording Functionality

The extension should allow users to start and stop recording audio. When recording, the extension should capture audio from the Chrome tab. It should display a timer showing the elapsed recording time.

### File Upload

Users should be able to upload .wav audio files. The extension should validate uploaded files to ensure they are in the correct format (.wav) and within a specified duration limit (60 seconds). After successful validation, the extension should display the filename and send the file to a Fast API endpoint for processing.

### Audio Processing

Upon stopping the recording or uploading a file, the extension should encode the audio data into a .wav format. It should send the encoded audio data to a FastAPI endpoint for further processing.

### API Integration

The extension should communicate with a FastAPI server running locally for file upload and processing. It should handle server responses, displaying API response data or error messages on the user interface.

### User Interface (UI)

The extension should have a user-friendly interface with clear instructions and feedback messages. It should provide buttons for starting and stopping recording, as well as for uploading files. Display elements such as the timer and filename should be updated dynamically based on user actions.

### Error Handling

The extension should handle errors gracefully, providing informative error messages to users in case of failures during recording, file upload, or API communication.

It should revert to a consistent state after encountering errors, allowing users to continue using the extension seamlessly.

### Security

Ensure that uploaded audio files are processed securely without exposing sensitive user data. Implement appropriate security measures to protect against potential threats such as data breaches or unauthorized access to audio recordings.

### Audio Processing Functions

Must implement functions to split WAV files into segments, extract MFCCs and LPCCs features, estimate VOT (Voice Onset Time), compute energy and zero-crossing rate (ZCR), and detect burst and periodicity onset. These functions are crucial for extracting relevant features from audio files.

### Prediction Functions

Defined functions must be able to process audio files, including loading models (SVM classifier and scaler), extracting features, and making predictions using the SVM classifier. These functions handle error cases gracefully and log relevant information for debugging purposes.

### Batch Analysis

The analyze audio batch function orchestrates the batch processing of audio files using concurrent execution. It efficiently utilizes multiple threads to analyze multiple audio files simultaneously, improving overall throughput. It needs CPU resources to execute.

### API Endpoints

A FastAPI endpoints for uploading files, collecting data feedback, and handling CORS (Cross-Origin Resource Sharing) middleware. These endpoints facilitate interaction with the backend server, enabling clients to upload files, receive predictions, and provide feedback on data.

### Logging and Error Handling

The code incorporates robust logging mechanisms to track the execution flow, capture errors, and provide insights into the processing pipeline. This logging is instrumental in troubleshooting issues and monitoring server performance.

### Dataset Management and Collection

Manages file uploads, temporary file storage, and metadata storage efficiently. Temporary files are cleaned up after processing, and metadata is updated to reflect user choices and processing outcomes.

## Non-Functional Requirements

These are the criteria that assure the system's efficiency and have an impact on the technical requirements. The non-functional requirements that the application must meet are listed below.

### Look and Feel

The server should provide clear and concise responses to client requests, including error messages that are informative and user-friendly. Logging and debugging information should be presented in a structured and readable format, facilitating easy analysis by developers.

### Usability

APIs should have intuitive endpoints and well-defined input/output formats, making it easy for clients to interact with the server. Error responses should be descriptive, helping clients understand what went wrong and how to rectify the issue.

### Accessibility

APIs should adhere to accessibility standards such as providing alternative text for images and ensuring keyboard navigation support for web interfaces. Documentation should be accessible to users with disabilities, possibly including options for screen readers and alternative formats.

### Performance Requirements

The server should be able to handle multiple concurrent requests efficiently, ensuring low latency and high throughput. Response times for API requests should be within acceptable limits, even under heavy load conditions.

### Operational and Environmental Requirement

The server should be deployable on various operating systems and cloud platforms, ensuring compatibility and scalability. It should gracefully handle changes in environmental conditions, such as network interruptions or server failures, without data loss or service disruption.

### Maintainability and Support Requirement

Code should be well-documented and follow best practices to facilitate maintenance by developers. Regular updates and patches should be provided to address security vulnerabilities and improve functionality. Support channels should be available for users to report issues and receive timely assistance.

### Security Requirement

The server should implement authentication and authorization mechanisms to protect sensitive data and prevent unauthorized access.

APIs should validate user input and sanitize data to prevent injection attacks such as SQL injection or XSS (Cross-Site Scripting).

Transmission of data between client and server should be encrypted using secure protocols such as HTTPS.

### Legal Requirements

The server should comply with relevant data protection regulations such as GDPR (General Data Protection Regulation) or HIPAA (Health Insurance Portability and Accountability Act), depending on the nature of the data being processed.

Intellectual property rights should be respected, and appropriate licenses should be obtained for any third-party libraries or components used in the server.

# Chapter 4

# System Design

## Introduction

Effective communication is crucial when designing complex systems, and visual representations often convey intricate details more vividly than words alone. This chapter delves into the modeling process, meticulously detailing how various components of the application have been conceptualized. Recognizing the adage "a picture is worth a thousand words," diagrams play a pivotal role in facilitating a comprehensive understanding of the system's architecture and pinpointing areas for potential enhancements. The subsequent sections present the relevant diagrams, providing a visual aid that complements the textual descriptions, thereby offering a holistic perspective on the system's design.

## Architect Diagram

An architecture diagram is a visual representation that illustrates the high-level structure, components, and relationships within a system or application. It serves as a blueprint or roadmap that depicts the overall design and organization of the various elements that make up the software or hardware architecture.

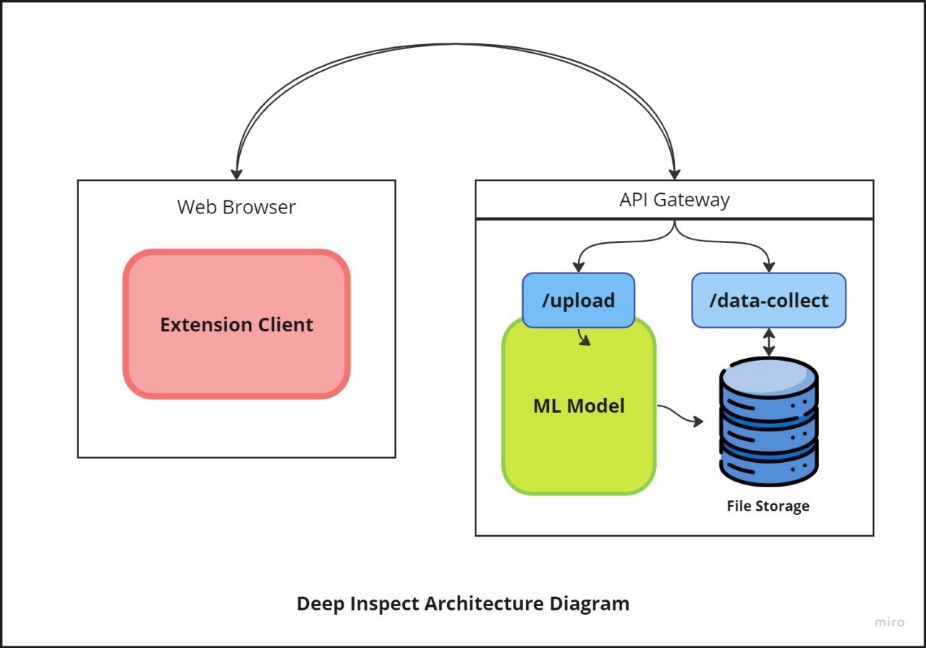


Figure 2 Architect Diagram

## Data Flow Diagram

The data flow diagram for the “Deep Inspect” system illustrates how data moves through various components. At the outset, user input is collected in the “Data Collection” module. Next, the audio data undergoes processing in the “Audio Processor,” potentially splitting it into segments. Relevant features are then extracted from the audio in the “Feature Extraction” step. The heart of the system lies in the “AI/ML Model,” which uses these features to make predictions. Notably, there’s a feedback loop labeled “Prediction,” suggesting continuous learning. Finally, the system generates a response for the user, completing the cycle. Additionally, data storage outside this client-side subsystem allows for further analysis or archival purposes.

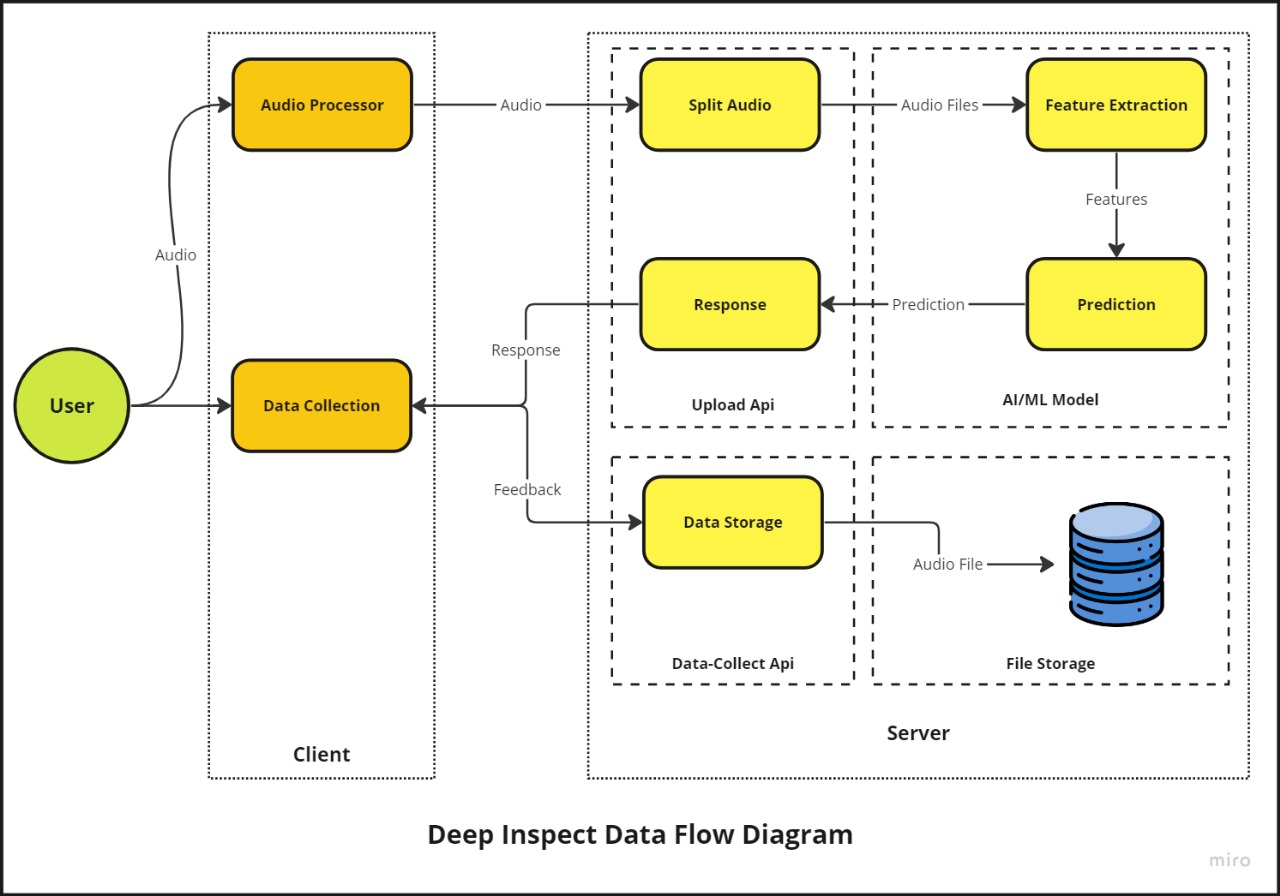


Figure 3 Data Flow Diagram

## Use Case Diagram

In this specific diagram, the User is depicted as an actor, and there are three main use cases: "Upload Audio," "Audio Results," and "Provide Feedback." The arrows connecting the User to the use cases indicate the interactions or associations between the actor and the system's functionalities. The diagram shows that the User can upload audio files to the Deep Inspect system, receive audio analysis results, and provide feedback on the system's performance or functionality.

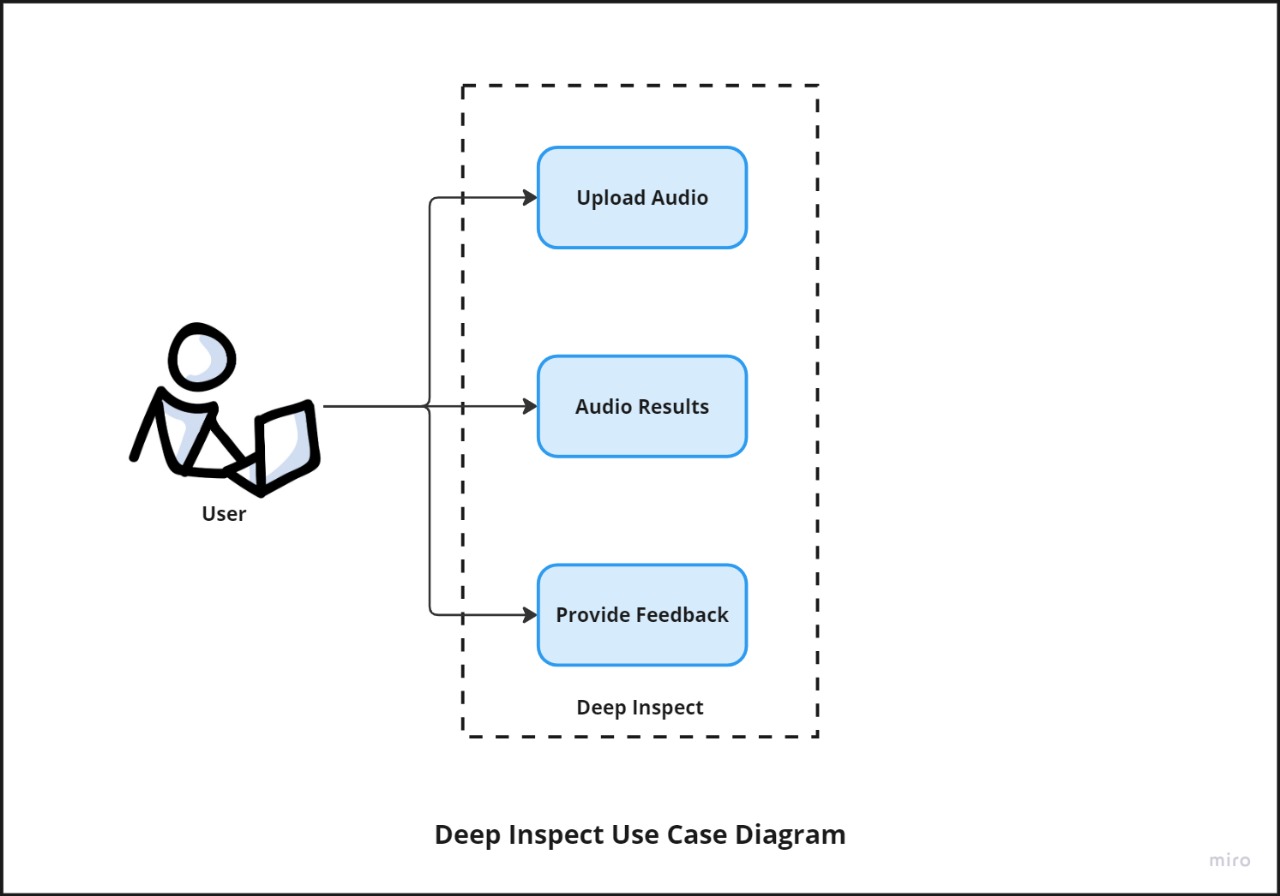


Figure 4 Use case Diagram

## Activity Diagram

The Activity Diagram illustrates the overall workflow or process flow of the Deep Inspect system. It starts with the "Start" node, followed by the "Open Deep Inspect" activity. The user can then either "Record Audio" or "Upload Audio" directly. These activities lead to the "Audio processing" step, which involves validating the audio file "File Validation".

After file validation, the process splits into two parallel activities: "File Splitting" and utilizing an "AI/ML Model" for analysis. The results from these activities are then used for "Prediction" or classification of the audio file.

Based on the prediction, the system "Generate Metadata" and allows the user to "Get Feedback." The metadata is then used to "Update file" information, followed by "File Upload" and "File Save" activities.

Finally, the process ends with the "End" node, indicating the completion of the workflow.

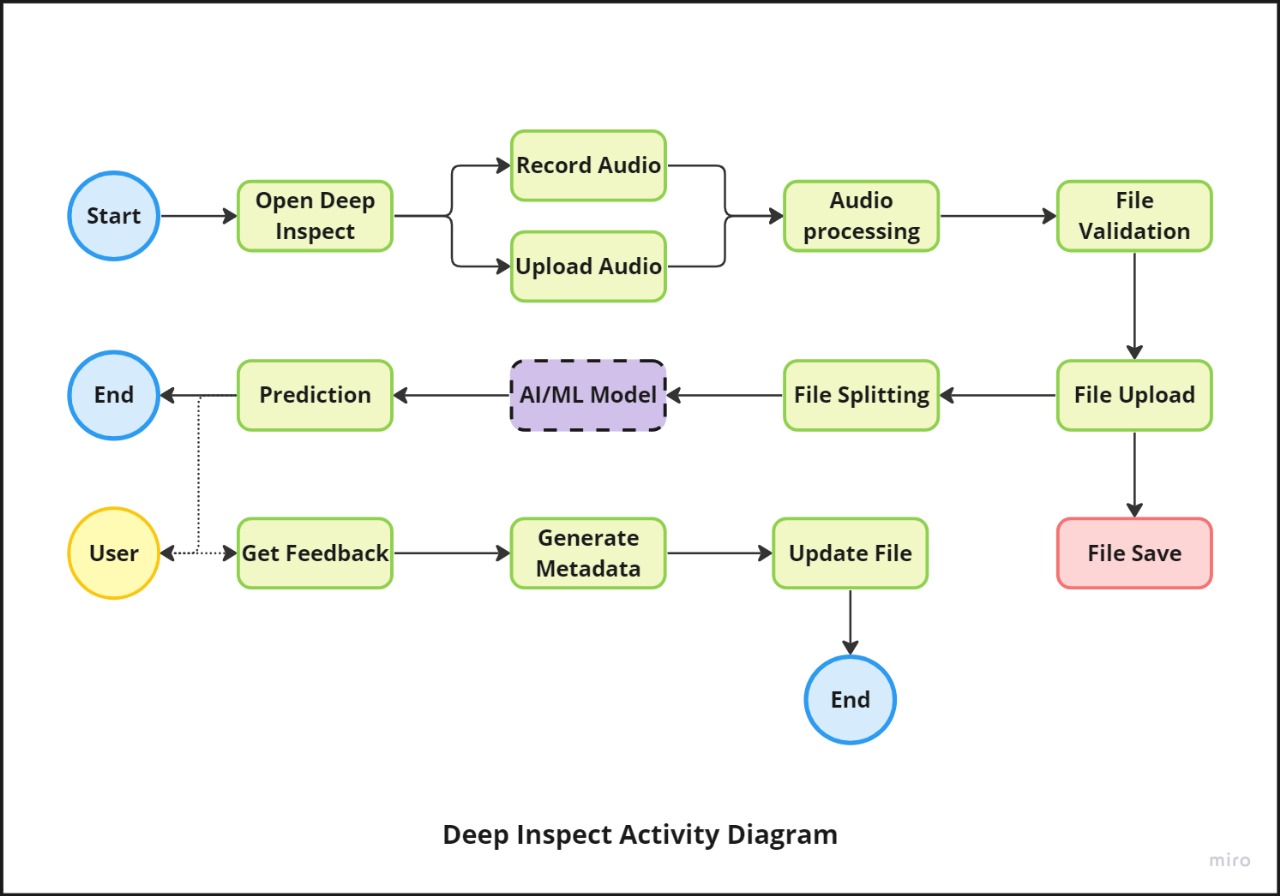


Figure 5 Activity Diagram

## Model Training Diagram

This diagram illustrates the architecture of our audio classification system using machine learning techniques. The process begins by splitting the available audio data into training and testing sets. Various audio features are then extracted, including Mel-Frequency Cepstral Coefficients (MFCCs), Linear Prediction Cepstral Coefficients (LPCCs), Voice Onset Time (VOT), and burst and periodicity onset times. These features are combined into a 43-dimensional feature vector, which is then scaled using a Standard Scaler for normalization. The scaled feature vectors are fed into a Support Vector Machine (SVM) classifier with a Radial Basis Function (RBF) kernel, which is trained on the training data. The trained SVM model can then be evaluated using performance metrics such as Accuracy, Precision, ROC-AUC and EER. This concise architecture diagram provides a clear overview of the end-to-end audio classification pipeline, from data preprocessing to model training and evaluation.

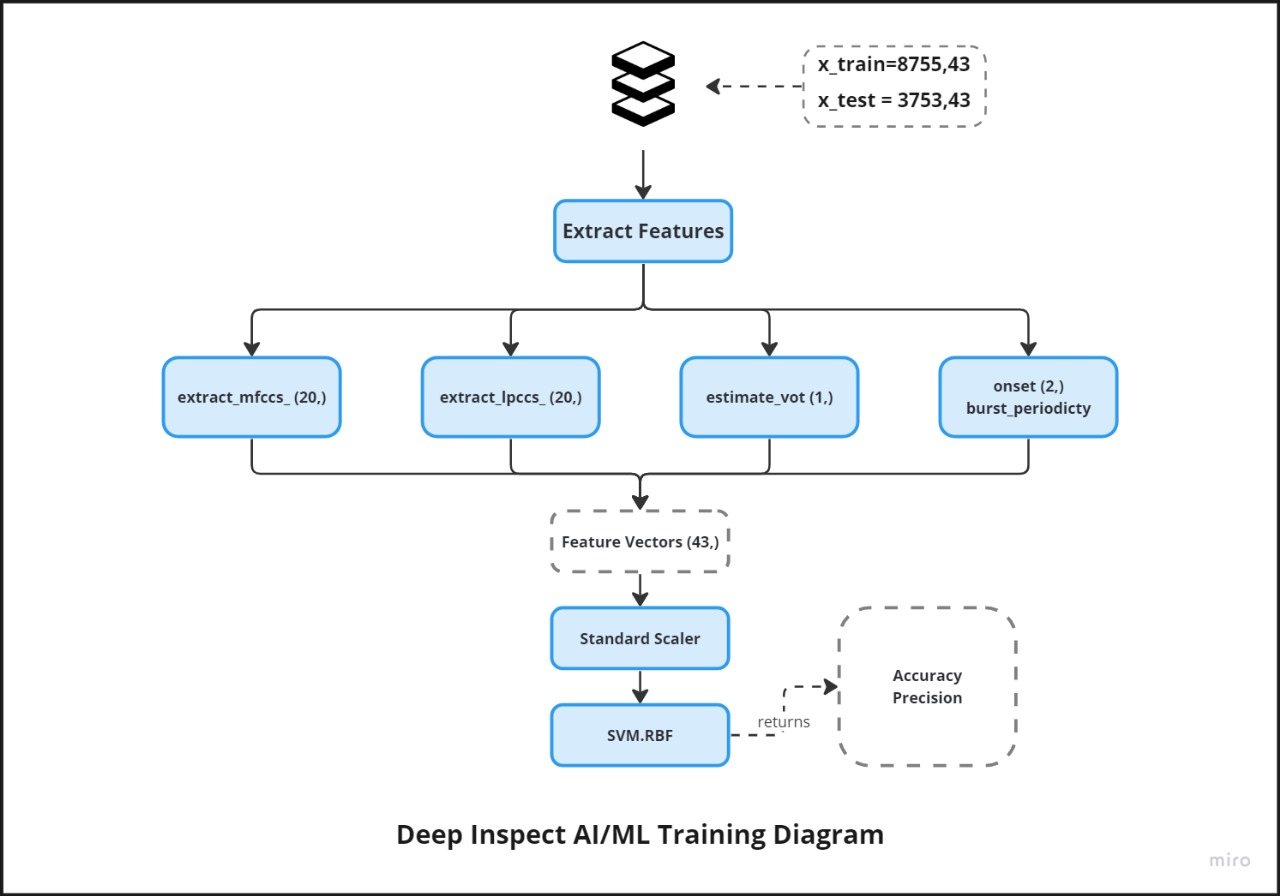


Figure 6 Model Training

## Model Inference Diagram

The diagram illustrates the AI/ML prediction flow of the Deep Inspect system. It begins by analyzing an audio batch, loading the SVM classifier and scaler models. It validates if the input audio file exists and is in WAV format. If valid, it extracts features from the audio, scales them using the scaler, and makes a prediction using the SVM classifier, classifying the audio as genuine or deepfake. Finally, it collects the prediction and probability scores, returning them as the output.

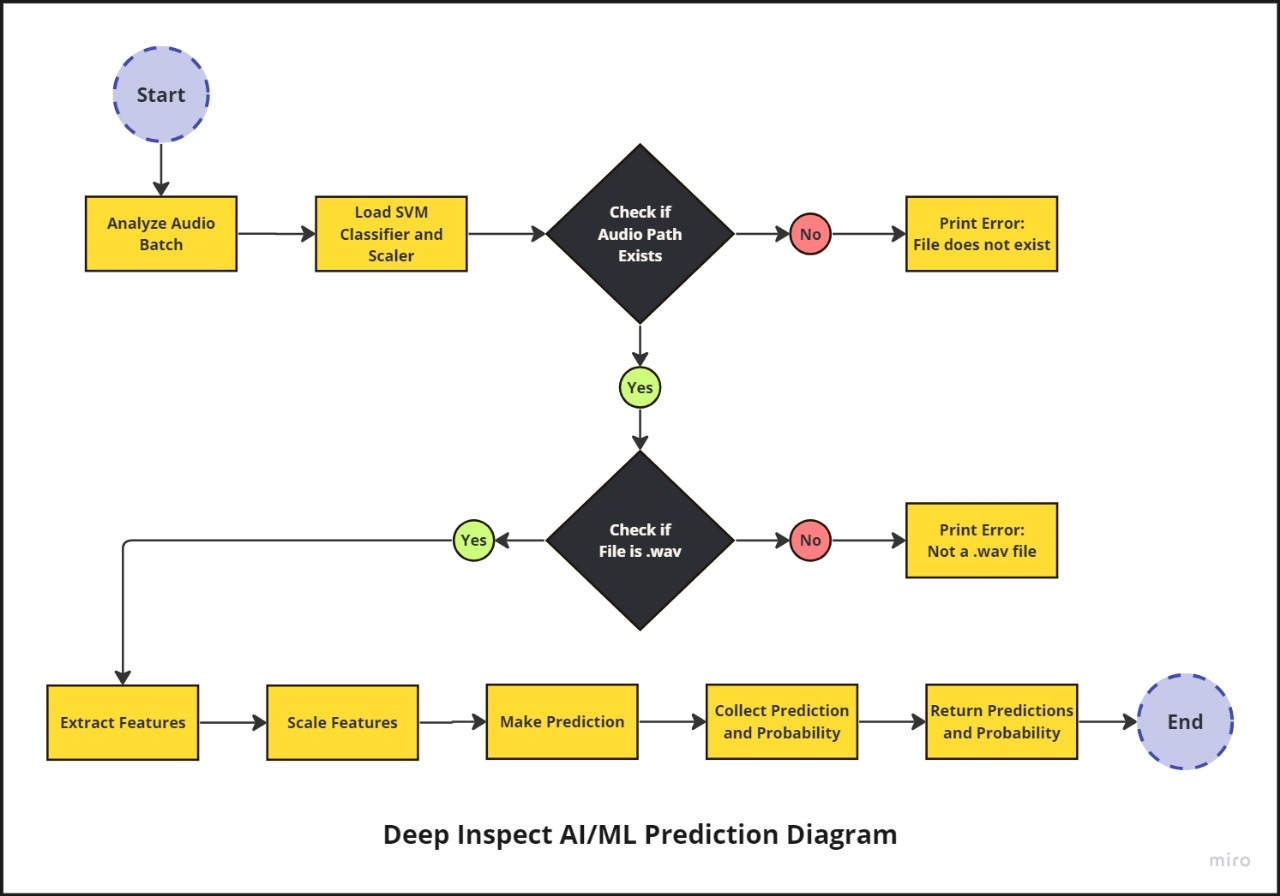


Figure 7 Model Inference

**Chapter 5:**

**Implementation**

**Chapter 5:**

# Implementation

## 5.1 Flow Control

This diagram depicts the flow control and all the activities happening in our system.

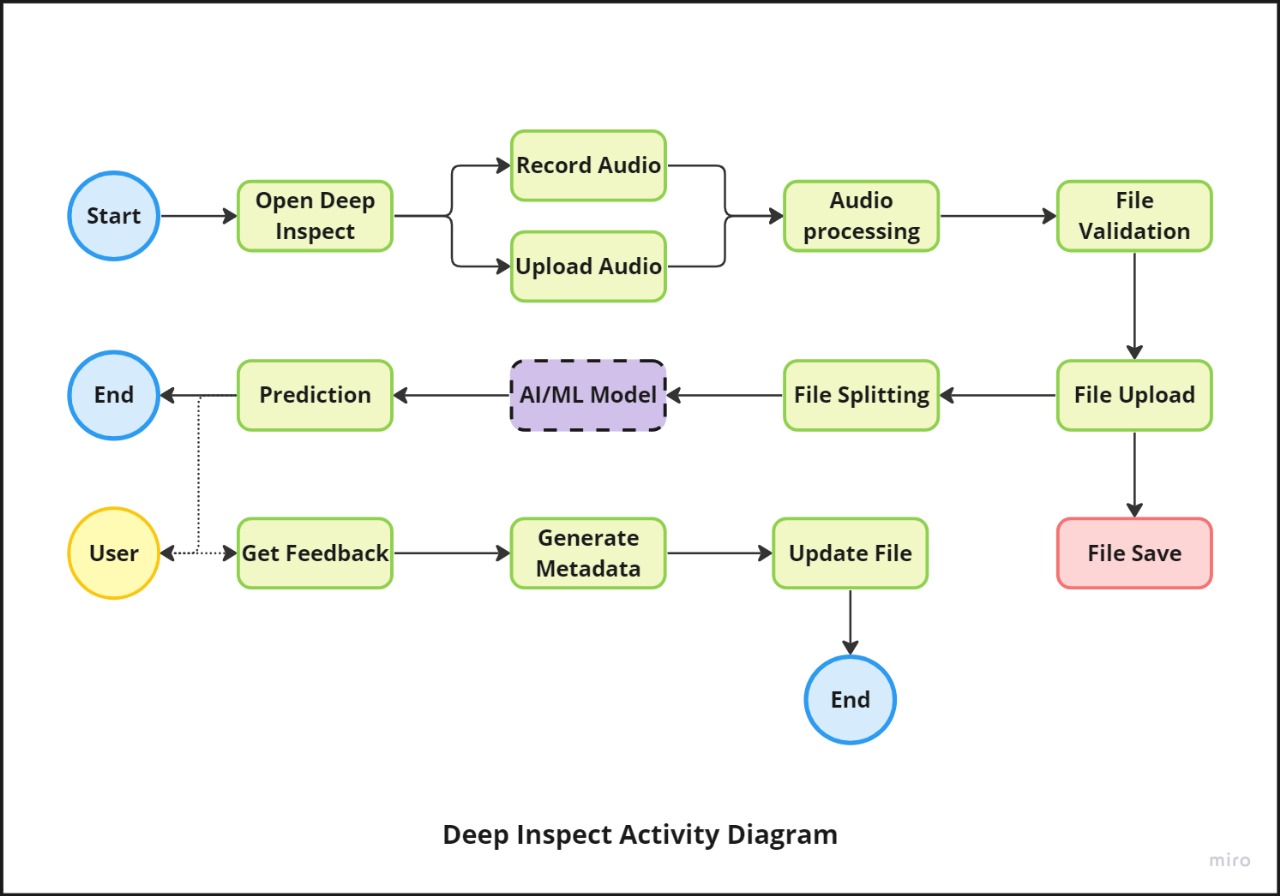


Figure 8 Flow Control

## 5.2 Components, Libraries, Web Services

In our project, we utilized a range of advanced tools and libraries to achieve accurate and efficient audio analysis and deepfake detection. At the core of our audio processing pipeline is **Librosa**, a powerful Python library for analyzing audio and music. Librosa is essential for loading audio files and extracting key features such as Mel-frequency cepstral coefficients (MFCCs) and Linear Predictive Coding (LPC). These features are fundamental in breaking down audio files into manageable segments and extracting meaningful data for further analysis.

To handle audio file operations, we used **Soundfile**, a library that allows reading and writing sound files in various formats. This is particularly useful for saving segmented audio clips extracted from the original input files. Additionally, we integrated **Parselmouth**, which provides a Python interface to **Praat**, a software tool used for speech analysis. **Parselmouth** is crucial for estimating **Voice Onset Time (VOT)**, a key metric in our audio analysis process.

**Joblib** plays a vital role in our project by enabling us to save and load trained machine learning models and scalers. This ensures that our models can be easily persisted and reused without the need for retraining, streamlining the workflow. Our machine learning efforts are powered by Scikit-learn, one of the most popular machine learning libraries in Python. We use **Scikit-learn** to train our Support Vector Machine (SVM) classifier for detecting deepfake audio, leveraging its comprehensive tools for model training, evaluation, and feature scaling.

To create a responsive and efficient web service API, we turned to **FastAPI**, a modern web framework known for its performance and ease of use. FastAPI allows us to handle file uploads, process audio files asynchronously, and return predictions quickly. This framework enhances the overall performance of our application and ensures a smooth user experience.

Numerical computations and array handling are efficiently managed using Numpy, a fundamental library in Python for mathematical operations. **Numpy** is extensively used throughout our project for manipulating arrays and performing various calculations on the extracted audio features.

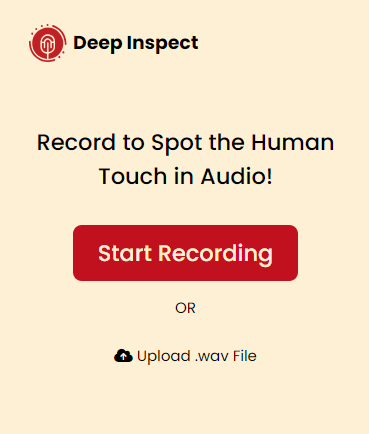
For monitoring and debugging, we rely on Python's built-in logging module. Logging helps us keep track of events during code execution, facilitating effective debugging and performance monitoring. We also utilize **RotatingFileHandler** from the **logging.handlers** module to manage log files, ensuring they do not grow indefinitely by rotating them when they reach a specified size.

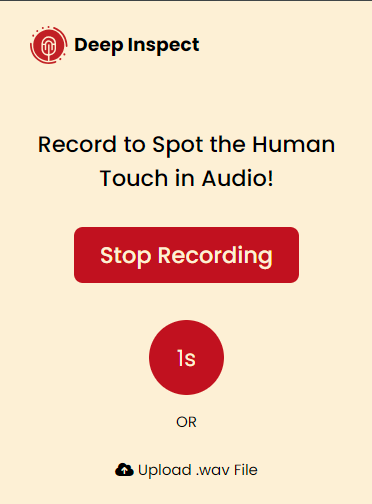
To optimize the processing of multiple audio files, we leverage **concurrent.futures**, a module that provides a high-level interface for asynchronously executing **callables**. This parallelization significantly reduces the time required for batch analysis, enhancing the efficiency of our system.

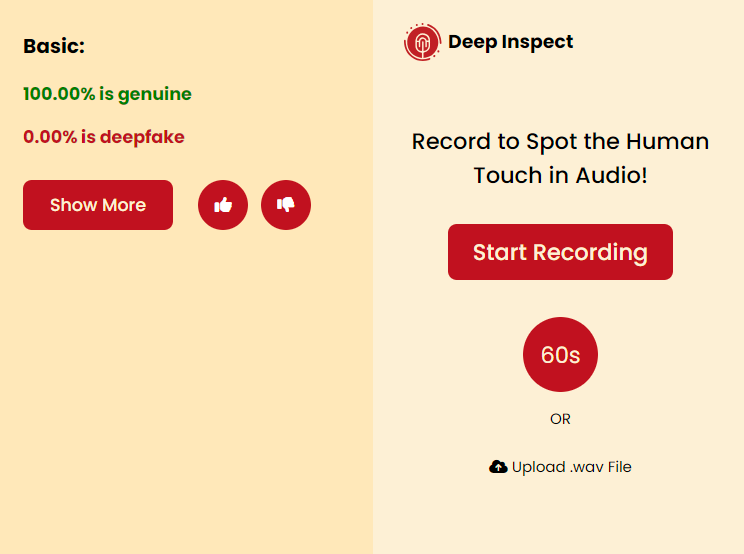
## 5.3 Code Layer

In this segment, we’ll delve into the various components of the code that elucidate the functioning of our application. For clarity, we’ve segregated the code into two distinct layers. The presentation layer, which handles the user interface, and the data management layer, which deals with data processing and storage.

### 5.3.1 Presentation Layer







|  |
| --- |
| <html>    <head>      <script src="home.js"></script>      <link rel="stylesheet" href="home.css" />      <link rel="preconnect" href="https://fonts.googleapis.com" />      <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />      <link        href="https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap"        rel="stylesheet"      />      <link        rel="stylesheet"        href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css"        integrity="sha512-SnH5WK+bZxgPHs44uWIX+LLJAJ9/2PkPKZ5QiAj6Ta86w+fsb2TkcmfRyVX3pBnMFcV7oQPJkl9QevSCWr3W6A=="        crossorigin="anonymous"        referrerpolicy="no-referrer"      />    </head>    <body>      <div id="right">        <div class="top">          <div class="logo">            <svg id="Layer\_1" data-name="Layer 1" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 45.53 46.21">              <defs>                <style>                  .cls-1 {                    fill: #c12126;                  }                </style>              </defs>              <path                class="cls-1"                d="M186.06,204c0-2.14,0-4.14,0-6.14,0-.16.35-.36.58-.45a24.79,24.79,0,0,0,3.21-1.24A7.94,7.94,0,0,0,194,189c0-2.06,0-4.11,0-6.17a8.7,8.7,0,0,0-6.4-8.38,8.86,8.86,0,0,0-11.29,8c-.12,2.41,0,4.83,0,7.24,0,3.34,2.85,6.86,6.77,7.69,1.11.24,1.11.24,1.11,1.35V204c-1.07-.16-2-.26-3-.46a19.12,19.12,0,0,1-14.35-23.56c2.39-8.53,10.7-14.64,19.07-14.23a18,18,0,0,1,14.22,7.83c4.58,6.41,5.31,13.35,1.69,20.44-2.83,5.56-7.59,8.7-13.73,9.74C187.51,203.84,186.85,203.88,186.06,204Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M184.26,195.61a6.56,6.56,0,0,1-6.21-6.18c-.05-1.36,0-2.72,0-4.08,0-.56.2-.71.72-.7,1.94,0,1.6-.17,1.62,1.57,0,1.31,0,2.63,0,3.94,0,.71.39.68.88.7s.78-.14.77-.73q0-3.12,0-6.25A3,3,0,0,1,184,181a3.09,3.09,0,0,1,3.38.59,2.76,2.76,0,0,1,1,2.17c0,2.11,0,4.22,0,6.32,0,.64.14.91.85.92s.88-.27.87-.91c0-1.77,0-3.54,0-5.44a18,18,0,0,1,1.82,0c.16,0,.4.31.41.48,0,1.6.11,3.2,0,4.78a6.82,6.82,0,0,1-5.9,5.75,1.09,1.09,0,0,1-.32-.06V184.7c0-.82-.57-1.28-1.22-.9a1.46,1.46,0,0,0-.51,1.06c0,3.1,0,6.22,0,9.33Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M162.4,184.58c0-.51,0-1,0-1.54a1,1,0,0,1,2.09-.07c.09,1.13.09,2.26.18,3.38a21.13,21.13,0,0,0,16.58,18.87,19.91,19.91,0,0,0,11.22-.9c1.1-.4,1.64-.27,1.89.43s-.12,1.25-1.16,1.6c-7.71,2.6-14.93,1.65-21.47-3.24C165.58,198.51,162.7,192.17,162.4,184.58Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M207.91,184.81v1a1,1,0,0,1-1,1.11,1,1,0,0,1-1.09-1c-.09-.9-.06-1.8-.18-2.69q-1.66-12.9-13.63-18c-.31-.13-.62-.25-.92-.41a1,1,0,0,1,.84-1.91,32.6,32.6,0,0,1,3.84,1.77,23.56,23.56,0,0,1,11.32,14.66c.49,1.8.51,3.71.82,5.55Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M178.07,183a6.31,6.31,0,0,1,1.56-4.49,6.92,6.92,0,0,1,7-2.49,6.79,6.79,0,0,1,5.4,4.82,2.66,2.66,0,0,1,.1.38c.32,1.7.27,1.74-1.46,1.73a.77.77,0,0,1-.88-.62,4.87,4.87,0,0,0-8.13-1.82,5.66,5.66,0,0,0-1,1.84c-.17.48-.39.67-.87.65S178.66,183,178.07,183Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M200.54,201.77a1.49,1.49,0,0,1-1.43,1.34,1.29,1.29,0,0,1-1.18-1.28,1.32,1.32,0,0,1,1.34-1.29A1.17,1.17,0,0,1,200.54,201.77Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M204.08,194.77a1.31,1.31,0,0,1,.06,2.62,1.22,1.22,0,0,1-1.24-1.27A1.28,1.28,0,0,1,204.08,194.77Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M178,165.27a1.23,1.23,0,1,1-.05-2.46,1.28,1.28,0,0,1,1.24,1.3A1.19,1.19,0,0,1,178,165.27Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M186.13,162.83a1.19,1.19,0,0,1-1.19,1.23,1.21,1.21,0,0,1-1.22-1.21,1.27,1.27,0,0,1,1.15-1.3A1.3,1.3,0,0,1,186.13,162.83Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M167.46,173.43a1.11,1.11,0,0,1-1.17,1.15,1.23,1.23,0,0,1-1.2-1.24,1.33,1.33,0,0,1,1.23-1.32A1.36,1.36,0,0,1,167.46,173.43Z"                transform="translate(-162.4 -161.55)"              />              <path                class="cls-1"                d="M172.42,167.73a1.22,1.22,0,0,1-1.24,1.19,1.1,1.1,0,0,1-1.09-1.14,1.21,1.21,0,0,1,1.18-1.25A1.24,1.24,0,0,1,172.42,167.73Z"                transform="translate(-162.4 -161.55)"              />              <text x="-162.4" y="-161.55" />            </svg>          </div>          <h1 class="name">Deep Inspect</h1>        </div>        <div class="container">          <h2 class="heading">Record to Spot the Human Touch in Audio!</h2>          <button id="share-audio-button">Start Recording</button>          <div id="timer">0s</div>          <p id="OR">OR</p>          <div class="fileupload">            <label for="upload-wav-button" class="upload-button">              <i class="fas fa-cloud-upload-alt"></i> Upload .wav File            </label>            <input type="file" accept=".wav" id="upload-wav-button" />            <p id="fileName"></p>          </div>        </div>      </div>      <div id="left">        <div id="response-container" class="response-container"></div>      </div>    </body>  </html>  //Home.css  \* {    font-family: 'Poppins', sans-serif;    font-style: normal;  }  body {    background-color: #fdf0d5;    display: flex;    flex-direction: row-reverse;    margin: 0;    padding: 0;  }  #right {    width: 300px;    min-height: 350px;  }  #left {    display: none;    background-color: #ffe8b9;    width: 300px;    min-height: 450px;    height: fit-content;    padding: 10px 10px 20px 20px;    box-sizing: border-box;    word-wrap: break-word; */\* Ensure long words wrap to the next line \*/*    overflow-wrap: break-word; */\* Ensure long words wrap to the next line \*/*  }  .probability {    font-size: 14px;  }  .genuinePercentage {    color: green;    font-size: 14px;    font-weight: bold;  }  .advBtnCont {    display: flex;  }  .like {    color: white;    height: 40px;    width: 40px;    background-color: #c1121f;    border-radius: 100%;    display: flex;    justify-content: center;    align-items: center;    margin-top: 10px;    margin-left: 20px;  }  .dislike {    color: white;    height: 40px;    width: 40px;    background-color: #c1121f;    border-radius: 100%;    display: flex;    justify-content: center;    align-items: center;    margin-top: 10px;    margin-left: 10px;  }  .like:hover {    opacity: 80%;  }  .dislike:hover {    opacity: 80%;  }  .deepfakePercentage {    color: #c1121f;    font-size: 14px;    font-weight: bold;  }  .ResHead {    font-weight: bold;    font-size: 16px;  }  #response-container {    font-size: 14px;  }  .top {    display: flex;  }  .name {    margin-top: 23px;    margin-left: 5px;    font-size: 15px;  }  .logo {    margin-left: 25px;    margin-top: 20px;    width: 30px;  }  .advanceBtn {    margin-top: 10px;    background-color: #c1121f;    color: #fdf0d5;    border-radius: 7px;    font-weight: 500;    font-size: 14px;    height: 40px;    width: 120px;    border: none;  }  .advanceBtn:hover {    opacity: 80%;  }  .container {    display: flex;    flex-direction: column;    justify-content: center;    align-items: center;    padding-left: 20px;    padding-right: 20px;    padding-top: 30px;  }  .heading {    text-align: center;    font-weight: 500;    font-size: 18px;  }  #share-audio-button:hover {    opacity: 80%;  }  #share-audio-button {    margin-top: 10px;    background-color: #c1121f;    color: #fdf0d5;    border-radius: 7px;    font-weight: 500;    font-size: 18px;    height: 45px;    width: 180px;    border: none;  }  .icon {    margin-right: 5px;  }  #timer {    display: none;    background-color: #c1121f;    color: #fdf0d5;    font-size: 18px;    border-radius: 100%;    height: 60px;    width: 60px;    justify-content: center;    align-items: center;    margin-top: 30px;  }  .fileupload {    text-align: center;    padding: 3%;  }  input[type='file'] {    display: none;  }  label {    cursor: pointer;  }  #fileName {    color: #c1121f;    font-weight: bold;  }  // Home.js  . let intervalId  let recorder  let isRecording = false *// Track recording state*  let chunks = []  let stream *// Keep track of the captured stream*  let recordingTimeout *// To store the timeout ID*  let advance = false  function sendToFastAPI(*blob*, *filename*) {    const button = document.getElementById('share-audio-button')    const originalText = button.textContent  *// Show loader on the button*    button.textContent = 'Uploading...'    button.disabled = true    const formData = new FormData()    formData.append('file', *blob*, *filename*)    fetch('http://127.0.0.1:8000/upload', {      method: 'POST',      body: formData,    })      .then((*response*) => {  *if* (!*response*.ok) {  *throw* new Error('Network response was not ok')        }  *return* *response*.json()      })      .then((*data*) => {        console.log('File upload successful:', *data*)        displayApiResponse(*data*) *// Function to display API response on the UI*      })      .catch((*error*) => {        console.error('Error uploading file:', *error*)        displayApiError(*error*) *// Function to display error message on the UI*      })      .finally(() => {  *// Restore button text*        button.textContent = originalText        button.disabled = false      })  }  function calculatePercentages(*data*) {    let genuineTotal = 0    let deepfakeTotal = 0    const totalCount = *data*.length  *data*.forEach((*pair*) => {      genuineTotal += *pair*[0]      deepfakeTotal += *pair*[1]    })    const genuinePercentage = (genuineTotal / totalCount) \* 100    const deepfakePercentage = (deepfakeTotal / totalCount) \* 100    const str = `<p class="gpercentage">Genuine Percentage: ${genuinePercentage}</p>                <p class="predictions">deepfakepercentage: ${deepfakePercentage}</p>                `  *return* str  }  function displayApiResponse(*data*) {    const responseContainer = document.getElementById('response-container')    const left = document.getElementById('left')    left.style.display = 'inline'    const predictions = *data*.predictions    const genuineCount = predictions.filter((*pred*) => *pred* === 0).length    const deepfakeCount = predictions.filter((*pred*) => *pred* === 1).length    const totalPredictions = predictions.length    const genuinePercentage = ((genuineCount / totalPredictions) \* 100).toFixed(2)    const deepfakePercentage = ((deepfakeCount / totalPredictions) \* 100).toFixed(2)  *// Function to update the UI*    function updateUI() {      responseContainer.innerHTML = ''  *if* (advance) {        const advanceContent = `          <p class="ResHead">Advance:</p>          <p class="Fname">Filename: ${*data*.filename}</p>          <p class="Fpath">File Path: ${*data*.file\_path}</p>          <p class="genuinePercentage">${genuinePercentage}% is genuine</p>          <p class="deepfakePercentage">${deepfakePercentage}% is deepfake</p>          <p class="predictions">Predictions: ${*data*.predictions}</p>          <h5>Probabilities: </h5>          <p class="probability">${calculatePercentages(*data*.probabilities)}</p>      `        responseContainer.insertAdjacentHTML('beforeend', advanceContent)      } *else* {        left.style.width = '' *// Reset width when not in advance mode*        const basicContent = `            <p class="ResHead">Basic:</p>            <p class="genuinePercentage">${genuinePercentage}% is genuine</p>            <p class="deepfakePercentage">${deepfakePercentage}% is deepfake</p>        `        responseContainer.insertAdjacentHTML('beforeend', basicContent)      }      const buttonContainer = document.createElement('div')      buttonContainer.classList.add('advBtnCont')      const showMoreButton = document.createElement('button')      showMoreButton.classList.add('advanceBtn')      showMoreButton.textContent = 'Show More'      const likeButton = document.createElement('button')      likeButton.classList.add('like')      likeButton.style.border = 'none'      likeButton.innerHTML = '<i class="fa-solid fa-thumbs-up"></i>'      likeButton.addEventListener('click', () => {  *// Call API with user's choice (like)*        sendDataFeedback(data.filename, data.file\_path, 'like')      })      const dislikeButton = document.createElement('button')      dislikeButton.classList.add('dislike')      dislikeButton.style.border = 'none'      dislikeButton.innerHTML = '<i class="fa-solid fa-thumbs-down"></i>'      dislikeButton.addEventListener('click', () => {  *// Call API with user's choice (dislike)*        sendDataFeedback(data.filename, data.file\_path, 'dislike')      })      showMoreButton.addEventListener('click', () => {        advance = !advance *// Toggle the advance state*        updateUI() *// Update the UI*      })      buttonContainer.appendChild(showMoreButton)      buttonContainer.appendChild(likeButton)      buttonContainer.appendChild(dislikeButton)      responseContainer.appendChild(buttonContainer)    }  *// Initial UI update*    updateUI()  }  function sendDataFeedback(*filename*, *filePath*, *choice*) {  *// Prepare data to send*    const formData = { filename, filePath, choice }  *// Convert the object to JSON string*    const jsonData = JSON.stringify(formData)  *// Send data to the API*    fetch('http://localhost:8000/data-collect', {      method: 'POST',      headers: {        'Content-Type': 'application/json',      },      body: jsonData,    })      .then((*response*) => {  *if* (!response.ok) {  *throw* new Error('Network response was not ok')        }  *return* response.json()      })      .then((*data*) => {        console.log('Data feedback sent successfully:', data)  *// Handle response if needed*      })      .catch((*error*) => {        console.error('Error sending data feedback:', error)  *// Handle error if needed*      })  }  function displayApiError(*error*) {    const responseContainer = document.getElementById('response-container')    responseContainer.innerHTML = `      <p>Error: ${error.message}</p>    `  }  function generateFilename() {    const date = new Date()    const dateString = date.toISOString().replace(/:/g, '-')    const randomHash = Math.random().toString(36).substring(2, 15)  *return* `recording\_${dateString}\_${randomHash}.wav`  }  function startRecording(*stream*) {    const context = new AudioContext({ sampleRate: 48000 })    const source = context.createMediaStreamSource(stream)    fileName.innerText = ''    context.audioWorklet      .addModule('audio-processor.js')      .then(() => {        const audioProcessor = new AudioWorkletNode(context, 'audio-processor')        source.connect(audioProcessor)        audioProcessor.connect(context.destination)        recorder = new MediaRecorder(stream)        recorder.ondataavailable = (*e*) => {          chunks.push(e.data)        }        recorder.onstop = () => {          clearInterval(intervalId)          clearTimeout(recordingTimeout)          const filename = generateFilename()          sendToFastAPI(new Blob(chunks, { type: 'audio/wav' }), filename)  *if* (stream) {            stream.getTracks().forEach((*track*) => track.stop())            stream = null          }        }        recorder.start()        recordingTimeout = setTimeout(() => {          stopRecording()        }, 60000)      })      .catch((*error*) => {        console.error('Error loading AudioWorkletNode:', error)      })  }  function stopRecording() {  *if* (recorder && recorder.state === 'recording') {      recorder.stop()    }    isRecording = false    document.getElementById('share-audio-button').textContent = 'Start Recording'  }  function captureTabAudio() {    let timeElapsed = 0    const timerElement = document.getElementById('timer')    timerElement.textContent = `${timeElapsed}s`    timerElement.style.display = 'flex'    intervalId = setInterval(() => {      timeElapsed++      timerElement.textContent = `${timeElapsed}s`    }, 1000)    chrome.tabCapture.capture({ audio: true, video: false }, (*capturedStream*) => {  *if* (!capturedStream) {        clearInterval(intervalId)        console.error('Failed to capture tab audio.')  *return*      }      stream = capturedStream      startRecording(stream)    })  }  function handleFileUpload() {  *// Get the uploaded file*    let uploadedFile = this.files[0]    const timerElement = document.getElementById('timer')    timerElement.style.display = 'none'  *// Check if the uploaded file is a .wav file*  *if* (uploadedFile && uploadedFile.type === 'audio/wav') {  *// Create a Blob from the file data*      let blob = new Blob([uploadedFile], { type: 'audio/wav' })  *// Create an object URL for the Blob*      let objectURL = URL.createObjectURL(blob)      console.log('File Blob:', blob)      console.log('Object URL:', objectURL)  *// Function to get the duration of the audio file*  *// getDirection function used from https://stackoverflow.com/a/41245574*      var getDuration = function (*url*, *next*) {        var \_player = new Audio(url)        \_player.addEventListener(          'durationchange',          function (*e*) {            console.log('Duration Change Event:', e)  *if* (this.duration !== Infinity) {              var duration = this.duration              console.log('Duration:', duration)              \_player.remove()              next(duration)            }          },          false        )        \_player.load()        \_player.currentTime = 24 \* 60 \* 60 *// Set a large current time*        \_player.volume = 0        \_player.play()  *// Waiting...*      }  *// Call getDuration function with the object URL and handle the duration*      getDuration(objectURL, function (*duration*) {        console.log('Received Duration:', duration)  *// Ensure the duration is finite and within the limit*  *if* (isFinite(duration) && duration <= 60) {  *// Rename the file*          let newName = generateFilename()          console.log('New Filename:', newName)  *// Display the new filename*          fileName.innerText = newName  *// Send the renamed file to the server*          sendToFastAPI(uploadedFile, newName)        } *else* {          alert('Please upload a .wav file that is no longer than 60 seconds.')        }      })    } *else* {      alert('Please upload a .wav file.')    }  }  function encodeToWav(*channelData*) {    const numChannels = 1 *// Mono audio*    const sampleRate = 48000 *// Same as the AudioContext sample rate*    const bytesPerSample = 2 *// 16-bit output*    const bitsPerSample = 16    const buffer = new ArrayBuffer(channelData.length \* numChannels \* bytesPerSample)    const view = new DataView(buffer)    let offset = 0  *for* (let i = 0; i < channelData.length; i++) {      const sample = channelData[i]      const value = Math.max(-1, Math.min(1, sample))      const output = value < 0 ? value \* 0x8000 : value \* 0x7fff      view.setInt16(offset, output, true)      offset += bytesPerSample    }    const wavBlob = new Blob([encodeWavHeader(buffer.byteLength, numChannels, sampleRate), buffer], { type: 'audio/wav' })    sendToFastAPI(wavBlob, generateFilename())  }  function encodeWavHeader(*length*, *numChannels*, *sampleRate*) {    const dataLength = length - 44 *// Length of the raw audio data*    const buffer = new ArrayBuffer(44)    const view = new DataView(buffer)    const writeString = (*offset*, *string*) => {  *for* (let i = 0; i < string.length; i++) {        view.setUint8(offset + i, string.charCodeAt(i))      }    }    const writeUint32 = (*offset*, *value*) => {      view.setUint32(offset, value, true)    }    const writeUint16 = (*offset*, *value*) => {      view.setUint16(offset, value, true)    }    writeString(0, 'RIFF') *// ChunkID*    writeUint32(4, 32 + dataLength) *// ChunkSize*    writeString(8, 'WAVE') *// Format*    writeString(12, 'fmt ') *// Subchunk1ID*    writeUint32(16, 16) *// Subchunk1Size*    writeUint16(20, 1) *// AudioFormat (1 = PCM)*    writeUint16(22, numChannels) *// NumChannels*    writeUint32(24, sampleRate) *// SampleRate*    writeUint32(28, sampleRate \* 2) *// ByteRate*    writeUint16(32, 2) *// BlockAlign*    writeUint16(34, 16) *// BitsPerSample*    writeString(36, 'data') *// Subchunk2ID*    writeUint32(40, dataLength) *// Subchunk2Size*  *return* view.buffer  }  document.addEventListener('DOMContentLoaded', function () {    let input = document.getElementById('upload-wav-button')    let fileName = document.getElementById('fileName')    input.addEventListener('change', handleFileUpload)    const button = document.getElementById('share-audio-button')    button.addEventListener('click', function () {  *if* (isRecording) {        stopRecording()      } *else* {        isRecording = true        chunks = []        document.getElementById('timer').textContent = '0s'        button.textContent = 'Stop Recording'        captureTabAudio()      }    })  })  function toggleShowMore() {    !advance    displayApiResponse()    console.log('advance clicked')  }  document.body.addEventListener('click', function (*event*) {  *if* (event.target && event.target.classList.contains('advanceBtn')) {      toggleShowMore()    }  })  // Manifest.json  {      "name": "Deep Inspect",      "version": "0.2",      "manifest\_version": 3,      "permissions": [          "tabs",          "activeTab",          "tabCapture"      ],      "host\_permissions": [          "https://\*/"      ],      "action": {          "default\_popup": "home.html",          "default\_action": "home.js",          "default\_icon":"logo128.png"      }    } |

*Figure 5.2 Chrome Extension Code*

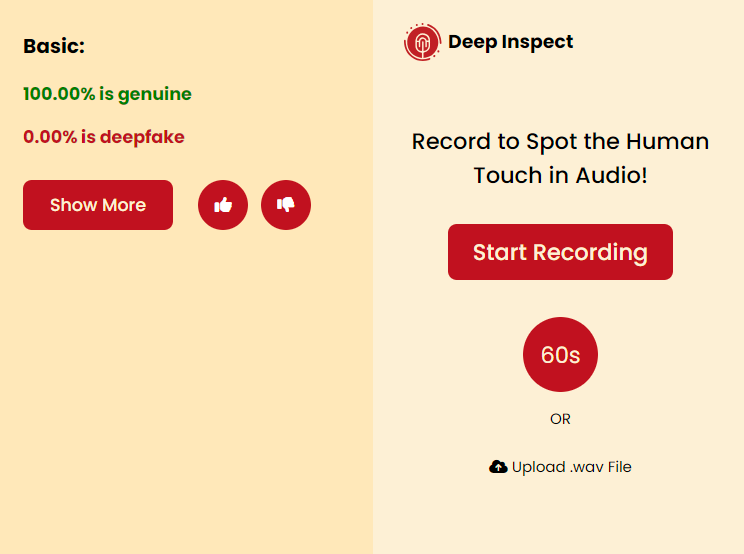
The provided code consists of a Chrome extension named "Deep Inspect" version 0.2. It has permissions to access tabs, active tabs, and capture tab data. The extension includes HTML, CSS, and JavaScript files.

The HTML file (`home.html`) contains a layout with two main sections: "right" and "left". The "right" section includes a logo and a container with a heading, a button to start recording audio, and an option to upload a .wav file. The "left" section is initially hidden and serves as a container for displaying API responses.

The CSS file (`home.css`) provides styling for the HTML elements, including the layout, button styles, and response container.

The JavaScript file (`home.js`) handles various functionalities of the extension. It includes functions to start and stop recording audio from the tab, capture audio from the tab, upload recorded or uploaded .wav files to a FastAPI server, display API responses, calculate and display percentages, handle user feedback (like/dislike), and generate filenames. Additionally, it contains functions for encoding audio data to .wav format and creating .wav file headers.

### 5.3.2 Business Logic Layer



|  |
| --- |
| // Main.py  *from* typing *import* List, Dict, Any  *import* os  *import* logging  *from* logging.handlers *import* RotatingFileHandler  *from* fastapi *import* Request  *from* fastapi *import* FastAPI, File, UploadFile  *from* fastapi.middleware.cors *import* CORSMiddleware  *from* inference *import* analyze\_audio\_batch, print\_predictions, split\_wav\_file, process\_predictions\_and\_probabilities  *import* numpy *as* np  *import* json  app = FastAPI()  *# Configure logging*  logging.basicConfig(  *level*=logging.INFO,  *format*='%(asctime)s - %(levelname)s - %(message)s'  )  *# Add a handler for main.log*  main\_log\_handler = RotatingFileHandler("main.log", *maxBytes*=10485760, *backupCount*=5)  main\_log\_handler.setLevel(logging.INFO)  formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')  main\_log\_handler.setFormatter(formatter)  logging.getLogger('').addHandler(main\_log\_handler)  FILE\_METADATA\_PATH = 'file\_metadata.json'  *# Allow requests from your Chrome extension's origin*  app.add\_middleware(      CORSMiddleware,  *allow\_origins*=["\*"],  *allow\_credentials*=True,  *allow\_methods*=["GET", "POST"],  *allow\_headers*=["\*"],  )  UPLOAD\_FOLDER = "uploads"  *# Define the folder where uploaded files will be stored*  *if* not os.path.exists(UPLOAD\_FOLDER):      os.makedirs(UPLOAD\_FOLDER)      logging.info(f"Created upload folder: {UPLOAD\_FOLDER}")  def save\_file\_to\_server(*file*: UploadFile) -> str:      file\_path = os.path.join(UPLOAD\_FOLDER, *file*.filename)  *with* open(file\_path, "wb") *as* f:          f.write(*file*.file.read())      logging.info(f"File saved to: {file\_path}")  *return* file\_path  def cleanup\_files(*file\_paths*: List[str]):      logging.info(*file\_paths*)  *for* file\_path *in* *file\_paths*:  *try*:  *if* os.path.exists(file\_path):                  os.remove(file\_path)                  logging.info(f"Deleted file: {file\_path}")  *else*:                  logging.warning(f"File not found: {file\_path}")  *except* Exception *as* e:              logging.error(f"Error deleting file {file\_path}: {e}")  def load\_file\_metadata():  *if* os.path.exists(FILE\_METADATA\_PATH):  *with* open(FILE\_METADATA\_PATH, 'r') *as* f:  *return* json.load(f)  *return* []  def save\_file\_metadata(*metadata*):  *with* open(FILE\_METADATA\_PATH, 'w') *as* f:          json.dump(*metadata*, f, *indent*=2)  @app.post("/upload")  async def upload\_file(*file*: UploadFile = File(...)) -> Dict[str, Any]:  *try*:  *# Save the file to the server*          file\_path = save\_file\_to\_server(*file*)          logging.info(f"File saved to: {file\_path}")  *# Normalize the file path for Python*          normalized\_file\_path = os.path.normpath(file\_path)          logging.info(f"Normalized file path: {normalized\_file\_path}")  *# Load existing file metadata*          file\_metadata = load\_file\_metadata()  *# Break down audio into 1-second clips and save*          input\_audio\_paths = split\_wav\_file(normalized\_file\_path)          logging.info(f"Created audio segments: {input\_audio\_paths}")          predictions, probabilities = analyze\_audio\_batch(input\_audio\_paths)          logging.info(f"Predictions: {predictions}")          logging.info(f"Probabilities: {probabilities}")          logging.info(print\_predictions(predictions, probabilities))          process\_predictions\_and\_probabilities(predictions, probabilities)  *# Clean up temporary files*          cleanup\_files(input\_audio\_paths)  *# Convert numpy arrays to lists for serialization*          predictions = predictions.tolist()          probabilities = probabilities.tolist()  *# Add new file metadata*          new\_metadata = {              "filename": *file*.filename,              "file\_path": normalized\_file\_path,              "choice": "unchecked",              "predictions": predictions,              "probabilities": probabilities          }          file\_metadata.append(new\_metadata)  *# Save updated file metadata*          save\_file\_metadata(file\_metadata)  *# Ensure predictions and probabilities can be serialized*          response = {              "filename": *file*.filename,              "file\_path": file\_path,              "predictions": predictions,              "probabilities": probabilities          }  *return* response  *except* Exception *as* e:          logging.error(f"Error processing file: {e}")  *return* {"error": str(e)}    @app.post("/data-collect")  async def collect\_data\_feedback(*request*: Request):  *try*:          request\_data = *await* *request*.json()          filename = request\_data.get("filename")          file\_path = request\_data.get("filePath")          choice = request\_data.get("choice")  *# Load existing file metadata*          file\_metadata = load\_file\_metadata()  *# Find the metadata for the given file*          file\_metadata\_entry = next((entry *for* entry *in* file\_metadata *if* entry["filename"] == filename and entry["file\_path"] == file\_path), None)  *if* file\_metadata\_entry:  *# Update the choice value*              file\_metadata\_entry["choice"] = choice  *# Save the updated file metadata*              save\_file\_metadata(file\_metadata)  *return* {"message": "Data feedback received successfully"}  *else*:  *return* {"error": "File metadata not found"}  *except* Exception *as* e:          logging.error(f"Error processing data feedback: {e}")  *return* {"error": str(e)} |

*Figure 5.3 FastAPI Backend*

The provided code consists of a FastAPI application for audio file processing and analysis. It includes functionalities for uploading audio files, splitting them into segments, analyzing audio batches for classification (genuine or deepfake), and collecting feedback on the analyzed data.

The application utilizes FastAPI for handling HTTP requests and responses. It includes endpoints for file upload (`/upload`) and data feedback (`/data-collect`). Upon uploading a file, it saves the file to a specified folder, splits it into 1-second segments, performs analysis using machine learning models, and saves metadata about the file and analysis results. Feedback can be collected on the analyzed data through the `/data-collect` endpoint.

The code is well-structured, with logging for error handling and monitoring. It also utilizes multithreading for parallel processing of audio segments to improve efficiency. Additionally, the code includes separate modules for feature extraction and analysis, facilitating modularity and reusability.

Overall, the application provides a robust framework for processing and analyzing audio files, particularly for detecting deepfake content.

### 5.3.3 Processing Layer

|  |
| --- |
| // Inference.py  *import* os  *import* numpy *as* np  *import* joblib  *import* librosa  *import* soundfile *as* sf  *import* parselmouth  *import* time  *from* concurrent.futures *import* ThreadPoolExecutor, as\_completed  *import* logging  *# Configure logging*  logging.basicConfig(*filename*='main.log', *level*=logging.DEBUG, *format*='%(asctime)s - %(levelname)s - %(message)s')  def split\_wav\_file(*path*):  *try*:          logging.info(f"Loading audio file: {*path*}")          y, sr = librosa.load(*path*, *sr*=None)          sample\_length = sr          num\_segments = len(y) // sample\_length          segmented\_audio = []  *for* i *in* range(num\_segments):              start = i \* sample\_length              end = (i + 1) \* sample\_length              segment = y[start:end]              temp\_path = f"segment\_{i}.wav"              sf.write(temp\_path, segment, sr)              segmented\_audio.append(temp\_path)              logging.info(f"Segmented file saved: {temp\_path}")  *return* segmented\_audio  *except* Exception *as* e:          logging.error(f"Error in split\_wav\_file: {e}")  *return* []  def extract\_mfccs\_(*audio\_data*, *sr*, *n\_mfcc*=20, *n\_fft*=2048, *hop\_length*=512):  *try*:          logging.info("Extracting MFCCs")          mfccs = librosa.feature.mfcc(*y*=*audio\_data*, *sr*=*sr*, *n\_mfcc*=*n\_mfcc*, *n\_fft*=*n\_fft*, *hop\_length*=*hop\_length*)  *return* np.mean(mfccs.T, *axis*=0)  *except* Exception *as* e:          logging.error(f"Error in extract\_mfccs\_: {e}")  *return* np.array([])  def compute\_lpcc(*lpc\_coeffs*, *sr*):  *try*:          logging.info("Computing LPCC")          lpcc\_coeffs = np.zeros\_like(*lpc\_coeffs*)          lpcc\_coeffs[0] = np.log(*lpc\_coeffs*[0])          i\_sr = np.arange(1, len(*lpc\_coeffs*)) / *sr*  *for* i *in* range(1, len(*lpc\_coeffs*)):              lpcc\_coeffs[i] = *lpc\_coeffs*[i] + np.dot(lpcc\_coeffs[1:i], *lpc\_coeffs*[i-1:0:-1]) \* i\_sr[i-1]  *return* lpcc\_coeffs  *except* Exception *as* e:          logging.error(f"Error in compute\_lpcc: {e}")  *return* np.array([])  def extract\_lpccs\_(*audio\_data*, *sr*, *order*=19):  *try*:          logging.info("Extracting LPCCs")          lpc\_coeffs = librosa.lpc(*audio\_data*, *order*=*order*)          lpcc\_coeffs = compute\_lpcc(lpc\_coeffs, *sr*)  *return* lpcc\_coeffs.T  *except* Exception *as* e:          logging.error(f"Error in extract\_lpccs\_: {e}")  *return* np.array([])  def estimate\_vot(*audio\_path*):  *try*:          logging.info(f"Estimating VOT for: {*audio\_path*}")          sound = parselmouth.Sound(*audio\_path*)          pitch = sound.to\_pitch()          pitch\_times = pitch.xs()          vot\_time = pitch\_times[0] *if* len(pitch\_times) > 0 *else* None  *return* vot\_time  *except* Exception *as* e:          logging.error(f"Error in estimate\_vot: {e}")  *return* None  def compute\_energy\_and\_zcr(*y*, *hop\_length*):  *try*:          logging.info("Computing energy and ZCR")          energy = np.array([np.sum(*y*[i:i+*hop\_length*]\*\*2) *for* i *in* range(0, len(*y*), *hop\_length*)])          zcr = np.array([np.sum(np.diff(*y*[i:i+*hop\_length*] > 0)) *for* i *in* range(0, len(*y*), *hop\_length*)])  *return* energy, zcr  *except* Exception *as* e:          logging.error(f"Error in compute\_energy\_and\_zcr: {e}")  *return* np.array([]), np.array([])  def find\_burst\_and\_periodicity(*energy*, *zcr*, *hop\_length*, *sr*):  *try*:          logging.info("Finding burst and periodicity")          energy\_diff = np.diff(*energy*)          burst\_frame = np.argmax(energy\_diff) + 1          zcr\_diff = np.diff(*zcr*)          periodicity\_frame = np.argmin(zcr\_diff) + 1          burst\_time = burst\_frame \* *hop\_length* / *sr*          periodicity\_time = periodicity\_frame \* *hop\_length* / *sr*  *return* burst\_time, periodicity\_time  *except* Exception *as* e:          logging.error(f"Error in find\_burst\_and\_periodicity: {e}")  *return* None, None  def detect\_burst\_and\_periodicity\_onset(*audio\_data*, *sr*, *hop\_length*=512):  *try*:          logging.info("Detecting burst and periodicity onset")          energy, zcr = compute\_energy\_and\_zcr(*audio\_data*, *hop\_length*)          burst\_time, periodicity\_time = find\_burst\_and\_periodicity(energy, zcr, *hop\_length*, *sr*)  *return* burst\_time, periodicity\_time  *except* Exception *as* e:          logging.error(f"Error in detect\_burst\_and\_periodicity\_onset: {e}")  *return* None, None  def extract\_features(*audio\_path*):  *try*:          logging.info(f"Extracting features from: {*audio\_path*}")          audio\_data, sr = librosa.load(*audio\_path*, *sr*=None)          mfcc\_features = extract\_mfccs\_(audio\_data, sr)          lpcc\_features = extract\_lpccs\_(audio\_data, sr)          vot = estimate\_vot(*audio\_path*)          burst\_time, periodicity\_time = detect\_burst\_and\_periodicity\_onset(audio\_data, sr)          vot\_feature = np.array([vot, burst\_time, periodicity\_time])          feature\_vector = np.concatenate((mfcc\_features, lpcc\_features, vot\_feature))  *return* feature\_vector  *except* Exception *as* e:          logging.error(f"Error in extract\_features: {e}")  *return* None  def process\_audio(*input\_audio\_path*, *scaler*, *svm\_classifier*):  *try*:          logging.info(f"Processing audio: {*input\_audio\_path*}")  *if* not os.path.exists(*input\_audio\_path*):              logging.error(f"File does not exist: {*input\_audio\_path*}")  *return* None, None  *elif* not *input\_audio\_path*.lower().endswith(".wav"):              logging.error(f"Not a .wav file: {*input\_audio\_path*}")  *return* None, None          features = extract\_features(*input\_audio\_path*)  *if* features is not None:              features\_scaled = *scaler*.transform(features.reshape(1, -1))              prediction = *svm\_classifier*.predict(features\_scaled)              probability = *svm\_classifier*.predict\_proba(features\_scaled)  *return* prediction[0], probability[0]  *else*:              logging.error(f"Unable to process input audio: {*input\_audio\_path*}")  *return* None, None  *except* Exception *as* e:          logging.error(f"Error in process\_audio: {e}")  *return* None, None  def analyze\_audio\_batch(*input\_audio\_paths*):      start\_time = time.time()      logging.info("Starting audio batch analysis")  *try*:          model\_filename = "./svm\_model\_142\_rbf.pkl"          scaler\_filename = "./scaler\_142\_rbf.pkl"          svm\_classifier = joblib.load(model\_filename)          scaler = joblib.load(scaler\_filename)          predictions = []          probabilities = []  *with* ThreadPoolExecutor() *as* executor:              futures = [executor.submit(process\_audio, input\_audio\_path, scaler, svm\_classifier) *for* input\_audio\_path *in* *input\_audio\_paths*]  *for* future *in* as\_completed(futures):                  prediction, probability = future.result()                  predictions.append(prediction)                  probabilities.append(probability)          end\_time = time.time()          elapsed\_time = end\_time - start\_time          logging.info(f"Total time taken: {elapsed\_time:.2f} seconds")  *return* np.array(predictions), np.array(probabilities)  *except* Exception *as* e:          logging.error(f"Error in analyze\_audio\_batch: {e}")  *return* np.array([]), np.array([])  def print\_predictions(*predictions*, *probabilities*):  *try*:          logging.info("Printing predictions and probabilities")          print(f"{'Index':<6} || {'Class Predicted':<16} || {'Probability of Real':<20} || {'Probability of Fake':<20}")          print("="\*76)  *for* i, (pred, prob) *in* enumerate(zip(*predictions*, *probabilities*)):              class\_predicted = "Genuine" *if* pred == 0 *else* "Deepfake"              prob\_real = prob[0]              prob\_fake = prob[1]              print(f"{i:<6} || {class\_predicted:<16} || {prob\_real:<20.4f} || {prob\_fake:<20.4f}")  *except* Exception *as* e:          logging.error(f"Error in print\_predictions: {e}")  def process\_predictions\_and\_probabilities(*predictions*, *probabilities*):  *try*:          logging.info("Processing predictions and probabilities")          count\_0 = np.count\_nonzero(*predictions* == 0)          count\_1 = np.count\_nonzero(*predictions* == 1)          avg\_genuine\_prob = np.mean(*probabilities*[:, 0])          avg\_deepfake\_prob = np.mean(*probabilities*[:, 1])  *return* [count\_0, count\_1, avg\_genuine\_prob, avg\_deepfake\_prob]  *except* Exception *as* e:          logging.error(f"Error in process\_predictions\_and\_probabilities: {e}")  *return* []  \_\_all\_\_ = ['analyze\_audio\_batch', 'print\_predictions', 'split\_wav\_file','process\_predictions\_and\_probabilities'] |

*Figure 5.4 Python Module to run model*

The inference.py file is essential for analyzing audio batches to discern genuine from deepfake content. It contains functions for feature extraction like MFCCs and LPCCs, crucial for identifying subtle audio nuances. Additionally, it facilitates segmentation of audio files into smaller chunks for efficient processing. The core function, process\_audio, uses an SVM classifier to label segments based on features. analyze\_audio\_batch coordinates batch analysis, optimizing processing efficiency. Finally, output functions refine predictions for easier interpretation. Overall, inference.py is pivotal in detecting deepfake content in audio files.

**Chapter 6:**

**Testing and Evaluation**

# Chapter 6: Testing and Evaluation

## 6.1 Introduction

Testing is a crucial phase in the development lifecycle of any application, ensuring that the software behaves as expected, meets the requirements, and is free of critical bugs. For our project, which involves sophisticated audio analysis and deepfake detection, thorough testing is essential to validate the accuracy and reliability of the system.

Given the complexity of our project, which includes a backend built with FastAPI and a frontend that interfaces with it, testing needs to be comprehensive and cover various aspects including unit tests, integration tests, and performance tests.

## 6.2 List of Test Scenarios

Throughout the development and testing phases of our audio analysis and deepfake detection application, we encountered various challenges and devised solutions to ensure the reliability and effectiveness of our system. Initially, in the backend audio processing tests, issues arose with memory and performance when extracting Mel-frequency cepstral coefficients (MFCCs) from audio files using the librosa library. To address this, we optimized the feature extraction process to handle large files more efficiently. Additionally, while estimating Voice Onset Time (VOT) with the Parselmouth library, occasional inaccuracies were observed, prompting us to implement pre-processing steps for audio cleaning before VOT estimation.

In the feature extraction and machine learning integration phase, we focused on extracting a comprehensive set of features, including MFCCs, LPCCs, and VOT, while training a Support Vector Machine (SVM) classifier for deepfake detection. Challenges such as biased predictions due to imbalanced data were mitigated through data augmentation and re-sampling techniques. Moreover, deployment challenges were addressed by encapsulating the application environment using configuring CORS middleware in FastAPI to ensure seamless interaction between the frontend and backend.

End-to-end functional testing was conducted to validate the entire workflow from uploading audio files to receiving deepfake detection results. Robust error handling and retry mechanisms were implemented to address inconsistencies in results due to varying network conditions and file sizes. Performance and load testing were carried out to assess the system's performance under high load conditions, leading to optimizations in handling concurrent requests efficiently using threading and task queues.

Finally, user acceptance testing (UAT) played a crucial role in validating the application with real users and gathering feedback. Iterative improvements were made based on user feedback, focusing on enhancing the user interface and providing detailed error messages to improve the overall user experience. By addressing these challenges and test scenarios iteratively, we ensured that our audio analysis and deepfake detection application is robust, reliable, and user-friendly, meeting the desired objectives effectively.

## 6.3 Performance and Evaluation

### 6.3.1 Functional Testing

In functional testing of our model and app we performed multiple testing methods. For evaluation of our model, we used Accuracy, Precision, Recall, F1-Score, EER, ROC-AUC curve and Cross Validation which are shown below.

|  |  |
| --- | --- |
| Evaluation Metrics | Results |
| Accuracy | 0.972% |
| Precision | 0.970% |
| Recall | 0.981% |
| F1-Score | 0.977% |
| EER | 0.03% |

Cross Validation Scores for Kstratified nfolds=5 are given below

|  |  |
| --- | --- |
| Kstratified Folds | Scores |
| Fold 1 | 0.9676259 |
| Fold 2 | 0.96842526 |
| Fold 3 | 0.97002398 |
| Fold 4 | 0.96561375 |
| Fold 5 | 0.97481008 |
| Mean CV Accuracy | 0.9692997940711805 |

#### **6.3.1.1 Confusion Matrix**

A confusion matrix is a powerful tool for evaluating the performance of a classification model, providing a visual representation of the model’s predictions compared to the actual values. It consists of four quadrants: true positives (TP) and true negatives (TN), which represent accurate predictions, and false positives (FP) and false negatives (FN), which indicate errors. Descriptive statistics derived from the matrix, such as accuracy, precision, recall, and F1 score, offer a nuanced understanding of the model’s strengths and weaknesses. Accuracy reveals the overall correctness, precision measures the exactness, recall assesses the completeness, and the F1 score balances precision and recall, especially when their importance is unequal. These metrics collectively help in fine-tuning the model for better performance. For instance, in a matrix with high values of TP and TN, and low values of FP and FN, the model is considered highly accurate and reliable.

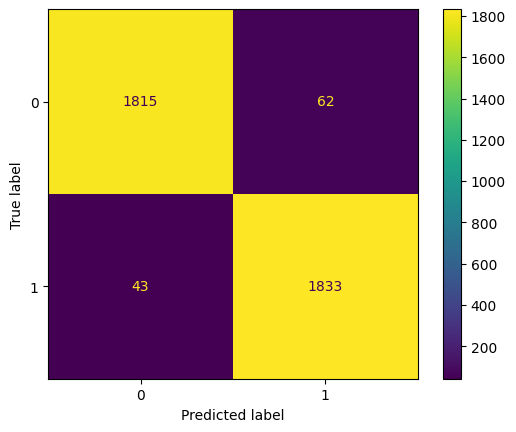


Figure 9 Confusion Metrics

#### **6.3.1.2ROC-AUC**

The ROC (Receiver Operating Characteristic) curve and AUC (Area Under the Curve) are essential tools for evaluating the performance of a binary classification model. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings, which helps in understanding the trade-offs between correctly identifying positives and falsely identifying negatives. The AUC provides a single scalar value that summarizes the overall ability of the model to discriminate between the positive and negative classes across all thresholds. An AUC of 1.0 signifies a perfect model that makes no mistakes, while an AUC closer to 0.5 indicates a model performing no better than random chance.

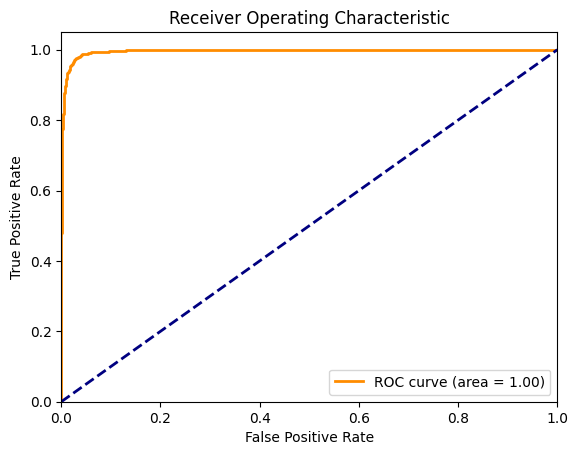


Figure 10 ROC-AUC

### 6.3.2 Non-Functional Testing

Non-functional testing plays a critical role in assessing the performance, scalability, security, and usability aspects of an application beyond its basic functionality. In the context of our audio analysis and deepfake detection application, several non-functional testing strategies were employed to ensure its robustness and reliability.

**Performance Testing:**

Performance testing was conducted to evaluate how the system performs under various workloads and stress conditions. Load testing, stress testing, and endurance testing were performed to assess the system's response time, throughput, and resource utilization under different levels of load. By simulating concurrent user interactions and analyzing system metrics, we identified potential bottlenecks and optimized the application for optimal performance.

**Scalability Testing:**

Scalability testing was conducted to assess the application's ability to handle increasing workload and scale resources accordingly. Vertical scalability, achieved by increasing the capacity of individual components, and horizontal scalability, achieved by adding more instances of components, were evaluated to ensure seamless scalability as the user base grows. By monitoring system performance under varying loads and scaling scenarios, we ensured the application can adapt to changing demands without compromising performance.

**Security Testing:**

Security testing was conducted to identify and mitigate potential vulnerabilities and ensure the confidentiality, integrity, and availability of data. Techniques such as penetration testing, vulnerability scanning, and security audits were employed to assess the application's resilience against common security threats such as SQL injection, cross-site scripting (XSS), and unauthorized access. By implementing robust authentication, authorization, and encryption mechanisms, we fortified the application against security breaches and data leaks.

**Usability Testing:**

Usability testing was conducted to evaluate the application's user interface, user experience, and accessibility aspects. Real users were involved in testing the application's intuitiveness, ease of navigation, and overall user satisfaction. Feedback was collected through surveys, interviews, and user observation sessions to identify usability issues and improve the application's design and usability. By incorporating user feedback and making iterative improvements, we ensured the application meets the needs and expectations of its target users.

By conducting comprehensive non-functional testing encompassing performance, scalability, security, and usability aspects, we ensured that our audio analysis and deepfake detection application not only meets functional requirements but also delivers a seamless, secure, and user-friendly experience to its users.

**Chapter 7:**

**Conclusion and Outlook**

**Chapter 7:**

**Conclusion and Outlook**

## 7.1 Introduction

In this section, we contextualize the project within the broader landscape of digital media manipulation and the proliferation of deepfake technology. We elucidate the challenges posed by the rampant spread of manipulated audio content and the consequent erosion of trust in media integrity. Furthermore, we highlight the limitations imposed by platforms like Google and Android OS, which restrict real-time audio processing capabilities, thus necessitating innovative solutions for offline deepfake detection.

**7.2 Achievements and Improvements**

Our project has achieved significant milestones in the realm of audio analysis and deepfake detection, marking a substantial leap forward in combating the proliferation of manipulated audio content. Through rigorous research and development, we have successfully devised innovative solutions to address the pressing challenges posed by deepfake technology.

**Achievements:**

**Sophisticated Audio Segmentation**: One of the key achievements of our project is the development of advanced audio segmentation algorithms. These algorithms enable the precise breakdown of audio files into manageable segments, facilitating more accurate analysis and detection of deepfake artifacts.

**Integration of Machine Learning Models:** We have successfully integrated state-of-the-art machine learning models into our system for deepfake detection. Through extensive training and optimization, these models exhibit high accuracy in differentiating between genuine and manipulated audio recordings, thereby bolstering the integrity of digital media content.

**Iterative Improvements:** Our project embodies a spirit of continuous improvement, with iterative enhancements aimed at enhancing performance, scalability, and user experience. By soliciting feedback from users and stakeholders, we have been able to identify areas for refinement and implement iterative updates to the system, ensuring its effectiveness in combating audio-based misinformation.

**Improvement Opportunities:**

**Real-Time Processing Constraints:** While our project excels in offline deepfake detection, there exist limitations in implementing real-time processing capabilities, particularly on mobile devices. Exploring strategies to optimize resource utilization and streamline processing algorithms could enable more efficient real-time detection mechanisms, enhancing the applicability of the technology in diverse scenarios.

**Enhanced User Interface:** Further improvements can be made to the user interface to enhance usability and accessibility. By employing intuitive design principles and incorporating user feedback, we can refine the interface to provide a seamless experience for users interacting with the deepfake detection system.

**Integration with Edge Devices:** Exploring integration opportunities with edge computing devices presents a promising avenue for extending the reach of our technology. By leveraging the computational capabilities of edge devices, we can deploy deepfake detection mechanisms closer to the source of audio content, thereby reducing latency and enhancing overall efficiency.

**Regulatory Compliance and Privacy:** Addressing regulatory compliance and privacy concerns surrounding the collection and analysis of audio data is paramount. By implementing robust data governance frameworks and adhering to established privacy standards, we can instill confidence in users regarding the ethical handling of their data, thereby fostering trust and adoption of the deepfake detection technology.

In summary, while our project has achieved notable success in advancing the state of deepfake detection, there exist ample opportunities for further refinement and innovation. By embracing a culture of continuous improvement and collaboration, we can realize the full potential of our technology in combating audio-based misinformation and upholding the integrity of digital media content.

## 7.3 Critical Review

## The development and implementation of the deepfake detection system represent a significant stride forward in the ongoing battle against audio-based misinformation. However, a critical review of the project reveals both strengths and areas for improvement, shedding light on key considerations for future development and deployment.

## The project demonstrates commendable technical innovation in the field of audio analysis and machine learning. By leveraging advanced algorithms and models, the system achieves impressive accuracy in identifying deepfake artifacts, contributing to the mitigation of digital misinformation. A notable strength of the project lies in its interdisciplinary approach, bringing together expertise from fields such as computer science, signal processing, and machine learning. This collaborative effort fosters diverse perspectives and facilitates the development of robust, multifaceted solutions to complex challenges.

## A critical assessment of the project also highlights areas for improvement. Despite the high accuracy achieved by the deepfake detection system, there remains a need for greater algorithmic transparency and interpretability. Providing insights into the decision-making process of the machine learning models can enhance trust and facilitate better understanding among users and stakeholders. Ethical considerations surrounding data privacy, bias mitigation, and societal impact require careful attention. Addressing these concerns necessitates robust governance frameworks, ethical guidelines, and mechanisms for accountability to ensure responsible development and deployment of the technology.

## As the volume and complexity of audio data continue to grow, ensuring scalability and performance becomes increasingly critical. Optimizing the system architecture, implementing efficient processing algorithms, and leveraging emerging technologies such as edge computing can enhance scalability and performance, enabling the system to handle large-scale deployments effectively. Compliance with regulatory requirements and standards governing the use of audio data is essential to mitigate legal risks and safeguard user rights. Strengthening compliance measures, conducting thorough risk assessments, and staying abreast of evolving regulations are imperative to ensure the project's long-term viability and sustainability.

## In conclusion, while the project represents a commendable effort in addressing audio-based misinformation, a critical review underscores the importance of continued refinement, ethical consideration, and regulatory compliance. By addressing the identified areas for improvement and embracing a culture of continuous learning and adaptation, the project can realize its full potential in combating digital misinformation and promoting a more trustworthy digital ecosystem.

## 7.4 Future Recommendations

Looking ahead, several avenues emerge for further enhancement and refinement of the deepfake detection system, paving the way for continued innovation and impact in the field of audio analysis and misinformation mitigation.

Firstly, there is a pressing need to explore novel approaches to address the challenge posed by real-time phone call audio detection. Despite advancements in audio analysis technology, limitations imposed by Google and the Android operating system present significant obstacles to real-time detection on mobile devices. To overcome this barrier, collaboration with industry stakeholders, exploration of alternative platforms, and research into lightweight, efficient algorithms tailored for mobile environments are essential.

Secondly, integrating the deepfake detection technology into edge devices represents a promising avenue for extending its reach and impact. With the emergence of AI chip-based Windows PCs embedding AI capabilities directly into the system, opportunities abound for deploying sophisticated audio analysis algorithms at the edge. By leveraging the computational power of edge devices and minimizing reliance on centralized processing, the system can achieve greater efficiency, responsiveness, and scalability.

Furthermore, automating the model training process using data collected from users and feedback on model outcomes holds immense potential for enhancing system performance and adaptability. By harnessing user-generated data to continuously refine and update the machine learning models, the system can evolve dynamically in response to emerging threats and evolving user needs. Implementing robust mechanisms for data anonymization, privacy protection, and user consent will be paramount to ensure ethical and responsible use of user-generated data.

Moreover, leveraging cryptography for securing audio data on the internet and tracing it back to its sources represents a critical area for future exploration. By integrating cryptographic techniques such as digital signatures, secure hash functions, and blockchain technology, the system can enhance data integrity, authenticity, and traceability, thereby mitigating the spread of misinformation and ensuring accountability in the digital sphere.

In conclusion, embracing these future recommendations presents an opportunity to propel the deepfake detection system to new heights of effectiveness, efficiency, and societal impact. By addressing the identified challenges and seizing emerging opportunities, the project can continue to lead the charge in combating audio-based misinformation and fostering a more trustworthy digital ecosystem for all.

**Bibliography**

[1] “Uncovering the Real Voice: How to Detect and Verify Audio Deepfakes | by Tan Jian Hong | HTX S&S COE | Medium.” Accessed: May 24, 2024. [Online]. Available: https://medium.com/htx-s-s-coe/uncovering-the-real-voice-how-to-detect-and-verify-audio-deepfakes-42e480d3f431

[2] J. Yi *et al.*, “Audio Deepfake Detection: A Survey,” Aug. 2023, Accessed: May 24, 2024. [Online]. Available: https://arxiv.org/abs/2308.14970v1

[3] S. Choudhary, C. R. Karthik, P. S. Lakshmi, and S. Kumar, “LEAN: Light and Efficient Audio Classification Network,” *INDICON 2022 - 2022 IEEE 19th India Council International Conference*, 2022, doi: 10.1109/INDICON56171.2022.10039921.

[4] M. Masood, M. Nawaz, K. M. Malik, A. Javed, and A. Irtaza, “Deepfakes Generation and Detection: State-of-the-art, open challenges, countermeasures, and way forward.”

[5] Z. Khanjani, G. Watson, and V. P. Janeja, “How Deep Are the Fakes?  Focusing on Audio Deepfake: A Survey”.

[6] Y. Takashima, Y. Fujita, S. Watanabe, S. Horiguchi, P. Garcia, and K. Nagamatsu, “End-to-End Speaker Diarization Conditioned on Speech Activity and Overlap Detection,” *2021 IEEE Spoken Language Technology Workshop, SLT 2021 - Proceedings*, pp. 849–856, Jun. 2021, doi: 10.1109/SLT48900.2021.9383555.

[7] I. D. Mienye and Y. Sun, “A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects,” *IEEE Access*, vol. 10, pp. 99129–99149, 2022, doi: 10.1109/ACCESS.2022.3207287.

[8] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A Survey of Model Compression and Acceleration for Deep Neural Networks,” Oct. 2017, Accessed: May 24, 2024. [Online]. Available: https://arxiv.org/abs/1710.09282v9

[9] S. Lyu, “DEEPFAKE DETECTION: CURRENT CHALLENGES AND NEXT STEPS”, Accessed: May 24, 2024. [Online]. Available: https://deepfakedetectionchallenge.ai.

[10] E. M. Bender, T. Gebru, A. Mcmillan-Major, S. Shmitchell, and S.-G. Shmitchell, “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? CCS CONCEPTS • Computing methodologies → Natural language processing. ACM Reference Format”, doi: 10.1145/3442188.3445922.

[11] P. Kawa, M. Plata, and P. Syga, “Defense Against Adversarial Attacks on Audio DeepFake Detection,” *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2023-August, pp. 5276–5280, Dec. 2022, doi: 10.21437/Interspeech.2023-409.

[12] Y. Zhang, B. Kang, B. Hooi, S. Yan, and J. Feng, “Deep Long-Tailed Learning: A Survey,” *IEEE Trans Pattern Anal Mach Intell*, vol. 45, no. 9, pp. 10795–10816, Oct. 2021, doi: 10.1109/TPAMI.2023.3268118.

[13] X. Yang, Z. Song, I. King, and Z. Xu, “A Survey on Deep Semi-Supervised Learning,” *IEEE Trans Knowl Data Eng*, vol. 35, no. 9, pp. 8934–8954, Sep. 2023, doi: 10.1109/TKDE.2022.3220219.

[14] R. Rafique *et al.*, “Deep fake detection and classification using error-level analysis and deep learning,” 123AD, doi: 10.1038/s41598-023-34629-3.

[15] H. Gharoun, F. Momenifar, F. Chen, and A. H. Gandomi, “Meta-learning approaches for few-shot learning: A survey of recent advances,” Mar. 2023, Accessed: May 24, 2024. [Online]. Available: https://arxiv.org/abs/2303.07502v1

[16] D. Cozzolino, M. Nießner, and L. Verdoliva, “Audio-Visual Person-of-Interest DeepFake Detection S) Same Subject D) Different Subject Video Embedding Space”, Accessed: May 24, 2024. [Online]. Available: https://github.com/grip-unina/poi-forensics.

[17] W. Ruan, X. Yi, and X. Huang, “Adversarial Robustness of Deep Learning: Theory, Algorithms, and Applications; Adversarial Robustness of Deep Learning: Theory, Algorithms, and Applications,” *Virtual Event*, 2021, doi: 10.1145/3459637.3482029.

[18] H. Y. Chen and W. L. Chao, “On Bridging Generic and Personalized Federated Learning for Image Classification,” *ICLR 2022 - 10th International Conference on Learning Representations*, Jul. 2021, Accessed: May 24, 2024. [Online]. Available: https://arxiv.org/abs/2107.00778v2

[19] J. Lee *et al.*, “Resource-Efficient Deep Learning: A Survey on Model-, Arithmetic-, and Implementation-Level Techniques,” *ACM Comput Surv*, vol. 55, no. 13 s, Dec. 2021, doi: 10.1145/3587095.