Helwan University
Faculty of Computers and Artificial Intelligence
Computer Science Department

DeepFake Audio Detection and Generation Android Application

A Graduation Project dissertation by:

[Anderw Ibrahim Ishac (202000167)]
[John Halim Hanna (202000242)]
[Habiba Abdelghany Sayed (202000257)]
[Essam Ahmed Mahmoud (202000557)]
[Kirollous Hany Eshak (202000683)]
[Mark Rafat Sarkies (202000690)]

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, at the General Department, the Faculty of Computers & Artificial Intelligence, Helwan University

Supervised by:
[Dr. Salwa Osama]

June 2024

جامعة حلوان

كلية الحاسبات والذكاء الإصطناعي

قسم علوم الحاسب

DeepFake Audio Detection and Generation Android Application

رسالة مشروع تخرج مقدم ة من:

[اندرو ابراهيم اسحق (٢٠٢٠٠٠١٦٧)]
[جون حليم حنا (٢٠٢٠٠٠٢٤٢)]
[حبيبة عبدالغني سيد (٢٠٢٠٠٠٢٥٧)]
[عصام احمد محمود (٢٠٢٠٠٠٥٥٧)]
[كيرلس هاني اسحق (٢٠٢٠٠٠٦٨٣)]
[مارك رأفت سركيس (٢٠٢٠٠٠٦٩٠)]

رسالة مقدمة ضمن متطلبات الحصول على درجة البكالوريوس في الحاسبات والذكاء الاصطناعي،
كلية الحاسبات والذكاء الاصطناعي، جامعة حلوان، بقسم علوم الحاسب

تحت إشراف:
[د. سلوى اسامة]

2024 يونيو

Abstract

The dawn of sophisticated AI tools has made a massive impact in digital media, as we are now seeing the emergence of deepfake audio - using AI to produce voice recordings so realistic that they are virtually indistinguishable from a real human voice. This poses potential danger with dissemination of misinformation and listen manipulation fraud. Existing audio deepfake detection methods have accuracy and scalability shortcomings. We present Our Proposed Solution, Fake Finder, an Android app which aims to detect, generate and clean deepfake audio signals to mitigate the challenges above. The central problem we sought to resolve was the widespread dissemination and abuse of deepfake audio. One can deploy malicious deepfake audio where he can harm reputations and extort individuals. For instance, a public figure could be falsely implicated through a fake audio recording, causing major reputational damage. To make it hard to defeat, research was done and a multitude of deepfake detection models were combined (LCNN with LFCC features, XceptionNet, MesoInception-4 and RawNet2) to ensure stability and precision in detection. Through experimentation it has been shown that LCNN with LFCC is more effective than other baselines, having more generalized performance, and has the lowest EER across all attacks. Our solution outputs a human-friendly application, that identifies deepfake audio with high precision, along with the audio file being a deepfake or not. This tool can be employed by various stakeholders, such as social media platforms, political campaigns, financial institutions, and the public, to safeguard against the malicious use of deepfake audio. For our proposed solution we represent Fake Finder, an Android application that combines all these features (deepfake audio detection the fake audio generation end audio denoising) in one app, enabling high quality audio generation and denoising methods and generalized deepfake audio detection model

Keywords

Deepfakes, Audio Deepfake Detection, Neural Networks, Spoofing Detection, Learning Computers, Finding Patterns, Testing, Results, Text-To-Speech, Speech-To-Speech, Generation, Filtration, denoising, Linear Frequency Cepstral Coefficients, Firebase, Kotlin, Android, Voice conversion, Voice cloning.

Acknowledgment

We as a team would like to express our deepest appreciation to the individuals who played a crucial role in completing this project. We are very grateful to our supervisor, Dr. Salwa Osama, for her guidance, support, and expertise throughout this journey. His insightful feedback and constant motivation have been invaluable in helping us achieve our goals.

We would also like to thank the TAs who provided their valuable assistance in various aspects of this project, which significantly impacted our work.

Moreover, we cannot express enough gratitude to our families and friends who stood by our side with great support and encouragement. Their belief in us has kept us motivated during the ups and downs of this journey.

Table of Contents

Figures

Tables

Chapter 1: Introduction

1.1 Introduction

In recent years, the advancement of artificial intelligence (AI) has led to remarkable developments in various fields, including media production.

One such development is the creation of deepfake audio, where AI techniques are used to generate synthetic audio recordings that mimic the voice of real individuals.

While this technology has potential for creative applications, it also presents significant challenges. Deepfake audio can be used maliciously to spread misinformation, damage reputations, and commit fraud.

To address these concerns, we have developed an Android application called FakeFinder which is designed to detect, generate, and filter deepfake audio, providing a comprehensive solution to mitigate the risks associated with this technology.

The importance of addressing deepfake audio cannot be overstated. Beyond its potential to mislead and manipulate, deepfake audio can also be used to exploit and harm individuals on a personal level.

For example, malicious actors can create fake audio recordings to trick, manipulate, or blackmail individuals, particularly targeting vulnerable groups such as women and girls.

These synthetic audio clips can be used to impersonate a victim's voice or the voice of someone they trust, leading to severe emotional distress and potential social or financial harm.

By developing FakeFinder, we aim to provide a tool that not only safeguards public discourse but also protects individuals from being victimized by such malicious tactics.

1.2 Motivation

The motivation behind the FakeFinder project arises from the pressing need to combat the detrimental effects of deepfake audio. Deepfake audio technology can create convincing imitations of a person's voice, which can be weaponized to deceive listeners and manipulate public opinion.

This poses a threat to individuals, organizations, and society at large. Our team is passionate about using AI to solve real-world problems, and we recognize the urgent need to develop tools that can detect and filter deepfake audio effectively. Deepfake audio can severely undermine trust in digital communications.

For instance, in journalism, the spread of fake audio recordings can discredit reputable news sources and mislead the public. Journalists rely on the authenticity of audio recordings to report accurate information.

FakeFinder can serve as a crucial tool for journalists to verify the authenticity of their sources, thus maintaining the integrity of their reports.

In the realm of social media, deepfake audio can fuel viral hoaxes or defamatory content, causing widespread panic or reputational damage to individuals and organizations.

By incorporating FakeFinder, social media platforms can identify and remove deepfake audio content, safeguarding users from exposure to false information.

Political campaigns are also at risk, as deepfake audio can be used to spread false statements attributed to political figures, thereby influencing public opinion and election outcomes.

By detecting such deepfakes, FakeFinder helps protect the democratic process and ensures that voters receive accurate information.

Financial institutions face another critical area of concern. Deepfake audio can be used in fraudulent schemes, where impostors use synthetic voice recordings to authorize transactions or solicit sensitive information.

FakeFinder can help financial institutions verify the authenticity of voice communications, thereby preventing fraud and protecting customers.

Furthermore, deepfake audio poses a significant threat to personal safety and privacy. Individuals, especially women and girls, can be targeted with deepfake audio for blackmail or manipulation.

By providing a reliable and user-friendly tool for detecting and filtering deepfake audio, FakeFinder aims to protect individuals from these threats and promote the ethical use of AI technologies.

1.3 Target Users

FakeFinder is designed to cater to a diverse group of users who are affected by the challenges posed by deepfake audio. The primary target users include:

Journalists and News Organizations: Journalists can use FakeFinder to verify the authenticity of audio recordings before publishing them. This helps prevent the dissemination of misinformation and protects the credibility of news outlets.

Social Media Platforms: Social media companies can integrate FakeFinder to identify and remove deepfake audio content. This ensures that users are not exposed to harmful or misleading information, enhancing the overall trustworthiness of the platform.

Political Campaigns: Political organizations can utilize FakeFinder to detect deepfake audio recordings that may be used to spread disinformation or tarnish a candidate's reputation. This helps maintain the integrity of the electoral process.

Financial Institutions: Banks and other financial institutions can use FakeFinder to prevent fraud by detecting manipulated audio recordings used in fraudulent transactions. This protects both the institutions and their customers from financial loss.

General Public: Individuals can use the standalone application to analyze and verify audio recordings they receive, helping them make informed decisions and protect themselves from deception.

1.4 Objectives

The FakeFinder project has several key objectives aimed at addressing the issue of deepfake audio:

Accurate Detection: Develop an AI-based system capable of accurately detecting deepfake audio, ensuring high precision and low false-positive rates.

User-Friendly Interface: Design an intuitive and accessible user interface that allows users to easily upload and analyze audio samples, making the technology accessible to non-experts.

Real-Time Analysis: Enable real-time or near-real-time analysis of audio recordings, allowing users to quickly verify the authenticity of audio content.

Comprehensive Reporting: Provide the detection results, including the likelihood of the audio being a deepfake and the rationale behind the assessment.

Integration and Scalability: Ensure that the system can be seamlessly integrated into various platforms and applications, allowing for widespread adoption and scalability.

Audio Generation and Filtration: Develop functionalities for generating and filtering deepfake audio, providing a comprehensive suite of tools for both detection and creation, which can be used for research and educational purposes.

1.5 Problem Definition

The proliferation of deepfake audio represents a significant problem in today's digital landscape. Deepfake audio involves the use of advanced AI algorithms to create synthetic voice recordings that sound nearly identical to the voices of real people.

These recordings can be used to spread misinformation, manipulate public opinion, damage reputations, and commit fraud. The current solutions available for detecting deepfake audio are often limited in their accuracy, accessibility, or both.

FakeFinder aims to address these limitations by providing a robust, accurate, and user-friendly tool for detecting and filtering deepfake audio.

The application leverages state-of-the-art AI techniques to analyze audio recordings and determine their authenticity. By offering a comprehensive solution that includes both detection and generation capabilities, FakeFinder helps mitigate the risks associated with deepfake audio and promotes the ethical use of this technology.

1.6 Overview of the Project

FakeFinder is an Android application designed to detect, generate, and filter deepfake audio. The system architecture consists of three main tiers: the presentation tier, the application tier, and the data tier.

Presentation Tier: This tier is responsible for the user interface and interaction with the user. It is implemented using Android XML, providing a seamless and intuitive experience for users. The presentation tier allows users to upload audio recordings, view analysis results, and generate new audio samples.

Application Tier: The core of the system, the application tier, contains the business logic and AI models used for audio analysis and generation. This includes modules for user authentication, registration, audio analysis, and audio generation. The voice module, implemented using TensorFlow Lite, performs the deepfake detection and generation tasks.

Data Tier: The data tier uses Firebase Realtime Database to store user data, audio samples, and analysis results. This ensures secure and efficient data management, allowing users to access their analysis history and generated audio samples.

Use Cases

- Safeguarding Public Figures

Imagine a scenario where a prominent politician is accused of making controversial statements in a leaked audio recording. The recording goes viral, causing public outrage and significant damage to the politician's reputation.

The politician's team, suspecting foul play, uses FakeFinder to analyze the audio. The application processes the recording and determines that it is a deepfake.

Armed with this evidence, the politician's team releases the findings to the public, accompanied by the result from FakeFinder explaining that the recording is synthetic.

This use case demonstrates how FakeFinder can help safeguard public figures from the malicious use of deepfake audio, ensuring that the truth prevails in the face of digital deception.

- Protecting Individuals from Blackmail

Consider a scenario where a young woman receives a phone call from someone claiming to have compromising audio recordings of her. The caller threatens to release these recordings unless she pays a ransom. The woman, suspecting that the recordings might be deepfake audio, uses FakeFinder to analyze the provided samples. FakeFinder quickly processes the audio and confirms that the recordings are indeed synthetic and not genuine.

Armed with this information, she confidently rejects the blackmail attempt and reports the incident to the authorities.

This use case highlights how FakeFinder can empower individuals to defend themselves against malicious actors using deepfake technology, providing a crucial defense mechanism for personal safety and privacy.

- Verifying Journalistic Integrity

Consider a journalist who receives an audio recording of an interview with a politician. Before publishing the recording, the journalist wants to verify its authenticity to ensure that it has not been tampered with.

The journalist uploads the recording to FakeFinder via the application's user interface. The application tier processes the audio using AI models to detect any signs of manipulation.

After analysis, FakeFinder provides if the recording is a deepfake or not. Based on this information, the journalist can make an informed decision about whether to publish the recording. This use case underscores the importance of FakeFinder in maintaining journalistic integrity and preventing the spread of misinformation.

- Enhancing Social Media Platforms

Social media platforms are often targets for spreading fake news and misinformation. Imagine a scenario where a deepfake audio clip goes viral on a social media platform, causing panic and confusion among users.

The platform's administrators use FakeFinder to analyze the suspicious audio content. The tool quickly identifies the clip as a deepfake and removes it from the platform, preventing further spread of false information.

Additionally, FakeFinder provides the Result that the administrators can share with users to explain why the content was removed. This use case demonstrates how FakeFinder can enhance the reliability of social media platforms by ensuring that users are protected from deepfake audio content.

- Securing Financial Transactions

Financial institutions face significant risks from deepfake audio used in fraudulent schemes. Imagine a bank receives a voice authorization for a large transaction that seems suspicious. The bank uses FakeFinder to verify the authenticity of the voice recording. The tool analyzes the audio and detects that it is a deepfake, preventing the fraudulent transaction from being processed.

The bank then takes appropriate measures to secure the affected account and alerts the customer. This use case highlights how FakeFinder can help financial institutions prevent fraud and protect their customers from financial loss.

Chapter 2: Similar Apps

2.1 Similar Apps or Websites and Their Pros and Cons

App/Website	Description	Pros	Cons
DeepVoice	An app that generates deepfake audio and voice transformations.	- High-quality deepfake audio generation - User-friendly interface	- Limited free features - Requires significant processing power
VoCoDetect	A web service for detecting deepfake audio and voice forgeries.	- Accurate detection of fake audio - Detailed analysis reports	- Subscription-based - Requires internet connection
AudioMender	An app focused on audio filtration and noise reduction.	- Effective noise reduction - Various audio enhancement tools	- Less focus on deepfake detection - Can be complex for new users
FakeApp	A comprehensive app for creating and detecting fake audio and video.	- Multi-media fake detection - High customization options	- Steeper learning curve - High resource usage
AudioAnalyzer	A tool for analyzing and manipulating audio recordings.	- Wide range of audio analysis tools - Supports various audio formats	- Not specifically tailored to deepfake audio - Basic interface

Table 1: Similar Apps

2.2 Comparison Between Fake Finder and Similar Apps or Websites

Feature	Our proposed solution	DeepVoice	VoCoDetect	AudioMender	FakeApp	Audio Analyzer
Voice Filtration	✔ Advanced filtration and noise reduction	✔ Basic filtration	✘ Not available	✔ Effective noise reduction	✔ Available	✔ Available
Deepfake Audio Generation	✔ High-quality deepfake audio	✔ High-quality	✘ Not available	✘ Not available	✔ Available	✘ Not available
Fake Audio Detection	✔ Accurate detection	✘ Not available	✔ Accurate detection	✘ Not available	✔ Available	✘ Not available
History of Audio Activities	✔ Available	✘ Not available	✘ Not available	✘ Not available	✔ Available	✘ Not available
User Profile Management	✔ Available	✘ Not available	✔ Available	✘ Not available	✔ Available	✘ Not available
User Interface	✔ Intuitive and modern	✔ User-friendly	✔ Detailed reports	✘ Can be complex	✘ Steeper learning curve	✘ Basic interface

Table 2: Comparison FakeFinder vs. Similar Apps

Chapter 3: Feasibility Study

3.1 Feasibility Analysis

3.1.1 Technical Feasibility

Objective: Assess whether the proposed technology and architecture are suitable for developing the application.

Existing Technology Stack:

- Programming Language: Kotlin
- Architecture Pattern: MVVM
- Backend: Firebase (Authentication, Realtime Database, Storage), Flask API
- Libraries: Retrofit, Kotlin Coroutines, Dagger Hilt, Navigation Component, Live Data

Assessment:

- Kotlin and Android Development: Kotlin is a modern, statically typed programming language that is fully supported by Android. It is suitable for building robust and maintainable applications.
- MVVM Architecture: MVVM is a widely adopted architecture pattern that facilitates a clean separation of concerns, making the app easier to manage and extend.
- Firebase Services: Firebase provides reliable and scalable backend services, including authentication, database, and storage, which are crucial for the app's functionality.
- Retrofit and Flask API: Retrofit is a robust HTTP client for Android, and Flask is a lightweight web framework for building APIs. Both are well-suited for the app's needs.

Conclusion: The proposed technology stack is technically feasible and appropriate for developing the application.

3.1.2 Operational Feasibility

Objective: Evaluate the ability of the organization to support the operation of the proposed application.

User Base:

- Individuals are interested in audio processing, including voice filtration, deepfake audio generation, and fake audio detection.
- Professionals in fields like media, security, and entertainment.

Operational Considerations:

- User Support: Providing user support for troubleshooting and guidance.
- Maintenance: Regular updates and maintenance of the app to ensure smooth operation.
- Training: Minimal training required for users due to intuitive UI and in-app guidance.

Conclusion: The organization can feasibly support the operation of the application with the existing resources and minimal additional training

3.2 Requirements Determination

3.2.1 Functional Requirements

- Voice Filtration:

- Users can upload audio files for filtration.

- The app filters out background noise and enhances audio quality.

- Users can download and share filtered audio files.

- Deepfake Audio Generation:

- Users can input text and upload an audio or upload two audios sample to generate deepfake audio.

- The app processes and generates the requested audio.

- Users can preview, download, and share the generated audio.

- Fake Audio Detection:

- Users can upload audio files for authenticity checks.

- The app analyzes the audio and provides a report on its authenticity.

- Users can view and download the detection report.

- History Management:

- Users can view the history of their generated, filtered, and detected audios.

- Users to manage (view, download) their history.

- Profile Management:

- Users can view and update their profile information.

- The app supports profile picture updates, password changes, and personal information management.

3.2.2 Non-Functional Requirements

- Performance:

- The app has fast response times for audio processing tasks.

- The app manages concurrent users efficiently.

- Security:

- User data, including audio files, securely stored and transmitted.

- Authentication ensures that only authorized users access their data.

- Usability:

- The app has an intuitive and user-friendly interface.

- Scalability:

- The backend able to scale to manage a growing number of users and audio files.

- The app architecture should support the addition of new features with minimal disruption.

- Reliability:

- The app has high availability with minimal downtime.

- Maintainability:

- The codebase is well-documented and modular to facilitate maintenance and updates.

- Regular updates planned to fix bugs and add enhancements.

Chapter 4: System Design and Architecture

4.1 Introduction

This chapter examines the system design and architecture of FakeFinder, an Android application for advanced audio processing tasks like voice filtration, deepfake audio generation, and fake audio detection. Utilizing Firebase for secure authentication, real-time database interactions, and efficient storage, the app is built to ensure seamless data flow and responsive user interfaces. We will discuss the architectural choices, component interactions, and data flow, providing a comprehensive understanding of the design principles that enhance the app's functionality and performance.

4.2 Diagrams

4.2.1 Use Case

4.2.1.1 Use Case Diagram

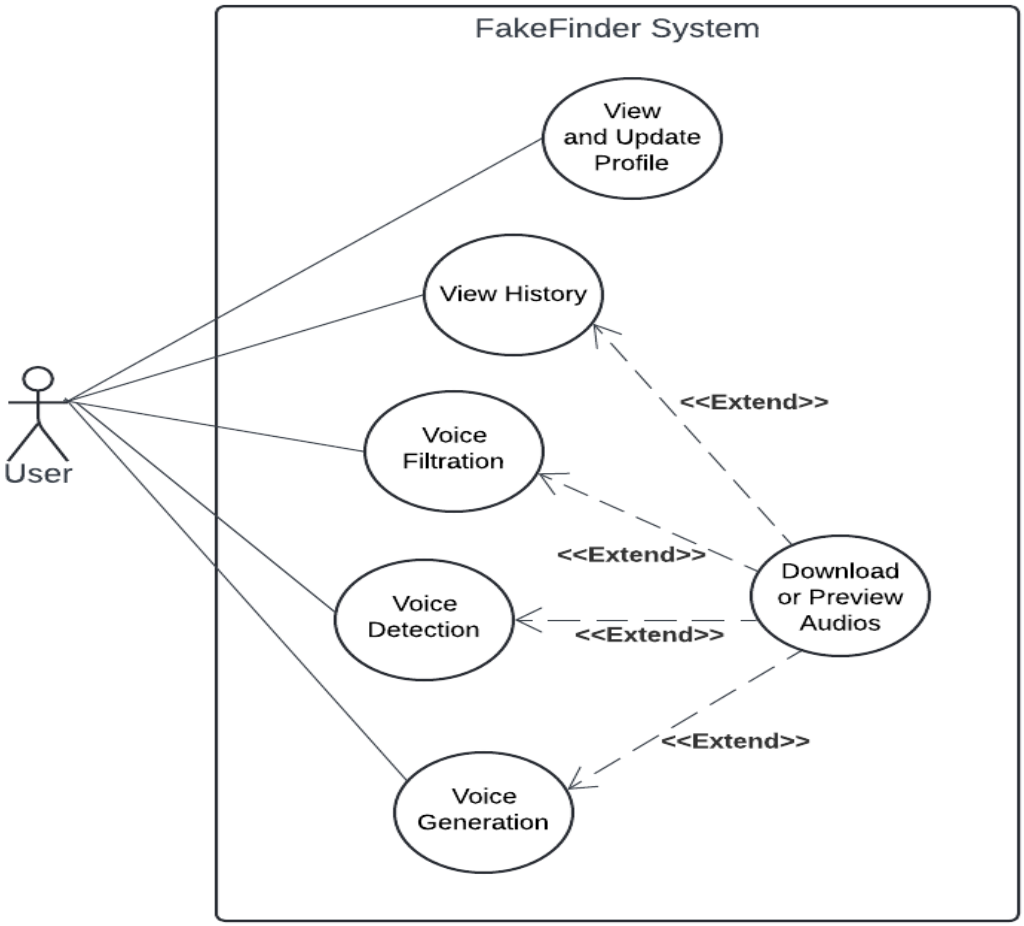

Figure 1: Use Case Diagram for the FakeFinder System

4.2.1.2 Use Case Description Table

Use Case	Actor	Description	Precondition	Postcondition
View and Update Profile	User	Allows the user to view and update their personal profile information.	User is logged into the system.	The user's profile information is updated and saved, or the user can see it.
View History	User	Enables the user to view their history of generated, filtered, and detected voices.	User is logged into the system.	User views the history of their voice activities.
Voice Filtration	User	Allows the user to filter a voice to identify and remove unwanted sounds or noise.	User provides voice input.	Filtered voice is saved and available for download or preview.
Voice Detection	User	Detects fake voices from the provided audio input.	User provides voice input.	Detection result and voice is saved and available for download.
Voice Generation	User	Allows the user to generate fake audio based on provided text and voice input or two voice inputs	User provides text and voice input or two voice inputs.	Generated voice is saved and available for download or preview.
Download or Preview Audios	User	Extends the functionalities of viewing history, voice filtration, voice detection, and voice generation by enabling the user to download or preview the processed audios.	User has accessed any related functionality (history, filtration, detection, or generation).	User downloads or previews the audio files.

Table 3: Use Case description

4.2.2 Activity Diagram

4.2.2.1 Deepfake Voice Detection

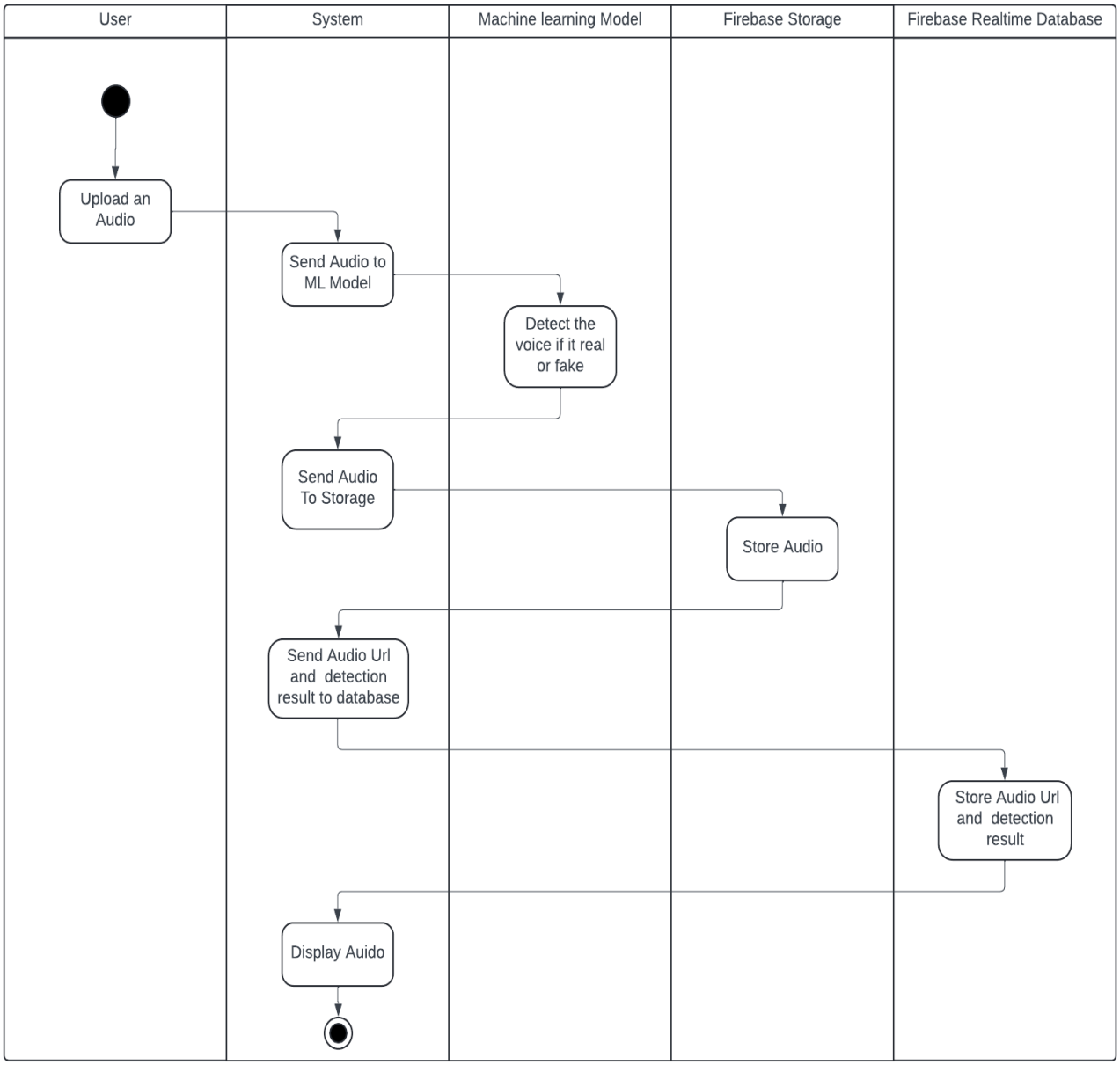

Figure 2.1: Activity Diagram for Fake Audio Detection in the FakeFinder System

4.2.2.2 Voice Filtration

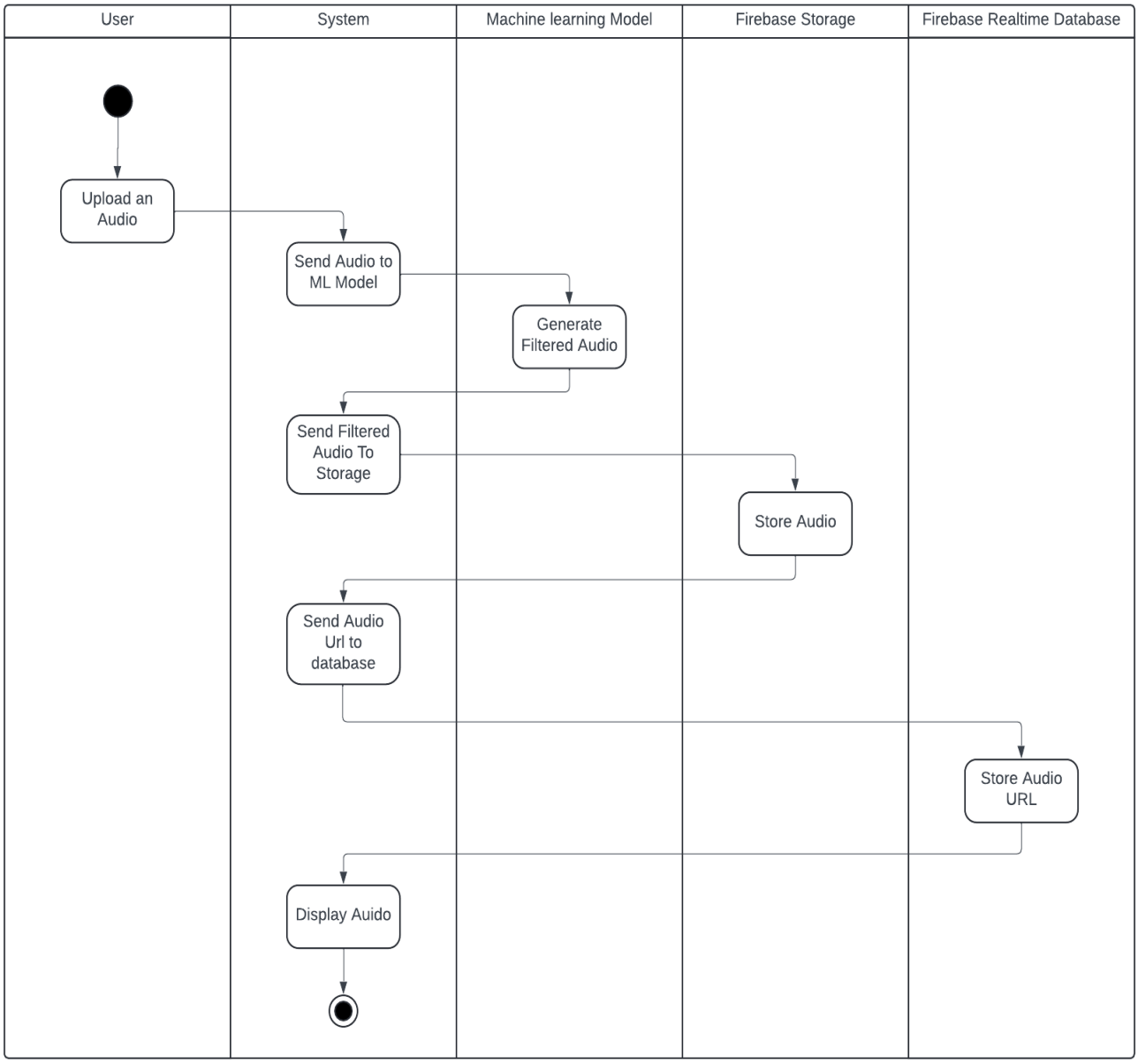

Figure 2.2: Activity Diagram to Filter Voice in the FakeFinder System

4.2.2.3 Deepfake Voice Generation

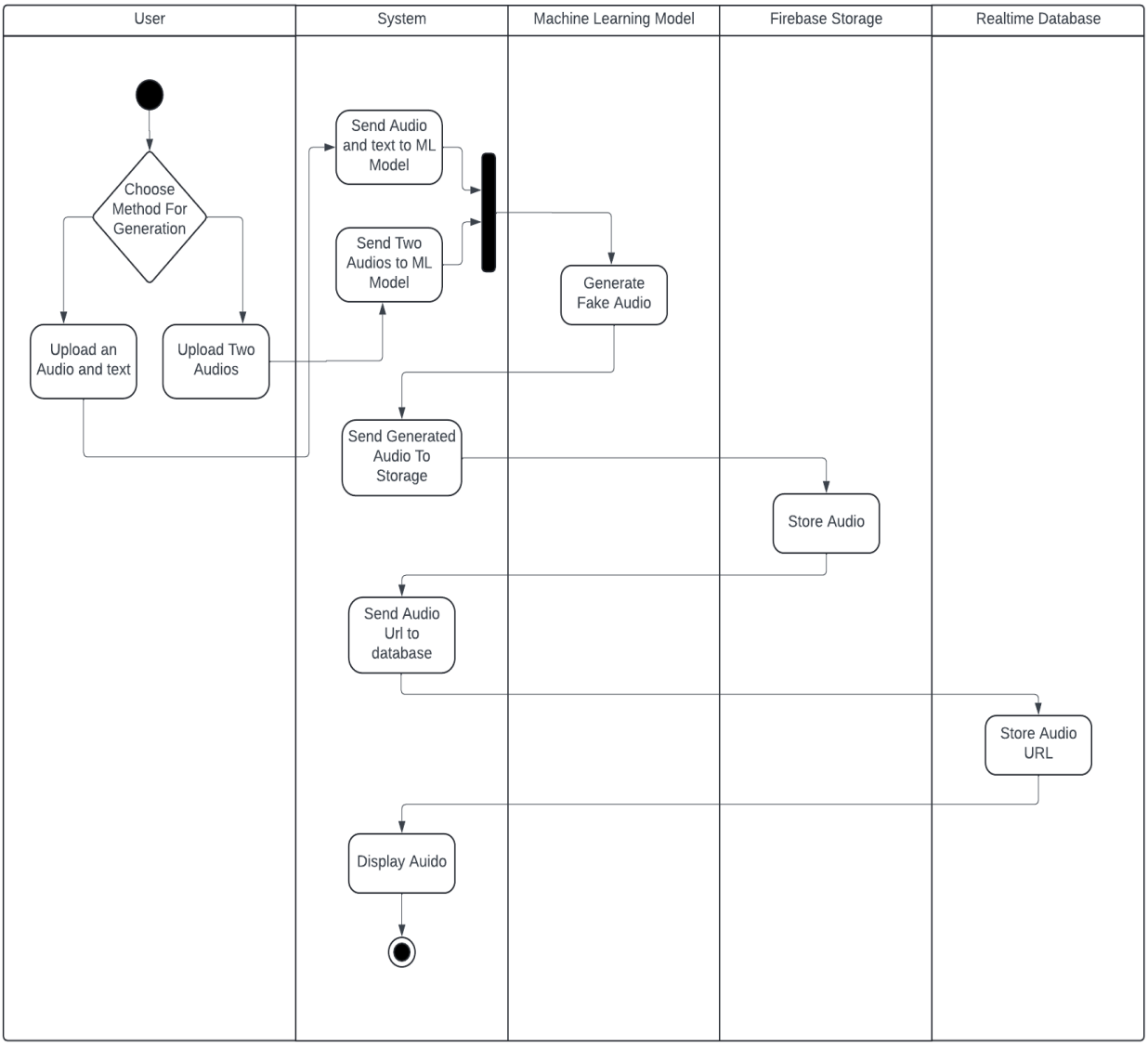

Figure 2.3: Activity Diagram for Generating Fake Audio in the FakeFinder System

4.2.2.4 Voices History

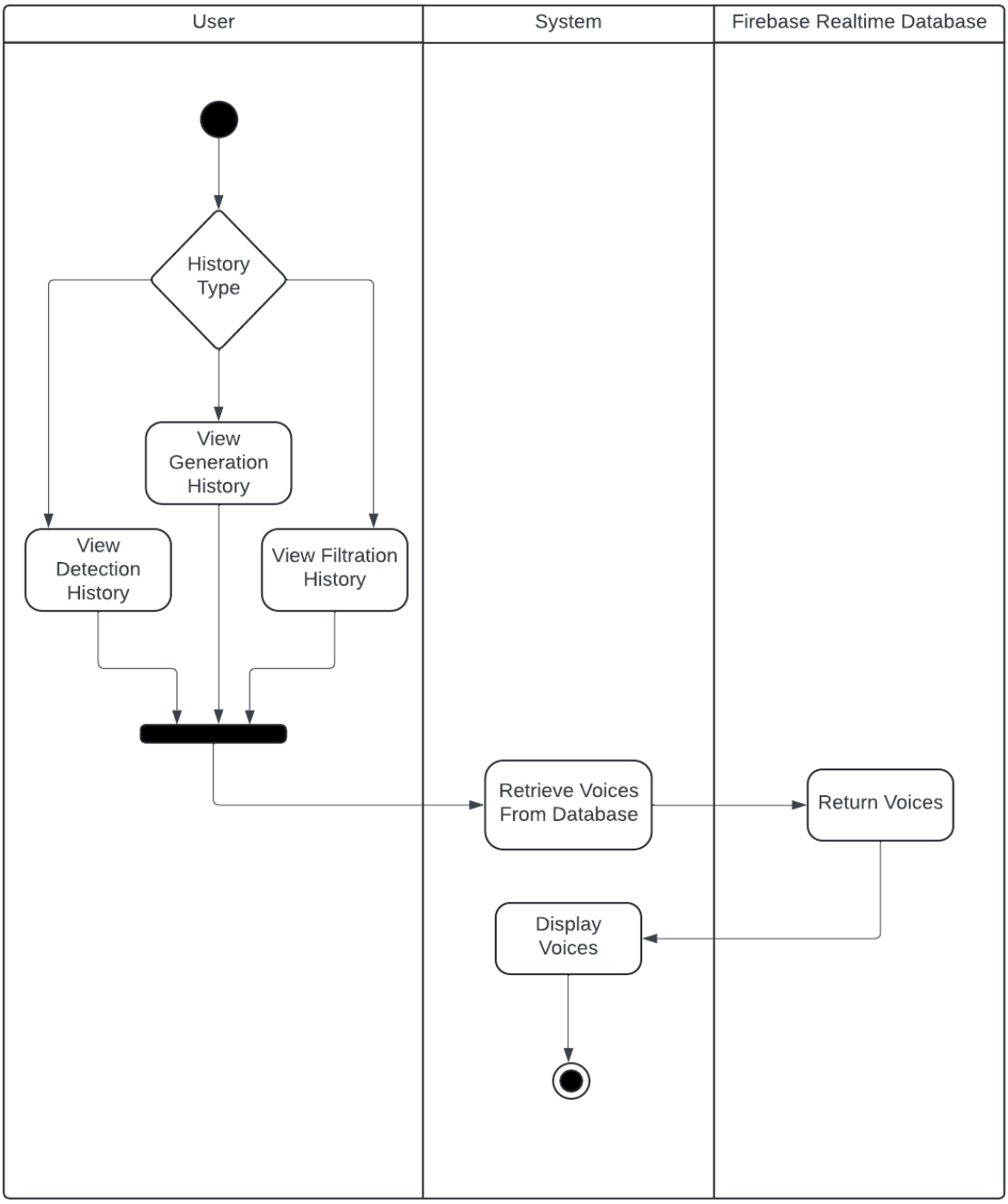

Figure 2.4: Activity Diagram for Viewing History in the FakeFinder System

4.2.2.5 Login and Signup

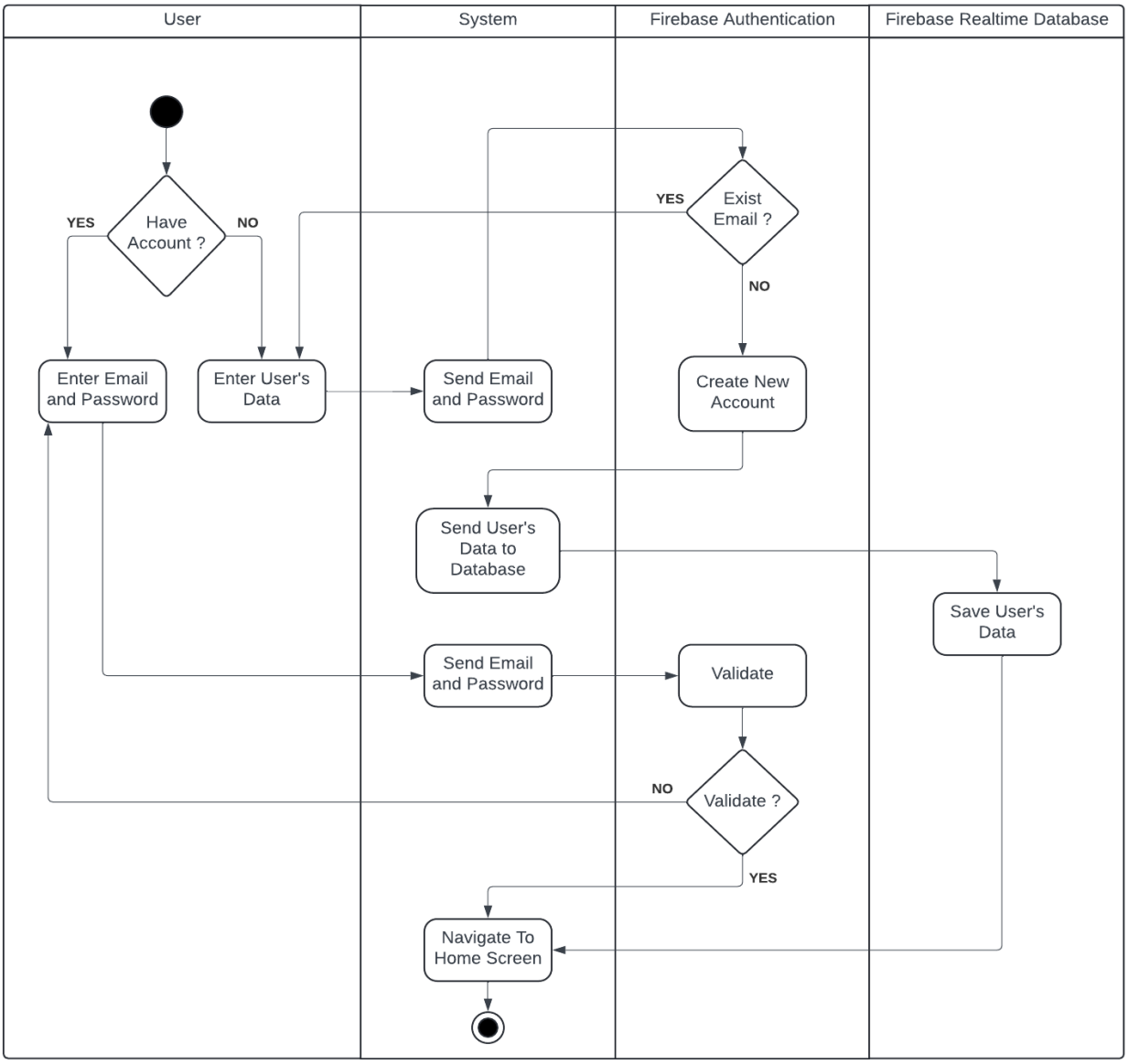

Figure 2.5: Activity Diagram for User Authentication in the FakeFinder System

4.2.3 Sequence Diagram

4.2.3.1 Deepfake Voice Detection

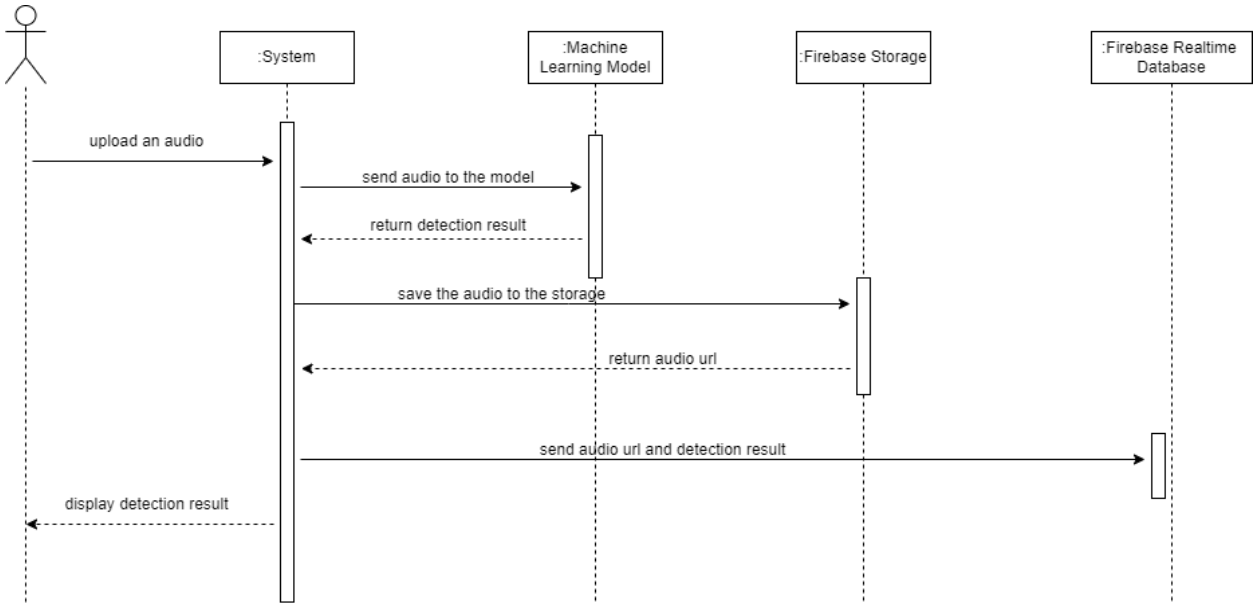

Figure 3.1: Sequence Diagram for Fake Voice Detection in the Fake Finder System

4.2.3.2 Voice Filtration

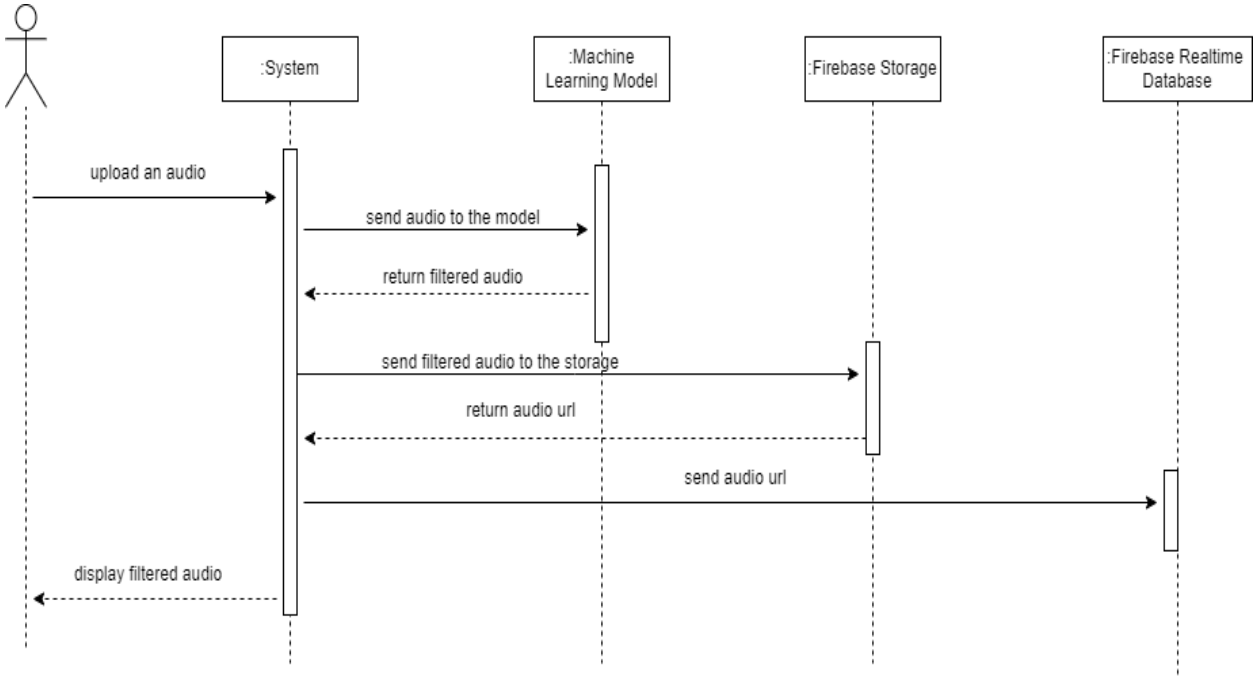

Figure 3.2: Sequence Diagram to Filter Voice in the Fake Finder System

4.2.3.3 Deepfake Voice Generation

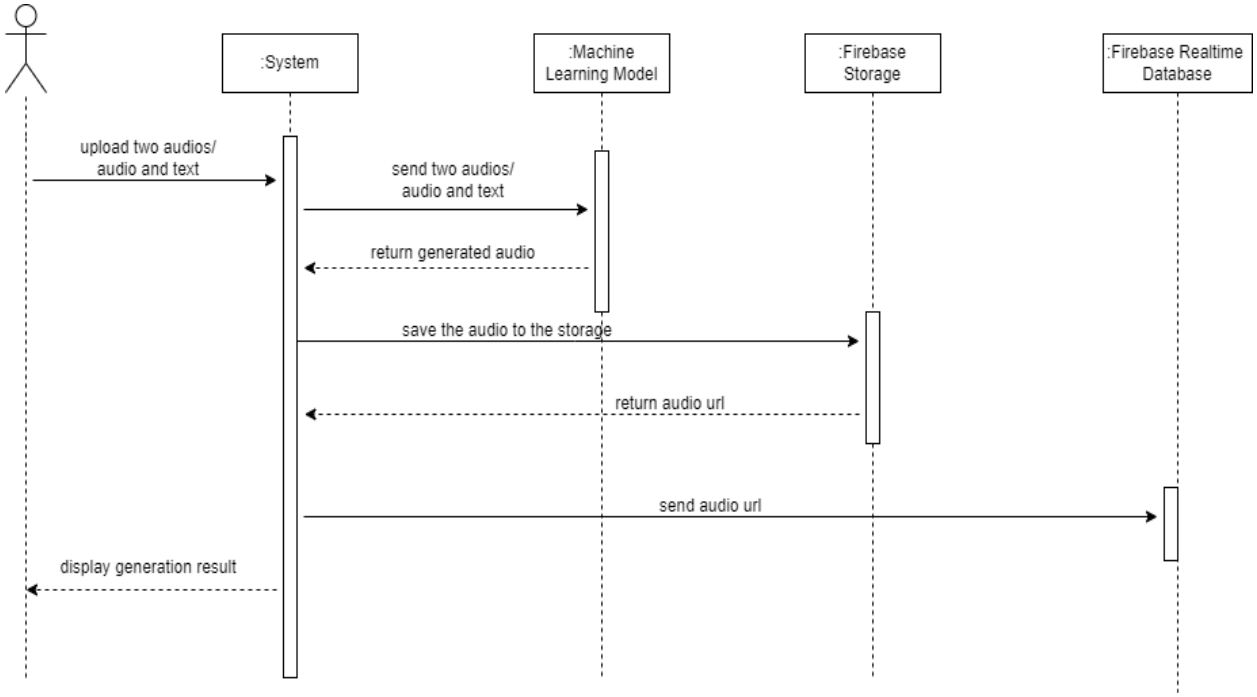

Figure 3.3: Sequence Diagram to Generate Fake Voice in the Fake Finder System

4.2.3.4 Voices History

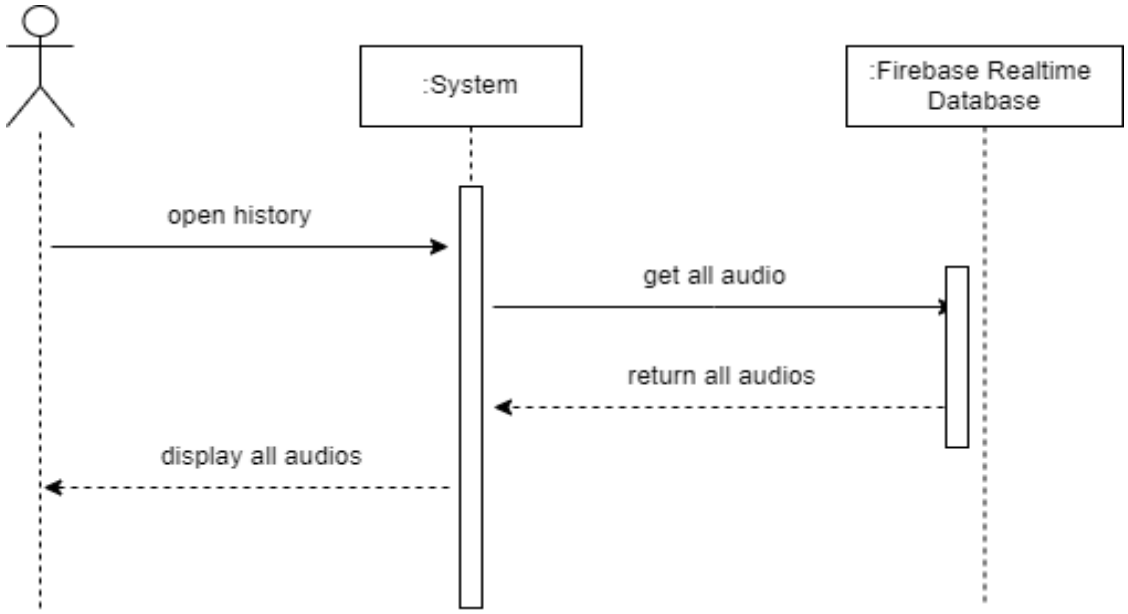

Figure 3.4: Sequence Diagram for Viewing History in the Fake Finder System

4.2.3.5 Login

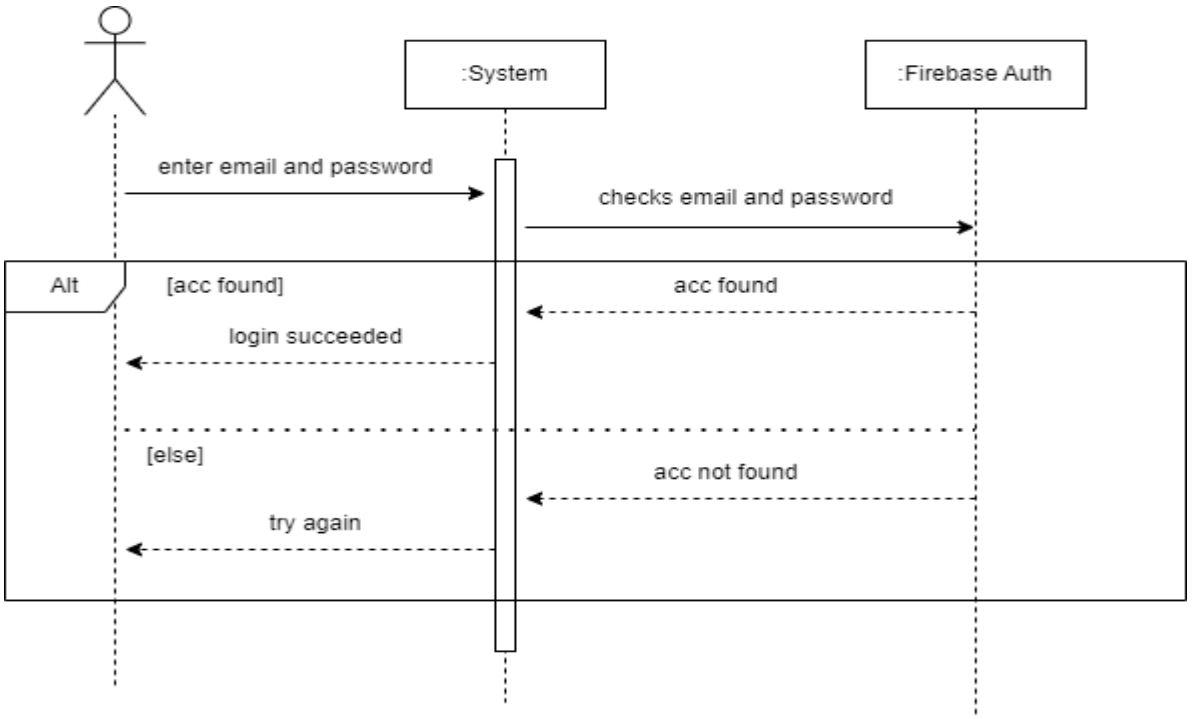

Figure 3.5: Sequence Diagram for the Login Process in the Fake Finder System

4.2.3.6 Signup

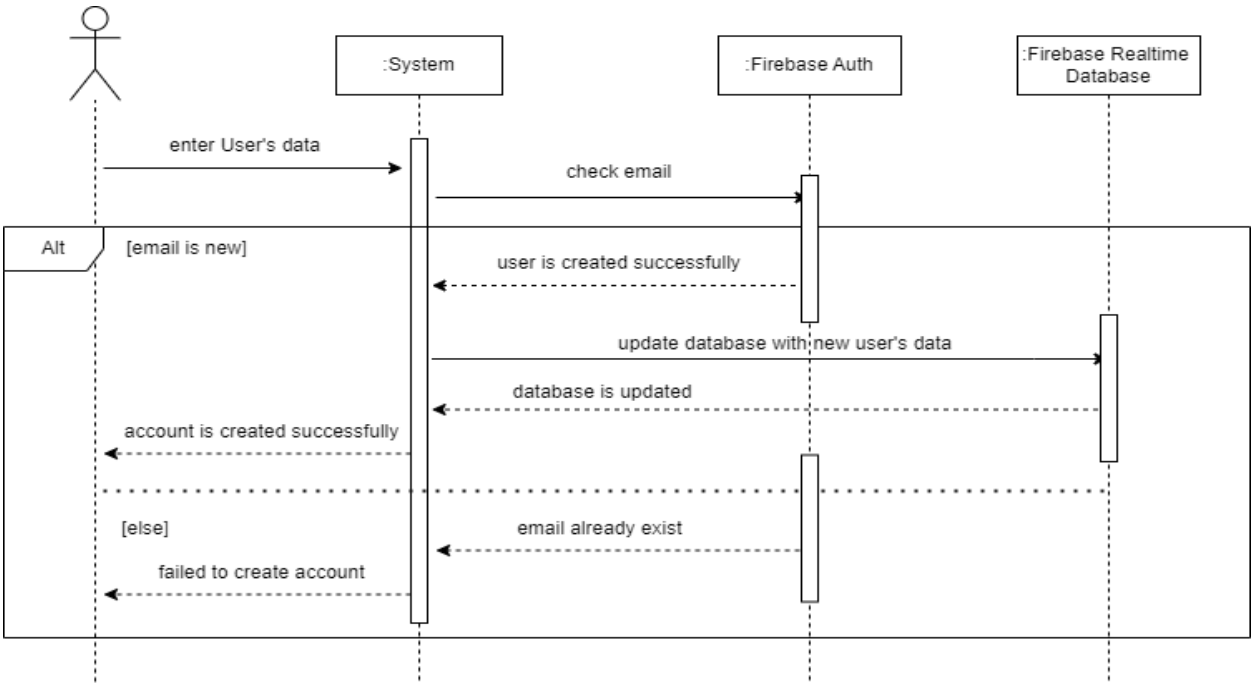

Figure 3.6: Sequence Diagram for User Registration in the Fake Finder System

4.2.4 Data Flow Diagram

4.2.4.1 Context Level (Level Zero)

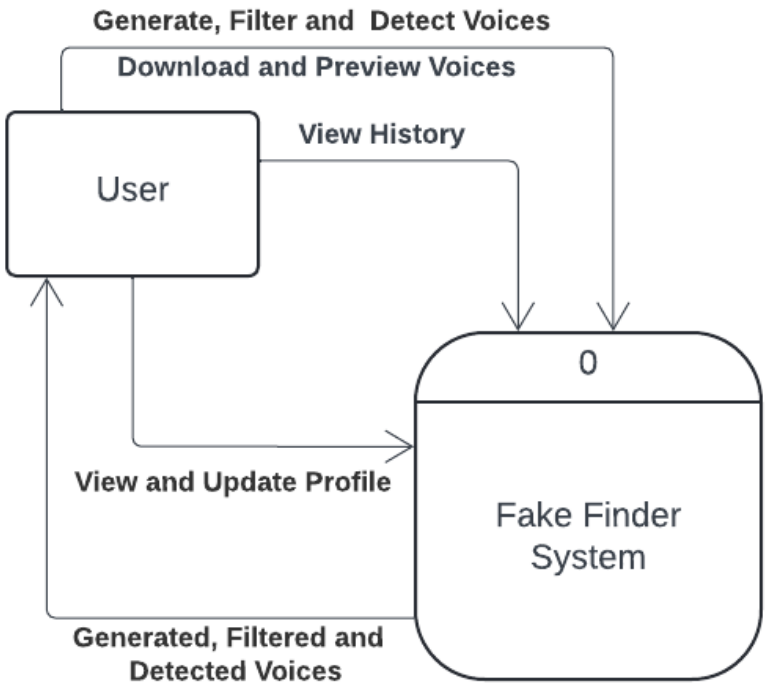

Figure 4.1: Level 0 Data Flow Diagram for the Fake Finder System

4.2.4.2 Level One

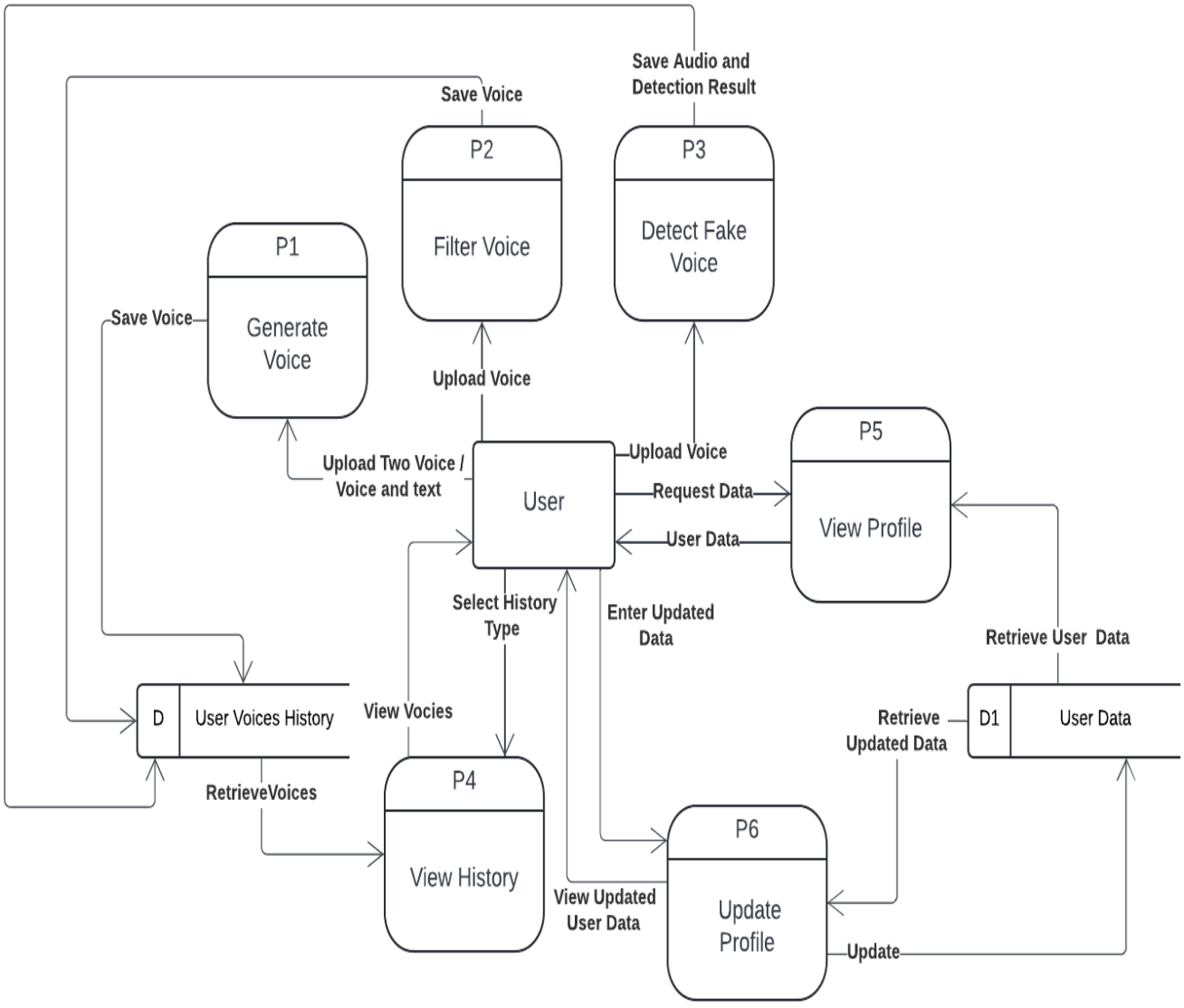

Figure 4.2: Level 1 Data Flow Diagram for the Fake Finder System

Chapter 5: Proposed Solution

5.1 System Architecture

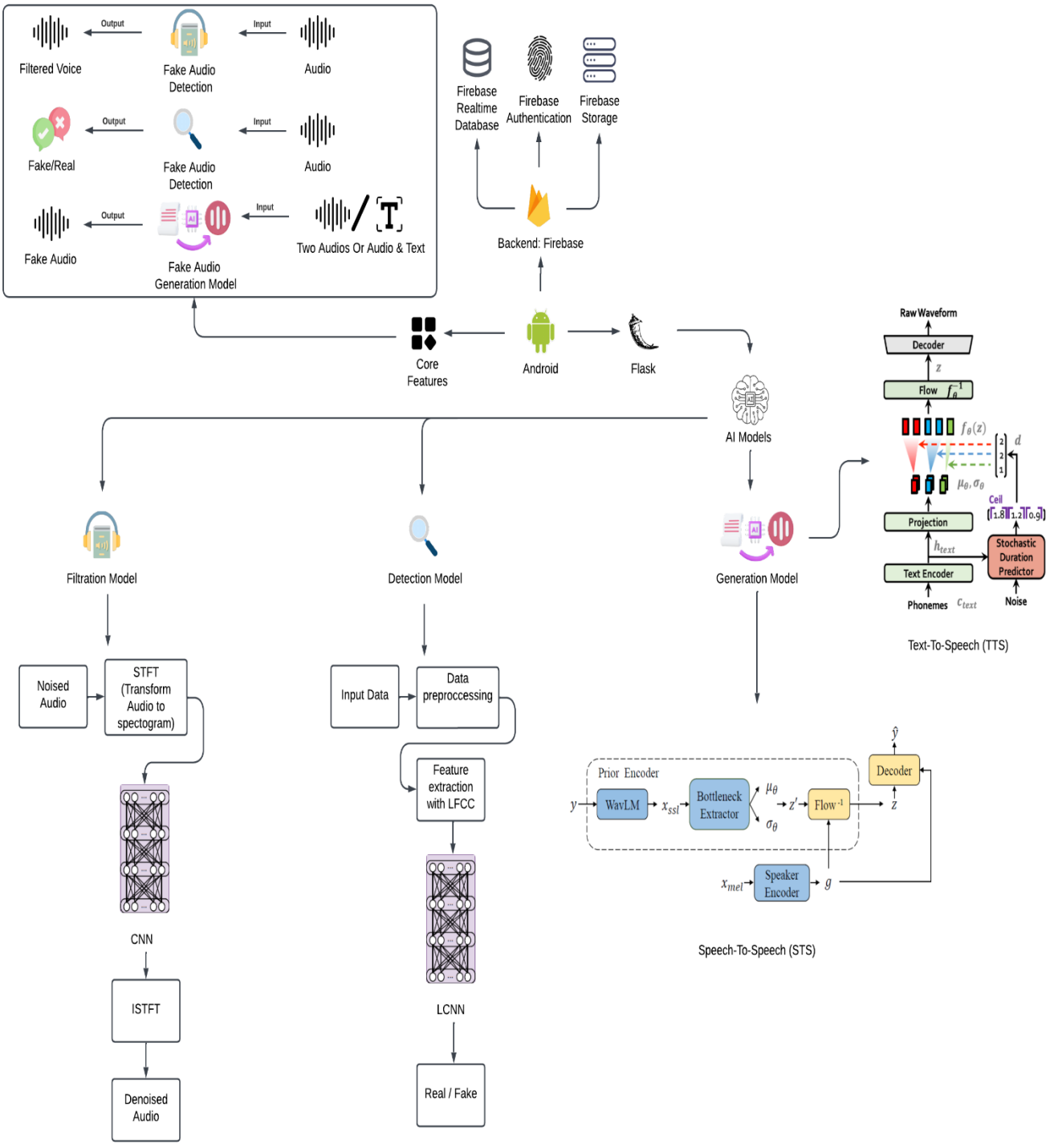

Figure 5: System Architecture

5.2 Architecture

5.2.1 MVVM Pattern

- Model: Manages the data and business logic.
- View: Displays data and captures user actions.
- View Model: Acts as a link between Model and View, handling data transformations and maintaining UI-related data.

5.2.2 Component Interaction

- View Model: Utilizes Live Data to notify the View of data changes.
- Repository: Handles data operations and serves as an abstraction layer between the View Model and data sources.

5.3 Libraries and Dependencies

5.3.1 Retrofit

Purpose: Retrofit is used for making network requests to Flask APIs that manage machine learning models for various audio processing tasks.

Key Features:

- Type-safe HTTP client.
- Support for JSON conversion.
- HTTP request annotations (GET, POST, etc.).
- Supports synchronous and asynchronous requests.

Usage: Define API interfaces and data models and create Retrofit instances to perform network operations seamlessly.

5.3.2 Kotlin Coroutines

Purpose: Kotlin Coroutines is used for managing background tasks, such as network requests and data processing, without blocking the main UI thread.

Key Features:

- Lightweight and efficient concurrency.
- Structured concurrency to manage lifecycle-aware operations.
- Easy integration with Android components and libraries.

Usage: Use suspend functions for network calls and viewModelScope for launching coroutines in View Models.

5.3.3 Dagger Hilt

Purpose: Dagger Hilt simplifies dependency injection, providing a standardized way to manage dependencies across the app.

Key Features:

- Reduces boilerplate code for dependency injection.
- Manages component lifecycles for proper resource management.
- Integrates seamlessly with Android components.

Usage: Annotate application and activity classes, define modules for providing dependencies, and inject them where needed.

5.3.4 Navigation Component

Purpose: The Navigation Component manages navigation and deep links within the app, ensuring a consistent navigation flow.

Key Features:

- Visual navigation graph for mapping out app navigation.
- Safe Args for type-safe argument passing between fragments.
- Deep link support for navigating to specific destinations within the app.

Usage: Define navigation graphs, set up NavController, and use Safe Args for passing data between navigation destinations.

5.3.5 Firebase Services

- Authentication: User login and registration.
- Realtime Database: Storing user data and audio history.
- Storage: Uploading and retrieving audio files.

5.4. Core Features

5.4.1 Voice Filtration

- Description: Filters out background noise and enhances audio quality.
- Implementation: Utilizes ML models hosted on Flask API.

5.4.2 Generation of Deepfake Audios

- Description: Generates synthetic voices using trained models.
- Implementation: Connects to backend Flask API for processing.

5.4.3 Fake Audio Detection

- Description: Detects whether an audio clip is real or fake.

- Implementation: Sends audio data to the backend API for analysis.

5.4.4 History Management

- Description: Allows users to view the history of generated, filtered, and detected audios.
- Implementation: Retrieves data from Firebase database.

5.4.5 Profile Management

- Description: View and update user profile information.
- Implementation: Firebase Realtime Database for storing and updating profile details.

5.5 Firebase Integration

5.5.1 Authentication

- Sign-Up/Sign-In: Using Firebase Authentication.

5.5.2 Realtime Database

- Data Storage: User data and audio history
- Structure: Organized by user IDs for efficient retrieval and updates.
- Profile Management: User profile information is stored and updated in the Realtime Database.

5.5.3 Storage

- Audio Files: Uploaded and retrieved from Firebase Storage.
- User Profile Picture: Upload and retrieved from Firebase Storage.
- Structure: Organize files by user IDs.

5.6 User Interface

5.6.1 Activities/Fragments

- Activities: Hosts fragments and manages navigation for each module.
- Voice Filter Fragment: Interface for voice filtration.
- Deepfake Generation Fragment: Interface for generating deepfake audios.
- Fake Audio Detection Fragment: Interface for detecting fake audios.
- History Activities: Displays user's audio history.
- Profile Fragment: Manages user profile view and updates.

5.6.2 Navigation

- Navigation Graph: Defines navigation paths and deep links.

5.7 API Integration

5.7.1 Retrofit Setup

- Base URL: Since the Flask APIs are running locally, the Base URL will be IP for the hosting device like (http://192.168.1.5:5000)
- Endpoints: Define endpoints for voice filtration, deepfake generation, and fake audio detection.

5.7.2 API Models

- Response Models: Define data classes for API responses.

5.8 Data Flow

5.8.1 Live Data and View Model

- Data Observation: Use Live Data to observe data changes in the View Model.
- Data Binding: Bind UI components to Live Data objects.

5.8.2 Coroutines

- Async Tasks: Perform network operations and data processing in background threads.

5.9 Code Structure

5.9.1 Package Organization

- Ui: Contains all UI-related classes (Activities, Fragments) and View Model classes.
- Data: Contains repository classes for data management and Retrofit service interfaces.
- di: Contains Dagger Hilt modules for dependency injection.
- model: Contains data models.
- utils: Contains utility classes.

5.10 Installation and Setup

5.10.1 Prerequisites

- Android Studio
- Firebase Account
- Flask API setup

5.10.2 Steps

1. Clone the Repository: git clone [https://github.com/AndrewIbrahim6/Fake-Finder.git] `
2. Open in Android Studio: Open the project in Android Studio.
3. Firebase Setup: Add your `google-services. json` file to the `app` directory.
4. Build the Project: Sync and build the project in Android Studio.
5. Run the App: Deploy the app to an Android device or emulator.

Chapter 6: Experiments

6.1 Deepfake Voice Detection

6.1.1 Introduction

This chapter details the implementation of the audio deepfake detection models used in our project. The goal is to achieve robust detection of audio deepfakes by leveraging diverse datasets and state-of-the-art machine learning models. The methodologies and findings are based on the research paper "Attack Agnostic Dataset: Towards Generalization and Stabilization of Audio DeepFake Detection." By employing multiple datasets and various deep learning architectures, we aim to enhance the generalization and stability of our detection system.

We have adopted a comprehensive approach to tackle the challenge of audio deepfake detection by using a variety of models and feature extraction techniques. Our implementation focuses on four primary deep learning models: Light Convolutional Neural Network (LCNN), XceptionNet, MesoInception-4, and RawNet2. Each of these models brings unique strengths and addresses different aspects of the deepfake detection problem.

LCNN is known for its efficiency and effectiveness in processing audio signals. It utilizes convolutional layers that are adept at capturing local patterns and features in the audio data. LCNN was evaluated using various front-end feature extraction techniques, including Linear-Frequency Cepstral Coefficients (LFCC), Mel-Frequency Cepstral Coefficients (MFCC), and spectrogram-based features. Among these, LFCC proved to be particularly effective in capturing both human-audible and high-frequency artifacts, leading to superior performance in terms of Equal Error Rate (EER) and stability.

XceptionNet, another powerful model, leverages depth wise separable convolutions, which significantly reduce the number of parameters and computational cost. This model primarily utilized LFCC features, which allowed it to achieve good generalization across different types of audios deepfakes. However, compared to LCNN, XceptionNet showed slightly less stability in its performance across different dataset folds.

MesoInception-4 is designed to capture mesoscopic properties of audio data, focusing on intermediate-level features that are crucial for distinguishing between real and fake audio. This model also relied on LFCC-based features but did not perform as well as LCNN and XceptionNet. The higher EER and less stable performance indicated that while MesoInception-4 can capture important features, it might not be as robust in varying conditions.

RawNet2 takes a different approach by processing raw audio signals directly, without the need for explicit feature extraction. This model aims to learn representations from the raw waveform, potentially capturing more nuanced details of the audio. Despite its innovative approach, RawNet2 exhibited higher EER compared to LCNN, suggesting that while direct waveform processing is promising, it may still benefit from refined feature extraction techniques.

6.1.2 Datasets

The datasets used in our study are critical for training and evaluating these models. We employed the Attack Agnostic Dataset, which combines two distinct datasets: WaveFake and ASVspoof 2019 LA Subset. Each dataset contributes unique characteristics and variations of audio deepfakes, ensuring a comprehensive evaluation of our models.

WaveFake focuses solely on audio deepfakes generated by seven different neural network architectures, providing a broad spectrum of fake audio samples. This dataset includes 18,100 bona fide samples and 101,700 fake samples, making it a significant resource for training robust detection models.

ASVspoof 2019 LA Subset is designed for audio spoofing detection and includes 19 different audio generation methods, ranging from text-to-speech (TTS) to voice conversion (VC) algorithms. With 12,483 bona fide and 108,978 fake samples, this dataset adds further diversity and complexity to our training data.

By combining these datasets, we created a comprehensive training set with 31,083 real and 210,678 fake samples, covering 26 different audio generation methods. This extensive dataset allowed us to evaluate the generalization capabilities of our models across a wide range of audio deepfake scenarios.

6.1.3 Model Architectures

Four different deep learning models were evaluated for their effectiveness in detecting audio deepfakes:

6.1.3.1 LCNN (Light Convolutional Neural Network)

The Light Convolutional Neural Network (LCNN) is designed to provide efficient and effective processing of audio signals for the purpose of deepfake detection. Its architecture is optimized to capture intricate patterns and features within audio data, distinguishing authentic samples from manipulated ones. Here, we delve into the architecture and functionality of the LCNN to illustrate how it achieves its high performance.

6.1.3.1.1 Convolutional Layers

The LCNN architecture begins with several convolutional layers, which are essential for detecting local patterns in the input audio features. These layers apply convolution operations using a set of learnable filters to the input data. Each filter scans the input,

detecting features such as edges, transitions, and other localized patterns within the audio signal. The key aspects of these layers include:

Filter Size and Stride: The choice of filter size and stride is crucial. Smaller filters can capture finer details, while larger filters provide a broader view of the signal.

Activation Functions: ReLU (Rectified Linear Unit) is commonly used as the activation function after each convolutional layer. ReLU introduces non-linearity, enabling the network to learn complex patterns.

Pooling Layers: Typically, max-pooling or average-pooling layers follow convolutional layers to reduce the spatial dimensions of the data, helping to manage computational load and control overfitting by summarizing the presence of features.

6.1.3.1.2 Front-End Feature Extraction

The LCNN utilizes various front-end feature extraction techniques to convert raw audio signals into formats suitable for convolutional processing:

Linear-Frequency Cepstral Coefficients (LFCC): LFCCs are computed by applying a linear filter bank to the power spectrum of the audio signal, followed by a discrete cosine transform. LFCCs capture both human-audible and high-frequency artifacts, making them highly effective for deepfake detection.

Mel-Frequency Cepstral Coefficients (MFCC): MFCCs are derived by applying a mel-scale filter bank to the power spectrum and then taking the discrete cosine transform. They focus on features within the human hearing range.

Spectrograms: These are generated using short-time Fourier transforms (STFT), providing a time-frequency representation of the audio signal. Spectrograms can highlight changes in frequency content over time.

6.1.3.1.3 Fully Connected Layers

After the convolutional and pooling layers, the data is passed through several fully connected layers. These layers act as a classifier, interpreting the high-level features extracted by the convolutional layers to make the final deepfake detection decision. The fully connected layers are characterized by:

Dense Connections: Each neuron in a fully connected layer is connected to every neuron in the preceding layer, allowing for complex pattern recognition.

Dropout Regularization: Dropout is often applied to prevent overfitting by randomly setting a fraction of input units to zero during training, ensuring the network learns robust features.

6.1.3.1.4 Output Layer

The final layer of the LCNN typically uses a sigmoid activation function to produce a probability score indicating the likelihood that the input audio is a deepfake. The output is a binary classification: real or fake.

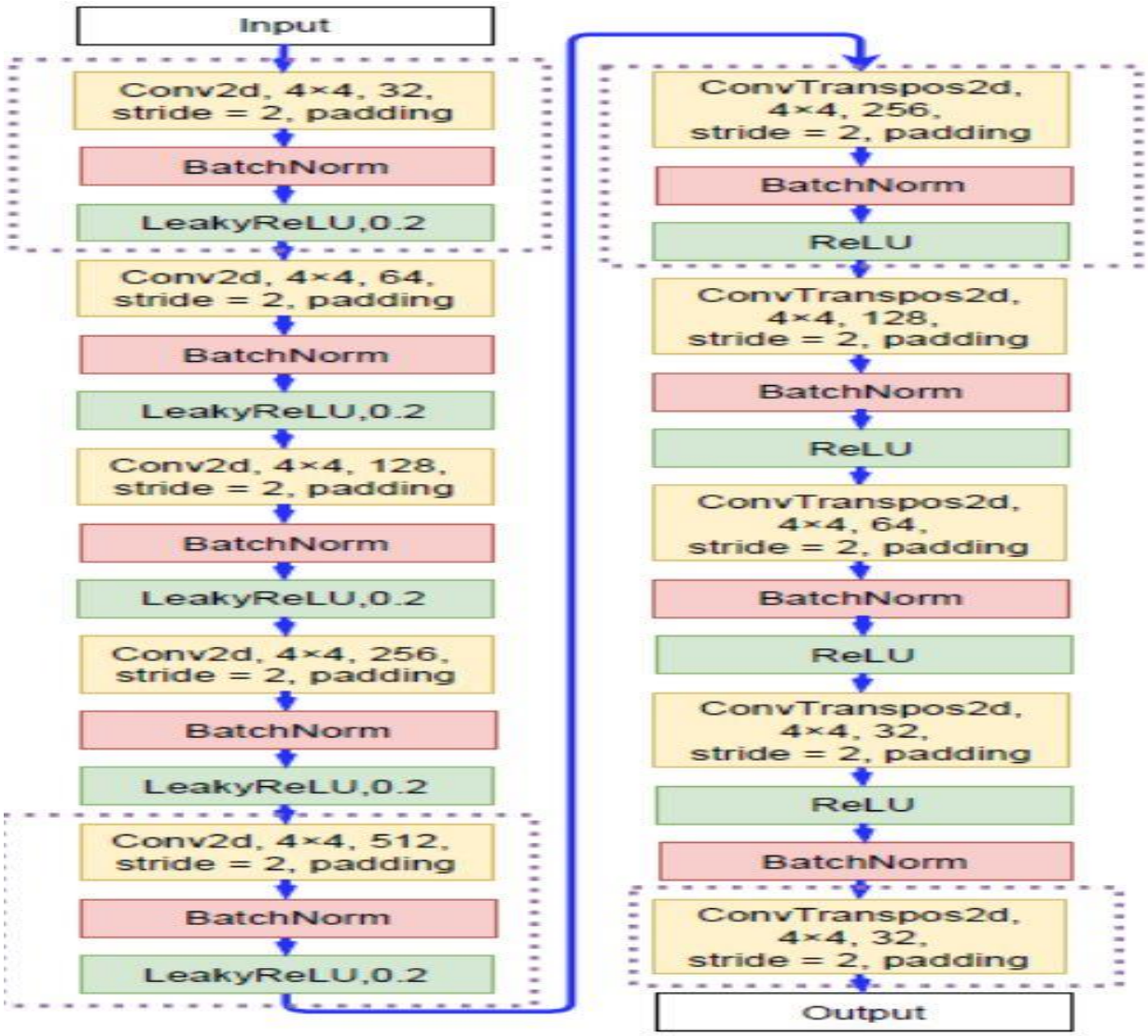

Figure 6: LCNN Architecture

6.1.3.1.5 Training and Optimization

The LCNN is trained using the Attack Agnostic Dataset, which includes a diverse set of audio deepfakes generated by various methods. The training process involves:

Loss Function: Binary cross-entropy is used as the loss function, measuring the difference between the predicted probability and the actual label.

Optimizer: The Adam optimizer is commonly used for its efficiency and effectiveness in training deep neural networks.

Batch Size and Epochs: The training typically involves a batch size of 128 and runs for several epochs until convergence, ensuring the model has sufficient exposure to the training data.

Performance Evaluation:

LCNN's performance is evaluated using the Equal Error Rate (EER), which is the rate at which the false acceptance rate (FAR) equals the false rejection rate (FRR). The model is tested across multiple dataset folds to ensure robustness and stability. In our study, the LCNN demonstrated the best generalization and stability, achieving the lowest EER across various audio deepfake scenarios.

By integrating these architectural elements and leveraging advanced feature extraction techniques, the LCNN effectively processes audio data, identifies deepfake characteristics, and provides reliable detection. Its superior performance underscores the importance of a well-designed convolutional network combined with robust feature extraction in the domain of audio deepfake detection.

6.1.3.2 XceptionNet

XceptionNet, or Extreme Inception, is an advanced deep learning model that builds upon the Inception architecture by using depth wise separable convolutions. This design allows XceptionNet to achieve high performance while maintaining computational efficiency. The architecture is specifically tailored for capturing complex patterns in data, making it highly suitable for tasks like audio deepfake detection.

6.1.3.2.1 Depthwise Separable Convolutions

XceptionNet replaces the standard convolution layers found in traditional CNNs with depthwise separable convolutions, which consist of two main operations:

Depthwise Convolution: Applies a single filter to each input channel separately. For example, if there are

M input channels and

D filters, this step produces

M×D output channels.

Pointwise Convolution: Uses a 1x1 convolution to combine the output channels from the depthwise convolution. This operation is responsible for mixing the information across different channels.

These two operations together drastically reduce the number of parameters and computational cost while maintaining the model's ability to capture intricate patterns.

6.1.3.2.2 Entry Flow, Middle Flow, and Exit Flow

XceptionNet's architecture is divided into three main flows:

Entry Flow: This section begins with standard convolutions to reduce the spatial dimensions of the input. It is followed by a series of depthwise separable convolution layers and max pooling layers, progressively reducing the spatial dimensions while increasing the depth.

Middle Flow: Comprising multiple blocks of depthwise separable convolutions, the middle flow performs the bulk of feature extraction. Each block consists of three depthwise separable convolution layers followed by a residual connection that helps preserve the gradients and improve training stability.

Exit Flow: The final section includes more depthwise separable convolutions followed by a global average pooling layer and a fully connected layer, which outputs the final predictions.

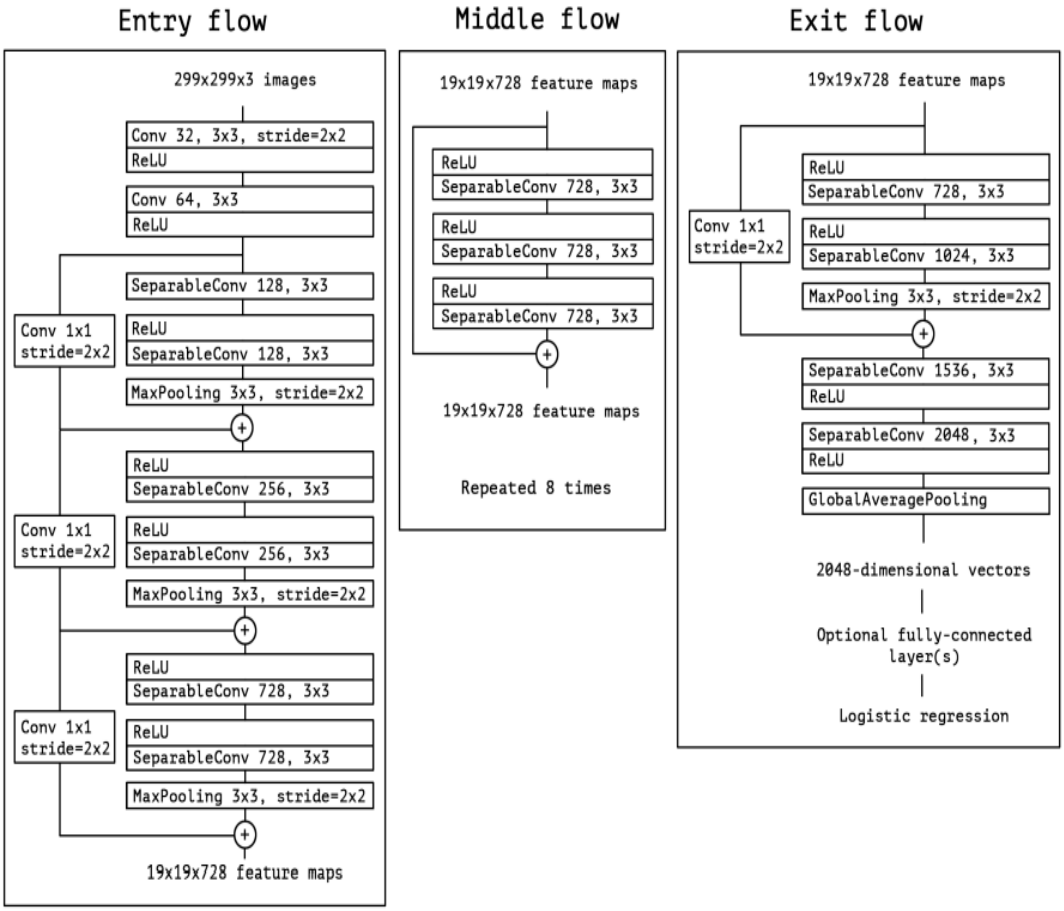

Figure 7: XceptionNet Architecture

6.1.3.2.3 Batch Normalization and Activation Functions

Throughout the network, batch normalization is applied after each convolutional layer to stabilize and accelerate the training process. The ReLU (Rectified Linear Unit) activation function introduces non-linearity, allowing the network to learn complex representations.

6.1.3.2.4 Feature Extraction

For the task of audio deepfake detection, XceptionNet uses Linear-Frequency Cepstral Coefficients (LFCC) as the primary front-end feature. LFCCs are effective at capturing both low and high-frequency components of the audio signal, which are crucial for identifying subtle artifacts introduced by deepfake generation methods.

6.1.3.2.5 Fully Connected Layers

After the convolutional and pooling layers, the output is passed through fully connected layers. These layers are responsible for combining the extracted features and making the final classification decision. The final layer typically uses a sigmoid activation function to produce a probability score indicating whether the audio is real or fake.

6.1.3.2.6 Training and Optimization

The XceptionNet model is trained using the Attack Agnostic Dataset, which combines various types of audios deepfakes. The training process involves:

Loss Function: Binary cross-entropy is used as the loss function, measuring the difference between the predicted probability and the actual label.

Optimizer: The Adam optimizer is employed for its efficiency in handling sparse gradients and large datasets.

Batch Size and Epochs: The training is conducted with a batch size of 128 and spans several epochs to ensure thorough learning.

Performance Evaluation

XceptionNet's performance is evaluated primarily using the Equal Error Rate (EER). Although it demonstrates good generalization across different types of audios deepfakes, it shows slightly less stability compared to the LCNN model. This variance in stability may be due to the inherent complexity of depthwise separable convolutions, which, while efficient, can sometimes lead to less consistent performance across diverse dataset folds.

By leveraging depthwise separable convolutions and an advanced architecture, XceptionNet effectively captures complex patterns in audio data. Its use of LFCC for feature extraction further enhances its ability to detect subtle anomalies in audio signals, making it a powerful tool for audio deepfake detection. However, its slight instability highlights the need for careful tuning and potential integration with other models to ensure robust performance across all scenarios.

6.1.3.3 MesoInception-4

MesoInception-4 is an advanced neural network architecture designed specifically for detecting deepfake content, particularly in multimedia contexts like images and audio. Its design focuses on capturing mesoscopic properties, which are intermediate-level features that can effectively distinguish between real and manipulated content.

6.1.3.3.1 Mesoscopic Feature Extraction

MesoInception-4 emphasizes the extraction of mesoscopic features. These features lie between low-level details (like edges and textures) and high-level concepts (like object shapes). This focus allows the network to identify subtle inconsistencies and artifacts introduced during the creation of deepfakes.

6.1.3.3.2 Inception Modules

The architecture incorporates inception modules, which were originally introduced in the GoogLeNet/Inception architecture. These modules are composed of parallel convolutional layers with different kernel sizes (1x1, 3x3, and 5x5), allowing the network to capture features at multiple scales simultaneously. In MesoInception-4, the inception modules help in efficiently extracting diverse features from the audio data.

6.1.3.3.3 Network Structure

MesoInception-4's structure can be broken down into the following components:

Initial Convolutional Layers: The network starts with standard convolutional layers that prepare the input for the more complex processing stages. These layers help in initial feature extraction and reducing the spatial dimensions.

Inception Blocks: Multiple inception blocks are used, each containing parallel convolutional layers with different kernel sizes. This design helps in capturing a variety of features at different scales, making the network robust to various types of artifacts.

Dense Layers: After the inception blocks, the extracted features are passed through fully connected (dense) layers. These layers combine the features and prepare them for the final classification task.

6.1.3.3.4 Feature Extraction

For audio deepfake detection, MesoInception-4 utilizes Linear-Frequency Cepstral Coefficients (LFCC) as the front-end feature extraction technique. LFCCs are effective in capturing both human-audible and high-frequency components, which are essential for detecting artifacts in deepfake audio.

6.1.3.3.5 Activation Functions and Batch Normalization

Throughout the network, ReLU (Rectified Linear Unit) activation functions are used to introduce non-linearity, allowing the network to learn complex patterns. Batch normalization is applied after convolutional layers to stabilize and accelerate the training process.

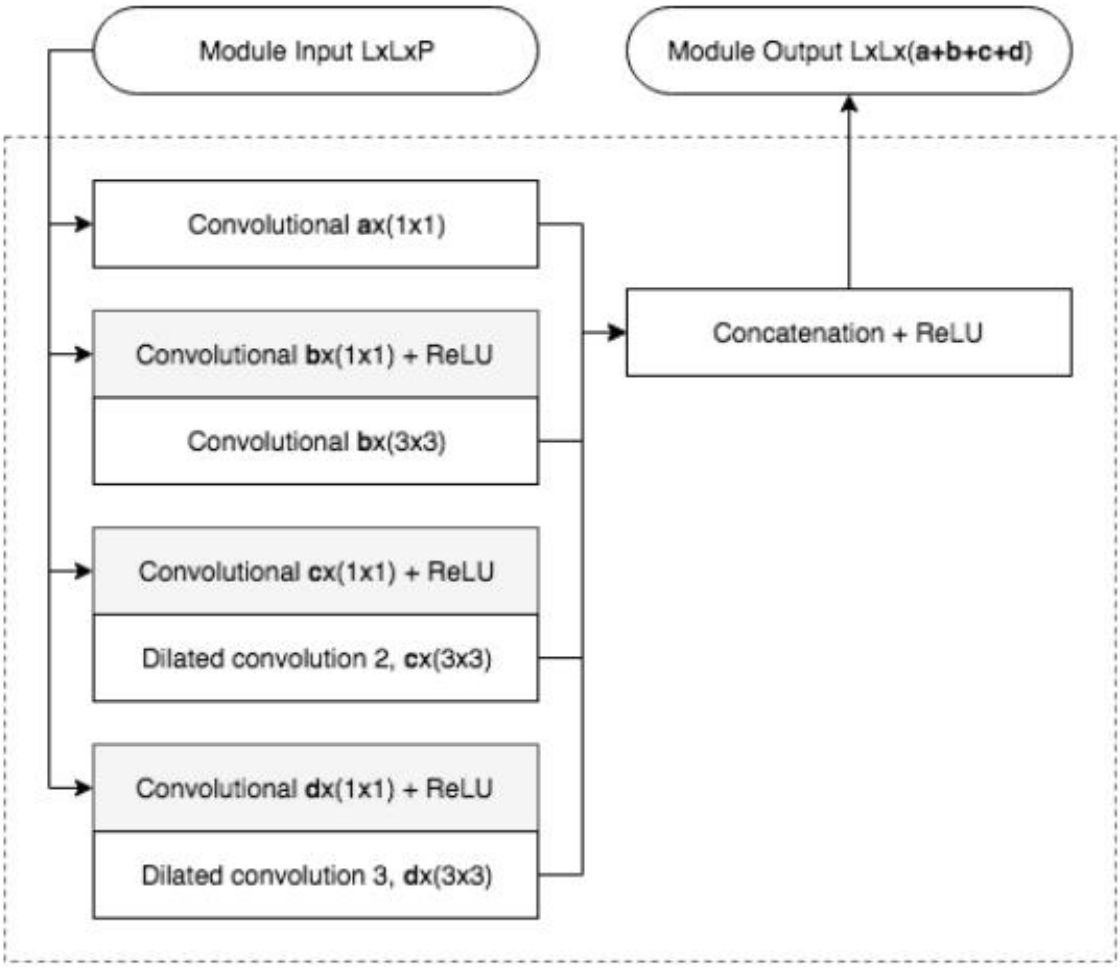

Figure 8: MesoInception-4 Architecture

6.1.3.3.6 How MesoInception-4 Works

MesoInception-4 works by processing the input audio through a series of convolutional and inception layers, extracting features at multiple scales. Here's a step-by-step breakdown of its operation:

Input Processing:

The input audio is first converted into LFCC features. These features represent the cepstral representation of the audio signal, capturing both temporal and spectral properties.

Convolutional Layers:

Initial convolutional layers process the LFCC features, reducing the dimensionality and highlighting important patterns in the data.

Inception Modules:

Inception modules process the features from the convolutional layers. Each module contains parallel convolutions with different kernel sizes, allowing the network to capture features at multiple scales. This multi-scale feature extraction is crucial for identifying both local and global inconsistencies in the audio signal.

Dense Layers:

The output from the inception modules is passed through dense layers. These layers integrate the multi-scale features and prepare them for classification.

Output Layer:

The final layer of the network is a fully connected layer with a sigmoid activation function, which outputs a probability score indicating whether the audio is real or fake.

6.1.3.3.7 Performance Evaluation

1. Equal Error Rate (EER):

MesoInception-4 showed higher EER compared to models like LCNN, indicating that it has a higher rate of misclassifications where the false acceptance rate equals the false rejection rate.

2. Stability:

The model exhibited less stability across different dataset folds. This instability might be due to the network's sensitivity to variations in the training data, which can lead to fluctuations in performance.

3. Generalization:

While MesoInception-4 can capture important mesoscopic features, its generalization across different types of audios deepfakes was not as strong as some of the other models. This suggests that while the model can detect certain types of manipulations effectively, it may struggle with others.

6.1.3.4 RawNet2

RawNet2 is a cutting-edge neural network architecture designed specifically for the task of audio deepfake detection, utilizing raw audio signals directly as input. Unlike traditional methods that rely on pre-processed features like spectrograms or cepstral coefficients, RawNet2 processes the raw waveform, aiming to capture intricate details in the audio signal that might be lost in feature extraction processes.

6.1.3.4.1 End-to-End Processing

RawNet2 is an end-to-end model, meaning it takes raw audio as input and directly processes it to make predictions. This approach avoids potential information loss that can occur during traditional feature extraction.

6.1.3.4.2 Network Structure

The architecture of RawNet2 consists of several key components, each designed to handle different aspects of the raw audio signal:

Initial Convolutional Layers: The first few layers of RawNet2 are standard convolutional layers that process the raw audio waveform. These layers apply filters to the input signal, capturing basic audio features such as edges and temporal patterns.

SincNet Layer: A unique aspect of RawNet2 is the inclusion of the SincNet layer. This layer uses parameterized sinc functions as filters, which are specifically designed to model band-pass filters. The SincNet layer is particularly effective at capturing frequency components of the audio signal, mimicking the way human auditory perception works.

Residual Blocks: The core of RawNet2 consists of multiple residual blocks, which are inspired by the ResNet architecture. These blocks enable the network to learn deeper representations by allowing gradients to flow more easily through the network. Each residual block contains a series of convolutional layers, batch normalization, and ReLU activations, followed by a shortcut connection that bypasses the block, adding the input directly to the output.

Temporal Convolutional Network (TCN): Following the residual blocks, RawNet2 incorporates a Temporal Convolutional Network. TCNs are particularly suited for sequential data like audio signals, as they can capture long-range dependencies and temporal patterns effectively.

Fully Connected Layers: The output from the TCN is flattened and passed through fully connected layers. These layers integrate the features extracted from the convolutional and residual layers, preparing them for the final classification task.

Output Layer: The final layer is a fully connected layer with a sigmoid activation function, which outputs a probability score indicating the likelihood of the audio being a deepfake.

6.1.3.4.3 Activation Functions and Batch Normalization

Throughout the network, ReLU (Rectified Linear Unit) activation functions introduce non-linearity, allowing the model to learn complex patterns. Batch normalization is applied after convolutional layers to stabilize and accelerate the training process.

Layer	Input:**64000** samples		Output shape
Fixed Sinc filters	Conv(**129**,1,128) Maxpooling(3) BN & LeakyReLU		(**21290**,128)
Res block	BN & LeakyReLU Conv(3,1,128) BN & LeakyReLU Conv(3,1,128) Maxpooling(3) FMS	× 2	(**2365**,128)
Res block	BN & LeakyReLU Conv(3,1, **512**) BN & LeakyReLU Conv(3,1, **512**) Maxpooling(3) FMS	× 4	(**29**,512)
GRU	GRU(1024)		(1024)
FC	1024		(1024)
Output	**1024**		**2**

Figure 9: RawNet Architecture

6.1.3.4.4 How RawNet2 Works

RawNet2 processes the raw audio waveform through a series of convolutional and residual layers, extracting features directly from the raw signal. Here's a step-by-step breakdown of its operation:

Input Processing:

The raw audio signal is fed directly into the network without any pre-processing or feature extraction.

Initial Convolutional Layers:

These layers apply filters to the raw waveform, capturing initial patterns and temporal features.

SincNet Layer:

The SincNet layer applies sinc function-based filters to the audio signal. These filters are effective in capturing the frequency components of the audio, which are crucial for identifying deepfake artifacts.

Residual Blocks:

The residual blocks process the filtered signal, extracting deeper and more complex features. The shortcut connections in these blocks help in preserving the gradient flow, making it easier to train the network.

Temporal Convolutional Network (TCN):

The TCN captures long-range dependencies and temporal patterns in the audio signal, which are essential for understanding the context and detecting subtle manipulations.

Fully Connected Layers:

The features extracted from the TCN are flattened and passed through fully connected layers. These layers integrate all the extracted features and prepare them for the final decision-making process.

Output Layer:

The final layer produces a probability score indicating whether the audio is genuine or a deepfake.

6.1.3.4.5 Performance Evaluation

1. Equal Error Rate (EER):

RawNet2 showed relatively stable results but with a higher EER compared to models like LCNN. This higher EER indicates a higher rate of misclassification where the false acceptance rate equals the false rejection rate.

2. Stability:

The model provided stable performance across different dataset folds, suggesting that it is robust to variations in the training data.

3. Generalization:

While RawNet2 performed well in capturing detailed features from raw audio, its generalization across different types of audios deepfakes was not as strong as some of the other models. This suggests that while the model can detect certain types of manipulations effectively, it may struggle with others.

6.1.4 Feature Extraction

Feature extraction is pivotal to the effectiveness of deepfake detection. In the context of audio deepfake detection, it involves transforming raw audio signals into a more manageable and informative format for machine learning models. This section explores three key feature extraction techniques: Linear-Frequency Cepstral Coefficients (LFCC), Mel-Frequency Cepstral Coefficients (MFCC), and Spectrograms. Each technique captures distinct aspects of the audio signal, each uniquely contributing to the detection process.

6.1.4.1 LFCC (Linear-Frequency Cepstral Coefficients)

1. Overview:

LFCC is a feature extraction technique designed to capture both human-audible and high-frequency artifacts present in audio signals. Unlike MFCC, which scales frequencies according to the mel scale (more aligned with human hearing), LFCC maintains a linear frequency scale, providing a more detailed representation of the high-frequency components of the signal.

2. How It Works:

Pre-emphasis: The raw audio signal is first pre-emphasized to boost the higher frequencies, which tend to have lower amplitudes. This step helps in balancing the frequency spectrum.

Framing: The pre-emphasized signal is divided into small overlapping frames. Each frame is short enough to assume that the signal is stationary within that window.

Windowing: A window function (typically Hamming or Hanning) is applied to each frame to reduce the spectral leakage.

Fourier Transform: A Fast Fourier Transform (FFT) is performed on each windowed frame to convert the time-domain signal into the frequency domain.

Power Spectrum: The power spectrum is calculated by squaring the magnitude of the FFT result.

Linear Filter Banks: The power spectrum is passed through a set of linear filter banks spaced evenly across the frequency range. This step captures the energy in each frequency band.

Logarithm: The logarithm of the filter bank energies is taken to mimic the logarithmic perception of loudness by the human ear.

Discrete Cosine Transform (DCT): Finally, a DCT is applied to the log filter bank energies to obtain the LFCCs. The DCT helps in decorrelating the filter bank coefficients and compressing the information.

3. Effectiveness:

LFCCs are particularly effective in capturing high-frequency artifacts that might be introduced during the deepfake generation process. These high-frequency components are critical for distinguishing between real and fake audio, especially in cases where the deepfake methods fail to accurately reproduce the high-frequency details.

4. Performance:

LFCC demonstrated the best overall performance in terms of Equal Error Rate (EER) and stability across multiple dataset folds. This makes it a robust choice for audio deepfake detection tasks.

6.1.4.2 MFCC (Mel-Frequency Cepstral Coefficients)

1. Overview:

MFCC is a widely used feature extraction technique in audio processing, particularly in speech recognition. It focuses on features within the human hearing range by mapping the frequencies to the mel scale, which approximates the human ear's sensitivity to different frequencies.

2. How It Works:

Pre-emphasis: Like LFCC, the raw audio signal is pre-emphasized to boost higher frequencies.

Framing and Windowing: The signal is divided into frames and windowed to minimize spectral leakage.

Fourier Transform: An FFT is applied to each frame to obtain the frequency spectrum.

Power Spectrum: The power spectrum is computed.

Mel Filter Banks: The power spectrum is passed through a set of mel-spaced filter banks. The mel scale is a nonlinear scale that places more emphasis on frequencies perceived as more important by human hearing.

Logarithm: The logarithm of the filter bank energies is taken.

Discrete Cosine Transform (DCT): A DCT is applied to obtain the MFCCs, which represent the short-term power spectrum of the sound.

3. Effectiveness:

MFCCs are effective in capturing the perceptually relevant features of the audio signal. However, their emphasis on the mel scale means they might not capture high-frequency artifacts as effectively as LFCCs.

4. Performance:

While MFCCs focus on features within the human hearing range, they tend to be less stable and generally provide a higher EER compared to LFCCs. This is due to their reduced sensitivity to high-frequency artifacts, which can be crucial for detecting audio deepfakes.

6.1.4.2 Spectrogram

1. Overview:

A spectrogram is a visual representation of the spectrum of frequencies in a signal as it varies with time. It provides a detailed view of the signal's frequency content and how it changes over time, making it a powerful tool for analyzing complex audio signals.

2. How It Works:

Framing and Windowing: The raw audio signal is divided into frames and windowed.

Short-Time Fourier Transform (STFT): An STFT is performed on each frame, producing a series of complex-valued frequency spectra. The STFT captures both time and frequency information by analyzing the signal in short, overlapping time segments.

Magnitude Spectrum: The magnitude of each complex-valued spectrum is taken to obtain the amplitude of the frequencies.

Color Mapping: The magnitudes are often mapped to colors or intensities, resulting in a visual representation where the x-axis represents time, the y-axis represents frequency, and the color intensity represents magnitude.

3. Effectiveness:

Spectrograms provide a comprehensive view of the signal's frequency content over time, capturing both steady-state and transient features. This makes them particularly useful for identifying temporal patterns and anomalies in the audio signal.

4. Performance:

While spectrograms offer detailed insights into the frequency content of the audio signal, they performed the worst individually in terms of EER. However, their performance improved when combined with other features like LFCCs, suggesting that spectrograms can complement other features by providing additional temporal and spectral information.

6.1.4.3 Summary

In summary, each of these feature extraction techniques offers unique advantages for audio deepfake detection:

LFCCs are effective in capturing both human-audible and high-frequency artifacts, providing the best overall performance in terms of EER and stability.

MFCCs focus on features within the human hearing range and are widely used in speech recognition, but they are less effective in detecting high-frequency artifacts introduced by deepfake methods.

Spectrograms offer a detailed view of the frequency content over time, capturing both steady-state and transient features. Although they performed the worst individually, they can enhance performance when combined with other features.

6.1.5 Training and Evaluation

Each model was trained using the Attack Agnostic Dataset, with the data split into three folds to evaluate generalization and stability:

Training Process:

Models were trained for 5 epochs using the Adam optimizer and binary cross-entropy loss function.

Training used a batch size of 128.

Each model was trained on a different fold to ensure robustness against various attacks.

Evaluation Metrics:

Equal Error Rate (EER) was the primary metric.

Stability was assessed by analyzing standard deviation of EER across different folds.

6.1.6 Results

The LCNN model with LFCC front-end showed the best performance, achieving low EER and high stability. The results for the main architectures on the evaluation set are summarized in this table:

Model	Fold	EER (%)	Std. Dev. (%)
LCNN + LFCC	1	9.526	0.728
	2	9.523	0.720
XceptionNet	1	16.206	1.476
	2	12.766	1.171
MesoInception-4	1	38.581	6.660
	2	36.381	5.936
RawNet2	1	19.443	1.358
	2	23.793	2.046

Table 4: Comparison Detection Models

6.1.7 Difficulties

Throughout the implementation process, we encountered several significant challenges, ranging from dataset preparation to model training and evaluation. These difficulties and the strategies employed to overcome them are discussed below in detail:

1. High Computational Resource Requirements:

One of the major issues was the need for high computational resources to train the models. Deep learning models, particularly those used for audio deepfake detection, require substantial computational power for both training and evaluation. This is due to the complex nature of audio data and the depth of the neural networks involved. Training models like LCNN, XceptionNet, MesoInception-4, and RawNet2 on large datasets demand considerable GPU resources and memory capacity.

Strategy to Overcome:

We leveraged cloud-based platforms such as Google Colab and Kaggle Notebooks, which provide access to powerful GPUs for free or at a low cost. This allowed us to train our models without the need for expensive hardware. Additionally, we optimized our code and employed techniques like data augmentation and model checkpointing to manage computational load efficiently.

2. Finding the Right Datasets:

Another major challenge was sourcing appropriate datasets for the project. Unlike image and video deepfakes, audio deepfakes have not been as extensively studied, resulting in a scarcity of large, high-quality datasets. The available datasets often lacked the diversity required to train robust models capable of generalizing well to various types of audios deepfakes.

Strategy to Overcome:

We used the Attack Agnostic Dataset, which combines three distinct datasets, including WaveFake and the ASVspoof 2019 LA Subset. By combining these datasets, we were able to create a comprehensive training set that covered a wide range of audio generation methods. This approach helped mitigate the dataset scarcity issue and provided the diversity needed for robust model training.

3. Lack of Resources and Literature:

The field of audio deepfake detection is relatively new, and there is a lack of extensive research and resources compared to image and video deepfakes. This made it challenging to find relevant studies, tools, and baseline models to guide our implementation.

Strategy to Overcome:

We conducted extensive literature reviews using academic databases and platforms like Google Scholar to gather information from the few available research papers on audio deepfake detection. Additionally, we participated in online forums and communities focused on audio processing and deep learning to seek advice and share insights with other researchers working on similar problems.

6.1.8 Tools

The implementation of our DeepFake detection system necessitated the use of various tools and technologies. Below is an overview of the key tools utilized, including programming languages, libraries, and frameworks:

1. Programming Languages and Development Environments:

Android Studio & Kotlin: We developed the front-end Android application using Android Studio with Kotlin as the programming language. Kotlin was chosen for its modern features and seamless integration with Android development.

Postman: This tool was used to test the network integration between our Android app and the machine learning back end. Postman allowed us to simulate API requests and verify the responses, ensuring robust communication between the client and server.

2. Machine Learning Platforms:

Google Colab: We used Google Colab extensively for training our machine learning models. Colab provides free access to GPUs, which was essential for handling the computationally intensive tasks associated with deep learning model training. It also offers a collaborative environment where we could easily share notebooks and code.

Kaggle Notebooks: Like Google Colab, Kaggle Notebooks provided another platform with free GPU access. Kaggle also hosts a wide array of datasets and competitions, which we leveraged to find additional resources and benchmark our models.

3. Libraries and Frameworks:

TensorFlow and Keras: These deep learning libraries were used to build and train our models. TensorFlow provided the underlying computational framework, while Keras offered a user-friendly interface for defining and training neural networks.

Librosa: This Python library was utilized for audio processing tasks such as feature extraction. Librosa provides tools for loading audio files, extracting features like MFCCs and LFCCs, and performing signal transformations.

SciPy and NumPy: These scientific computing libraries were used for various data manipulation and analysis tasks, including handling audio data and performing mathematical operations required during feature extraction and model evaluation.

4. Research and Documentation:

Google Scholar: We used Google Scholar extensively to find and review research papers related to audio deepfake detection. This helped us stay updated with the latest advancements in the field and incorporate state-of-the-art techniques into our project.

Online Forums and Communities: Platforms like Stack Overflow, Reddit, and specialized AI forums provided a space to ask questions, share experiences, and find solutions to technical problems encountered during the implementation.

6.1.9 Conclusion

In this chapter, we detailed the implementation and evaluation of four deepfake detection models using the Attack Agnostic Dataset. The models included Light Convolutional Neural Network (LCNN), XceptionNet, MesoInception-4, and RawNet2, each leveraging different feature extraction techniques and architectural strengths.

LCNN with LFCC Front-End:

Among the models, LCNN with LFCC front-end emerged as the most effective solution. This combination demonstrated robust and stable performance across various attacks, achieving the lowest EER and highest generalization capabilities. The efficiency of LFCC in capturing a broad spectrum of audio features played a crucial role in the superior performance of LCNN.

XceptionNet and MesoInception-4:

While XceptionNet also utilized LFCC features and achieved good generalization, it exhibited slightly less stability compared to LCNN. MesoInception-4, designed to capture mesoscopic properties of audio data, showed higher EER and less stable performance, indicating that while it could capture important intermediate-level features, it was not as robust under varying conditions.

RawNet2:

RawNet2, which processes raw audio signals directly, offered a unique approach by attempting to learn representations from the raw waveform. Despite its innovative method, RawNet2 had higher EER compared to LCNN, suggesting that explicit feature extraction still holds an advantage in the current context of audio deepfake detection.

Dataset Utilization:

The use of the Attack Agnostic Dataset, which combined WaveFake and the ASVspoof 2019 LA Subset, provided a diverse and comprehensive training set. This dataset played a critical role in enabling the models to generalize across a wide range of audio deepfake scenarios, covering multiple audio generation methods and variations.

These findings highlight the critical importance of selecting appropriate feature extraction methods and using diverse, comprehensive datasets for training deepfake detection models. The robustness and stability of the LCNN with LFCC front-end make it a particularly promising approach for future developments in audio deepfake detection. As the field progresses, continued exploration of feature extraction techniques and model architectures will be essential to further enhance the detection capabilities and ensure resilience against evolving audio deepfake technologies.

6.2 Voice Filtration

6.2.1 Abstract

Most deep learning-based models for speech enhancement have mainly focused on estimating the magnitude of spectrogram while reusing the phase from noisy speech for reconstruction. This is due to the difficulty of estimating the phase of clean speech. To improve speech enhancement performance, we tackle the phase estimation problem in three ways. First, we propose Deep Complex U-Net, an advanced U-Net structured model incorporating well-defined complex-valued building blocks to deal with complex-valued spectrograms. Second, we propose a polar coordinate-wise complex-valued masking method to reflect the distribution of complex ideal ratio masks. Third, we define a novel loss function, weighted source-to-distortion ratio (wSDR) loss, which is designed to directly correlate with a quantitative evaluation measure. Our model was evaluated on a mixture of the Voice Bank corpus and DEMAND database, which has been widely used by many deep learning models for speech enhancement. Ablation experiments were conducted on the mixed dataset showing that all three proposed approaches are empirically valid. Experimental results show that the proposed method achieves state-of-the-art performance in all metrics, outperforming previous approaches by a large margin.

6.2.2 Dataset

To perform direct performance comparison Noise and clean speech recordings were provided from the Diverse Environments Multichannel Acoustic Noise Database and the Voice Bank corpus respectively, each recorded with sampling rate of 48kHz.

Mixed audio inputs used for training were composed by mixing the two datasets with four signal- to-noise ratio (SNR) settings (15, 10, 5, and 0 (dB)), using 10 types of noise (2 synthetic + 8 from DEMAND) and 28 speakers from the Voice Bank corpus, creating 40 conditional patterns for each speech sample. The test set inputs were made with four SNR settings different from the training set (17.5, 12.5, 7.5, and 2.5 (dB)), using the remaining 5 noise types from DEMAND and 2 speakers from the Voice Bank corpus. Note that the speaker and noise classes were uniquely selected for the training and test sets.

Noise train input	11600 files
Clean trainset wav	11600 files
Noise test input	824 files
Clean test wav	824 files

Table 5: Dataset Composition for Voice Filtration Model

6.2.3 Pre-processing

The original raw waveforms were first sampled from 48kHz to 16kHz. For the actual model input, complex-valued spectrograms were obtained from the down sampled waveforms via STFT with a 64ms sized Hann window and 16ms hop length.

6.2.4 Model Architecture

6.2.4.1 DCU net

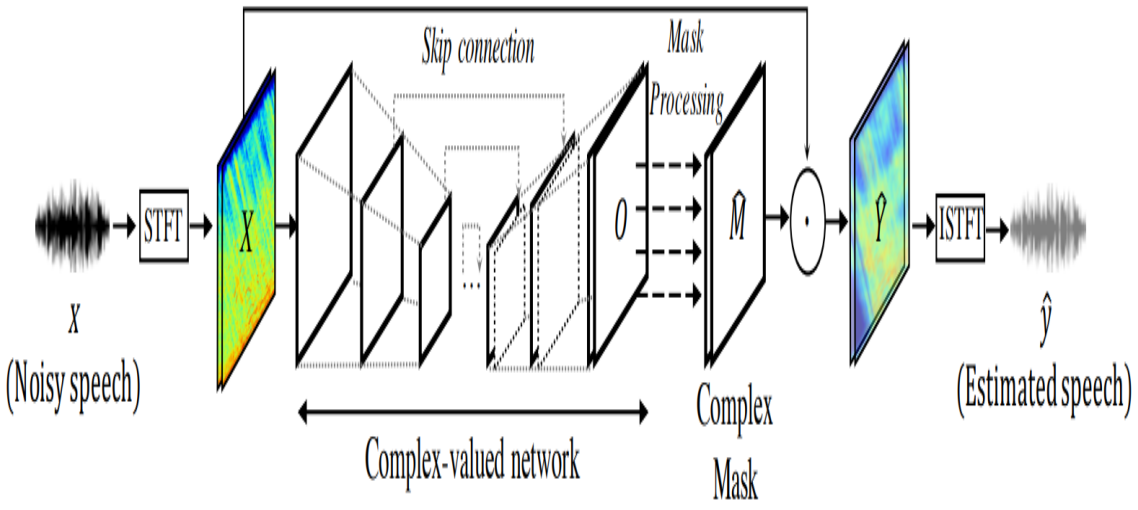

Figure 10: DcuNet Architecture

The U-Net structure is a well-known architecture composed as a convolutional autoencoder with skip-connections, originally proposed for medical imaging in computer vision community Furthermore, the use of real-valued U-Net has been shown to be also effective in many recent audio source separation tasks such as music source separation Stoller et al., 2018; Takahashi et al., 2018b), and speech enhancement Complex U-Net (DCU net) is an extended U-Net, refined specifically to explicitly handle complex domain operations. In this section, we will describe how U-Net is modified using the complex building blocks originally proposed by Complex-valued Building Blocks. Given a complex-valued convolutional filter W = A+ Ib with real-valued matrices A and B, the complex convolution operation on complex vector h = x + I y with W is done by $W * h = (A * x - B * y) + i (B * x + A * y)$. In practice, complex convolutions can be implemented as two different real-valued convolution operations with shared real value. convolution filters. Details are illustrated in the Appendix.

6.2.5 Evaluation

We performed qualitative evaluations by obtaining preference scores between the proposed DCU net (Large-DCUnet-20) and baseline methods. 15 utterance samples with different noise levels were selected from the test set and used for subjective listening tests. For each noisy sample, all possible six pairs of denoised audio samples from four different algorithms were presented to the participants in a random order, resulting in 90 pairwise comparisons to be made by each subject. For each comparison, participants were presented with three audio samples - original noisy speech and two denoised speech samples by two randomly selected algorithms - and instructed to choose either a preferred sample (score 1) or “can’t decide” (score 0.5).

6.2.6 Result

	CSIG	CBAK	COVL	PESQ	SSNR
Wiener (Scalart et al., 1996)	3.23	2.68	2.67	2.22	5.07
SEGAN (Pascual et al., 2017)	3.48	2.94	2.80	2.16	7.73
Wavenet (Rethage et al., 2018)	3.62	3.23	2.98	-	-
MMSE-GAN (Soni et al., 2018)	3.80	3.12	3.14	2.53	-
Deep Feature Loss (Germain et al., 2018)	3.86	3.33	3.22	-	-
DCUnet-20 (ours)	4.24	4.00	3.69	3.13	15.95
Large-DCUnet-20 (ours)	**4.34**	**4.10**	**3.81**	**3.24**	**16.85**

Table 6: DcuNet Result

1. CSIG: Measures the overall intelligibility of the enhanced speech signal.
2. CBAK: Assesses the clarity of the enhanced speech signal.
3. COVL: Measures the overall loudness of the enhanced speech signal.
4. PESQ: Assesses the overall perceived quality of the enhanced speech signal from a listener's perspective.
5. SSNR: Measures the signal-to-noise ratio.

- Training loss: 0.0117
- Testing loss: 0.0117

6.2.7 Conclusion

We proposed Deep Complex U-Net which combines two models to deal with complex-valued spectrograms for speech enhancement. In doing so, we designed a new complex-valued masking method optimized with a novel loss function, weighted-SDR loss. Through ablation studies, we showed that the proposed approaches are effective for more precise phase estimation, resulting in state-of-the-art performance for speech enhancement. Furthermore, we conducted both quantitative and qualitative studies and demonstrated that the proposed method is consistently superior to the previously proposed algorithms.

6.3 Deepfake Voice Generation

6.3.1 TTS(Text-To-Speech)

6.3.1.1 Tactron1

6.3.1.1.1 Abstract

A text-to-speech synthesis system typically consists of multiple stages, such as a text analysis frontend, an acoustic model, and an audio synthesis module. Building these components often requires extensive domain expertise and may contain brittle design choices, an end-to-end generative text-to-speech model that synthesizes speech directly from characters. Given<text, audio> pairs, the model can be trained completely from scratch with random initialization. We present several key techniques to make the sequence-to-sequence framework perform well for this challenging task. Tactron1 achieves a 3.82 subjective 5-scale mean opinion score on US English, outperforming a production parametric system in terms of naturalness. In addition, since Tactron1 generates speech at the frame level, it's substantially faster than sample-level autoregressive methods.

6.3.1.1.2 Model Architecture

The diagram provides a high-level overview of the different stages involved in this process.

The model takes characters as input and outputs the corresponding raw spectrogram, which is then fed to the Griffin-Lim reconstruction algorithm to synthesize speech.

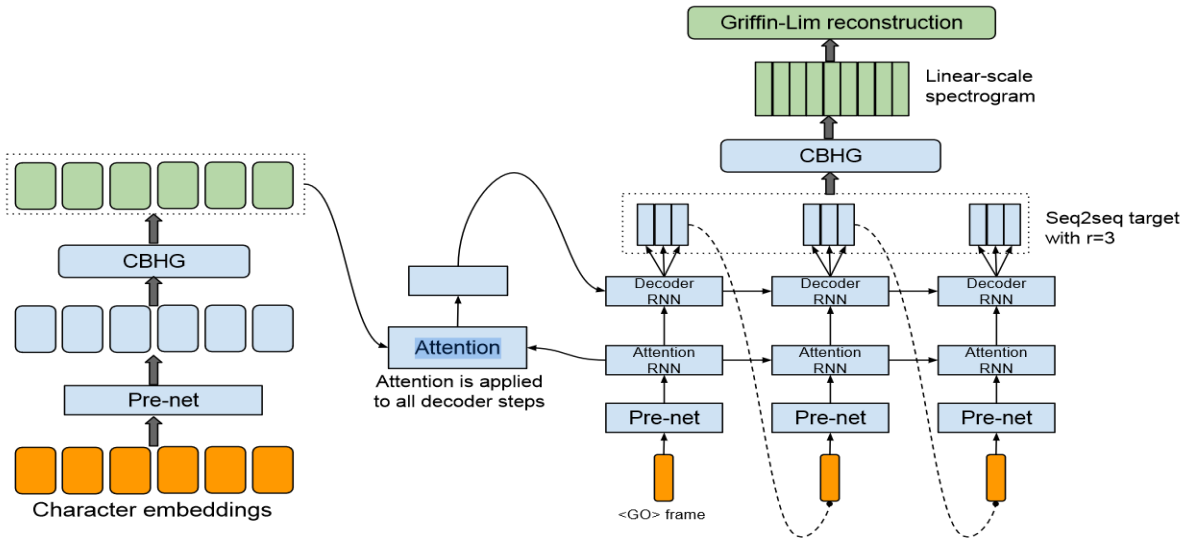

Figure 11: Tacotron-1 Architecture

- Character embeddings: This is the first stage of the model, where each character in the input text is converted into a numerical vector.
- Pre-net: This stage applies a pre-net function to the character embeddings. A pre-net is a type of neural network that is used to learn a non-linear transformation of the input data.
- Encoder RNN: This is a recurrent neural network (RNN) that encodes the character embeddings into a sequence of hidden states.
- Attention mechanism: The attention mechanism allows the decoder RNN to focus on specific parts of the encoder output when generating the spectrogram. This is important for tasks like speech synthesis, where the model needs to pay attention to different parts of the input text in order to generate the correct sounds.
- Decoder RNN: This is another RNN that decodes the hidden states from the encoder RNN into a sequence of mel spectrograms. Mel spectrograms are a type of spectrogram that is based on the human auditory system.
- Griffin-Lim reconstruction: This stage converts the mel spectrogram back into a waveform, which is the actual audio signal.
- CBHG
- is a powerful module for extracting representations from sequences.

6.3.1.1.3 Dataset

We train Tacotron1 on an internal North American English dataset, which contains about 24.6 hours of speech data spoken by a professional female speaker. The phrases are text normalized, e.g. “16” is converted to “sixteen”.

6.3.1.1.4 Post processing

The post-processing net's task is to convert the seq2seq target to a target that can be synthesized into waveforms. Since we use Griffin-Lim as the synthesizer, the post-processing net learns to predict spectral magnitude sampled on a linear-frequency scale. Another motivation of the post processing net is that it can see the full decoded sequence. In contrast to seq2seq, which always runs from left to right, it has both forward and backward information to correct the prediction for each individual frame. In this work, we use a CBHG module for the post-processing net, though a simpler architecture likely works as well. The concept of a post-processing network is highly general. It could be used to predict alternative targets such as vocoder parameters, or as a Wave Net-like neural vocoder (van den Oord et al., 2016; Mehri et al., 2016; Arik et al., 2017) that synthesizes waveform samples directly.

We use the Griffin-Lim algorithm 3-(Griffin & Lim, 1984) to synthesize waveform from the predicted spectrogram. We found that raising the predicted magnitudes by a power of 1.2 before feeding to Griffin-Lim reduces artifacts, likely due to its harmonic enhancement effect. We observed that Griffin-Lim converges after 50 iterations (in fact, about 30 iterations seems to be enough), which is reasonably fast. We implemented Griffin-Lim in TensorFlow (Abadi et al., 2016) hence it's also part of the model. While Griffin-Lim is differentiable (it does not have trainable weights), we do not impose any loss on it in this work. We emphasize that our choice of Griffin-Lim is for simplicity; while it already yields strong results, developing a fast and high-quality trainable spectrogram to waveform inverter is ongoing work.

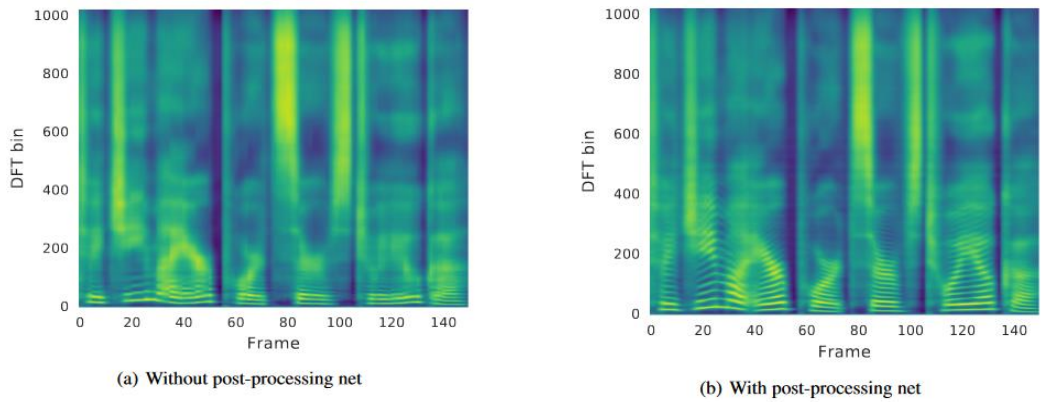

Figure 12: Tacotron-1 Post Processing

6.3.1.2 Tactron2

6.3.1.2.1 Abstract

A neural network architecture for speech synthesis directly from text. The system is composed of a recurrent sequence-to-sequence feature prediction network that maps character embeddings to mel-scale spectrograms, followed by a modified WaveNet model acting as a vocoder to synthesize time-domain waveforms from those spectrograms. Our model achieves a mean opinion score (MOS) of 4.53 comparable to a MOS of 4.58 for professionally recorded speech. To validate our design choices, we present ablation studies of key components of our system and evaluate the impact of using mel spectrograms as the conditioning input to WaveNet instead of linguistic, duration, and F0 features. We further show that using this compact acoustic intermediate representation allows for a significant reduction in the size of the WaveNet architecture.

6.3.1.2.2 Model Architecture

Our proposed system consists of two components, (1) a recurrent sequence-to-sequence feature prediction network with attention which predicts a sequence of mel spectrogram frames from an input character sequence, and (2) a modified version of WaveNet which generates time-domain waveform samples conditioned on the predicted mel spectrogram frames.

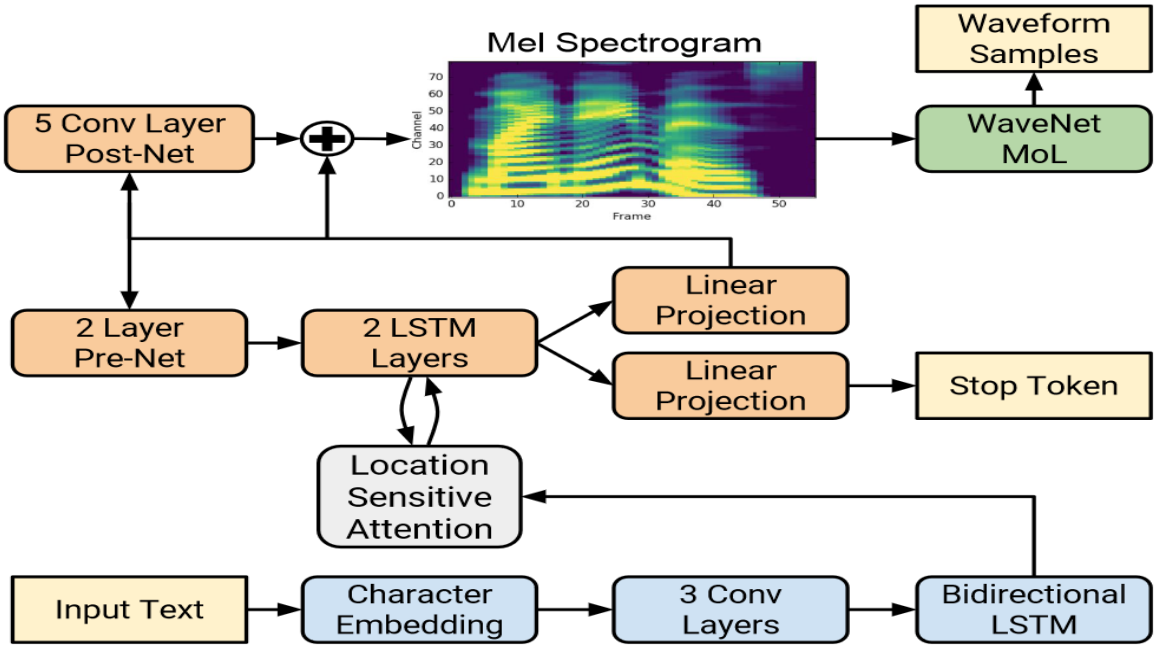

Figure 13: Tacotron-2 Architecture

Our proposed system consists of two components, shown in Figure 1:

(1) a recurrent sequence-to-sequence feature prediction network with attention which predicts a sequence of mel spectrogram frames from an input character sequence, and (2) a modified version of WaveNet which generates time-domain waveform samples conditioned on the predicted mel spectrogram frames.

- Input Text: This is the text that you want to convert into speech.
- Character Embedding: This is the first stage of the model, where each character in the input text is converted into a numerical vector. Each vector captures information about the character, such as its sound and position in the word.
- 3 conv layers: A series of 1D convolutional layers can be applied to the sequence of character embeddings. These layers aim to extract local features from the character sequence.
- Bidirectional LSTM: A bidirectional LSTM is used to process the sequence, potentially following the convolutional layers.
- Pre-net: This stage applies a pre-net function to the character embeddings. A pre-net is a type of neural network that is used to learn a non-linear transformation of the input data.
- 2 Layer LSTM: This refers to a two-layer Long Short-Term Memory (LSTM) network, a type of recurrent neural network (RNN) that is able to learn long-term dependencies in data. LSTMs are well-suited for tasks like speech synthesis, where the model needs to be able to remember the pronunciation of words in order to generate the correct sounds. Here, the two layers work together to process the character embeddings and capture their contextual information.
- Location-sensitive attention: This mechanism allows the model to focus on specific parts of the encoded text (character embeddings) when generating different parts of the spectrogram. For instance, when generating the spectrogram for the beginning of a word, the model will pay more attention to the beginning characters of the encoded text.
- Mel Spectrogram: This is a type of spectrogram that is based on the human auditory system. It is a visual representation of the sound, where the frequency of the sound is shown on the y-axis and time is shown on the x-axis. The darkness of each area in the spectrogram represents the amplitude of the sound at that frequency and time.
- WaveNet: This is a generative model that is used to generate the audio waveform from the mel spectrogram. The WaveNet model is like a decoder that learns to invert the process of creating a spectrogram from a sound wave.
- Post-net: This stage is used to further improve the quality of the generated waveform by making small adjustments to it.

6.3.1.2.3 Dataset

We train all models on an internal US English dataset [12], which contains 24.6 hours of speech from a single professional female speaker. All text in our datasets is spelled out. e.g., “16” is written as “sixteen”, i.e., our models are all trained on normalized text.

6.3.1.2.4 Post processing

Since it is not possible to use the information of predicted future frames before they have been decoded, we use a convolutional post-processing network to incorporate past and future frames after decoding to improve the feature predictions. However, because WaveNet already contains convolutional layers, one may wonder if the post-net is still necessary when Wave Net is used as the vocoder. To answer this question, we compared our model with and without the post-net, and found that without it, our model only obtains a MOS score of 4.429 ± 0.071, compared to 4.526 ± 0.066 with it, meaning that empirically the post-net is still an important part of the network design.

6.3.1.3 Evaluation

When generating speech in inference mode, the ground truth targets are not known. Therefore, the predicted outputs from the previous step are fed in during decoding, in contrast to the teacher-forcing configuration used for training.

We randomly selected 100 fixed examples from the test set of our internal dataset as the evaluation set. Audio generated on this set are sent to a human rating service similar to Amazon’s Mechanical Turk where each sample is rated by at least 8 raters on a scale from 1 to 5 with 0.5-point increments, from which a subjective mean opinion score (MOS) is calculated. Each evaluation is conducted independently from each other, so the outputs of two different models are not directly compared when raters assign a score to them.

Note that while instances in the evaluation set never appear in the training set, there are some recurring patterns and common words between the two sets. While this could potentially result in an inflated MOS compared to an evaluation set consisting of sentences generated from random words, using this set allows us to compare to the ground truth. Since all the systems we compare are trained on the same data, relative comparisons are still meaningful.

Table 1 shows a comparison of our method against various prior systems.

To better isolate the effect of using Mel spectrograms as features, we compare to a Wave Net conditioned on linguistic features We also compare to the original Tacotron2 that predicts linear spectrograms and uses Griffin-Lim to synthesize audio, as well as concatenative [30] and parametric [31] baseline systems, both of which have been used in production at Google. We find that the proposed system significantly outperforms all other TTS systems, and results in an MOS comparable to that of the ground truth audio.

System	MOS
Parametric	3.492 ± 0.096
Tacotron (Griffin-Lim)	4.001 ± 0.087
Concatenative	4.166 ± 0.091
WaveNet (Linguistic)	4.341 ± 0.051
Ground truth	4.582 ± 0.053
Tacotron 2 (this paper)	$\mathbf{4.526 \pm 0.066}$

Figure 14: Tacotron-2 Evaluation

6.3.1.4 Your TTS

6.3.1.4.1 Abstract

Your TTS brings the power of a multilingual approach to the task of zero-shot multi-speaker TTS. Our method builds upon the VITS model and adds several novel modifications for zero-shot multi-speaker and multilingual training. We achieved state-of- the-art (SOTA) results in zero-shot multi-speaker TTS and results comparable to SOTA in zero-shot voice conversion on the VCTK dataset. Additionally, our approach achieves promising results in a target language with a single-speaker dataset, opening possibilities for zero-shot multi-speaker TTS and zero-shot voice conversion systems in low-resource languages. Finally, it is possible to fine-tune the Your TTS model with less than 1 minute of speech and achieve state-of-the-art results in voice similarity and with reasonable quality. This is important to al- low synthesis for speakers with a very different voice or recording characteristics from those seen during training.

6.3.1.4.2 Model Architecture

- Training Procedure

1. Input Text and Language ID:
    - Input text is converted into character embeddings.
    - Language ID is converted into language embeddings.
2. Transform-Based Encoder:
    - The character and language embeddings are combined and processed through a transformer-based encoder consisting of 10 transformer blocks.
    - The output from the encoder is a sequence of latent variables mt\mathbf{m}_tmt.
3. Linear Projection:
    - The encoder output mt\mathbf{m}_tmt is linearly projected to align with the target phoneme duration space.
4. Monotonic Alignment Search:
    - This module aligns the encoded text features with the target acoustic features to produce a latent sequence zp\mathbf{z}_pzp.
5. Posterior Encoder:
    - It takes the linear spectrogram as input and processes it through multiple Wave Net residual blocks to produce another latent variable z\mathbf{z}z.

6. Flow-Based Decoder:

- The latent variable zp\mathbf{z}_pzp from the alignment search is transformed using affine coupling layers in the flow-based decoder.
- The decoder includes four affine coupling layers, each containing four Wave Net residual blocks.

7. Stochastic Duration Predictor:

- It predicts the durations d\mathbf{d}d of the phonemes stochastically, using a noise input to ensure variability.
- The predicted durations are used to guide the alignment process.

8. Speaker Encoder and Embedding:

- A pre-trained speaker encoder extracts speaker embeddings from a reference waveform.
- These speaker embeddings are used to condition the model to generate speech in the desired speaker's voice.

9. HiFi-GAN Generator:

- The latent variable z\mathbf{z}z is then passed to the HiFi-GAN generator, which converts it into a waveform.

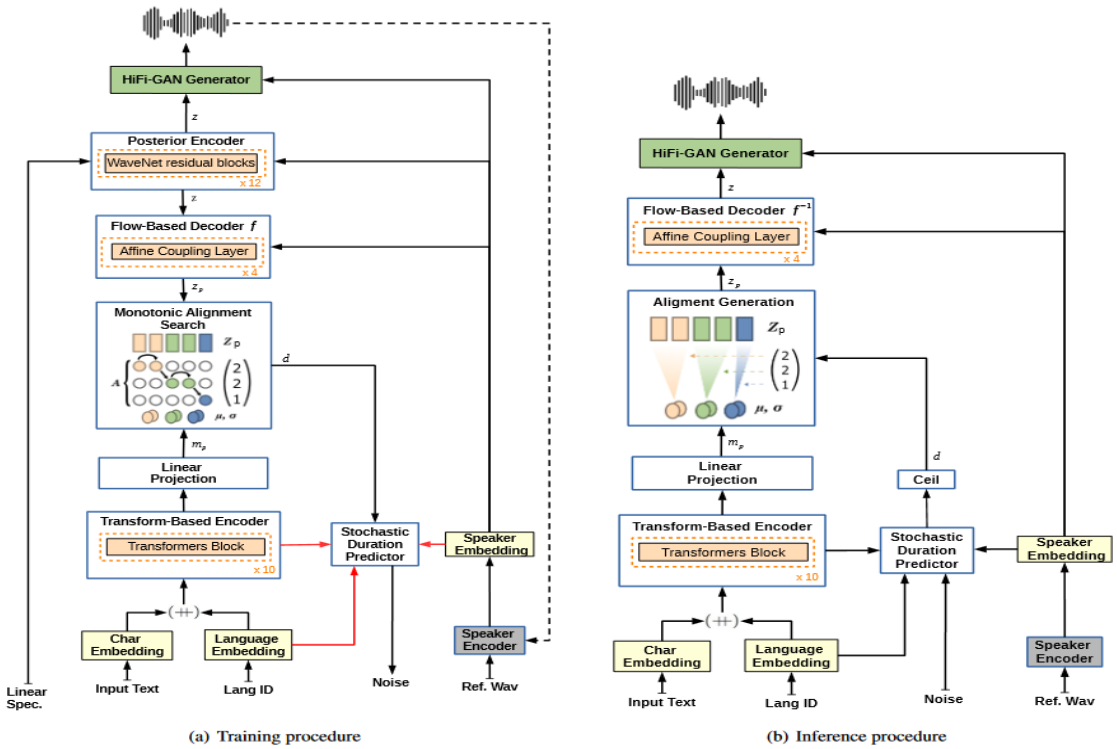

Figure 15: Your TTS Architecture

- Inference Procedure

1. Input Text and Language ID:
    - Like the training procedure, input text is converted into character embeddings.
    - Language ID is converted into language embeddings.
2. Transform-Based Encoder:
    - The character and language embeddings are processed through the same transformer-based encoder.
3. Linear Projection:
    - The output from the encoder is linearly projected to align with the phoneme duration space.
4. Alignment Generation:
    - The model uses the stochastic duration predictor to estimate the durations d\mathbf{d}d of the phonemes.
    - These durations are used to generate the latent sequence zp\mathbf{z}_pzp.
5. Flow-Based Decoder:
    - The latent sequence zp\mathbf{z}_pzp is processed through the flow-based decoder with affine coupling layers to produce the latent variable z\madhab{z}z.
6. Speaker Encoder and Embedding:
    - A reference waveform is passed through a speaker encoder to extract the speaker embedding.
    - These embedding conditions the model to produce speech in the target speaker's voice.
7. HiFi-GAN Generator:
    - The latent variable z\madhab{z}z is converted into a waveform by the HiFi-GAN generator, resulting in the final synthesized speech.

6.3.1.4.3 Dataset

6.3.1.4.3.1 VCTK Dataset

In the VCTK dataset experiments, the best similarity results were achieved with experiment 1 (monolingual) and experiment 2 with Speaker Consistency Loss (SCL) (bilingual). SCL improved similarity in two out of three experiments, although confidence intervals overlap, making the analysis inconclusive. Notably, SECS for all experiments on the VCTK dataset were higher than ground truth, possibly due to dataset characteristics like breathing sounds. Overall, the best experiments maintained good quality and similarity for unseen speakers, outperforming previous studies.

English: VCTK dataset, which contains 44 hours of speech and 109 speakers, sampled at 48KHz. We divided the VCTK dataset into train, development (containing the same speakers as the train set) and test. For the test set, we selected 11speakers that are neither in the development nor the training set.

French: FR set of the M-AILABS dataset which is based on LibriVox6. It consists of 2 female (104h) and 3 male speakers (71h) sampled at 16KHz.To evaluate the zero-shot multi-speaker capabilities of our model in English, we use the 11 VCTK speakers reserved for testing. To further test its performance outside of the VCTK domain, we select 10 speakers (5F/5M) from subset test-clean of LibriTTS dataset.

6.3.1.4.3.2 Libri TTS Dataset

Experiment 4 achieved the best similarity in the LibriTTS dataset, likely due to the broader coverage of voice diversity with approximately 1.2k speakers. However, MOS achieved the best result for the monolingual case, attributed to the higher quality of the VCTK dataset compared to other datasets.

6.3.1.4.3.3 Portuguese MILS Dataset

Experiment 3+SCL achieved the highest MOS metric for the Portuguese MLS dataset, indicating good quality in zero-shot multi-speaker synthesis despite being trained on a single-speaker dataset of medium quality. Sim-MOS results were relevant, although SECS and Sim-MOS did not entirely agree, possibly due to confidence intervals

6.3.1.4.4 Experiments

This experimental setup outlines a series of training experiments conducted with a TTS (Text-to-Speech) model called Your TTS. The goal appears to be to improve the model's performance by training it on various datasets and utilizing transfer learning techniques.

Here's a breakdown of the experiments:

1. Experiment 1: This experiment involves training Your TTS using the VCTK dataset, which is monolingual (presumably English).
2. Experiment 2: In addition to the VCTK dataset, the model is trained using the TTS-Portuguese dataset, making it bilingual.
3. Experiment 3: This experiment extends the training to include not only VCTK and TTS-Portuguese but also the M-AILABS French dataset, making it trilingual.
4. Experiment 4: Building upon the model trained in Experiment 3, this experiment further enhances the training by incorporating 1151 additional English speakers from the Libri TTS dataset, specifically from the train-clean-100 and train-clean-360 partitions.

For each experiment, transfer learning is employed, where the model is initialized from a pre-trained state and then further trained on the specific dataset(s) for a certain number of steps. Additionally, fine-tuning is performed using a Speaker Consistency Loss (SCL) method. The learning rate and decay strategy are carefully managed using the AdamW optimizer with specific parameters.

During training, an NVIDIA TESLA V100 32GB GPU is utilized with a batch size of 64. Weighted random sampling is employed for multilingual experiments to ensure a balanced representation of languages in each batch.

The rationale behind these experiments seems to be to enhance the TTS model's ability to generalize across languages and speakers by exposing it to diverse datasets and utilizing transfer learning techniques. This is particularly important given the limitations of the VCTK dataset in terms of speaker variety and recording conditions.

6.3.1.4.5 Evaluation

Evaluate synthesized speech quality using a Mean Opinion Score (MOS) study, as in. To compare the similarity between the synthesized voice and the original speaker, we calculate the Speaker Encoder Cosine Similarity (SECS) between the speaker embeddings of two audios extracted from the speaker encoder. It ranges from -1 to 1, and a larger value indicates a stronger similarity Following previous works we compute SECS using the speaker encoder of the Resemble package, allowing for comparison with those studies. We also report the Similarity MOS (Sim-MOS) following the works of and Although the experiments involve 3 languages, due to the high cost of the MOS metrics, only two languages were used to compute such metrics: English, which has the largest number of speakers, and Portuguese, which has the smallest number. In addition, following the work of we present such metrics only for speakers unseen during training.

The evaluation of synthesized speech quality was conducted through a Mean Opinion Score (MOS) study and the calculation of Speaker Encoder Cosine Similarity (SECS) to measure the similarity between synthesized voices and original speakers. Additionally, Similarity MOS (Sim-MOS) was computed following previous works.

Due to the high cost of MOS metrics, evaluations were performed using two languages: English and Portuguese, with metrics calculated only for speakers unseen during training.

MOS scores were obtained through crowdsourcing, utilizing native English contributors for English evaluations and native Portuguese contributors for Portuguese evaluations. Speaker embeddings were extracted from reference audios, and MOS, SECS, and Sim-MOS were calculated based on synthesized sentences.

Table: SECS, MOS and Sim-MOS with 95% confidence intervals for all our experiments

	VCTK			LIBRITTS			MLS-PT		
EXP.	SECS	MOS	SIM-MOS	SECS	MOS	SIM-MOS	SECS	MOS	SIM-MOS
GROUND TRUTH	0.824	4.26±0.04	4.19±0.06	0.931	4.22±0.05	4.22±0.06	0.9018	4.61±0.05	4.41±0.05
ATTENTRON ZS	(0.731)	(3.86±0.05)	(3.30 ±0.06)	–	–	–	–	–	–
SC-GLOWTTS	(0.804)	(3.78±0.07)	(3.99±0.07)	–	–	–	–	–	–
EXP. 1	**0.864**	4.21±0.04	4.16±0.05	0.754	**4.25±0.05**	3.98±0.07	–	–	–
EXP. 1 + SCL	0.861	4.20±0.05	4.13±0.06	0.765	4.21±0.04	4.05±0.07	–	–	–
EXP. 2	0.857	**4.24±0.04**	4.15±0.06	0.762	4.22±0.05	4.01±0.07	0.740	3.96±0.08	3.02±0.1
EXP. 2 + SCL	**0.864**	4.19±0.05	**4.17±0.06**	0.773	4.23±0.05	4.01±0.07	0.745	4.09±0.07	2.98±0.1
EXP. 3	0.851	4.21±0.04	4.10±0.06	0.761	4.21±0.04	4.01±0.05	0.761	4.01±0.08	**3.19±0.1**
EXP. 3 + SCL	0.855	4.22±0.05	4.06±0.06	0.778	4.17±0.05	3.98±0.07	0.766	**4.11±0.07**	3.17±0.1
EXP. 4 + SCL	0.843	4.23±0.05	4.10±0.06	**0.856**	4.18±0.05	**4.07±0.07**	**0.798**	3.97±0.08	3.07±0.1

Figure 16: Your TTS Evaluation

6.3.2 STS(Speech-To-Speech)

6.3.2.1 VITS Model

6.3.2.1.1 Abstract

The TTS model we implemented, known as Variational Inference with Adversarial Learning for End-to-End Text-to-Speech (VITS), effectively integrates Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) to generate high-quality, natural-sounding speech from text inputs. The prominent features of this model include:

- Variational Inference:

This component is pivotal in modeling the uncertainty within latent variables, facilitating the generation of diverse speech patterns from identical text inputs.

Variational inference, as described in the context of the VAE framework by Kingma and Welling (2013), employs a stochastic gradient ascent method for scalable posterior inference in large datasets. This approach leverages reparameterization and inference networks to achieve a highly scalable learning procedure, enhancing the flexibility and expressiveness of the posterior distribution through techniques like normalizing flows.

- Adversarial Learning:

GANs play a crucial role in this system by refining the realism of the generated speech waveforms.

The adversarial component, introduced by Goodfellow et al. (2014), involves training a generator to produce realistic outputs that a discriminator attempts to distinguish from real data, thereby pushing the generator towards producing highly realistic samples.

- Stochastic Duration Predictor:

This component captures the variability in speech duration, effectively modeling the natural rhythm and timing variations found in human speech.

The predictor addresses the inherent stochastic nature of speech, ensuring that the generated speech maintains a natural and fluent cadence.

Theoretical Foundation

The VITS model also draws on advanced concepts in variational inference, particularly the Inverse Autoregressive Flow (IAF) technique proposed by Kingma et al. (2016). This technique enhances the flexibility of the posterior distribution by applying a sequence of invertible transformations, each parameterized by autoregressive neural networks. The key insights from the IAF methodology include:

Improved Variational Inference with IAF: IAF leverages autoregressive neural density estimators, such as RNNs, MADE, Pixel CNN, and WaveNet models, to transform a simple initial random variable into a complex distribution with a computable probability density.

The transformation is parallelizable and maintains a simple Jacobian determinant, making it suitable for high-dimensional latent spaces.

- Normalizing Flows:

Normalizing flows iteratively transform an initial variable through a chain of invertible transformations, each step enhancing the flexibility and expressiveness of the resulting distribution.

This process allows for efficient computation of probability densities and sampling, critical for scalable and effective variational inference in complex models.

- Parallelizability and Computational Efficiency:

The IAF methodology emphasizes the importance of parallelizable operations across the dimensions of the latent space, significantly improving computational efficiency.

This ensures that the transformations maintain tractable density computations, crucial for large-scale applications like end-to-end TTS.

6.3.2.1.2 Model Architecture

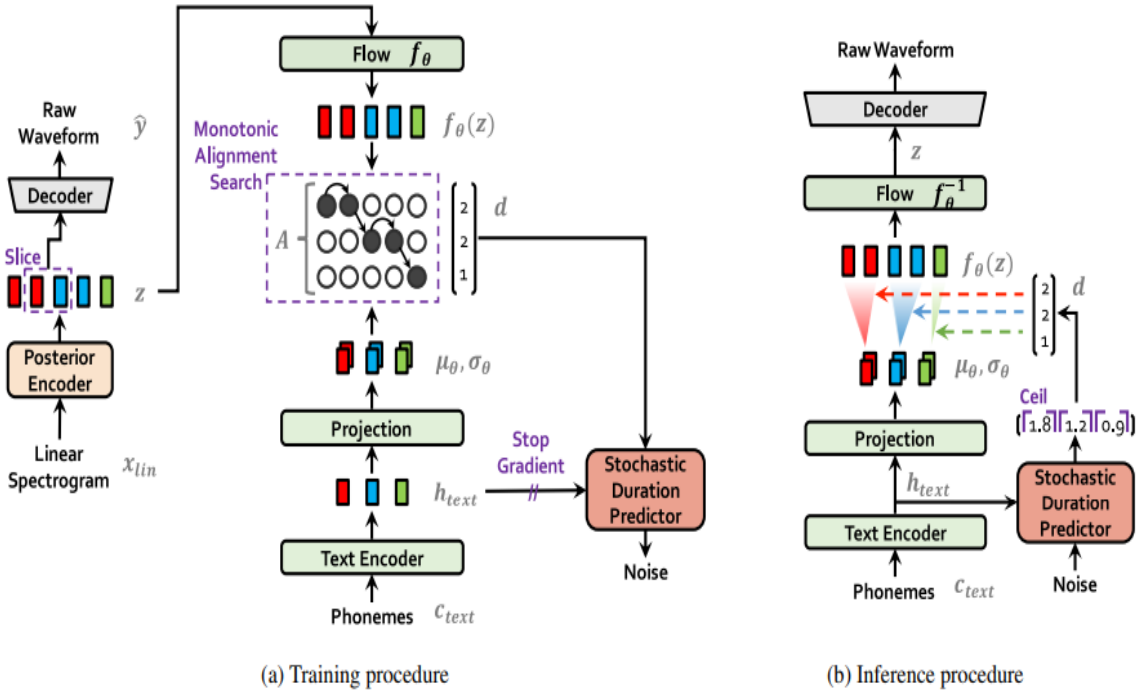

Figure 17: VITS Architecture

The overall architecture of the VITS model consists of several interconnected components:

- Posterior Encoder: For the posterior encoder, we use the non-causal WaveNet residual blocks used in Wave Glow and Glow-TTS. A WaveNet residual block consists of layers of dilated convolutions with a gated activation unit and skip connection. The linear projection layer above the blocks produces the mean and variance of the normal posterior distribution. For the multi-speaker case, we use global conditioning in residual blocks to add speaker embedding.
- Prior Encoder: The prior encoder consists of a text encoder that processes the input phonemes and a normalizing flow that improves the flexibility of the prior distribution. The text encoder is a transformer encoder that uses relative positional representation instead of absolute positional encoding. We can obtain the hidden representation through the text encoder and a linear projection layer above the text encoder that produces the mean and variance used for constructing the prior distribution. The normalizing flow is a stack of affine coupling layers consisting of a stack of WaveNet residual blocks. For simplicity, we design. the normalizing flow to be a volume-preserving transformation with the Jacobian determinant of one. For the multispeaker setting, we add speaker embedding to the residual blocks in the normalizing flow through global conditioning.

- Decoder: Based on the HiFi-GAN generator, it up samples the latent variables to produce the final speech waveform.
- Discriminator: A multi-period discriminator that evaluates the realism of the generated waveforms.
- Stochastic Duration Predictor: The stochastic duration predictor estimates the distribution of phoneme duration from a conditional input. For the efficient parameterization of the stochastic duration predictor, we stack residual blocks with dilated and depth-separable convolutional layers. We also apply neural spline flows, which take the form of invertible nonlinear transformations by using monotonic rational-quadratic splines, to coupling layers. Neural spline Flows improve transformation expressiveness with a similar number of parameters compared to commonly used affine coupling layers. For the multi-speaker setting, we add a linear layer that transforms speaker embedding.

6.3.2.1.3 Dataset

We trained our model using the LJ Speech dataset, which contains approximately 24 hours of speech from a single speaker. The audio was converted to 16-bit PCM format with a sample rate of 22 kHz. Preprocessing steps included converting text to International Phonetic Alphabet (IPA) sequences and generating Mel-spectrograms from raw waveforms using Short-time Fourier Transform (STFT).

6.3.2.1.4 Experiment

The training process involves optimizing the combined loss function, which includes the reconstruction loss, KL-divergence, adversarial loss, and feature matching loss. We used the Adam optimizer and trained the model with mixed precision on multiple GPUs. The training data consisted of preprocessed text and corresponding speech waveforms, with mel-spectrograms used for reconstruction loss computation.

6.3.2.1.5 Evaluation

- Speech Synthesis Quality

We conducted crowd-sourced Mean Opinion Score (MOS) tests to evaluate the quality of generated speech. The MOS tests involved listeners rating the naturalness of audio samples on a scale from 1 to 5. The results demonstrated that VITS outperformed other TTS systems and achieved a similar MOS to ground truth.

Model	MOS (CI)
Ground Truth	4.46 (±0.06)
Tacotron 2 + HiFi-GAN	3.77 (±0.08)
Tacotron 2 + HiFi-GAN (Fine-tuned)	4.25 (±0.07)
Glow-TTS + HiFi-GAN	4.14 (±0.07)
Glow-TTS + HiFi-GAN (Fine-tuned)	4.32 (±0.07)
VITS (DDP)	4.39 (±0.06)
VITS	4.43 (±0.06)

Table 7: VITS Evaluation

The results indicate that the stochastic duration predictor in VITS generates more realistic phoneme durations compared to the deterministic duration predictor. Additionally, the end-to-end training method enhances sample quality.

- Ablation Study

An ablation study was conducted to evaluate the effectiveness of different model components. The study showed that removing the normalizing flow in the prior encoder significantly decreased the MOS, highlighting the importance of prior distribution flexibility. Replacing the linear-scale spectrogram with the Mel-spectrogram also resulted in quality degradation.

Model	MOS (CI)
Ground Truth	4.50 (±0.06)
Baseline	4.50 (±0.06)
without Normalizing Flow	2.98 (±0.08)
with Mel-spectrogram	4.31 (±0.08)

Table 8: Ground Truth, Baseline, without Normalizing Flow, and with Mel-spectrogram Evaluation

- Generalization to Multi-Speaker Text-to-Speech

To verify the model's ability to learn and express diverse speech characteristics, we compared it with other models on the VCTK dataset. The evaluation showed that VITS achieved higher MOS, demonstrating its capability to handle multi-speaker TTS.

Model	MOS (CI)
Ground Truth	4.38 (±0.07)
VITS	4.38 (±0.06)

Table 9: Ground Truth and VITS Evaluation

These findings demonstrate that the VITS model effectively synthesizes high-quality, natural-sounding speech and generalizes well to different speakers, making it a robust choice for deepfake audio generation.

6.3.2.2 Freevc Model

6.3.2.2.1 Abstract

Voice conversion (VC) can be achieved by first extracting source content information and target speaker information, and then reconstructing waveform with this information. However, current approaches normally either extract dirty content information with speaker information leaked in or demand a large amount of annotated data for training. Besides, the quality of reconstructed waveform can be degraded by the mismatch between conversion model and vocoder. In this paper, we adopt the end-to-end framework of VITS for high-quality waveform reconstruction and propose strategies for clean content information extraction without text annotation. We disentangle content information by imposing an information bottleneck to Wav LM features and propose the spectrogram-resize based data augmentation to improve the purity of extracted content information. Experimental results show that the proposed method outperforms the latest VC models trained with annotated data and has greater robustness. Index Terms— voice conversion, self-supervised learning, information bottleneck, data augmentation.

6.3.2.2.2 Model Architecture

Free VC leverages several components to achieve voice conversion. This section delves into the key modules:

- Prior Encoder: Responsible for extracting content information from the source waveform.
- Posterior Encoder (follows VITS): Analyzes the data for training purposes (details omitted for brevity).
- Decoder (follows VITS): Reconstructs the waveform based on extracted information (details omitted).
- Discriminator (follows VITS): Evaluates the generated waveform's quality during training (details omitted).
- Speaker Encoder: Captures speaker identity for voice conversion.

We'll focus on the unique aspects of FreeVC's architecture the Prior Encoder and Speaker Encoder:

- Prior Encoder

The prior encoder aims to extract content information, independent of the speaker, from the raw waveform. Here's a breakdown of its components:

1. WavLM Model: This pre-trained model processes the raw waveform and generates a 1024-dimensional representation (xssl) containing both content and speaker information.
2. Bottleneck Extractor: This component acts as an information bottleneck. It takes the xssl representation and reduces its dimensionality to d (much smaller than 1024). This process forces the model to discard speaker information and focus on content.
3. Normalizing Flow: This step refines the low-dimensional representation by incorporating speaker information encoded in the g (speaker embedding). It utilizes affine coupling layers to improve the complexity of the prior distribution.

- Speaker Encoder

FreeVC offers two options for the speaker encoder:

1. Pre-trained Speaker Encoder (Preferred): This leverages a pre-trained speaker verification model for superior performance. It's trained on vast speaker datasets and is widely used in voice conversion tasks.
2. Non-pretrained Speaker Encoder: This option trains the speaker encoder alongside the rest of the model from scratch. It employs a simpler Long Short-Term Memory (LSTM) architecture. The underlying assumption is that a clean content representation allows the speaker encoder to effectively learn missing speaker information during training.

6.3.2.2.3 Dataset

The experiments were conducted on two primary datasets: VCTK and LibriTTS.

The VCTK corpus was exclusively used for training, comprising data from 107 speakers. Among these speakers, 314 utterances (2 sentences per speaker) were randomly selected for validation, 10,700 utterances (10 sentences per speaker) for testing, and the remaining data for training purposes. The LibriTTS dataset's test-clean subset was utilized solely for testing purposes.

6.3.2.2.4 Experiments

1. Preprocessing

All audio samples underwent down sampling to 16 kHz. Subsequently, linear spectrograms and 80-band Mel-spectrograms were calculated using short-time Fourier transform (FFT). The parameters for FFT, window, and hop size were set to 1280, 1280, and 320, respectively. Additionally, a dimension of 192 was set for the bottleneck extractor.

2. Augmentation and Vocoder

For data augmentation, a Speech Reconstruction (SR)-based approach was employed, with a resize ratio (r) ranging from 0.85 to 1.15. Post-augmentation, a HiFi-GAN v1 vocoder [25] was utilized to transform the modified Mel-spectrogram into waveform.

3. Training Configuration

The training of models was executed for up to 900k steps on a single NVIDIA 3090 GPU. A batch size of 64 with a maximum segment length of 128 frames was employed.

4. Model Variants

Three versions of the proposed method were tested:

1. FreeVC-s: The proposed model employing a non-pretrained speaker encoder.
2. FreeVC: The proposed model incorporating a pretrained speaker encoder.
3. FreeVC (w/o SR): The proposed model utilizing a pretrained speaker encoder but trained without SR-based data augmentation.

6.3.2.2.5 Evaluation

1 Subjective Evaluation

Fifteen participants were invited to evaluate the naturalness and speaker similarity of the generated speech. Evaluation was conducted using a 5-scale Mean Opinion Score (MOS) and Similarity Mean Opinion Score (SMOS). Evaluation scenarios encompassed seen-to-seen, unseen-to-seen, and unseen-to-unseen speaker scenarios.

2 Objective Evaluation

Objective evaluation was conducted using three metrics:

- Word Error Rate (WER) and Character Error Rate (CER): Calculated between source and converted speech utilizing an Automatic Speech Recognition (ASR) model.
- F0-PCC (Pitch Correlation Coefficient): Indicates the Pearson correlation coefficient between the pitch (F0) of source and converted speech.

3.1 Speech Naturalness and Speaker Similarity

The MOS and SMOS results demonstrated the superiority of the proposed models over baseline models across all evaluation scenarios in terms of both speech naturalness and speaker similarity. Notably, the proposed models exhibited robustness, particularly evident when source speech quality was low.

3.2 Speech Intelligence and F0 Variation Consistency

The proposed models achieved lower WER and CER compared to baseline models, indicating proficient preservation of linguistic content. Additionally, higher F0 variation consistency was observed, showcasing the effectiveness of the proposed method in maintaining prosody.

In conclusion, the YourTTS generation model, particularly the FreeVC variant, displayed promising results in deepfake audio generation, outperforming baseline models in various evaluation metrics and demonstrating robustness across different scenarios

Chapter 7: Conclusion

The FakeFinder project is a major advancement in tackling the challenges of deepfake audio. By using advanced machine learning models and a solid software framework, it offers a practical and effective solution for detecting and generating deepfake audio. The project's careful evaluation and successful use of various detection models highlight its contribution to improving the security and integrity of audio communications.

For deepfake detection, we implemented and chose the Light Convolutional Neural Network (LCNN) model with Linear-Frequency Cepstral Coefficient (LFCC) features. This combination proved to be the most effective due to its high precision, robust performance, and stability across different types of audio manipulations. For audio generation, we used Text-to-Speech (TTS) and Speech-to-Speech (STS) models, selected for their ability to produce high-quality synthetic audio and their flexibility in generating various types of deepfake audio.

We chose to develop FakeFinder as an Android application to leverage the widespread use of Android devices. Android's vast user base and flexible development environment make it an ideal platform for reaching a broad audience and providing a user-friendly interface. Making it an app ensures accessibility, allowing users to easily detect and generate deepfake audio on-the-go.

This project not only shows that AI can effectively combat deepfake threats but also paves the way for future improvements in this area. By integrating cutting-edge technology with practical applications, FakeFinder sets a strong foundation for enhancing the security and integrity of audio communications.

Chapter 8: Future Work

In this chapter, we will outline potential directions for future work and enhancements, building upon the foundations laid by this project. The proposed future work involves several innovative aspects that aim to expand the capabilities and applications of the developed system.

1. Models Generation for TTS and STS:

One significant area for future work is the development and enhancement of models for Text-to-Speech (TTS) and Speech-to-Text (STS) systems. These models are crucial for a wide range of applications, including virtual assistants, accessibility tools, and interactive gaming environments. Future efforts should focus on:

Advanced Model Architectures: Exploring and implementing advanced neural network architectures such as Transformer-based models (e.g., Tacotron 2) to improve the quality and efficiency of TTS and STS systems.

Language-Specific Enhancements: Tailoring models specifically for the Arabic language, addressing its unique phonetic and syntactic characteristics to enhance accuracy and naturalness.

Multimodal Integration: Developing systems that can integrate text and speech seamlessly, enabling more sophisticated interactions and more robust performance in diverse scenarios.

2. Extension to All Platforms:

Another critical aspect of future work is extending the developed models and systems to be compatible across various platforms, including web, mobile, and desktop environments. This extension will make the technology more accessible and user-friendly. Key areas of focus include:

Cross-Platform Compatibility: Ensuring that the models and applications are optimized for different operating systems and devices (iOS, Android, Windows, macOS, Linux).

Scalability and Performance: Enhancing the scalability of the systems to handle a large number of concurrent users and ensuring that performance remains robust across different platforms.

3. Integration with Gaming:

Integrating TTS and STS systems into gaming platforms offers an exciting opportunity to create more immersive and interactive gaming experiences. This integration could enable features such as:

Voice-Controlled Gameplay: Allowing players to control game actions and interact with characters using voice commands.

Dynamic Narration: Utilizing TTS to provide real-time narration and character dialogue, making games more engaging and accessible.

Enhanced Multiplayer Communication: Improving communication between players in multiplayer games through accurate and real-time STS systems.

4. Real-Time Processing:

Achieving real-time processing capabilities is essential for the practical deployment of TTS and STS systems in interactive applications. Future work should aim to:

Latency Reduction: Implementing optimizations to reduce latency in processing and response times, ensuring that interactions feel instantaneous and natural.

Efficient Resource Utilization: Developing models that can perform efficiently on devices with limited computational resources without compromising performance.

5. Arabic Dataset Utilization:

To support the above advancements, the utilization of comprehensive Arabic datasets is crucial. Future work should focus on:

Dataset Expansion: Collecting and curating large-scale, high-quality Arabic speech and text datasets to train more robust and accurate models.

Dialectal Variations: Addressing the variations in Arabic dialects to ensure that the models can understand and generate speech across different regions and contexts.

Open Dataset Initiatives: Contributing to open dataset initiatives to foster collaboration and accelerate progress in Arabic language processing technologies.

6. Commercial Future Work:

To maximize the impact of this technology, future work should include commercial applications, such as:

Product Development: Transforming the developed models and systems into a product available on app stores, making it accessible to a broader audience and enabling practical use cases.

7. Comprehensive Deepfake Detection:

Building on FakeFinder's capabilities, future work should aim to create a comprehensive deepfake detection system for text, images, and audio. This involves:

Generalization of Detection Models: Enhancing the generalization capabilities of detection models as more diverse model-generated datasets become available.

Arabic Deepfake Generation: Focusing on generating and detecting deepfakes specifically for the Arabic language to address unique challenges and improve detection accuracy in this context.

By pursuing these future work directions, we can significantly advance the state of TTS, STS, and deepfake detection technologies, especially in the context of the Arabic language and across various application domains.

Chapter 9: References

[1] Kawa, P., Plata, M., & Syga, P. (2022). Attack Agnostic Dataset: Towards Generalization and Stabilization of Audio DeepFake Detection.

[2] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[3] & Yamagishi, J. (2021). A comparative study on recent neural spoofing countermeasures for synthetic speech detection.

[4] Afchar, D., Nozick, V., Yamagishi, J., & Echizen, I. (2018). Mesonet: a compact facial video forgery detection network. In 2018 IEEE International Workshop on Information Forensics and Security (WIFS) (pp. 1–7).

[5] Tak, H., Patino, J., Todisco, M., Nautsch, A., Evans, N., & Larcher, A. (2021). End-to-end anti-spoofing with rawnet2. In ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

[6] Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Saurous, R. A., Agiomyrgiannakis, Y., & Wu, Y. (2017). Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions.

[7] Kong, J., Kim, J., & Bae, J. (2020). HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis.

[8] Kumar, K., Kumar, R., De Boissiere, T., Gestin, L., Teoh, W. Z., Sotelo, J., De Brebisson, A., Bengio, Y., & Courville, A. (2019). MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis.

[9] Casanova, E., Weber, J., Shulby, C., Junior, A. C., Gölge, E., & Ponti, M. A. (2021). YourTTS: Towards Zero-Shot Multi-Speaker TTS and Zero-Shot Voice Conversion for everyone.

[10] Kim, J., Kong, J., & Son, J. (2021). Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech.

[11] Wang, Y., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q., Agiomyrgiannakis, Y., Clark, R., & Saurous, R. A. (2017). Tacotron: Towards End-to-End Speech Synthesis.

[12] Li, J., Tu, W., & Xiao, L. (2022). FreeVC: Towards High-Quality Text-Free One-Shot Voice Conversion.

[13] Choi, H., Kim, J., Huh, J., Kim, A., Ha, J., & Lee, K. (2019). Phase-aware Speech Enhancement with Deep Complex U-Net.