ADDRESSING VOICE-BASED FRAUD WITH AI

Combating Identity Personification through AI-Generated Voice Prints

Authors

Krishna Sabbu
Padma Naresh
Chandra Sekhar Javalkar
Naresh Palabatla
Sai Tejaswi
Ravi Kumar

Introduction

Purpose of the Document

This document serves as a comprehensive guide outlining the approach and strategies to address the challenge of identity personification using AI-generated voice prints within the banking domain.

Its primary purpose is to provide a clear roadmap for developing effective countermeasures to safeguard customers and institutions against voice-based fraud.

Overview of the Problem to be Solved

In the banking domain, the threat of identity personification through AI-generated voice prints poses significant risks, potentially leading to financial losses for both customers and banks alike. As AI technology advances, the ability to mimic human voices with remarkable accuracy has become increasingly prevalent, making it easier for malicious actors to impersonate legitimate customers. The challenge at hand is to develop robust solutions capable of detecting and preventing such fraudulent activities.

Goals and Objectives of the Project

The overarching goal of this project is to develop innovative countermeasures that can effectively combat identity personification through AI-generated voice prints. To achieve this goal, the following objectives have been established

Objective/Task 1: AI-Based Voice Verification

Develop AI algorithms capable of distinguishing between authentic and AI-generated voice prints. This task aims to create sophisticated algorithms capable of accurately identifying subtle differences between genuine human voices and those generated by AI systems.

Objective/Task 2: Speech Pattern Analysis

Analyse speech patterns, intonation, and other subtle vocal cues to identify anomalies indicative of AI-generated speech. This objective focuses on leveraging advanced analysis techniques to detect deviations from natural human speech, which may indicate fraudulent activity.

Objective/Task 3: Liveness Detection

Incorporate liveness detection techniques to ensure that voice samples are produced by a live person and not a recording or AI simulation. This task aims to enhance the security of voice authentication systems by verifying the presence of a live individual during voice authentication attempts.

Additional Details on Functionality

The evaluation of countermeasures against identity personification using AI-generated voice prints will be based on the following criteria

Accuracy: The ability to correctly identify AI-generated voice prints and distinguish them from authentic voice prints.

Generalizability: The system's ability to detect AI-generated voice prints across various voice domains, accents, and speaking styles.

Robustness: The system's resilience against adversarial techniques and its ability to adapt to evolving AI-generated voice prints.

By addressing these objectives and criteria, the project aims to significantly mitigate the risks associated with voice-based fraud within the banking sector, ultimately enhancing security and trust for customers and institutions alike.

Problem Statement

Description of the Problem to be Addressed

The challenge at hand is to develop robust and effective countermeasures against identity personification using AI-generated voice prints. With the advancement of artificial intelligence technology, there has been a significant increase in the threat of voice-based fraud. Malicious actors can exploit AI capabilities to mimic human voices with remarkable accuracy, posing a substantial risk to individuals and organizations alike. The problem statement revolves around detecting and preventing unauthorized access and impersonation facilitated by AI-generated voice prints.

Scope of the Project

The scope of the project entails the development of a REST API designed to analyse audio samples and determine the authenticity of the voice detected. The API will accept audio samples as input and provide an analysis of the detected voice, including attributes such as voice type, confidence scores, emotional tone, and background noise level. The primary goal is to create a scalable and efficient solution capable of accurately differentiating between human and AI-generated voices in real-time.

Request Format:

```
{
"sample": "audio.mp3"
}
```

The request consists of an audio file encoded in the MP3 and WAV format, representing the voice sample to be analysed.

Expected Response:

```
{
  "status": "success",
  "analysis": {
    "detectedVoice": true,
    "voiceType": "human",
    "confidenceScore": {
      "aiProbability": 5,
      "humanProbability": 95
    },
    "additionalInfo": {
      "emotionalTone": "neutral",
      "backgroundNoiseLevel": "low"
    }
  },
  "responseTime": 200
}
```

The response contains an analysis of the detected voice, including whether it is identified as human or AI-generated, confidence scores indicating the probability of each classification, and additional information such as emotional tone and background noise level. The response time indicates the duration taken to process the request and generate the response.

Stakeholders Involved

Development Team: Responsible for designing, implementing, and testing the REST API for voice analysis.

Project Managers: Oversee the project's progress, allocate resources, and ensure adherence to timelines and objectives.

Quality Assurance Team: Conduct testing to validate the functionality and performance of the REST API.

End Users: Individuals and organizations seeking to integrate voice analysis capabilities into their applications for enhanced security against identity personification and fraud.

Overall Approach

High-level Description of the Approach to Solving the Problem

The approach to solving the problem of identity personification using AI-generated voice prints involves training an AI model on large datasets using various classifiers such as RandomForestClassifier, Sequential classifier, etc. The trained model will be capable of distinguishing between human and AI-generated voices, detecting emotional tones (e.g., Angry, Disgust, Neutral, Happy), identifying spoken languages (e.g., English, Spanish, Arabic), and detecting music within audio samples. Once the model is trained, it will be saved to disk and utilized for prediction tasks.

Overview of the System Architecture

The system architecture consists of two main high-level components: training and prediction. During the training phase, Python scripts utilizing TensorFlow, Scikit-learn, and LSTM (Long Short-Term Memory) neural networks will be employed to train the AI model on large datasets. The trained model will then be saved to disk for future use. In the prediction phase, the saved model will be loaded and utilized to predict various attributes such as voice type, emotional tone, spoken language, and music detection from audio samples.

Technologies and Tools to be used

Technologies:

Python: Used for scripting and implementation of machine learning algorithms. TensorFlow, Scikit-learn: Machine learning libraries utilized for building and training AI models.

Google Cloud Vertex AI: Platform for training and deploying machine learning models at scale.

Jupyter Labs: Interactive development environment for data science and machine learning tasks.

Google Cloud Run: Serverless platform for deploying and scaling containerized applications.

Docker: Containerization technology used for packaging the application and its dependencies.

Tools:

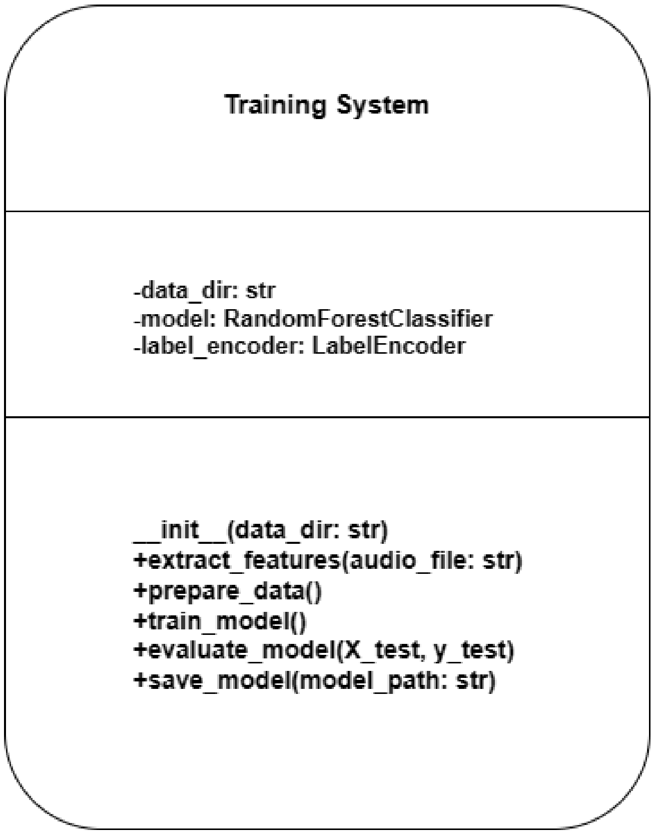
PyCharm: Integrated development environment (IDE) for Python development, providing features for code editing, debugging, and version control.

Streamlit: Open-source framework for building interactive web applications for data science and machine learning tasks.

This comprehensive set of technologies and tools will facilitate the development, training, deployment, and management of the AI model for voice analysis, ensuring scalability, efficiency, and accuracy in detecting identity personification through AI-generated voice prints.

Class Diagrams

Training Phase Class Diagram:


Training System
-data_dir: str
-model: RandomForestClassifier
-label_encoder: LabelEncoder
__init__(data_dir: str)
+extract_features(audio_file: str)
+prepare_data()
+train_model()
+evaluate_model(X_test, y_test)
+save_model(model_path: str)

This class diagram outlines the structure of the TrainingSystem class, along with its attributes and methods for training the AI model. It encapsulates the functionality for data preparation, model training, evaluation, and model saving.

Attributes:

-data_dir: str: Directory containing audio files for each language.

-model: RandomForestClassifier: The trained model instance.
-label_encoder: LabelEncoder: Encoder for encoding language labels.

Methods:

+__init__(data_dir: str): Initializes the TrainingSystem with the provided data directory.

+extract_features(audio_file: str): Extracts features from the audio file.

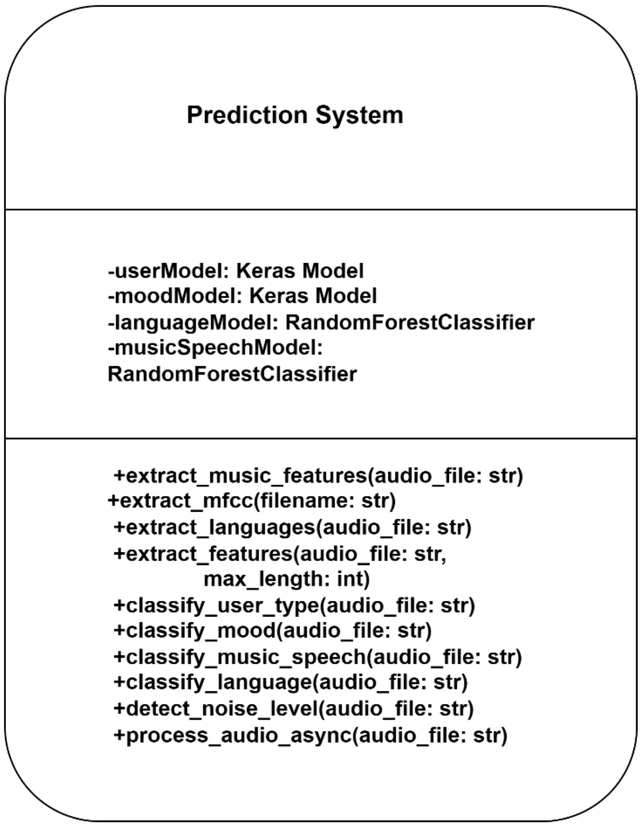
+prepare_data(): Prepares the data by extracting features and labels from audio files.

+train_model(): Trains the AI model using the prepared data.

+evaluate_model(X_test, y_test): Evaluates the trained model using the test data.

+save_model(model_path: str): Saves the trained model to the specified path.

Prediction Phase Class Diagram:


Prediction System
-userModel: Keras Model
-moodModel: Keras Model
-languageModel: RandomForestClassifier
-musicSpeechModel: RandomForestClassifier
+extract_music_features(audio_file: str)
+extract_mfcc(filename: str)
+extract_languages(audio_file: str)
+extract_features(audio_file: str, max_length: int)
+classify_user_type(audio_file: str)
+classify_mood(audio_file: str)
+classify_music_speech(audio_file: str)
+classify_language(audio_file: str)
+detect_noise_level(audio_file: str)
+process_audio_async(audio_file: str)

This class diagram outlines the structure of the PredictionSystem class, along with its attributes and methods for processing audio files and generating predictions for user type, mood, language, and audio type. It encapsulates the functionality for feature extraction, model prediction, and result aggregation

Attributes:

-userModel: Keras Model: Pre-trained model for classifying user type (AI or human).

-moodModel: Keras Model: Pre-trained model for classifying mood/emotional tone.

-languageModel: RandomForestClassifier: Pre-trained model for language identification.

-musicSpeechModel: RandomForestClassifier: Pre-trained model for classifying audio type (music or speech).

Methods:

+extract_music_features(audio_file: str): Extracts music features from the audio file.

+extract_mfcc(filename: str): Extracts MFCC features from an audio clip.

+extract_languages(audio_file: str): Extracts language features from the audio file.

+extract_features(audio_file: str, max_length: int): Extracts features from the audio file with a specified maximum length.

+classify_user_type(audio_file: str): Classifies the user type (AI or human) based on the audio file.

+classify_mood(audio_file: str): Classifies the mood/emotional tone based on the audio file.

+classify_music_speech(audio_file: str): Classifies the audio type (music or speech) based on the audio file.

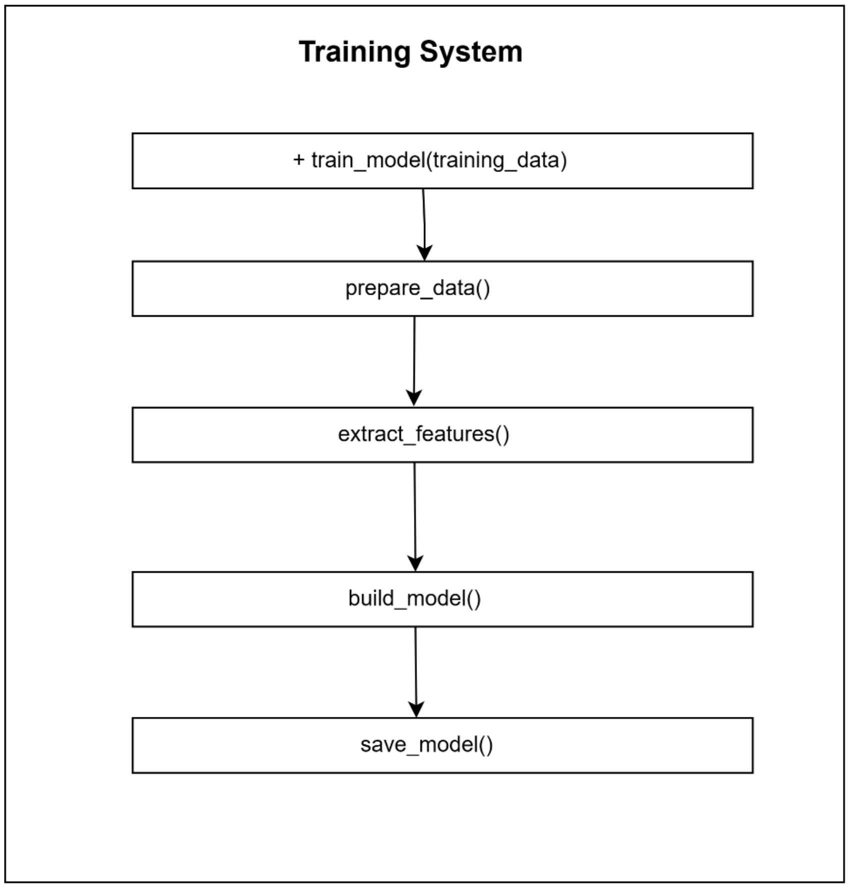
+classify_language(audio_file: str): Classifies the language based on the audio file.

+detect_noise_level(audio_file: str): Detects the noise level in the audio file.

+process_audio_async(audio_file: str): Processes the audio file asynchronously and generates combined JSON with predictions.

Sequence Diagrams

This sequence diagram illustrates the flow of control among objects/modules within the Training System as it prepares data, builds, and trains the AI model during the training phase.



Explanation:

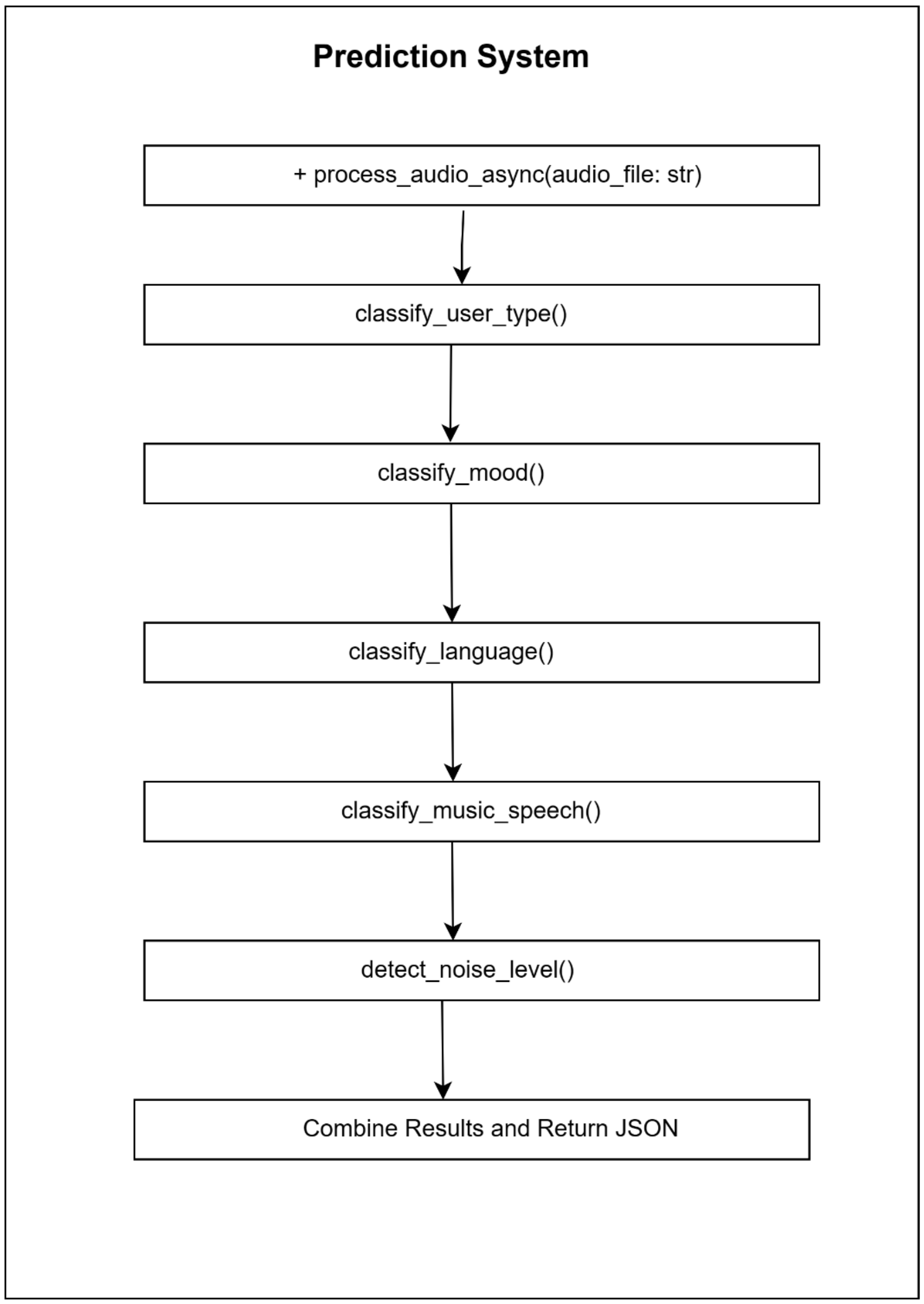
The process starts when the train_model() method is called.

Inside the train_model() method, data preparation is initiated by calling the prepare_data() method.

The prepare_data() method extracts features from audio files by calling the extract_features() method.
Once the data is prepared, the build_model() method is invoked to construct the architecture of the AI model.

After the model is built and trained, the save_model() method is called to save the trained model to disk.

This sequence diagram illustrates the flow of control among objects/modules within the Prediction System as it processes an audio file and generates predictions for user type, mood, language, audio type, and noise level.



Prediction System
+ process_audio_async(audio_file: str)
classify_user_type()
classify_mood()
classify_language()
classify_music_speech()
detect_noise_level()
Combine Results and Return JSON

Explanation:

The process starts when an audio file is received by the Prediction System.
The Prediction System asynchronously processes the audio file by calling the process_audio_async() method.
Inside the process_audio_async() method, various classification tasks are executed concurrently:
classify_user_type(): Classifies the user type (AI or human).
classify_mood(): Classifies the mood/emotional tone.
classify_language(): Classifies the language.
classify_music_speech(): Classifies the audio type (music or speech).
detect_noise_level(): Detects the noise level.
Each classification task returns its respective result.
The results from all classification tasks are combined into a single JSON object.
The combined JSON object containing all the predictions is returned as the final result.

Traceability Matrix

This traceability matrix provides a comprehensive mapping of requirements to design components for both the training and prediction systems. It ensures that each requirement is addressed by the corresponding design component(s) in the respective systems.

Training System Traceability Matrix:

Requirement	Design Component(s)
Ability to extract features from audio files	Training System, extract_features() method
Ability to prepare data for training	Training System, prepare_data() method
Ability to build AI model architecture	Training System, build_model() method
Ability to train AI model	Training System, train_model() method
Ability to save trained model	Training System, save_model() method

Prediction System Traceability Matrix:

Requirement	Design Component(s)
Ability to classify user type (AI or human)	Prediction System, classify_user_type() method
Ability to classify mood/emotional tone	Prediction System, classify_mood() method
Ability to classify language	Prediction System, classify_language() method
Ability to classify audio type (music or speech)	Prediction System, classify_music_speech() method
Ability to detect noise level	Prediction System, detect_noise_level() method
Ability to process audio files asynchronously	Prediction System, process_audio_async() method
Ability to extract features from audio files	Prediction System, extract_features() method

Implementation Plan

Training System:

Task: Define Data Collection Strategy
Description: Determine the sources of audio data for training.

Task: Feature Extraction Implementation
Description: Implement the feature extraction method to extract features from audio files.

Task: Data Preparation
Description: Prepare the training data by organizing audio files and corresponding labels.

Task: Model Architecture Design
Description: Design the architecture of the AI model for classification.

Task: Model Training
Description: Train the AI model using the prepared training data.

Task: Model Evaluation
Description: Evaluate the trained model's performance using validation data.

Task: Model Optimization
Description: Optimize the model's hyperparameters for better performance.

Task: Model Deployment
Description: Deploy the trained model to the production environment.

Prediction System:

Task: Model Loading
Description: Load pre-trained models for user type classification, mood classification, language identification, and audio type classification.

Task: Feature Extraction Setup
Description: Set up feature extraction methods for processing incoming audio files.

Task: Asynchronous Processing Implementation
Description: Implement asynchronous processing for handling multiple audio files simultaneously.

Task: Integration Testing
Description: Perform integration testing to ensure proper interaction between different components of the prediction system.

Task: API Development
Description: Develop APIs for receiving audio files and returning predictions.

Task: Frontend Integration
Description: Integrate prediction APIs with the frontend application.

Task: Performance Optimization
Description: Optimize the prediction system for better scalability and performance.

Testing Strategy

Training System:

Overview:

The testing strategy for the Training System focuses on ensuring the quality and reliability of the AI model trained for classification tasks. It involves testing each component of the training pipeline to verify its functionality and performance.

Types of Testing:

Unit Testing:

Test individual components such as feature extraction methods, data preparation functions, and model training algorithms.
Ensure each component works correctly in isolation.
Integration Testing:

Test the integration between different components of the training pipeline.
Verify the interaction and data flow between feature extraction, data preparation, model training, and model evaluation stages.
Model Evaluation Testing:

Evaluate the trained model's performance using validation datasets.
Measure metrics such as accuracy, precision, recall, and F1-score to assess classification performance.

Tools and Techniques:

Unit Testing Frameworks: Use frameworks like pytest or unittest for writing and executing unit tests.

Mocking Libraries: Utilize libraries like unit test.mock to mock external dependencies during testing.

Data Validation Techniques: Perform data validation checks to ensure the consistency and integrity of training datasets.

Model Evaluation Metrics: Calculate evaluation metrics using libraries like scikit-learn to assess the model's performance.

Prediction System:

Overview:

The testing strategy for the Prediction System focuses on validating the functionality and reliability of the prediction APIs and asynchronous processing capabilities. It involves testing the entire end-to-end flow of processing audio files and generating predictions.

Types of Testing:

Unit Testing:

Test individual components such as feature extraction methods, prediction algorithms, and asynchronous processing functions.
Ensure each component behaves as expected and handles edge cases gracefully.
Integration Testing:

Test the integration between prediction APIs, feature extraction methods, and model loading functionalities.
Verify the interaction and data flow between different modules of the prediction system.

End-to-End Testing:

Perform end-to-end testing to validate the entire prediction workflow from receiving audio files to returning predictions.
Test various scenarios, including different audio file formats, sizes, and processing times.

Tools and Techniques:

API Testing Tools: Use tools like Postman or Swagger for testing APIs and sending requests to prediction endpoints.

Testing Frameworks: Utilize testing frameworks like pytest or unittest for writing and executing unit and integration tests.

Load Testing Tools: Conduct load testing using tools like Apache JMeter to simulate concurrent requests and assess system performance under load.

Asynchronous Testing Techniques: Implement testing techniques to verify the correctness and reliability of asynchronous processing functionalities.

Conclusion:

In conclusion, the design and implementation plan for both the Training and Prediction systems have been outlined comprehensively. The Training System is responsible for preparing data, extracting features, designing, and training AI models, while the Prediction System handles the real-time processing of audio files to generate predictions.

For the Training System, tasks such as data collection, feature extraction, model training, and deployment have been identified, with responsibilities assigned to relevant teams. Similarly, for the Prediction System, tasks including model loading, asynchronous processing implementation, API development, and

performance optimization have been delineated, along with respective responsibilities.
Resource requirements for both systems have been specified, including Machine Learning Engineers, Data Preparation Teams, Software Engineers, Quality Assurance Teams, and DevOps Teams, each with distinct roles and expertise necessary for successful implementation.

Additionally, a testing strategy has been outlined for both systems, emphasizing the importance of various types of testing such as unit testing, integration testing, and performance testing. Tools and techniques for testing will be employed to ensure the robustness, accuracy, and reliability of the systems.

Overall, with a well-defined plan, clear delineation of tasks and responsibilities, and a comprehensive testing strategy, the Training and Prediction systems are poised for successful development, deployment, and operation, contributing to the effective detection and prevention of identity personification using AI-generated voice prints in the banking domain.