Voice Biometric Authentication System – Technical Report

1. Introduction

This system implements a voice-based biometric authentication mechanism using Python. It consists of three main scripts:

- `register.py`: Registers a user's voice.
- `authenticate.py`: Authenticates a user against previously registered voice data.
- `voice_utils.py`: Provides reusable utility functions for audio processing, feature extraction, and data management.

The system utilizes Mel-frequency cepstral coefficients (MFCC) for feature extraction and cosine similarity for matching embeddings. SQLite is used for persistent storage of user voice profiles.

---

2. System Components and Workflow

2.1 Registration Module (`register.py`)

Purpose: Capture and store a user's voiceprint (audio embedding) for future authentication.

Workflow:

1. **Database Initialization**:
    - Calls `init_db()` to create the necessary SQLite schema if it does not exist.
2. **User Input**:
    - Accepts and validates a username.
    - Terminates if input is empty.
3. **Audio Capture**:

- Records a 4-second audio sample via `record_audio(user)`.

4. **Preprocessing**:
   - Loads audio data from a `.npy` file using `load_audio_from_npy()`.
   - Performs a liveness check using `is_live(audio)`. If the check fails, the process terminates.
   - Applies basic noise suppression with `denoise_audio()`.
5. **Feature Extraction**:
   - Extracts MFCC features via `extract_features()` and averages across the time axis to obtain a fixed-size embedding.
6. **Data Storage**:
   - Embedding is serialized as a float32 byte array and stored in the `users` table using `save_embedding_to_db()`.

2.2 Authentication Module (`authenticate.py`)

Purpose: Verify a user's identity by comparing a live audio sample to the registered voiceprint.

Workflow:

1. **Database Initialization**:
   - Initializes the SQLite database.
2. **User Identification**:
   - Prompts for a username.
   - Loads the corresponding embedding using `load_embedding_from_db()`.
3. **Audio Capture and Preprocessing**:
   - Records a 4-second voice sample using `record_audio()`.

    - Applies the same preprocessing pipeline: liveness check → denoising → feature extraction.

4. **Similarity Computation**:
    - Uses `compare_embeddings()` to compute the cosine similarity between the stored and current embeddings.
5. **Decision Threshold**:
    - The user is authenticated if the similarity score ≥ 0.75.
6. **Visualization**:
    - Optionally renders a bar chart using `plot_score()` to indicate the similarity score.

2.3 Utility Module (`voice_utils.py`)

Purpose: Provides audio processing, feature extraction, database management, and visualization functions.

Key Components:

- **Audio Handling**:
    - `record_audio(filename)`: Captures audio using the `sounddevice` library and saves it as a `.npy` file.
    - `load_audio_from_npy(filename)`: Loads a NumPy array from the given filename.
- **Feature Extraction**:
    - `extract_features(audio)`: Extracts 13-dimensional MFCC features and returns their mean vector over time.
- **Similarity Measurement**:
    - `compare_embeddings(emb1, emb2)`: Returns the cosine similarity between two vectors.
- **Noise Reduction**:

    - `denoise_audio(audio)`: Uses a simple spectrogram-based thresholding mechanism to remove low-energy components.

- **Liveness Detection**:
    - `is_live(audio)`: Evaluates signal energy and silence ratio to detect whether the recording appears to be live.

- **Database Operations**:
    - `init_db()`: Sets up a `users` table with fields `username` (TEXT) and `embedding` (BLOB).
    - `save_embedding_to_db(username, embedding)`: Stores the user's voiceprint as a float32 binary blob.
    - `load_embedding_from_db(username)`: Retrieves and decodes the stored voiceprint.

- **Visualization**:
    - `plot_score(score)`: Plots a bar chart of the similarity score, using color coding to indicate authentication success or failure.

---

3. Technical Specifications

Aspect	Specification
Sample Rate	16,000 Hz
Recording Duration	4 seconds
Audio Format	Mono, 32-bit float
Feature Vector Size	13 (MFCC)
Embedding Storage	Serialized NumPy float32 array
Similarity Metric	Cosine similarity
Authentication Threshold	0.75
Storage Backend	SQLite (local)

---

4. Evaluation

4.1 Strengths

- **Modular Architecture**: Clearly separated responsibilities across modules.
- **Lightweight**: Minimal dependencies and fast execution.
- **Easy Deployment**: No external server or cloud infrastructure required.
- **Visual Feedback**: Score plotting helps in understanding similarity metrics.

4.2 Limitations

- **Liveness Detection**:
    - Relatively primitive and could be circumvented by playback attacks.
- **Security**:
    - Embeddings are stored in plaintext without encryption or access control.
- **Noise Handling**:
    - Basic denoising approach might not perform well under varied real-world noise conditions.
- **Scalability**:
    - SQLite is not suitable for large-scale applications or concurrent user access.

---

5. Recommendations for Enhancement

- **Advanced Liveness Detection**:
    - Introduce techniques like voice spoofing detection, playback challenge-response, or deep learning-based classifiers.
- **Data Security**:

    - Encrypt stored embeddings.
    - Add user authentication for database access.
- **Improved Audio Preprocessing**:
    - Use spectral subtraction, Wiener filters, or neural denoisers for better noise resilience.
- **Web Integration**:
    - Develop a RESTful API or web interface for accessibility across devices.
- **Evaluation Metrics**:
    - Add metrics such as False Acceptance Rate (FAR) and False Rejection Rate (FRR) to benchmark system performance.

---

6. Conclusion

This system provides a functional baseline for voice-based user authentication. Its straightforward design, minimal resource requirements, and modularity make it a good candidate for prototyping or educational purposes. However, deployment in a real-world or production-grade environment would require enhancements in security, scalability, and robustness.