**Documentation**

**Overview**

**integrated\_system.py** is the main orchestrator script for a voice-based AI assistant system. It performs the following functions:

1. Records audio from the user
2. Detects emotion in the user's voice
3. Transcribes the audio to text
4. Analyzes the sentiment of the text
5. Determines the appropriate tone for the AI's response
6. Generates a response (using Gemini or a fallback)
7. Synthesizes the response into speech (using ElevenLabs or a fallback)
8. Plays the audio back to the user

The system is modular and leverages three main components:

* **voice\_emotion\_detector.py** – Voice emotion detection
* **text\_sentiment\_checker.py** – Text sentiment analysis
* **tone\_switcher.py** – Determining and managing the AI's speaking tone

**Main Flow in integrated\_system.py**

**1. Initialization**

* API keys for Gemini and ElevenLabs are loaded.
* Audio and model components are initialized.
* Helper modules (VoiceEmotionDetector, TextSentimentChecker, ToneSwitcher) are instantiated.

**2. Audio Recording**

* The system records a short audio clip from the user.

**3. Voice Emotion Detection**

* The recorded audio is passed to VoiceEmotionDetector.detect\_emotion\_from\_file.
* Returns the detected emotion (e.g., happy, sad, angry, neutral) and a confidence score.

**4. Transcription**

* The audio is transcribed to text using the Whisper model.

**5. Text Sentiment Analysis**

* The transcript is analyzed for sentiment using TextSentimentChecker.analyze\_transcript.
* Provides a sentiment label and score.

**6. Tone Selection**

* The transcript and detected emotions are fed into ToneSwitcher, which determines the best tone for the AI's response (e.g., calm, cheerful, neutral).

**7. Response Generation**

* The system uses the Gemini API (if available) to generate a conversational response. If not available, a simulated response is provided.

**8. Voice Synthesis**

* The response is synthesized into speech using the ElevenLabs API (if available), with parameters tailored to the selected tone.
* If ElevenLabs is unavailable or errors occur, a simulated voice fallback is used.

**9. Playback**

* The synthesized or simulated audio is played back to the user.

**10. Error Handling**

* Errors are caught and logged, with fallback mechanisms in place to ensure the system remains usable.

**How integrated\_system.py Uses Each Component**

**1. voice\_emotion\_detector.py**

**How it is used:**

The main script calls self.voice\_detector.detect\_emotion\_from\_file(audio\_file) after recording the user's audio.

**How it works:**

* Loads the audio file using librosa.
* Extracts features (energy, zero-crossing rate, spectral centroid, etc.).
* Applies rule-based logic to classify the emotion.
* Returns the emotion and a confidence score.

**2. text\_sentiment\_checker.py**

**How it is used:** After transcribing the user's speech, the transcript is passed to self.text\_checker.analyze\_transcript(transcript).

**How it works:**

* Analyzes the text using NLP techniques (could be rule-based, ML, or API-based).
* Returns a sentiment label (e.g., positive, negative, neutral) and a confidence score.

**3. tone\_switcher.py**

**How it is used:** The transcript and detected emotion are provided to self.tone\_switcher.update\_transcript(transcript) and get\_current\_tone().

**How it works:**

* Maintains conversational context and recent emotions/sentiments.
* Decides the most appropriate tone for the AI's next response.
* Provides parameters for voice synthesis (style, rate, pitch).

**Error Handling**

**General Approach:**

* All major steps (audio, transcription, emotion detection, sentiment analysis, response generation, voice synthesis, playback) are wrapped in try/except blocks.
* Errors are logged with descriptive messages.
* The system falls back to simulated responses or voices when external services are unavailable or fail.

**ElevenLabs-specific Handling:**

* If the ElevenLabs API key is missing, the system logs a warning and uses simulated voice output.
* If API errors occur (e.g., free tier restrictions, network issues), the error is caught and logged.
* For known ElevenLabs errors (like free tier usage disabled or unusual activity), the system disables ElevenLabs for the session and notifies the user.
* For other errors, a temporary fallback to simulated voice is used.

**Summary Table**

|  |  |  |
| --- | --- | --- |
| **Component** | **Purpose** | **How it's Used in Main Script** |
| **voice\_emotion\_detector** | Detect emotion from user's voice | Detects emotion after recording audio |
| **text\_sentiment\_checker** | Analyze sentiment of transcribed text | Analyzes transcript after speech-to-text |
| **tone\_switcher** | Decide tone for AI response | Sets parameters for voice synthesis |
| **ElevenLabs API** | Synthesize AI response into speech | Used for TTS, with robust error handling |
| **Gemini API** | Generate AI response text | Used for response, fallback if unavailable |

**Difference between Eman’s and enhanced version integration:**

The key difference between text\_sent.py (Eman’s) and text\_sentiment\_checker.py (Enhanced version) lies in their complexity, flexibility, and intended use within your project. text\_sent.py is a simple, standalone script that uses Hugging Face's transformers pipeline to perform sentiment analysis on text, returning only three possible scores: 1 for positive, -1 for negative, and 0 for neutral. It is straightforward and mainly suited for quick, basic sentiment checks or demonstrations. In contrast, text\_sentiment\_checker.py is a more robust and modular class. It not only performs sentiment analysis using a similar pipeline but also provides a confidence-weighted score between -1 and +1, includes error handling, timing control (to avoid excessive checks), and additional methods for more advanced usage.

**Conclusion**

The **integrated\_system.py** script is a robust, modular system for conversational AI with emotion and sentiment awareness. It leverages specialized modules for emotion, sentiment, and tone, and is designed to gracefully handle errors and fallback to simulated behaviors when external APIs are unavailable.