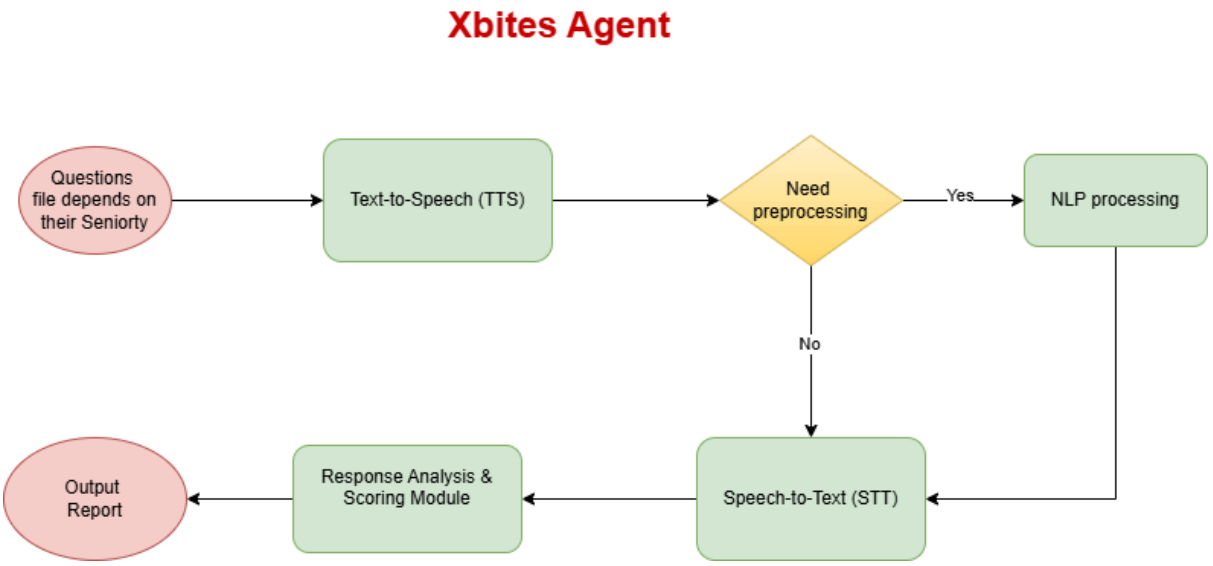
AI Agenet Interviewer

Introduction

The Audio AI Interviewer is designed to simulate a live interview environment by interacting with candidates through audio-based questions. The system evaluates candidates' responses against predefined criteria, assigns scores based on competencies, and generates a final report. This document presents the technical design and architecture of the system, detailing the tools, frameworks, and methodologies used.

End-to-End Architecture



Steps:

Step 1: Load Interview Questions from JSON File depends on seniority

- The system reads predefined questions from a JSON file.
- Each question is associated with:
    - A maximum score (e.g., 10 points).
    - A set of competencies (e.g., technical depth, clarity).
- Tools Used: `json` module in Python.
- We can insert and extract data from into json files

Step 2: Convert the Question into an Audio Recording (TTS)

- The system converts the text question into an audio file using a Text-to-Speech (TTS) model (SeamlessM4T).
- The audio is saved as a `.wav` file with a 16 kHz sample rate
- The audio started to play to the user so he could listen to the quesiton.

Step 3: Play the Question Audio and Prompt the User to Answer

- The system plays the generated audio question for the candidate.
- The candidate is then given 45 seconds to respond.

Step 4: Record the Candidate's Response

- The system records the candidate's answer using their microphone.
- The recording is saved in a standardized format (.wav, 16 kHz).

Step 5: Convert the Audio Response into Text (STT)

- The recorded answer is transcribed into text using Speech-to-Text (STT).
- The text undergoes cleaning and preprocessing to remove noise or filler words.

Description

Question Modification :

- Input: Predefined questions selected based on candidate seniority.
- Process:
    - The cleaned text is converted into an audio file using a Text-to-Speech model (SeamlessM4T).
    - Audio files are generated in a universally supported format (e.g., `.wav`) at a standard sample rate of 16 kHz.
- Output: The generated audio serves as the question to be played to the candidate.
- Tools Used:
    - `transformers` library for SeamlessM4T Text-to-Speech.
    - `scipy.io.wavfile` for saving audio files.

Candidate Response:

- Input: Candidate's recorded audio.
- Process:
    - Candidates respond to the questions by recording their answers.
    - The system supports audio inputs in all standard formats (e.g., `.wav`, `.mp3`) and normalizes them to a sample rate of 16 kHz.
- Output: The recorded audio is saved as `.wav` files for further processing.
- Tools Used:
    - `sounddevice` for recording responses.
    - `pydub` for handling and converting audio formats.
    - `torchaudio` for resampling audio to the required 16 kHz.

Response Evaluation:

- Input: Transcribed response.
- Process:
    - The recorded audio is transcribed into text using a Speech-to-Text model (SeamlessM4T).
    - The transcribed text undergoes minimal preprocessing (e.g., cleaning) before evaluation.
- Output: The system generates a cleaned text version of the candidate's response.
- Tools Used:
    - `transformers` library for SeamlessM4T Speech-to-Text.
    - `torchaudio` for audio preprocessing.
    - Python utilities for text cleaning.

Report Generation:

- Input: Evaluation scores and justifications.
- Process:
    - The transcribed response is evaluated based on predefined competencies (e.g., Technical Depth, Clarity).
    - An overall score is generated by combining competency scores.
    - Candidates are ranked based on their overall score.
- Output: A structured evaluation with justifications for each score and an overall ranking.
- Tools Used:
    - Together API (Llama) for generating evaluations.
    - A scoring prompt template (`scoring_prompt_system`) ensures consistent evaluation criteria.

Output: Report Generation

- Process:
    - Evaluation results, including question text, candidate answers, scores, and justifications, are compiled into structured reports.
    - Reports are saved as CSV files for further analysis or review.
- Output: A CSV file containing detailed evaluations and rankings.
- Tools Used:
    - pandas for data aggregation and CSV generation.

Summary of Tools and Frameworks

Component	Frameworks /tools	Purpose
Text Cleaning	**Python utilities (re) , unicodedata, Inflect**	**Preprocesses and normalizes questions**
TTS	**transformers (SeamlessM4T)**	**Converts text questions into audio.**
Audio Recording	**sounddevice, pydub**	**Records candidate responses and handles audio formats.**
STT	**transformers (SeamlessM4T), torchaudio**	**Transcribes responses into text.**
Evaluation & Score	**LLama -3.370b Together AI**	**Evaluates responses and generates scores and justifications.**
Reporting	**pandas**	**Creates structured CSV reports**

Enhancements and Upcoming Improvements

1. Expanding Evaluation Metrics

Current Limitation:

- The current system primarily evaluates **correctness** based on predefined competencies.
- It does not consider **fluency, coherence, confidence, or reasoning depth** in responses.

Proposed Enhancement:

- Introduce additional scoring metrics, such as:
    - **Fluency**: Measures the clarity and natural flow of the response.
    - **Coherence**: Evaluates how well-structured and logical the answer is.
    - **Confidence Level**: Detects hesitation or uncertainty in speech.
    - **Technical Depth & Accuracy**: Measures the depth of understanding in technical concepts.
    - **Recall and Truthfulness** : measure the hallucinations of user and how accurate he reach to bullet point
-
- **Implementation Plan:**
    - Use **NLP techniques** (e.g., sentence embedding comparisons) to measure coherence.
    - Analyze **speech patterns** (e.g., pauses, hesitation markers) for confidence scoring.
    - Integrate **domain-specific QA models** to verify technical correctness.

2. Updating the Prompt for Better Evaluation

Current Limitation:

- The current **evaluation prompt** focuses on correctness but does not guide the AI in scoring advanced aspects like fluency or reasoning depth.

Proposed Enhancement:

- Refine the **evaluation prompt** by adding:
    - Specific **criteria for fluency, coherence, and reasoning**.
    - Examples of **strong vs. weak responses** to help the model generate better justifications.
    - A more structured **weighting system** to balance different competencies.

Implementation Plan:

- Modify the **scoring prompt template** to include new evaluation factors.
- Experiment with **few-shot learning techniques** to provide better model examples.

3. Fine-Tuning or Using a More Advanced Model

Current Limitation:

- The **SeamlessM4T** model works well, but there are cases where transcription errors or evaluation inconsistencies occur.
- Current evaluation relies on **OpenAI's API**, which might have **limitations in domain-specific assessment**.

Proposed Enhancement:

- **Option 1: Fine-Tune a Model**
  - Fine-tune **Whisper or SeamlessM4T** on domain-specific interview datasets to improve transcription accuracy.
  - Fine-tune an **LLM (like Llama 3)** for better response evaluation tailored to **technical interviews**.
- **Option 2: Use More Robust Models**
  - Consider integrating **Google Cloud TTS/STT** for more **robust transcription and synthesis**.
  - Use **Claude or Gemini AI** instead of OpenAI for more **context-aware response evaluation**.

Implementation Plan:

- Train on **real interview datasets** for accuracy improvements.
- Benchmark **OpenAI vs. Claude vs. Gemini AI** on evaluation consistency.
- Integrate **Google's Speech API** for comparison.

4. Building a Web Application for a Better User Experience

Current Limitation:

- The system currently runs via **a script**, which limits accessibility for non-technical users.

Proposed Enhancement:

- Develop a **simple web-based interface** where users can:
  - Upload **candidate audio** and receive an **evaluation report**.
  - Select **different interview types** (e.g., technical, behavioral).
  - View **real-time rankings** of multiple candidates.

Implementation Plan:

- **Frontend:** Use **React.js or Streamlit** for an intuitive UI.
- **Backend:** Implement a **FastAPI or Flask server** for handling requests.
- **Storage:** Save reports in a database (e.g., PostgreSQL or MongoDB).

5. Handling Multi-Language Support

Current Limitation:

- The system only supports **English interviews**.

Proposed Enhancement:

- Expand support for **multilingual AI interviews** by:
    - Integrating **SeamlessM4T's multilingual capabilities**.
    - Adding **language-specific evaluation prompts** to adapt scoring.
    - Supporting **candidate language preference selection**.

Implementation Plan:

- Extend **TTS/STT** components to handle **multiple languages**.
- Train evaluation models to **handle different linguistic structures**.

Conclusion

The Audio AI Interviewer system successfully automates the process of conducting and evaluating technical interviews using speech-to-text, text-to-speech, and AI-driven evaluation. By following a structured pipeline—from loading predefined questions, generating audio, recording candidate responses, transcribing text, evaluating answers, and generating reports—the system ensures a streamlined and objective assessment process.

One of the key strengths of this system is its ability to evaluate candidates beyond simple correctness by incorporating AI-driven scoring, justifications, and ranking mechanisms. The ability to fine-tune evaluation criteria, expand scoring metrics, and scale the system to multiple users makes it a powerful tool for conducting technical assessments efficiently.

While the system is already functional, future enhancements such as more advanced evaluation models, web application deployment, and multi-language support can further improve its usability and accuracy. By integrating these improvements, the system can evolve into a comprehensive AI-powered interviewing tool that enhances hiring decisions, reduces bias, and ensures fair assessments for all candidates.