Mini Clinical AI Assistant – Documentation

1. Solution Overview

The solution implements a mini Clinical AI Assistant. The assistant processes a short clinical audio note, transcribes it, extracts structural clinical fields, and generates a SOAP note using GPT-4o. The goal is to simulate an EMR-ready pipeline for automated document extraction.

2. Architecture & Flow

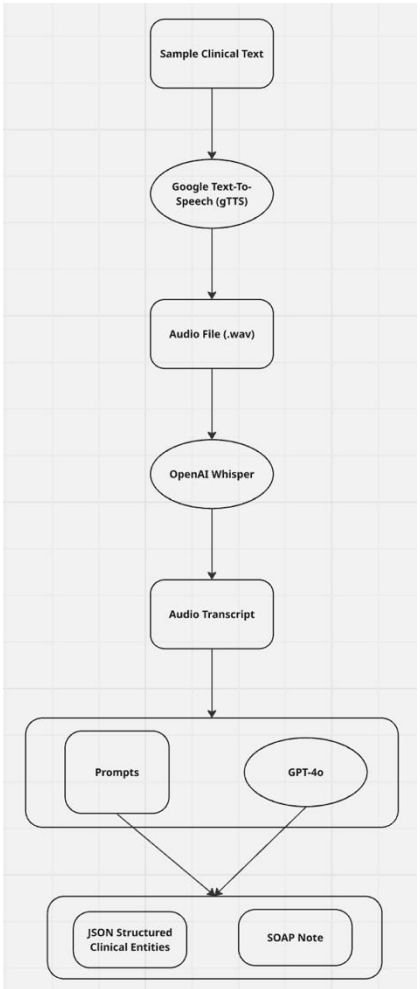

Fig 1: Architecture Diagram

1. Used Google Text-To-Speech to convert a sample clinical text to an audio file in .wav format
2. Performed transcription on the generated audio file using OpenAI Whisper Turbo model
3. Developed separate LLM prompts to extract clinical entities and to generate SOAP note from the transcript.
4. To generate SOAP notes, two different prompts were written. One of them generates a structured JSON output and the other generates a plain text output.
5. Used a GPT4o model along with the prompts to extract the required clinical entities and generate a SOAP note from the transcript.
6. The Clinical entities were extracted as a structured JSON using a custom JSON parser applied on top of the model's original response.
7. The results of the entity extraction and SOAP note generation are stored under the folder outputs/ in the source code provided.

The solution has also been updated with a simple RAG functionality that enables querying past/historical diagnosis extracts (JSON) to recommend treatment approaches.

Fig 1 illustrates the flow of the solution

3. Tools & Technologies

Component	Tool/Library
Audio Text to Speech	gTTS
Audio Transcription	OpenAI Whisper
LLM	OpenAI GPT-4o
Parsing & saving to files	Python (Custom classes)
Vectorstore	FAISS

4. Prompt Templates

The solution makes use of two prompt templates:

a. Clinical Information Extraction Prompt – To describe the required entities and define the extraction schema
b. SOAP Note Generation Prompt
    i. Text Version – Defines each SOAP section: Subjective, Objective, Assessment and Plan

ii. JSON Version – Same as above but instructs the LLM to return a JSON with the keys {“subjective”, “objective”, “assessment”, “plan}

The prompts used in the solution can be found under the **prompts/** folder in the source code.

5. Scaling to Real EMR Product

In a real world scenario, clinical documentation may consist of multiple unstructured documents and data sources beyond clinical transcripts, lab reports, referrals, etc. To scale this solution for real world use, the following considerations need to be made:

i. **Multi-template Extraction Pipelines**: A prompt-based extraction framework would be used, where each document type is associated with a corresponding extraction prompt template. This would require building a model for automatic file type or template detection before selecting the appropriate prompt template.
ii. **Vector Database for Retrieval & Querying**: The Extracted EMR reports should be stored in an efficient vector database. This enables efficient contextual retrieval for downstream applications like RAG, question-answering or summarization.
iii. **Transition from Monolith to Microservices**: The current solution is a single pipeline, which is fine for prototyping but not scalable. In production, key components like the transcription service and LLM inference should be containerized and deployed as independent microservices**.** This would enable independent scaling and fault isolation.
iv. **High Availability & Load Balancing**: To handle requests from thousands of clinicians concurrently, multiple replicas of both the transcription and generative model services would be required. These would be load-balanced to optimize response time and resource utilization.
v. **Integrating into an Agent Framework**: In a real EMR system, this pipeline could serve as the foundational layer for a Clinical RAG Agent**.** The agent would consist of multiple tools such as a tool to query internal EMR data and a tool to query on external medical knowledge bases.