**SOAP Note Generation - Generative AI**

We've devised a novel approach for automating SOAP note generation through patient-doctor conversations, utilizing state-of-the-art language models (LLMs).

Our methodology employs two key techniques:

1. RAG
2. Advanced prompting.

Before delving into these techniques, it's essential to grasp the dataset utilized. We've leveraged an open-source dataset containing patient-doctor conversation, along with corresponding SOAP notes, patient demographics. This dataset is divided into train and test subsets.

Here is the data source : [adesouza1/soap\_notes](https://huggingface.co/datasets/adesouza1/soap_notes).

**Methodology**

Initially, we experimented with various LLM models such as OpenAI, Gemini, Mistral AI, and Claude-3, as we were constrained by a limited quota for accessing these models. After testing each one and depleting our quota, we found that the **Claude-3** model performed the best and still had ample quota remaining. As for the framework, we utilized **Langchain** to integrate and interface the LLM with our project.

**RAG - Retrieval-Augmented Generation**

The method relies on a retrieval mechanism to generate responses. Put simply, it involves using content retrieved from a knowledge bank—content closely resembling the query—as context for generating responses using a large language model (LLM).

We're employing a similar approach, utilizing conversational training data as our knowledge bank. Our aim is to retrieve the most similar conversation and SOAP note pair to provide context for the LLM, enabling it to generate responses by learning from this context for test conversations.

RAG involves several operations before yielding a response:

1. Data Splitting: In our approach, we split the data row by row, treating each row as a separate chunk.
2. Storing Split Data: We utilize FAISS as a vector index to store the split data.
3. Retrieval: FAISS inherently offers functionality to search for the most similar data to a given query from the vector store.

The retrieval process encompasses various advanced methods, such as the multi-query method. In our experimentation, we employed this method, which utilizes the LLM to generate versions of the query and retrieves the common content among them. However, this approach exhausts the LLM quota. Hence, we opted to utilize the FAISS retrieval method exclusively. Alternatively, one can train a neural network specifically for retrieval purposes.

1. Generation: The retrieved data serves as context for the LLM to generate a response. Specifically, we retrieve the most similar conversation along with its corresponding SOAP note, combining them to provide context. Subsequently, the SOAP note of the test conversation is generated.

**Advanced Prompting**

Unlike RAG, Prompting doesn't involve intricate operations for generation. It entails assigning a task to the LLM, which then generates results based on its training.

In our approach, we utilized one-shot prompting to generate SOAP notes from conversations. In this method, we provide the LLM with an example of a similar task we want it to perform. Additionally, offering more information about the task enhances the LLM's ability to generate a more precise response.  
  
Here is an example of our one-shot prompting.

"""Write SOAP Notes from the provided conversation.

<conversation>

Here is the conversation from which you need to write SOAP notes:

{conversation}

</conversation>

<example>

Here is an example of conversation and its respective SOAP notes:

Conversation: {conversation\_x}

SOAP Notes: {soap\_x}

</example>

<format>

Here is the format of the response

Subjective (S): Patient's reported symptoms and medical history.

Objective (O): Measurable and observable clinical data.

Assessment (A): Professional interpretation and diagnosis.

Plan (P): Strategy for treatment and management.

</format>

""”

Conversation : Conversation for which SOAP Note has to be generated.

Conversation\_x : example of the conversation

Soap\_note\_x : Corresponding soap note of conversation\_x

Passing more details in the format, helps LLM to understand better about the task and the format in which LLM needs to respond.

**Evaluation**

Assessing LLMs can pose challenges and create confusion. However, when you possess a reliable reference for generation, you can employ metrics such as BLEU and ROUGE scores. These metrics rely on overlapping methods to signify precision and recall, respectively.

Given that we had a source of truth in our test data, we generated responses for the top 10 conversations from that data. Limiting the scope to 10 conversations helped conserve the LLM quota.

The average reported 1-gram **ROUGE and BLEU** scores for the RAG method are **0.60 and 0.604**, respectively. Conversely, for the advanced prompting method, the average reported 1-gram **ROUGE and BLEU** scores are **0.58 and 0.55**, respectively. These scores indicate that the **RAG method outperforms** the advanced prompting method, as evidenced by the higher average scores.

Note: For a more deep dive on these evaluations please refer to **test\_evaluate.csv** in the project folder.

**Deliverables**

1. soap\_notes\_development (Jupyter Notebook) : This Notebook consists of testing multiple LLM, retrieval techniques, prompting and evaluation on Test Data.
2. soap\_notes.py (Python File) : This python code file consists of backend - helper functions, vectorstore, LLM for Chatbot.
3. app\_rag.py & app\_prompt.py (Python Files) : These python codes consist of streamlit codes to be utilized as chatbot, app\_rag consist of RAG based chatbot and app\_prompt consist of advanced prompting based chatbot.
4. test\_evaluate.csv (Worksheet): The worksheet consists of test conversation and their reference call\_notes, extracted call\_conversation and call\_notes pair from RAG, RAG based generation, advanced prompting based generation and their respective BLEU and ROUGE Scores.

Before running codes in your environment please make sure to install required libraries using requirements.txt (*pip install -r requirements.txt*).

If you want to launch the chatbot, please use these commands in your command prompt in the same directory of the project folder. *streamlit run app\_rag.py* for **RAG based chatbot** and *streamlit run app\_prompt.py* for **advanced prompting based chatbot**