Contents

1. Overview

1.1 Overview of assignment

The aim of this assignment is to develop a Large Language Model (LLM) Chatbot using ChatGPT for Singapore Finance Minister Budget 2024. This document outlines the design considerations, implementation and architectural diagram of the chatbot.

As ChatGPT model was specified for the use of this assignment, the embedding model selected for use was text-embedding-3-small, while gpt-4o-mini was selected as the main LLM engine for this assignment. The LangChain framework was used to implement the chatbot, and Streamlit was used as the front-end framework for the users to interact with.

A brief overview of the entire flow is shown in the diagram below.

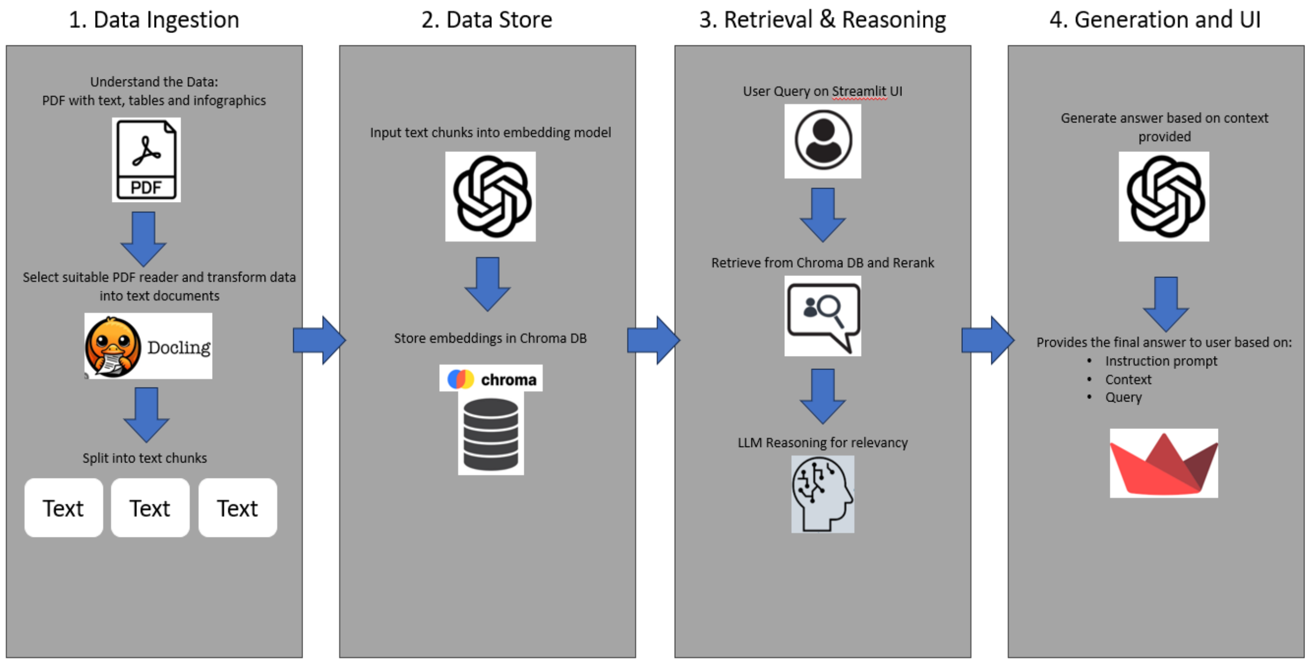

Figure 1: Brief Overview

2. Data Ingestion

2.1 Data Investigation

The files for Singapore Budget 2024 were downloaded from the Ministry of Finance website at https://www.mof.gov.sg/singapore-budget/budget-2024. The downloaded files which were selected as the source of information for the chatbot were in PDF file format, comprising of the following categories: 1) Budget Statement and Speech, 2) Annexes, 3)

Budget Booklet. Upon further investigation of the PDF Files, the structure of information in the PDF were largely texts for the statement and speech, tables for the annexes and infographics from the Budget Booklet. A suitable PDF reader has to be selected to be able to handle such structures of data.

2.2 PDF Reader Selection

Three PDF readers were selected for comparison and the details of the tests are stored in Data_Exploration.ipynb. The three PDF readers are as follows: 1) PyPDFLoader, 2) Unstructured, 3) Docling. PyPDF Loader is known for its speed and efficiency in parsing PDF files, while Unstructured and Docling is known for parsing unstructured data.

Figure 2 below is a screenshot of the Table 2 from annexb1.pdf, Figure 3, Figure 4 and Figure 5 are screenshots that show the data that was extracted with PyPDFLoader, Unstructured and Docling respectively. Docling was evaluated to be the better library to be used for this assignment as it was better at handling unstructured data as it better retaining context from the tables in the PDF files.

Table 2: B2024 COL Special Payment

Singaporeans Aged 21 and Above in 2024, and Own No More than One Property	Assessable Income		
	Up to $22,000	More than $22,000 and up to $34,000	More than $34,000 and up to $100,000
B2024 COL Special Payment	$400	$300	$200

Figure 2: B2024 COL Special Payment from annexb1.pdf

```
ble 2 for the scheme details. \n \nThe B2024 COL Special Payment will benefit about 2.5 million adult Singaporeans. \n \nTable 2: B2024 COL Special Pay
ment \nSingaporeans Aged \n 21 and Above in 2024, and \nOwn No More than One \nProperty \nAssessable Income  \nUp to $22,000 \nMore than \n$22,000 and
up \nto $34,000 \nMore than \n$34,000 and up \nto $100,000 \nB2024 COL Special Payment $400 $300 $200 \nNotes: \n1. Eligible adult Singaporeans must be
```

Figure 3: Data extracted with PyPDFLoader

```
Table 2: B2024 COL Special Payment

Singaporeans Aged Assessable Income 21 and Above in 2024, and More than More than Own No More than One Up to $22,000 $22,000 and up | $34,000 and up Pr
operty to $34,000 to $100,000 B2024 COL Special Payment $400 $300 $200
```

Figure 4: Data extracted with Unstructured

```
0,000, will receive the one-off cash payment of between $200 and $400 in September 2024 . Refer to Table 2 for the scheme details.\nThe B2024 COL Speci
al Payment will benefit about 2.5 million adult Singaporeans.\nTable 2: B2024 COL Special Payment\n\nB2024 COL Special Payment, Assessable Income.Up to
$22,000 = $400. B2024 COL Special Payment, Assessable Income.More than $22,000 and up to $34,000 = $300. B2024 COL Special Payment, Assessable Income.M
ore than $34,000 and up to $100,000 = $200\nNotes:\n1. Eligible adult Singaporeans must be residing in Singapore.\n2. Assessable Income will be based o
```

Figure 5: Data extracted with Docling

2.3 Split, Embedding and Store

After extracting the data with Docling, the data was split into smaller chunks using the RecursiveCharacterTextSplitter. While there are various other text splitting methods, this is the most effective method in splitting at more meaningful points which can preserve the context and meaning better than other splitting methods.

Subsequently, the split text was put into the “text-embedding-3-small” embedding model, and finally stored into the Chroma vector database.

3. RAG implementation

3.1 Retrieval Process

When the user sends a message in the chatbot, the following retrieval process will happen:

1) The prompt will be used to retrieve the top 10 chunks based on similarity search from the vector database. (10 chunks so that it is more effective with a Reranker, not more than 10 since it is not expected to be many relevant chunks based on the amount of data stored)
2) Based on the top 10 retrieved chunks, a Reranker is used to rank not just based on semantics but also the context, and returns top 3 chunks to be used as context into the start of the LLM chain.

3.2 LLM Reasoning

As one of the requirements for the assignment is for the LLM to not answer any unrelated questions aside from the Budget 2024 context, two LLM chains were created to try to prevent the chatbot from answering anything that is unrelated as follows:

1) With the 3 chunks extracted after reranking, the LLM is required to decide if the context for each of the chunk extracted is relevant to the question or not. This forms as a relevance check and acts as the “first line of defense” against unrelated questions when prompted to the LLM.
2) In this first chain, the LLM engine is enforced to only reply with “relevant” or “irrelevant”. Refer to Figure 6 below.

```
class RelevanceResponse(BaseModel):
    response: str = Field(title="Determines if context provided is relevant", description= "Output only 'relevant' or 'irrelevant'.")
relevance_prompt = PromptTemplate(
    input_variables=["query", "context"],
    template="Given the query '{query}' and the context '{context}', determine if the context is relevant to the query. Output only 'relevant' or 'irrelevant'."
)
# Relevance chain to be forced with a stuctured output
relevance_chain = relevance_prompt | llm.with_structured_output(RelevanceResponse)
```

Figure 6: First Chain to prevent answering irrelevant questions

3) If all 3 chunks were concluded to be irrelevant to the question, the context that will be provided during the generation chain would be “No relevant context found”. The LLM should then be able to firmly reply the user that it is unable to answer.

```
# Step 3a: If no relevant contexts found, direct the LLM to respond with no answer found
if not relevant_contexts:
    print("No relevant contexts found. Generating without using any provided context")
    input_data = {"query": query, "context": "No relevant context found."}
    response = generation_chain.invoke(input_data) # Run chain
    return response
```

Figure 7: No context provided if LLM decides no relevant chunks

4) Else, if any of the 3 chunks were decided to be relevant by the LLM, all 3 chunks will then be provided to the LLM as context during the generation chain. In the generation chain, there is also an instruction prompt on top of the context and query provided. The instruction prompt specifies the LLM not to answer anything outside of the context provided. The summary of the overall RAG flow is shown in the diagram below.

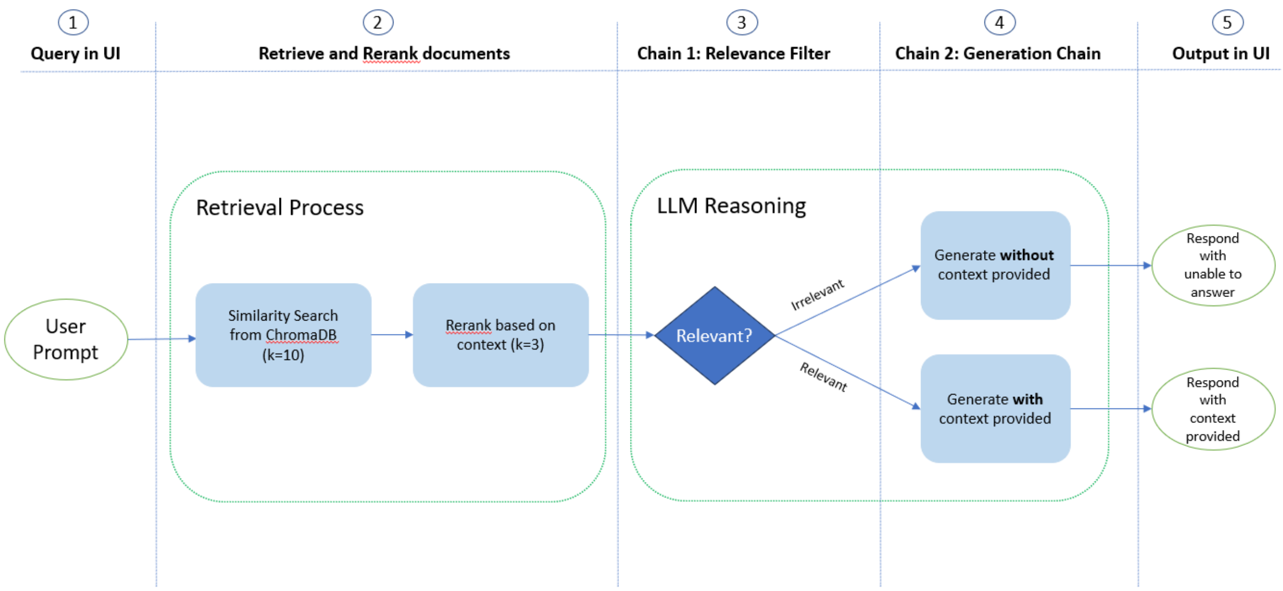

Figure 8: Overview of the RAG process

4. Results

4.1 Prompt and Results

The prompts were then fed to the Chatbot through the Streamlit UI. Results were as shown below:

Question 1: Am I eligible for the Majulah Package?

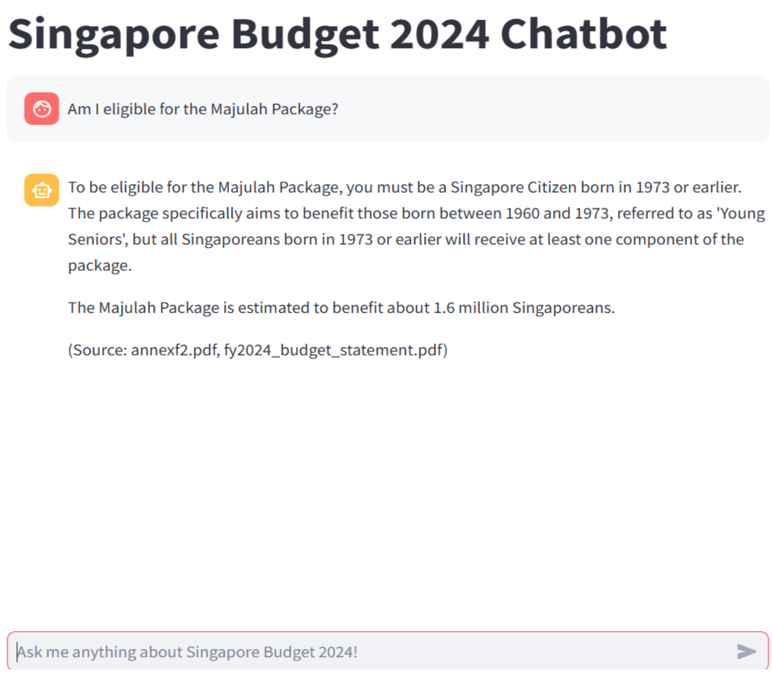

Figure 9: Results from Question 1

Question 2: What are the payouts I can expect to receive in December 2024?

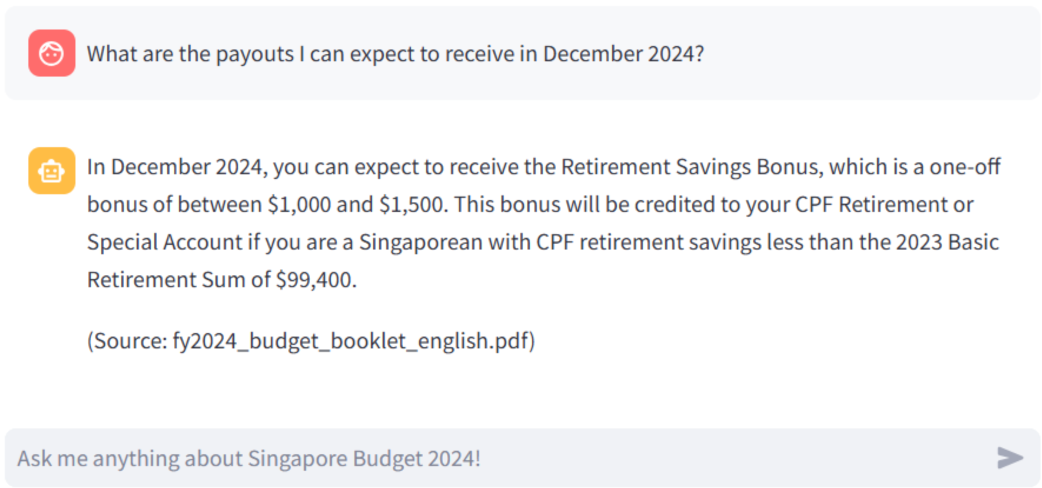

Figure 10: Results from Question 2

Question 3: What are the Key reasons for high inflation over the last two years?

Figure 11: Results from Question 3

5. Conclusion, Tests and Future Work

5.1 Conclusion

As seen from the results above, the Chatbot was able to answer most of the questions apart from question 2, which failed to mention about the Cash component as well as the CPF Medisave component. Upon further investigation, the reason for this was due to the data that is stored in the Chroma Database failed to capture the information. This is largely due to the PDF parser failing to capture the unstructured data from the infographics. Hence, the chatbot will have difficulty answering any questions pertaining to the dates which could mostly be found in the budget booklet since it would have lacked the context for it.

A workaround would be to manually create the data into a text file, capturing the information from the infographics, then eventually splitting the text and storing the data into the Chroma DB. This manual creation of data was not performed for this assessment

5.2 Further Tests

Further tests were conducted to check for the response of the chatbot. The chatbot was able to answer questions related to information that was captured in tables such as from Figure 2 (screenshot from Annexb1) above, and the chatbot would also not answer anything outside of context from Singapore Budget 2024.

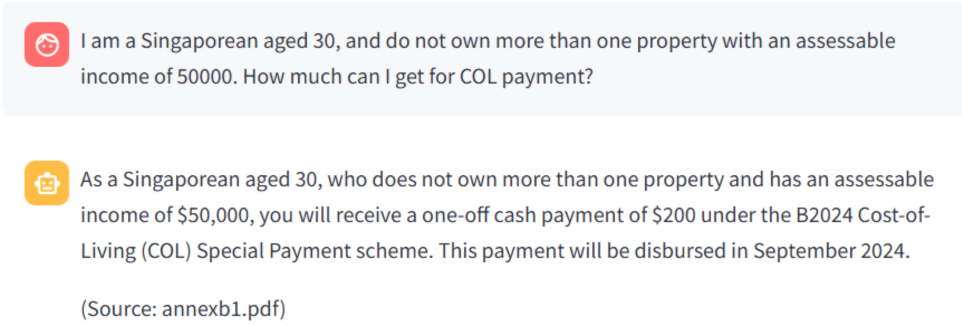

Figure 12: Additional Test Conducted

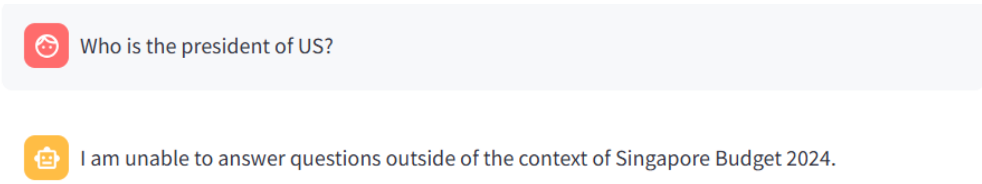

Figure 13: Non-Budget 2024 Question Response

5.3 Future Work

There can be further enhancement which can be made if this app is to be deployed to the public. The use of libraries such as asyncio would benefit the app when it receives multiple requests, and multithreading can also be used if high volume is expected. This could also be deployed onto cloud platforms, such as AWS for increased security and improved access and connectivity.