Identifying User Requirements Using LLMs

Dustin O'Brien
Department of Computer Science
Salisbury University
Salisbury, United States
dobrien6@gulls.salisbury.edu

Spencer Presley
Department of Computer Science
Salisbury University
Salisbury, United States
spresley1@gulls.salisbury.edu

Xiaohong Wang
Department of Computer Science
Salisbury University
Salisbury, United States
xswang@salisbury.edu

Shuangquan Wang
Department of Computer Science
Salisbury University
Salisbury, United States
spwang@salisbury.edu

***Abstract*— Saltwater intrusion caused by rising sea levels, droughts, and increased freshwater pumping in the coast and tidal regions poses a significant threat to people's daily lives and industrial and agricultural production. Many organizations and individuals have come together to develop various products to reduce its impact. A major challenge in this is that different water users have different water usage types such as irrigation, power generation, sand & gravel washing, product manufacturing, drinking water, etc. and it is essential to learn and understand the needs of each user group so that appropriate services can be provided and used effectively. To effectively identify the diverse requirements from the water users, in this paper, we proposed a method to use Large Language Model (LLM) to analyze water user interview transcripts and construct an AI-powered chat system. The system supports quick and accurate understanding of client water usage requirements under the situation of saltwater intrusion.**

I. Introduction

Saltwater intrusion is an increasingly pressing climate issue, characterized by the mixing of saltwater with freshwater due to rising sea levels[1], droughts, and increased freshwater pumping.[2] This phenomenon poses a significant threat to people's daily lives and industrial and agricultural production.

These concerns have generated demand for products to monitor, visualize, and predict water salt concentration. However, due to the diverse and user-specific demands of the water usage clients, it is important to identify each client's unique needs through user interviews, surveys, etc. To quickly and correctly synthesize the results of those user interviews, we propose to use LLMs to develop an interface to analyze the results from user interviews and generate summaries in a simple and concise manner.

In this research, we plan to develop a user-friendly chat system that allows service providers and product developers to better understand water usage clients' needs and demands related to saltwater intrusion. This proposed system uses transcript obtained from interviewing a number of clients who represent various organizations. The interview transcript data was anonymized to ensure no sensitive data is shared with LLM providers. The interview data is fed to an LLM to construct the proposed AI-powered chat system. The design of this product is inherently non-domain specific, allowing it to theoretically be used for user requirement analysis in other application domains.

We have seen a growth in models designed to help administrative and specialized industries constructing specialized LLM architectures. One example comes from financial software company Bloomberg who is working on a local LLM for their own financial usage. They have shown that there exists a great deal of practicality and benefits to their work. This research aims to achieve some of the same benefits to this approach, but by using tools to improve performance of existing LLMs, making it more affordable and more accessible.

The rest of this paper is organized as follows. Section II introduces the technologies used in this project. Section III & IV describe the backend and frontend systems of this application respectively. Section V describes the user interview processing. Section VI discusses results and Section VII concludes this paper.

II. Why use LLMs?

A. What is an LLM?

LLMs are a recently developed type of AI that is focused on the understanding and generation of human language.[3] In other words, they are capable of taking an input text and generating an output text based on that input text. A common application of LLMs is chat-bots, which takes a user prompt and creates their own text response. The most popular of these is OpenAI's ChatGPT.[3]

B. How do LLMs fulfill our needs?

For our research, we need to extract requirements about water usage from interview text documents. An LLM architecture is well-suited for this task due to it's ability to understand contextual meaning across text. Specifically, we plan to develop an OpenAI powered chat-bot for its simplicity, versatility, and performance. This chat-bot takes user questions, queries documents for relevant information, and generate responses that not only pull from the context it learned during training but also from the client interview transcripts. OpenAI's models are notorious for their ability to understand complex pieces of text and extract details from them. This will see the chat-bot be able to offer insights related to the wants and needs expressed by the clients in their interviews.

C. What is Langchain?

Due to the rapid growth and popularity of LLMs and chat-bots, user-friendly mechanisms for handling and working with these models have become necessary. LangChain has emerged as one of the most popular frameworks, providing many helpful tools and components that make it quick and simple to incorporate complex, recent innovations, and strategies into modern LLM-based applications.[4] Our research aligns with the

987-1-6654-7345-3/22/$31.00 ©2024 IEEE

type of application intended for LangChain, as it requires these tactics for efficient and effective LLM explanations.

III. BACKEND CHAT-BOT

A. Document Processing and Storage

In order to use and access documents given to the system, it is necessary to define a consistent file format that we could easily access and convert various possible interview document formats into. A good choice is the simple plain text or *.txt* format. Therefore, we created an array of file converters to turn possible file formats, such as *.docx*, *.doc*, *.csv*, *.md*, *.xlsx*, *.xls*, and *.pdf*, into plain text. Documents were then split into sections, embedded by OpenAI's text-embedding-ada-002 embedding model and stored within a local FAISS vector database, as shown in Figure 1.

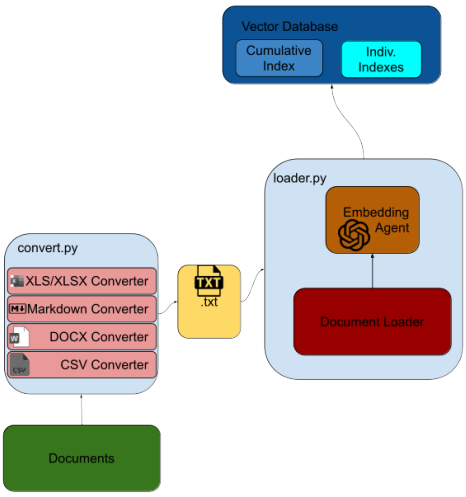

Figure 1. Structure of document loading system

By default, vector databases return the top k most relevant documents, which lead to an unnecessarily large amount of data having to be processed by the LLMs. This is because, if there exists less than k documents that are relevant to prompt, the querying algorithm will begin to select irrelevant documents. Documents upon retrieval must be scored based on relevance to allow for filtering the results to only include the most relevant ones.

B. Chat-bot Construction

To use our chat-bot backend, we process user input prompts by querying the vector database to retrieve a set of relevant document sections. These sections are incorporated into the user given prompt, which is then sent to OpenAI for processing. OpenAI generates a response based on the enriched prompt, which we subsequently use as chat-bot outputs.

However, functional issues arise from this schema, the most significant one being the lack of contextual continuity in conversations. For example, if a user encounters an issue and requests further clarification on a specific point, the LLMs only receives the current prompt along with a set of potentially relevant documents. This approach disregards the context provided by previous interactions, underscoring the need to incorporate prior conversations to maintain continuity from the end user's perspective.

To alleviate this, we integrate what is known as memory. Memory in the realm of LLM powered chat-bots involves storing a collection of previous prompts. When a new prompt is generated, it is added to this collection, and the entire set of prompts is sent to the LLMs. This approach ensures that all prior conversations are accessible to the LLM, thereby maintaining continuity and providing more contextually accurate responses. We implemented these structures using the LangChain framework.

C. Prompt Engineering

To keep the costs of running this program practical for individual and small organizational use, smaller LLM models are necessary. In our case, we opted for OpenAI's GPT-4o-mini model due to its significantly reduced price compared to GPT-4 and GPT-4o. However, as the size of the LLM decreases, so does its capability to store and understand large amounts of context.

One beneficial feature of memory is the storing of a system messages. Just as user and chat-bot conversation messages can be saved, it is often advantageous to store the first message as an explanation of the LLM's purpose and specific requirements. This approach helps mitigate some limitations of smaller models by clearly defining certain expectations. For example, if identifying clients in a meeting is a central task for the LLM, it can be specified that individuals are rarely representatives of only themselves, thus the LLM should avoid listing individual names. While system messages can be utilized without storing them in memory, this requires they be passed in on every query of the LLM, further taking up valuable context space.

Another advantage of introducing a system message is the creation of personas. A persona is a role assigned to the LLM to help it better understand its purpose and how it should interact with the user. These personas are included in the memory via the system message before any prompts are sent. In our case, it was beneficial to inform the chat-bot that its primary purpose is to explain client criteria and needs. This ensures that its responses focus on the clients' needs and concerns, even if these are not explicitly stated by the user. [7]

In this project, the prompt constructed a persona of a document analyzing assistant. On top of personas it was found to be advantageous to set guidelines of what was needed from responses such as giving follow up questions, and the length of inference that the LLM should make in regards to client needs. We can also specify unwanted behavior discovered during testing of bot examples, including fabricating information, mentioning source documents, and overly lengthy explanations.

D. Specialized Memory

Token amount overall is a major problem in a setting like this. As handling of large amounts of information requires increased computation and necessity for increased amount of

context to understand, therefore the high token count results in increasingly less practical and more costly bots. Various memory solutions have been applied to deal with this issue. Two main ones are summary and entity memory. Firstly, summary memory will store a summarized version of the conversation that GPT appends new information to after each part of conversation. The second form of memory is entity memory, which over the lifespan of the conversation keeps track of relevant "entities" which encapsulates a person, place, thing, or idea that may be continuously referenced throughout the conversation. Entities can then be queried similar to vector database and appended to as new information comes out.

These forms of memory then allow for novel implementation. An example in our research was investigating using summarization memory to shrink documents into only relevant details. This can be increasingly practical due to documents high token count and large quantities of overly specific data such an individual's name, specific times, etc.

E. Web Searching

Another problem the chat-bot may run into is a lack of context in regards to some aspects of an interview. If a user wanted to know more about a specific organization which may not be contained within the documents, the model would be incapable to answer the question properly. Furthermore, the user's question may require up-to-date information that wasn't available at the time of training for the LLM. To increase the LLMs ability to understand context, it is beneficial to give the LLM access to documents outside of the ones provided.

Our application has two main ways of handling this. Firstly, when web searching is needed, our AI can request web documents via the DuckDuckGo API. The second is ArXiv's search. ArXiv is an online academic and scholarly documents database ran by Cornell University[6]. It is highly beneficial to our chat-bot especially in areas such as Water Salinity where the knowledge is often highly technical and complicated and interview data is unlikely to go into those technical details, and DuckDuckGo may not return academic resources.

IV. Front-end Methods

A. User Interface

The front-end for our application has a few requirements and goals to make it as simple as possible for the end user and for future development. Firstly, we wanted the front-end to be able to be used as many various types of applications, such as a GUI Desktop Application and Website given its need to be used within various areas of research and confine to their needs. We also wanted fast development and therefore needed a language that is easy to work with, this lead us to use the Next.js web framework due to its robust, already-existing infrastructure.

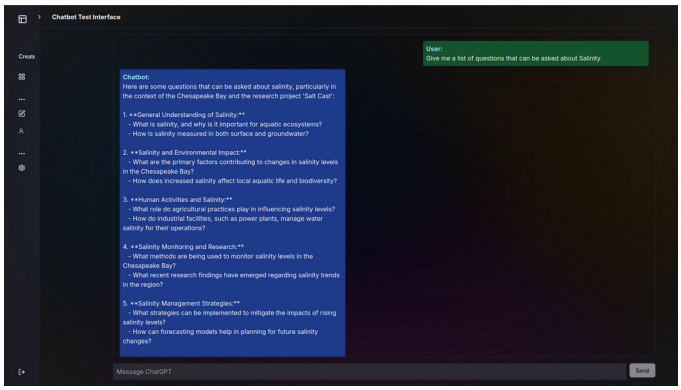

Figure 2. Example of chat interface

The overall goal of the User Interface was to keep layout clean, user-friendly, and fairly simplistic. Fortunately, we have seen large amounts of LLM-oriented applications and websites have achieved similar goals effectively. The largest of these is OpenAI's ChatGPT. Inspired by its user interface, we have opted for a layout with a large chat box taking up majority of the screen. This prevents visual noise from taking away user's attention and centers the application towards the chat.

B. Conversation Loading

Another practical feature we implemented was the ability to save and continue previous conversations. The functionality for this however is more complex and implemented in large part by us. The way in which this works is to store the conversations in a SQL database where every saved conversation has a unique key that is generated during run time. These conversations then are loaded into the frontend and reconstructed into the various chat messages and saved into chat history for backend purposes. The frontend then is just an array of buttons allowing for user to choose desired chat session, as shown in Figure 3.

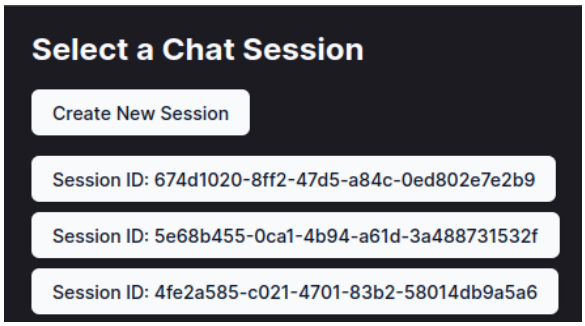

Figure 3. Chat selection user interface

V. Client Interview Documents Analysis

A. Client Identification

With the design of specialized chat-bots such as this, it opens the possibilities for generation of specialized algorithms and structures to maximize the intended capabilities. One of the major ideas of this project was the development of specialized documents that are easily understandable and precomputed such that many of the questions likely to be asked by users can be answered quickly. Given the client-oriented perspective, these generated documents will be a simple list of every client needs and the relevant clients. The two parts identify the clients and list of needs of the client, as shown in Figure 4.

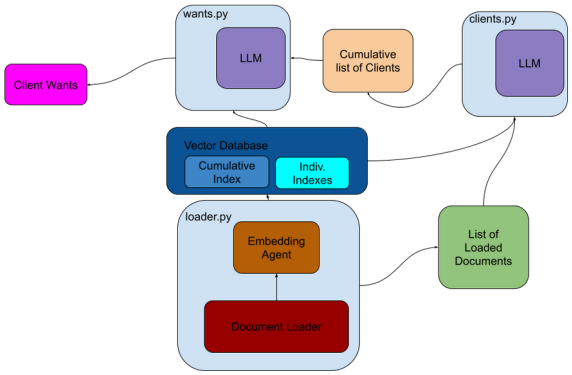

Figure 4. Identified clients and their needs

The first section of this program was to extract data on who each of the clients are. A naive approach for this is to simply send a prompt to OpenAI requesting names of the clients from documents. However, this solution has two major flaws. The first one is a limited number of documents that can be sent at one time. Therefore, large amounts of data will not be seen and cannot be analyzed. Second, due to the anonymization of data, people in meetings such as names like ‘interviewee1’ will be assumed to be the same person when it is very probable to be different people in different interviews.

One solution to this problem is to keep track of documents stored and run through each document individually. This requires independent vector databases for each document and logging of which interviews exist. Another novel strategy to identify clients is the usage of entity memory. The idea is that all clients are entities. Identifying clients will simply be pulling entities from entity memory that are associated with a given document. These structures though cause a secondary issue, that being defining who the ‘client’ is in a given document. In order to identify who the client organization is, we must scan the documents and identify references to other potential clients who are not actually present in the meeting. Once identified the LLM can be informed that these references are not clients themselves, making the results received more consistent and accurate.

B. Need Identification

Once clients have been identified, the naive approach of identifying needs becomes significantly more logical. This is because, given client names, very few documents are likely to pertain to each the smaller clients within the Salinity field. We did however discover major differences between the usage of smaller singular document based databases and large cumulative databases. We have chosen to be oriented towards cumulative databases to maximize information provided to the LLM.

With this system, a document generation becomes simple opting for markdown as preferred format allowing for both Human readable and LLM readable format to be built with very little required effort. These documents have since been loaded into LLM.

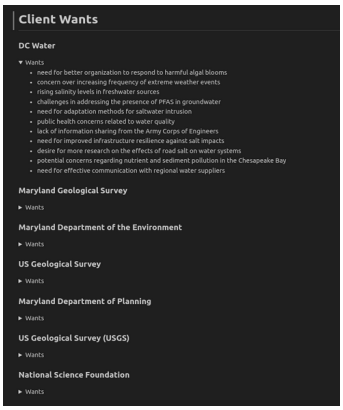

Figure 5 Client needs example

As seen in Figure 5 we see the output generated from this structure overall the number of clients seems to line up with expected output from meetings although with a repetition in USGS. Wants derived from the interviews also seem to line up with products and information seen in the transcripts. Overall, we have found these results to be satisfactory.

VI. Results

A. Experimental Results

Overall, we have found that modern LLMs have become very proficient at identifying and understanding complex topics such as salinity. They can easily understand documents given to it and use information from those documents in response to user queries, as shown in Figure 6.

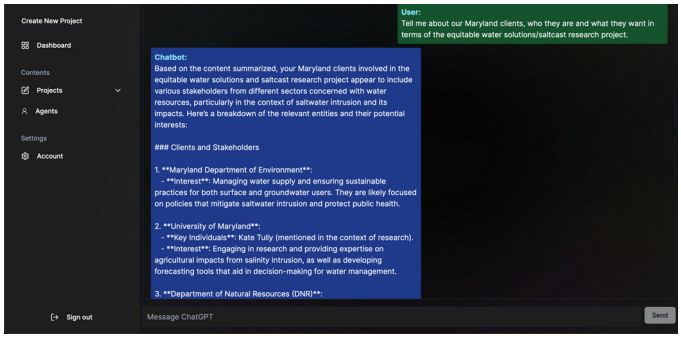

Figure 6. Example output of the developed chat-bot

From the example output in Figure 6, we can see that the Chat-bot is capable of taking a topic very specific to water usage and output relevant information about each of the clients and specifics about them. The modern LLM architecture as indeed hit a point where it has become powerful, malleable, and simple enough and specialized, locally-made systems can be practical for organizational usage.

B. Chat-bot Findings

Throughout the research many discoveries were made in regards to vector databases, web searching, and specialized memory. Firstly, we have found that there are significant problems in regards to document querying. Given the following prompt “Please give me an explanation of the needs of Maryland Department of Planning”, we found that on default similarity search without score has a precision of 50% of documents given were relevant. Followed by a recall of 10% of relevant sections of documents were properly given. The latter result is to be somewhat expected given the limited amount of space. We have found that document scoring to be ineffective at removing irrelevant documents as all retrieved documents had good scores.

Summary memory was found highly effective for conversations. Long conversations had little to no penalization. However, when used to summarize documents before sending to AI, we saw significant decreases in information kept in summarized documents.

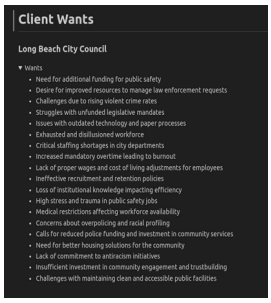

Figure 7. Example of Long Beach client needs

Figure 7 shows client needs generated on the second system where only 1 of 5 clients were identified in the meeting.

C. Document Generation Findings

In regards to documents, the overall structure of extracting clients followed by extracting each client needs works as a good solution. However, specifics of the program such as attempting to identify interviewees vs directly asking for client show very different results. Our observations have shown that the first system significantly reduces the number of clients given. Such a system works extremely well on interview datasets. However, using test dataset of Long Beach County City Council meetings where there exists larger amounts of clients, the second strategy became more effective, as shown in Figure 7.

VII. Conclusion

We have found that LLMs have proven to be effective at dissecting and understanding user interview documents. But we have observed that many of the recently created support structures around them such as vector databases, entity memory, and at times summary memory have proved to be less effective than desired to be used in a task such as this. Despite these shortfalls, overall, specialized chat-bots, particularly around client needs identification, could be a practical and fairly reliable tool that can be used by organizations on various products, research and future endeavors.

References

[1] S. W. Chang, T. P. Clement, M. J. Simpson, and K.-K. Lee, “Does sea-level rise have an impact on saltwater intrusion?,” *Advances in Water Resources*, vol. 34, no. 10, pp. 1283–1291, Oct. 2011, doi: https://doi.org/10.1016/j.advwatres.2011.06.006.

[2] Water Resources, “Saltwater Intrusion,” *www.usgs.gov*, Mar. 02, 2019. https://www.usgs.gov/mission-areas/water-resources/science/saltwater-intrusion

[3] IBM. “What Are Large Language Models? | IBM.” *Www.ibm.com*, 2023, www.ibm.com/topics/large-language-models.

[4] “LangChain.” *Www.langchain.com*, www.langchain.com/.

[5] S. Wu *et al.*, “BloombergGPT: A Large Language Model for Finance,” Dec. 2023. Available: https://arxiv.org/pdf/2303.175

[6] “arXiv.org e-Print archive,” *Arxiv.org*, 2019. https://arxiv.org/

[7] G. Sun, X. Zhan, and J. Such, “Building Better AI Agents: A Provocation on the Utilisation of Persona in LLM-based Conversational Agents,” Jul. 2024, doi: https://doi.org/10.1145/3640794.3665887