VReqST

Conversational Requirement Gathering for all the stages of VReqST.

In the earlier version of VReqST, the requirement analyst must populate the model template files in the GUI to generate the requirement specification. Considering the Requirement Analysts (RAs) are non-technical in practice, we introduced a conversational tool for capturing the specifications linked with model template files. When compared to the existing edition of VReqST, the proposed approach will aid the RAs to provide specifications through conversations and limit the specifications within the context of the VR technology domain.

- This conversational requirement specification feature is a Q&A type GUI that captures the scene, article, action-response, and timeline model template. The behavior step is continued using the earlier approach.
- Here the questions are populated using the underlying model template files of scene, article, action-response, and timeline on-demand. The questions are constructed from the respective model template JSON files.
- The responses for each field are validated against their validation provided in the respective validator JSON files. In case of errors or incorrect responses, proper error messages are displayed to make the user aware of that.
- The user is supposed to specify the requirements in a specific order i.e., scene, article, action-response, behavior and timeline. This is continued same from the earlier approach.
- The captured values of relevant attributes in the model template are equated to the respective attributes of the model template and are stored accordingly in the backend.
- Also, a website tour is implemented which makes the user aware of all the essential features of this conversational requirement gathering.

Retrieval Augmented Generation (RAG) to generate the model template files

Though in the previous version, we built a conversational tool for capturing the specifications, this task can be tedious and time consuming given the number of fields in each model template file as we used one-to-one mapping from fields to questions. Considering the use of ML in our task, we built a **Retrieval Augmented Generation** (RAG) application to create a conversational tool designed to assist in capturing and generating the VR requirements. As a part of experimentation, we focused on a particular domain i.e., chess game scene. This tool combines a language model, vector store, and custom prompting to streamline the process of gathering detailed specifications for VR development in the context of chess.

1. Setup and initialisation
    - The application leverages **LangChain** with the **OpenAI API** for language model processing and embedding creation. This sets up an environment for interactive question-answering.
    - Loaded the document from a chess-specific PDF file (`chess.pdf`). The document contains data from the wikipedia for now. These document serve as the knowledge base for the chess context, enabling the RAG model to provide domain-specific answers.
2. Document Processing and Chunking
    - The document is recursively split into manageable chunks of text, each of approximately 1000 characters with a 200-character overlap. This overlap ensures continuity in the context, providing richer information per chunk, which is beneficial when retrieving relevant segments.
    - These document chunks are indexed in a vector store (Chroma). It enables efficient similarity-based document retrieval using embeddings, which will later facilitate relevant information fetching.
3. Question Generation
    - Read the contents of `scene.json` and `scene-validator.json` files to capture the different attributes along with their validation rules. These are used for the

generation of questions.

- Create a question for each field in the VR requirements by referencing the validator file. It generates a basic prompt based on the query attribute along with the data-type instruction based on the typeof field and an additional comment providing each field's purpose in context.

4. Conversational Retrieval Augmented Generation

    - The RAG application sets up a retrieval chain where each question about the VR scene triggers a document retrieval step. This chain is constructed using a retriever, RAG prompt and `gpt-4o-mini` model.
    - The retriever pulls relevant information from the vector store by matching the content of each question against the embedded document chunks. A pre-defined RAG prompt is pulled from LangChain's model hub, and it structures the response generation for each question by leveraging the retrieved context. The `gpt-4o-mini` model processes the retrieved context and question prompt to generate coherent, domain-specific responses.

5. Prompt Composition and Response Generation

    - A main prompt provides guidance on the overall objective informing the model about the generation of scene requirement specifications for a chess game.
    - For each field in the scene template, a question is formed by combining the prompt with the generated question, creating a full inquiry. The response to each prompt is generated using the RAG chain, which are then printed and stored

6. Execution Flow and User Interaction

    - The application iterates over each field in the JSON template, asking specific questions and generating responses based on the chess domain context.
    - This iterative process allows the RAG application to collect all necessary data points for a comprehensive VR scene specification for a chess game, saving each response for future use in the requirements documentation.

7. Observations

- For many fields, the model gave a “I don’t know” answer meaning the model was not able to generate a suitable value for this particular field. This could be accounted for some VR domain specific attribute which the model is unaware of.
- Also noted the answer for two different main prompts where one was focused on detailed explanation and other on brief responses. Although, there were not much difference in the answers generated, the answers were correct according to the corresponding field in the scene.

Improving the RAG application using Few Shot Prompting

In the previous RAG application, although the model could generate the specifications for the chess game, there were many fields where the model generated “I don’t know” as the response. So, to improve this we explored different possibilities and introduced the idea of **few-shot prompting**. This technique involves providing the language model with a small number of examples to guide its response to a specific task. For this purpose, we used examples from prior scenes. This approach provides the model with structured examples for each field in the VR requirements, giving it a clearer sense of how to generate contextually appropriate responses.

1. Loading and structuring example data
    - Load the example scene files from a folder (`./examples`). These JSON files contain pre-filled scene data, each representing a prior VR scene configuration.
    - For each field in a JSON file, the field’s value along with the scene filename are stored in a corresponding entry in the examples dictionary.
2. Generating questions with few-shot examples
    - While generating the question for each field, we frame the question as we used to do earlier. But now we are also supposed to include the few-shot examples if available.

    - If there are previous examples available for the current field, each of them is formatted as an input-output pair to form the few-show examples. The final question is constructed with these few-shot examples along with the original question.
    - These formatted examples are added to the question prompt, creating a contextual framework that guides the model's response by illustrating expected answer formats and levels of detail.
3. Enhanced prompt text
    - The prompt text is refined to include the added few-shot examples. It now explicitly instructs the model to refer to the examples provided with each question. The prompt also provides the overall task context.
    - The model now uses these examples to understand the specificity, format and detail level expected in the responses. This would help the model to understand what is the expected value for each field.
4. Retrieval Augmented Generation with few-shot examples
    - The RAG chain setup remains similar but now operates with enriched questions that contain few-shot examples.
    - The model uses these example-enriched prompts to generate detailed, consistent answers. Each response is then printed and stored.
5. Observation
    - The quality of the answers have significantly improved and they are more specific now as compared to earlier. The number of "I don't know" responses have reduced a lot.
    - Also noted the answer for two different main prompts where one was focused on detailed explanation and other on brief responses. Although, there were not much difference in the answers generated, the answers were correct according to the corresponding field in the scene.