Clemson Capstone Michelin Proposal

AJ Garner, Jacob Cox, Troy Butler, Caroline Baker, Carson Crockett

Overview

Michelin and many other companies need a fast, automated, reliable way to migrate their legacy systems while ensuring their new systems capture the same business logic as the old ones. This is especially important for outward-facing systems that handle invoicing and ordering to be translated accurately to minimize customer dissatisfaction. As a result, this work is done manually, which is both time-consuming and expensive.

We propose a modern, automated system using modern generative AI technology to accelerate the legacy system migration process by providing users with fast convenient access to all of the business rules and documentation of the legacy system in the form of a convenient, easy-to-use AI chatbot built to answer user queries.

Retrieval augmented generation (RAG) is an emerging generative AI technique designed to improve the factual accuracy and depth of the knowledge possible from generative AI technology. RAG provides a flexible way to use popular pre-trained large language models to answer domain-specific questions about data not included in the model's training data.

By applying this approach to the legacy system's source code and documentation, we will create a user-friendly chat application able to analyze the legacy system to answer questions about its business rules and functionality with a high degree of accuracy.

System Architecture

Our proposed solution is composed of several components that work together to achieve the desired functionality by implementing a RAG pipeline to extract system business rules. These components include the elements of a traditional RAG pipeline as well as extensions necessary for our specific use case.

- **Query Translator:** The query translator is responsible for rewriting, iterating, and modifying the abstraction level of user-provided queries. This is necessary because user queries can be ambiguous, leading to inconsistent context retrieval performance.

- **Router**: Routers direct queries to the appropriate vector store. In cases where there is more than one type of data needing retrieval, it is necessary to determine to which data store each query should be directed to find relevant information to provide to the generator. Often, another LLM is given information about the types of data available and asked to decide which data source to use.

Clemson Capstone Michelin Proposal

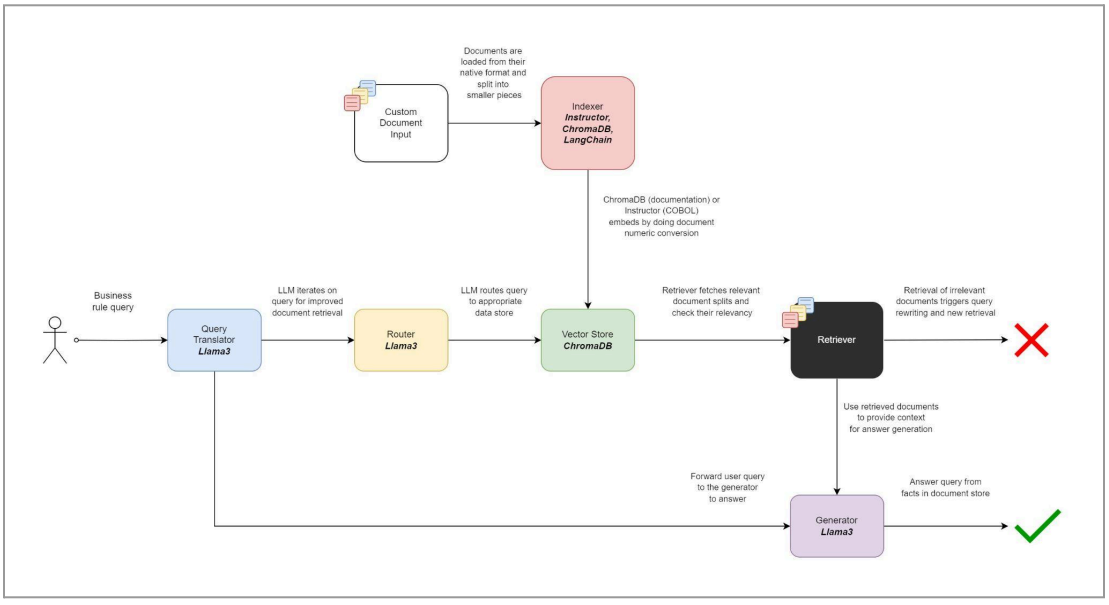
AJ Garner, Jacob Cox, Troy Butler, Caroline Baker, Carson Crockett

- **Vector Store**: Vector Stores are used to store the RAG documents. We will use vector stores to convert unstructured information like documentation and COBOL source code to their respective numeric representations to be stored for retrieval by comparing them against queries to find documents with similar information.

- **Indexer**: The indexer is responsible for storing documents for later retrieval. RAG relies upon performing comparing queries and the provided documents. Indexing is the process of converting the documents to a form where semantic comparisons can easily be made between the documents and queries during retrieval.

- **Retriever**: The retriever is responsible for fetching and assessing the quality and relevancy of the retrieved information. The core of RAG is the ability to look up relevant pieces of the provided documents as context for the LLM. The retriever performs similarity searches between queries and embedded documents to provide relevant context to the generator.

- **Generator**: The generator is the component of the RAG pipeline responsible for generating answers based on the provided query and the retrieved context using the LLM.



Clemson Capstone Michelin Proposal

AJ Garner, Jacob Cox, Troy Butler, Caroline Baker, Carson Crockett

Frameworks, Tools, and Techniques

- **LangChain:** A framework for creating applications that integrate with large language models. LangChain provides integrations with popular large language models and provides tooling to easily integrate these models with other ancillary components needed to build more complex LLM-based applications including most or all of the components necessary to implement a RAG pipeline. LangChain supports both Python and JavaScript – we will use Python. Access to most of the components in our system will be through LangChain's interfaces.

- **LangSmith:** LangSmith is a companion call-tracing and debugging platform featuring tight integration with the LangChain framework. We will be using LangSmith's integration features to test, debug, and monitor the behavior of our application during the development process.

- **Llama3:** The core of a RAG pipeline is a large language model that produces answers based on the query and the retrieved context. We are using Meta's Llama3 as our primary LLM because it is open-source, comparable to popular proprietary models, and can be run locally. These factors are favorable because they provide more freedom to make necessary adjustments.

- **ChromaDB:** RAG relies on decomposing reference information into easily retrievable chunks to insert into a generator's context window. ChromaDB is a popular open-source vector database for embedding documents for retrieval in a RAG pipeline. Our decision to work with it stems from it being open source, well-supported within LangChain, and commonly used for our use case.

- **Palmetto:** Compute-intensive applications like those running one or more LLMs benefit from higher resource availability. Palmetto is Clemson University's high-performance computing platform. Many nodes are equipped with NVIDIA Tesla GPUs, enabling us to work with more sophisticated models and higher parameter counts, increasing application performance and output reliability.

- **Multi-Query Translation:** The performance of a RAG application depends on the retriever's ability to find documents relevant to the user's query to provide context to the generator to produce an answer. Since queries are written by users, Multi-Query Translation is a technique to improve answer consistency by using an LLM to write multiple versions of the original query to reduce ambiguity. Retrieval is done with each rewritten question and the total retrieved documentation combined when provided to the generator.

Clemson Capstone Michelin Proposal

AJ Garner, Jacob Cox, Troy Butler, Caroline Baker, Carson Crockett

- **Logical Routing:** In cases where a RAG application includes multiple data stores, it becomes necessary to determine to which data store each question should be routed based on its content. This is often done by prompting an LLM with appropriate context to route the question. Since this application retrieves information from system documentation as well as source code, it is necessary to have separate data stores and route questions to each appropriately.

- **Instructor:** This is an embedding model with the ability to generate text embeddings for a variety of different domains (documentation, source code, etc.) for different tasks. We think this model is a good fit for this application because it is open source, can be run locally, works in a variety of embedding scenarios, and is easily adjusted by rewording the prompt associated with any embedding task.

Implementation Plan

- **Tentative team member roles**

    - **AJ Garner:** Team lead, COBOL advisor
    - **Jacob Cox:** LangChain, indexing and embedding, corrective retrieval, system documentation
    - **Troy Butler:** Language models, source document organization
    - **Caroline Baker:** Usage documentation, query translation
    - **Carson Crockett:** COBOL advisor, query routing

- **Sprint 1 (9/23 – 9/27): Problem research and solution design**
    - Examine the problem and assess possible solutions.
    - Compile a list of possible approaches for solving the problem.
    - Compare the strengths and weaknesses of each approach.
    - Assess the available tooling and libraries for the chosen approach.
    - Organize team members around logical components of the proposed solution.
    - Author project proposal report documenting the solution and its organization.
    - Begin the process of acquiring appropriate tooling and dev environments.

- **Sprint 2 (9/30 – 10/4): Tooling and Hello World**
    - Gather learning materials and example code for LangChain and other components.
    - Assign learning material to team members by role ownership for study.
    - Compile annotated learning material from team into central project repository.
    - Bring all team members into the Palmetto development environment.

- UML diagrams illustrating logical system organization.
- Set up individual components for a minimal RAG pipeline and test separately.
- Assemble individual components into a minimal RAG pipeline.
- Compile findings, annotated documentation, and issues encountered.

- **Sprint 3 (10/7 – 10/11): System Refinement, Testing, and Feature Roadmap**
  - TBD

Roadblocks and Solutions:

- **COBOL query translation and similarity lookup:** We anticipate one of our greatest challenges to be implementing a query translation system that can convert English user queries about the system to a form that can be compared against the embedded source code to retrieve relevant snippets from the vector store. Additionally, the router needs to determine when to direct queries to the vector store for documentation and when to direct them to the vector store for source code. This is especially challenging because we anticipate many instances where queries bound for different data stores share a high degree of similarity, and there may be instances where retrieval is required from both data stores.

  Proposed solution: We expect a substantial portion of the work done this semester will be on building a query translation and routing system suitable for our needs. Our current solutions are based on existing routing techniques that use an LLM and information about the available data to make retrieval decisions. This approach will be explored further in future sprints.

- **Response accuracy and performance testing:** Since this system needs to deliver highly accurate responses due to its critical role in business rule extraction, it is paramount that this system consistently produces responses that are highly accurate descriptions of the legacy system's actual behavior. Due to the hallucinatory nature of large language models, we expect difficulty attaining complete factual accuracy in responses from our early systems.

  Proposed solution: There are several well-known approaches to improving the accuracy of LLM output, especially in the context of an RAG system. The first and most promising approach is CRAG, or Corrective Retrieval Augmented Generation in which the retriever plays a more active role by using its own LLM to assess the relevance of the retrieved information concerning the original query. We plan to extend this idea further by using more LLMs to verify the accuracy of generator output and further back this effort with manual test cases made from known-valid business rules.