Architectural Design of LangChain Apps

Carlos Giai

September 2023

This document elucidates the architectural framework underpinning the construction of LangChain applications. Focusing on the intricate engineering principles employed, it offers insights into how these applications harness the capabilities of Large Language Models (LLMs) for comprehensive document analysis based on the nature of the queries, the documents to be analyzed, and the way LangChain's resources can be put together to optimize results. References to the sources of the LangChain API framework are also provided here.

Contents

Part I
Text Document Processing

1 Architecture Breakdown

1.1 Document Loading

1. Detect the type of document (PDF, TXT, PowerPoint, Website HTML, etc.).
2. Use appropriate libraries or methods to extract raw text and metadata from the document.
3. For file loading, LangChain has its proprietary loaders. Additionally, Doctran also provides some loaders.

1.2 Text Preprocessing

1. Depending on the document type, preprocess the text:
   - Remove headers, footers, ads, navigation menus, etc.
   - Handle tables, images, or charts if necessary.
   - Use tools like the RecursiveCharacterTextSplitter to chunk the input text for embeddings and vector storage.

1.3 File Property Extraction and Translation

1. Doctran offers file Property Extractors, Document Interrogation, and Document Translation capabilities.
2. Extracting metadata from documents can help in classification, data mining, and style transfer.
3. Document interrogation converts narrative documents into a Q&A format, increasing retrieval accuracy.
4. Doctran also provides tools to translate entire documents.

1.4 Embedding and Vector Storage

1. Convert each text chunk into a vector representation using an embedding method.
2. Store these vectors in a vector store for similarity searches.
3. LangChain offers custom output parsers.

4. There are various options for embeddings and vector stores, including OpenAIEmbeddings for embeddings and FAISS or Chroma for VectorStores.

5. Users have the option to utilize persist directories for vector stores.

1.5 Similarity Searches and Index Creation

1. For questions requiring specific details or comparisons, use stored vectors for similarity searches.

2. The similarity searches are primarily handled by the retriever.

3. A wrapper named VectorStoreIndexCreator encompasses embeddings, vector storage, text preprocessing, and retrievers.

1.6 Sources and Additional Tools

Below are some essential sources and tools related to LangChain's document processing capabilities. I added another source that specifies how to concatenate all of the aforementioned with a document comparison tools so as to ask these questions to two documents at a time:

References

[1] *Document Comparison Tool*, `https://python.langchain.com/docs/integrations/toolkits/document_comparison_toolkit`, How to implement these actions onto two documents at a time for comparison between the data inside of them.

[2] *RouterChain Paradigm in LangChain*, `https://python.langchain.com/docs/modules/chains/foundational/router`, A guide on the RouterChain paradigm in LangChain, which dynamically selects the appropriate chain for a given input.

[3] *Vector Stores and Searches*, `https://python.langchain.com/docs/modules/data_connection/vectorstores/`, Details on Maximum Marginal Relevance Searches and similarity_search_by_vector.

[4] *Retrievers in LangChain*, `https://python.langchain.com/docs/modules/data_connection/retrievers/`, Information on VectorStoreIndexCreation wrapper and its functionalities, along with RetrievalQA.

[5] *MultiRetrievalQAChain*, `https://python.langchain.com/docs/use_cases/question_answering/how_to/multi_retrieval_qa_router`, How to use the MultiRetrievalQAChain for question answering.

[6] *Doctran Extraction Properties*, https://python.langchain.com/docs/integrations/document_transformers/doctran_extract_properties, Using DoctranPropertyExtractor for extracting metadata for tasks like classification, data mining, and style transfer.

[7] *Document Interrogation with Doctran*, https://python.langchain.com/docs/integrations/document_transformers/doctran_interrogate_document, Converting narrative documents into Q&A format for improved retrieval.

[8] *Doctran Document Translation*, https://python.langchain.com/docs/integrations/document_transformers/doctran_translate_document, Tools to translate an entire document.

2 Semantic Analysis of the Input

Questions can often be classified more broadly into a few major categories. These overarching classifications help in making initial routing decisions for questions before delving into more nuanced categorizations. In light of simplification, two fundamental classifications are considered:

2.1 Descriptive (Factual) Questions

Nature: These questions seek direct, explicit information contained within the text. They don't necessarily require a deep level of analysis or inference but rather locate and relay the information.

Examples:

- "What does the text say about renewable energy?"
- "When did the described event happen?"
- "What does the text predict about the future of AI?"

Underlying Categories from Previous Lists: Specific Detail Questions, Temporal Questions, Predictive Questions.

2.2 Interpretive Questions

Nature: These questions require the system to analyze, infer, or draw conclusions based on the information in the text. They involve "reading between the lines" or analyzing the text's tone, sentiment, or deeper meanings.

Examples:

- "What is the main theme of the text?"
- "How does the author feel about the subject?"
- "Why does the author believe in this conclusion?"

Underlying Categories from Previous Lists: Qualitative Questions, Comparative Questions, Causative Questions, Opinion-based Questions.

2.3 Why we do this:

Our code will have to adopt an initial chain with good understanding of the above mentioned criteria (the two classes) to parse the inputs into either of these categories. The reason behind this is that the first category of questions, as you might have noticed, can be approached with an easy similarity search, whereas the second category needs a deeper and broader understanding of the text as a whole to make a reasonable judgement of the question and answer it will output.

We will later learn that for this type of questions (depending on their sub-category) we will need chains that run in parallel to provide summaries of each chunk to interpret in one go and give a good answer for. We can also sometimes concatenate this with a Maximum Marginal Relevance Search (MMRS) which allows for retrieving a number of the the most relevant chunks to the the input.

Part II
Tabulated Data Processing

3 Nature of the Queries

Queries on tabulated data such as CSVs, SQL databases, FITS files, can yet again be broadly categorized into two main types: **Mechanical Queries** and **Interpretive Queries**.

3.1 Mechanical Queries

Nature: Mechanical queries are deterministic in nature and involve direct data processing. They produce consistent outputs for the same inputs and typically involve data extraction, transformation, and computation.

Processing Mechanism: For each mechanical query, there is either a hard-coded function behind the scenes to handle it or, for simpler queries, a native function of the framework can be used. Examples of native functions include `create_pandas_dataframe_agent`, `create_csv_agent`, or `create_python_agent`. In fact, the last of these is capable of executing any code as long as the prompt for it is reasonably simple.

Categories:

- Extraction Queries: Direct extraction of specific data points or subsets.
- Aggregation Queries: Mathematical or statistical operations on the data.

- Comparative Queries: Direct comparisons within the data.
- Transformation Queries: Specific, defined data alterations or conversions.
- Metadata Queries: Providing structural information about the dataset.
- Visualization Queries: Generating visual representations of the data.
- Predictive/Modeling Queries: Complex computations, e.g., machine learning models.

3.2 Interpretive Queries

Nature: Interpretive queries require deeper analysis and context understanding. They often involve a combination of mechanical processing followed by qualitative interpretation. The responses are more subjective and may vary based on context, model understanding, or specific algorithms.

Processing Mechanism: Interpretive queries first utilize a chain of mechanical queries to extract necessary data. This extracted data, along with the initial qualitative query and column context, is then fed into a Large Language Model (LLM). This can be orchestrated using a `SequentialChain` or a `SimpleSequentialChain`.

Categories:

- Relationship Queries: Involves both mechanical correlation computations and deeper analysis of relationships.
- Contextual Predictive Queries: Beyond mechanical model predictions, understanding and explaining its implications.
- Qualitative Analysis: Combination of data analysis and natural language generation to articulate insights.

References

[1] *CSV Toolkit in LangChain*, `https://python.langchain.com/docs/integrations/toolkits/csv`, Description and functionalities of the CSV toolkit in LangChain.

[2] *Pandas Toolkit in LangChain*, `https://python.langchain.com/docs/integrations/toolkits/pandas`, Overview and capabilities of the pandas toolkit integration in LangChain.

[3] *Python Agent in LangChain*, `https://python.langchain.com/docs/integrations/toolkits/python`, Description and functionalities of the python agent and python REPL tool in LangChain.

[4] *Sequential Chains in LangChain* `https://python.langchain.com/docs/modules/chains/foundational/sequential_chains`, Application of Sequential Chains in LangChain

Part III
LLM-powered document edition

4 LLM powered document edition

When integrating the capabilities of LLMs with an AI framework that allows for file editing and shell command execution, the range of possible queries expands significantly. Given the increased complexity and potential risks, careful design, classification, and safety checks are essential.

4.1 Types of Queries

1. **Data/File Retrieval Queries**: Fetch specific files or data points.
   Example: "Fetch the configuration file for SExtractor."

2. **File Editing/Manipulation Queries**: Change, add, or remove content from a file.
   Example: "Update the configuration file to set the parameter 'DETECT_MINAREA' to 5."

3. **Command Execution Queries**: Instruct the system to run specific commands or scripts.
   Example: "Run the SExtractor with the updated configuration."

4. **Tool/Software Interaction Queries**: Specific to a particular software or tool.
   Example: "Configure SExtractor for a two-band extraction."

5. **Analysis & Interpretation Queries**: Seek insights or summaries post-execution or post-editing.
   Example: "Did the SExtractor run successfully?"

6. **Safety & Verification Queries**: Ensure operations are safe and intentional.
   Example: "Backup the original configuration file before making changes."

7. **Automation & Scripting Queries**: Involve creating, editing, or running automated tasks or scripts.
   Example: "Create a script to run SExtractor weekly with the latest configuration."

4.2 Advanced Operations

1. **Directory Operations**: Interact with the filesystem to create, delete, move, or list directories and files. Using the Shell Tool, users can seamlessly traverse the file structure.

2. **Custom OS Operations**: The framework allows users to inject their own operating system-specific operations, expanding its capabilities beyond the built-in functions.

3. **File Locator**: Leveraging an LLMChain with an output parser, the framework can find a file based on an initial path and the query input. For instance, a user might provide partial or complete file names, and the system locates it efficiently.

4. **Intelligent File Editing**: Once the desired file is located, the framework can interpret the user's request, understand the relationship between the document's content and the query, and make line-specific edits. For example, changing a parameter like 'MAG_MAX' to a specific value in a configuration file. Post-editing, the framework can even execute the file if requested.

4.3 Broad Categories

1. **Mechanical Operations**: Direct tasks involving file retrieval, editing, and command execution.
   Includes: Data/File Retrieval, File Editing/Manipulation, Command Execution, Tool/Software Interaction, Automation & Scripting.

2. **Interpretive & Safety Operations**: Tasks requiring analysis, verification, and interpretation.
   Includes: Analysis & Interpretation, Safety & Verification.

In practice, user queries may involve a combination of both categories. Given the potential risks associated with file editing and command execution, robust safety checks, user verification steps, and clear boundaries are essential. Especially, the usage of a `HumanApprovalCallbackHandler` might be of help. This callback allows for approval from a user before executing the commands.

References

[1] *Shell tool operations in LangChain* `https://python.langchain.com/docs/integrations/tools/bash`, Application of terminal operations, commands in the terminal and directory changes.

[2] *Human Approval Callback Handler* `https://python.langchain.com/docs/integrations/tools/bash`, Callback that allows for user approval before executing a tool with an AI agent.