DL Short Course Notes

Anri Lombard

2023

Table of Contents

LangChain for LLM Application Development

Introduction

The creator of LangChain is Harrison Chase, a former Harvard student.

LangChain is:

1. an open-source framework for building LLM applications with either Python or JavaScript
2. focussed on composition and modularity

It makes the process of making modular components for specific use cases easier.

Models, Prompts, and Parsers

Models refer to the LMM models that are used to generate text.

Prompts refer to the inputs that are passed into the model.

- Prompt templates allow for the reuse of good prompts to do specific tasks. Langchain has a set of template prompts that can be used.

Parsers refer to the output from the model that needs to be parsed into a format that can be used downstream.

- Prompt templates also support output parsing.
- Output is initially given as a long string, but can be parsed into a dictionary with LangChain.

Memory

Store past conversations in memory so that it has a more conversational flow.

- The way it stores memory is with ConversationBufferMemory.
- The LLM itself is stateless (it does not remember anything from the past). Wrapper code gives the full conversation so far as context to the LLM so it could generate a response. As a conversation gets longer the context gets longer and becomes expensive to provide. LangChain solves this with ConversationBufferMemory, which specifies in a parameter how many conversation steps should be remembered.

Memory Types

1. **ConversationBufferMemory**: allows for storing of messages and then extracts the message in a variable.
2. **ConversationBufferWindowMemory**: keeps list of interactions of the conversation over time and only uses last k interactions.
3. **ConversationTokenBufferMemory**: keeps buffer of recent interactions in memory and uses token length rather than number of interactions to determine when to flush interactions.
4. **ConversationSummaryMemory**: creates summary of the conversation over time.

Chains

Combines LLM with prompt.

- chain.run formats the prompt and feeds it to the LLM, then returns the result of the chain.
- The simple sequential chain is where there are multiple chains with a single input and single output each.
- For multiple inputs and/or outputs
- A router chain routes input to the correct chain

Question and Answer

LLMs can only inspect a few thousand words at a time. This is why we need to use embeddings and vector stores.

A **Vector store** is a database of vectors that can be queried for similar vectors.

An **Embedding** is a vector representation of a piece of text which allows us to compare it to other pieces of text.

A **Vector database** is a database of vectors that can be queried for similar vectors.

The **Stuff Method / Stuffing** is a method to simply stuff all data into a prompt as context to pass to the language model.

- Pros: It makes a single call to the LLM, then the LLM has access to all the data at once.
- Cons: Very large or many documents may cause a prompt to exceed the context length.

Additional methods:

1. Map_reduce: takes many chunks and their respective questions, then passes those to another language model to summarize the answers into a final answer.
    a. Very fast, but requires many calls
2. Refine: also takes the chunk approach, but builds upon the previous answer iteratively.
    a. Not as fast since answers are not independent, but rather depends on result of previous calls.
3. Map_rerank: single call for each document, then score the answers and pick the highest score as the final answer.

Evaluation

Applications using LLMs need to be evaluated with some criteria to test their accuracy.

One very effective way to do this is LLM-assisted evaluation (see notebook).

Agents

Langchain "agents" refer to a specific component of the Langchain system. An agent is essentially a stateless wrapper around an agent prompt chain, such as MRKL, and it plays a crucial role in the functioning of the system. The agent is responsible for formatting tools into the prompt, as well as parsing the responses that are obtained from the chat model. It receives user input and returns a response that corresponds to an "action" to take and a corresponding "action input"[1].

Furthermore, in some applications, an agent can be used when there's a need for not just a predetermined chain of calls to language models (LLMs) or other tools, but potentially an unknown chain that depends on the user's input. The agent has access to a suite of tools, and it can decide which, if any, of these tools to call based on the user input.

Conclusion

LangChain, as explored in this course by deeplearning.ai, empowers us to swiftly construct and deploy impressive applications powered by large language models (LLMs). It leverages the capabilities of state-of-the-art LLMs, combines them with unique functionalities like chains, agents, and custom prompts, and presents a streamlined interface for rapid development. While the course provides a solid introduction, it only scratches the surface of LangChain's full potential. There is a vast array of features and capabilities that remain to be discovered and harnessed, which could revolutionize the way we build and interact with LLM applications.