Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast

Article *in* International Conference on Applied Engineering and Natural Sciences · July 2023

DOI: 10.59287/icaens.1127

2 authors:

Oguzhan Topsakal
Florida Polytechnic University
40 PUBLICATIONS **278** CITATIONS

SEE PROFILE

T. Cetin Akinci
University of California, Riverside
134 PUBLICATIONS **917** CITATIONS

SEE PROFILE

All Sciences Proceedings

5th International Conference on Applied Engineering and Natural Sciences

July 10-12, 2023 : Konya, Turkey

All Sciences Proceedings
http://as-proceeding.com/

https://www.icaens.com/

Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast

Oguzhan Topsakal[1*], and Tahir Cetin Akinci [2]

[1]*Computer Science Department, Florida Polytechnic University, FL, USA*

[2]*WCGEC, University of California at Riverside, CA, USA*

[*]*(otopsakal@floridapoly.edu) Email of the corresponding author*

Abstract **–** This study focuses on the utilization of Large Language Models (LLMs) for the rapid development of applications, with a spotlight on LangChain, an open-source software library. LLMs have been rapidly adopted due to their capabilities in a range of tasks, including essay composition, code writing, explanation, and debugging, with OpenAI's ChatGPT popularizing their usage among millions of users. The crux of the study centers around LangChain, designed to expedite the development of bespoke AI applications using LLMs. LangChain has been widely recognized in the AI community for its ability to seamlessly interact with various data sources and applications. The paper provides an examination of LangChain's core features, including its components and chains, acting as modular abstractions and customizable, use-case-specific pipelines, respectively. Through a series of practical examples, the study elucidates the potential of this framework in fostering the swift development of LLM-based applications.

Keywords – Large Language Models, LangChain, Concepts, Application, ChatGPT, NLP, GPT

I. INTRODUCTION

The past decade has witnessed an unparalleled evolution in the realm of artificial intelligence (AI). This period, characterized by the ascendancy of deep learning through the utilization of neural networks, has resulted in significant enhancements in capabilities pertaining to image and speech recognition. One salient milestone highlighting this progress is the ImageNet Large Scale Visual Recognition Challenge, which effectively demonstrated the prowess of AI capabilities in image recognition. [1][2]

Another major milestone in the AI landscape is the successful implementation of reinforcement learning, as exemplified by DeepMind's AlphaGo and AlphaZero. [3] These innovations have demonstrated extraordinary performance in complex games, such as Go and Chess, using self-play algorithms, thereby signifying a leap forward in reinforcement learning techniques. In parallel, the evolution of generative models has facilitated the creation of convincingly realistic synthetic multimedia content.

During the same period, the field of natural language processing (NLP) experienced remarkable transformations. The advent of advanced models, exemplified by the likes of BERT (Bidirectional Encoder Representations from Transformers) by Google [4] and GPT (Generative Pretrained Transformer) by OpenAI [5], and T5 (Text-to-Text Transfer Transformer) by Google [6] has fostered significant improvements in machine translation, sentiment analysis, and text generation, thus ushering in a new era for NLP. BERT, GPT, T5 and similar technologies all utilized transformers architecture and were trained on huge amount of data and hence named as Large Language Models (LLMs). [7] As LLMs were trained using more and more data, and encompassed more parameters, their capabilities increased. For example, GPT-1 (June 2018), GPT-2 (February 2019), and GPT-3 (June 2020) had 117

million, 1.5 billion, and 175 billion parameters, respectively.

A large language model (LLM) is a subtype of artificial intelligence model that generates text with human-like proficiency. [7] Characterized by a sizable number of parameters and trained on expansive text corpora, these models are equipped to produce contextually pertinent and grammatically coherent outputs.

Utilizing machine learning techniques such as deep learning, these models are trained to predict subsequent words in a sentence based on prior context, thereby enabling the generation of comprehensive sentences and paragraphs that bear a resemblance to human-authored text.

Even though the LLMs present limitations, such as occasionally producing erroneous or illogical outputs (also called hallucination), they achieved rapid success due to their performance in doing various tasks such as composing essays, writing, explaining, and debugging code. The recent OpenAI's LLM, ChatGPT, made the technology known to most, acquiring millions of users in a short amount of time. The capabilities of GPTs have become even more impressive with the release of GPT4. [8].

Many started to think about how to leverage this technology to provide solutions for fields like education, research, customer service, content creation, healthcare, entertainment, etc. It became possible to develop AI applications much faster than ever before by interacting with an LLM. However, custom AI apps require more than just interacting via a web interface. A recent open-source software library called LangChain, started providing solutions for the steps of developing a custom AI app utilizing LLMs and gained much attention from the AI community. [9][10] In this article, we describe the capabilities of LangChain and provide a primer on developing large language model applications rapidly utilizing LangChain.

II. Materials and Method

LangChain is a framework for developing applications utilizing large language models, and its goal is to enable developers to conveniently utilize other data sources and interact with other applications. To enable this, LangChain provides components (modular abstractions) and chains (customizable use case-specific pipelines). We first provide an overview of components and then describe several use cases.

A. Components

Next, the main components of LangChain, such as Prompts, Memory, Chains, and Agents are explained.

A.A.1 Prompts

A "prompt" is the input to a LLM. They are generally generated dynamically when used in an LLM application and includes user's input (question), a set of few shot examples to help the language model generate a better response, and instructions for the LLM regarding how to process the input that comes from the user. LangChain provides several classes to construct prompts utilizing several specialized Prompt Templates. A prompt template refers to a reproducible way to generate a prompt. It contains a text string ("the template") that can take in a set of parameters from the end user and generates a prompt.

Table 1. Examples of prompts. The examples are adapted from the DeepLearning.AI course on LangChain.

Task	Prompt
Extracting information	For the following text, extract the following information: gift: Was the item purchased as a gift for someone else? Answer True if yes, False if not or unknown. delivery_days: How many days did it take for the product to arrive? If this information is not found, output -1. price_value: Extract any sentences about the value or price. text: {text}
Writing a response	Write a follow-up response to the following summary in the specified language: Summary: {summary} Language: {language}

A.A.2 Models

Large Language Models (LLMs) are the main type of models utilized in LangChain. They accept a text string (prompt) and output a text string. There are other types of models that are used in LangChain, namely, *Chat Models* and *Text Embedding Models*. Chat Models have more structured API processing chat messages, and text embedding models take text and return its corresponding embedding as a list of floats. The embeddings are required when we want to work

with our own documents, as discussed in the following Question Answering from Documents section.

A.A.3 Chains

The most important key building block of LangChain is the chain. The chain usually combines an LLM together with a prompt, and with this building block, you can also put a bunch of these building blocks together to carry out a sequence of operations on your text or on your other data.

A simple chain takes one input prompt and produces an output. Multiple chains can be run one after another, where the output of the first chain becomes the input of the next chain. Multiple chains can be concatenated using the Simple Sequential Chain class when there is one input and one output, as illustrated in Fig 1.

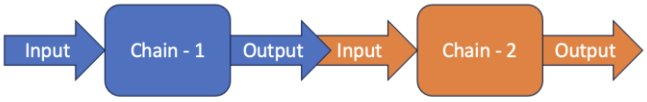

Fig. 1 Example of a Simple Sequential Chain

LangChain provides another class named SequentialChain, when there can be multiple inputs but one output, as illustrated in Fig 2.

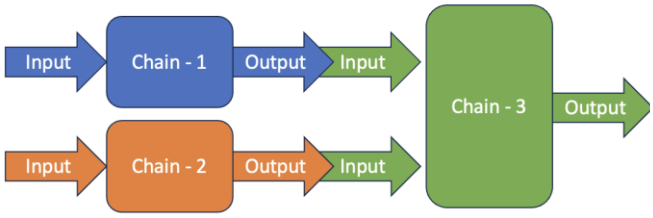

Fig. 2 Example of a Sequential Chain that gets two inputs and outputs one result.

A pretty common scenario is to use several chains and to route an input to a chain depending on what the input is. For example, if you have multiple chains, each of which specialized for a particular type of input, you could have a router chain which first decides which subchain to pass it to and then passes it to that chain. An example of router chain is depicted in Fig 3.

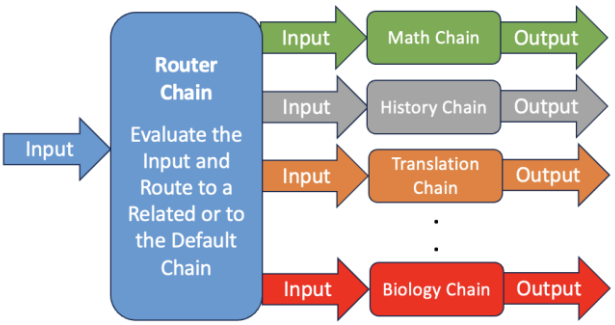

Fig. 3 Example of a Router Chain that routes the input to a chain that can answer the input the best.

Each chain can utilize a different or the same LLM and would be differentiated mostly based on their prompt. The prompts that go into an LLM as input include a description of the role of the chain. The input coming from the user is combined with the description of the expected output to form the prompt so that the chain behaves as expected. Table 1 gives examples of these prompt descriptions that are used along with the user inputs to let LLM produce a result.

Table 2. Examples of chain prompts. The examples are adapted from the DeepLearning.AI course on LangChain.

Name	Description
Math	You are a very good mathematician. You are great at answering math questions. You are so good because you can break down hard problems into their component parts, answer the component parts, and then put them together to answer the broader question.
History	You are a very good historian. You have an excellent knowledge of and understanding of people, events, and contexts from a range of historical periods. You can think, reflect, debate, discuss, and evaluate the past. You have respect for historical evidence and the ability to make use of it to support your explanations and judgments.

A.A.4 Memory

The language model itself is stateless and does not remember the conversation you've had so far. Each transaction (call) to the LLM's API endpoint is independent. The illusion of memory in chatbot systems is facilitated by supplementary code, which incorporates the context of preceding dialogues when interacting with the LLM. A *memory* component is needed that can store the previous conversations and pass them to the LLM with the next prompt.

In the LangChain framework, memory implementation can take multiple forms. The

'buffer' type delineates memory bounds based on a set quantity of conversational exchanges, whereas the 'token' type imposes limitations contingent on the number of tokens. The 'summary memory' type applies an abstracted summary of tokens when a certain threshold is exceeded. Beyond these memory types, developers have the discretion to archive the entire conversation within a conventional database or a key-value store. This can be beneficial for auditing purposes or to enhance system performance through reflective analysis of past interactions.

A.A.5 Question Answering from Documents

One of the most common and complex applications that are being built using an LLM is a system that can answer questions on top of or about a document. Given a piece of text, extracted from a PDF file, a webpage, a csv file or another type of document, apps can be developed to use an LLM to answer questions about the content of those documents to help users gain a deeper understanding and get access to the information that they need. Once LLMs can be utilized to process data that they were not originally trained for, there are many use cases that can be achieved.

There are various steps involved in answering questions from own documents, as depicted in Fig 4.

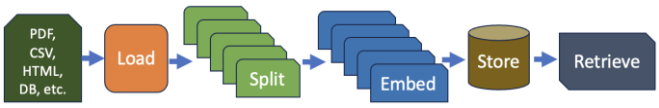

Fig. 4 Steps to answer questions from own documents.

LangChain provides functionality to easily achieve these steps to load, transform (split), store, and query your data. LangChain has the following classes that help to perform these functionalities.

- *Document Loaders:* Loads documents from many different sources, including CSV, PDF, HTML, JSON, Excel, GitHub, Google Drive, One Drive, XML, Wikipedia, and many more.
- *Document transformers:* Splits documents into smaller chunks so that they can be processed by LLMs.
- *Text embedding models*: Take unstructured text and turn it into a list of floating-point numbers that represent corresponding embeddings.
- *Vector stores*: Helps to store and search over embedded data.
- *Retrievers*: Helps to query your data based on embedding similarities.

Embeddings generate quantitative representations for textual units, encapsulating their semantic essence in a numerical format. When applied to similar textual content, these embeddings yield closely aligned vectors. This enables an evaluation of textual similarity within the vector space, facilitating an intuitive understanding of textual coherence. This becomes particularly instrumental when selecting text pieces to feed into a language model for query resolution.

These embeddings are stored in vector databases. When a user query comes in, it is converted into embeddings and then sent to the vector database to find the most semantically close text based on the similarity scores computed by getting a dot product of the query embedding and the embeddings stored in the vector database. The dot product can also be expressed as the product of the magnitudes of the vectors and the cosine of the angle between them, as shown in the following equation, where a and b are the vectors being compared [12]:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}|\,|\mathbf{b}|\,cos\alpha$$

There are several methods to pass the content of the documents and process them via LLM. The most used method is the stuff method which is simple and works well if the text chunks that were returned from the vector store similarly function fit into the context size of the LLM. The Fig 5. depicts the stuff method where retrieved similar documents are passed to LLM in a single prompt.

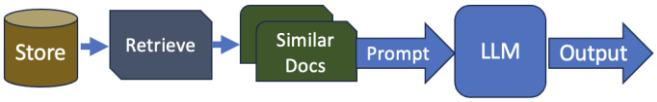

Fig. 5 Stuff method passes similar documents in a single prompt.

The other methods that can be used in processing documents when the retrieved documents do not fit in a single prompt are “map reduce,” “refine,” and “map rerank.”

Map reduce takes all the chunks, passes them along with the question to a language model, gets back a response, and then uses another language model call to summarize all of the individual responses into a final answer. Map reduce can

operate over any number of documents and can process individual questions in parallel. However, it requires more calls and does treat all the documents as independent, which may not always be the most desired thing. Map reduce process is depicted in Figure 6.

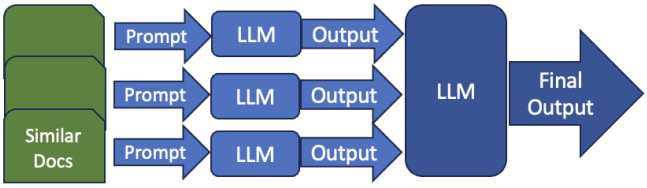

Fig. 6 Map reduce passes similar documents to multiple LLMs and then uses another LLM to finalize the output.

Refine is used to loop over many documents iteratively, building upon the answer from the previous document, which generally leads to longer answers. However, many calls are required, and the calls cannot be done in parallel, causing slower execution. The refine methods is depicted in Fig 7.

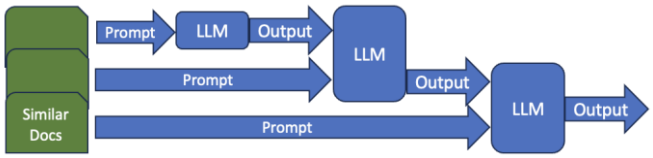

Fig. 7 Refine method iteratively builds the answer.

Map rerank does a single call to the language model for each document and asks for a score. Then it selects the document with the highest score. The map rerank method is shown in Fig. 8.

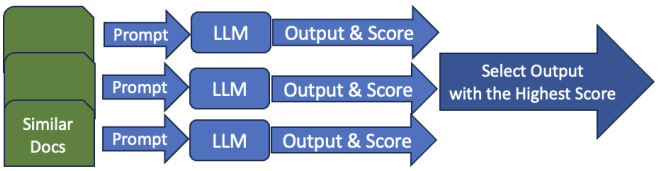

Fig. 8 Map rerank method gets an output score and then selects the output with the highest score.

A.A.6 Agents

When an app needs a flexible chain of calls to LLMs and other tools based on user input, agents can be utilized. An agent determines which tool from a suite of tools needs to be used for the user input. An agent can be either an *action agent* that decides on the next action using the outputs of all previous actions or a *plan-and-execute agent* that decide on the full sequence of actions upfront, then execute them all without updating the plan.

- *Action Agent* operates at a high level by receiving user input, determining the appropriate tool and its input, executing the tool and recording its output (termed as an 'observation'), and making decisions on subsequent steps based on the history of tool usage, inputs, and observations. This cycle repeats until the agent can directly respond to the user. These agents are encapsulated in agent executors that manage the sequence of actions and interactions with the tools.
- *Plan and Execute Agent* operates at a high level by receiving user input, devising a comprehensive sequence of steps, and implementing these steps in a sequential manner, where outputs from previous steps are utilized as inputs for subsequent ones. A common implementation involves using a language model as the planner and an action agent as the executor.

Within the framework of agent-based systems, the concepts of tools and toolkits play a significant role. Tools, defined as interfaces that facilitate agent-world interactions, constitute specific actions that an agent can perform, purposefully selected based on the agent's functional objective. Conversely, toolkits represent aggregated collections of synergistic tools, assembled to cater to particular use cases. Toolkits are specifically designed to consolidate tools that function effectively in unison for distinct tasks, and they encompass convenience methods for easy loading. For instance, an agent interfacing with a SQL database would necessitate a tool for executing queries and another for inspecting tables.

B. Use Cases

LangChain framework provides walkthroughs of common end-to-end use cases on the topics such as autonomous agents, chatbots, code understanding agents, extraction, question answering over documents, summarization, and analyzing structured data. Each of these categories provides several examples of how to utilize LangChain to implement the LLM app using Langchain. For example, the AutoGPT sample given under the ‘autonomous agents’ category provides a notebook implementing AutoGPT using LangChain that

aims to autonomously achieve whatever goal is given. An increasing number of examples of LangChain use cases are documented at the LangChain website. [12]

III. DISCUSSION

The rise of Large Language Models (LLMs), specifically OpenAI's ChatGPT, signifies a transformative phase in AI development with broad-ranging applications. LangChain, an open-source library, stands out due to its proficiency in integrating with diverse data sources and applications, making it an influential tool in the AI community.

LangChain's modular structure, with customizable pipelines for specific use cases, expedites the development process for LLM applications. This paper contributes to the discourse on LLM application development, aiming to spur further exploration of LangChain and similar tools. To help facilitate the development of LLM applications utilizing LangChain, we provide a sample Jupyter Notebook page showcasing many of the concepts described here at a GitHub page. [13]

IV. CONCLUSION

In conclusion, the advent and rapid adoption of Large Language Models, underscored by the rise of OpenAI's ChatGPT, signify a new frontier in the AI landscape. These models have exhibited proficiency across a multitude of tasks, setting a precedent for future advancements.

Particularly, LangChain, with its ability to streamline the development process of LLM applications, has demonstrated significant potential in the AI ecosystem. It has pioneered a new approach that enables developers to interact with various data sources and applications effortlessly. This flexibility and efficiency, characterized by LangChain's modular abstractions and customizable use-case-specific pipelines, make it an invaluable tool for the future of LLM applications.

This paper provides insights into LangChain's structure and usage in specific use-cases, elucidating its capacity to foster rapid development. It is hoped that the potential and capabilities identified will spur more exploration and innovation in the field of LLMs and serve as a basis for the development of more sophisticated applications, thus expanding the boundaries of what is possible with artificial intelligence.

REFERENCES

[1] OlgaRussakovsky, JiaDeng, HaoSu, JonathanKrause, SanjeevSatheesh, SeanMa, Zhiheng, Huang, AndrejKarpathy, AdityaKhosla, MichaelBernstein, et al. Imagenet large scale visual recognition challenge. IJCV, 2015.

[2] Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.

[3] Sean D. Holcomb, William K. Porter, Shaun V. Ault, Guifen Mao, and Jin Wang. 2018. Overview on DeepMind and Its AlphaGo Zero AI. In Proceedings of the 2018 International Conference on Big Data and Education (ICBDE '18). Association for Computing Machinery, New York, NY, USA, 67–71. https://doi.org/10.1145/3206157.3206174

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[5] Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018). Improving language understanding by generative pre-training.

[6] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, & Peter J. Liu (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research, 21(140), 1-67.

[7] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... & Wen, J. R. (2023). A survey of large language models. arXiv preprint arXiv:2303.18223

[8] Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., ... & Zhang, Y. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. arXiv preprint arXiv:2303.12712.

[9] Chase, H. LangChain LLM App Development Framework. https://langchain.com/ Accessed Jul 10th, 2023

[10] Chase, H. LangChain, Building applications with LLMs through composability, GitHub Repo, https://github.com/hwchase17/langchain Accessed Jul 10th, 2023

[11] Vector Similarity Explained, The Pinecone Vector DB, https://www.pinecone.io/learn/vector-similarity/, Accessed July 10th, 2023.

[12] LangChain Use Case Examples, https://docs.langchain.com/docs/category/use-cases , Accessed July 10th, 2023

[13] GitHub page for sample Jupyter Notebook page showcasing usage of LangChain framework, https://github.com/research-outcome/llm-langchain-examples Accessed Jul 10th, 2023

This is the text I want to add.