Large Language Model Training Human

Yu Wu, Yuhang Cai, Ani Yusuf

Technical University of Munich

Abstract

We have developed a chatbot utilizing the large language model that focuses on the culinary field. Unlike traditional recipe resources, our robot places a greater emphasis on imparting cooking knowledge and assisting users in learning about food culture, making it an educational software. Full code see our gitlab program repository: https://gitlab.lrz.de/lab-courses/nlp-lab-ws2023/teams-tobias/language-models-training-humans

1 Introduction

As large language models continue to evolve, their knowledge bases become increasingly comprehensive, enabling intelligent chatbots like ChatGPT to answer human queries across a wide range of subjects. In this sense, large language models can also transform into digital tutors for humans, imparting knowledge in various fields. Against this backdrop, our team, which boasts a diverse culinary culture and has huge enthusiasm of cooking topics, decided to venture into cooking as a starting point, leading to the development of a specialized intelligent assistant application for cooking tutorials named "The Chef." In this application we use langchain to build tools and chains by visiting OPEN-AI model so that the assistant can use them to answer the question reasied by the users.

2 Technology

2.1 OpenAI

Compared to other models such as Google Bert, Microsoft Transformer and AllenNlp, OpenAI GPT-4 models have better performance in accuracy (a mean average precision of 95.1% on benchmark datasets), speed (an average processing time of 250ms per text query), scalability(OpenAI models can handle large amounts of text data, making them highly scalable). Besides in real-world case, OpenAI models have been used for Question Answering, and Text Generation. Therefore we use OpenAI model (GPT-4-Turbo) for our application.

Furthur, we want our models to generate pictures, we use Dalle with 12-billion parameter version of GPT-3, which is trained to generate images from a text description using same transformer architecture. Dalle receives both the text and the image as a single stream of data containing up to 1280 tokens and is trained using maximum likelihood to generate all of the tokens, one after another. This will be helpful if we want to generate correlated pictures the contents.

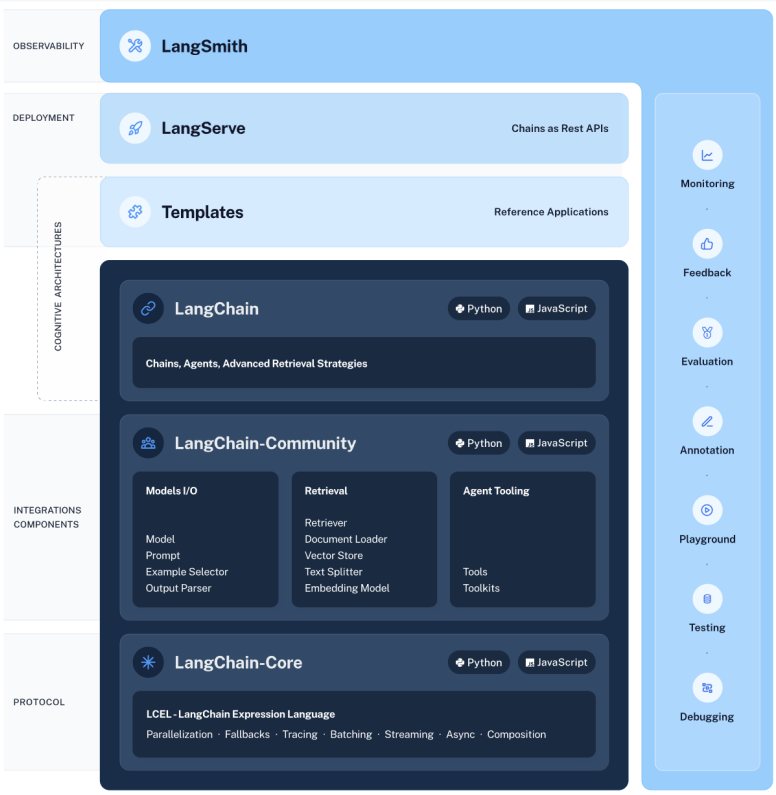

Figure 1: Langchain sturcture

2.2 Langchain

LangChain is a framework designed to simplify the creation of applications using large language models (LLMs). Compared with other tools, Langchain holds several advantages such as has more flexible and configurable interface, simpler memory structure and is more generic through retrievers. Langchain provides varieties of API and toolkits as the picture 1 shows [2].

What we use at most are agent, chain and memory:

- Agent: Agents in LangChain are built to interact with the real world. They are powerful tools that automate tasks and engage with real-world scenarios. LangChain agents can be used for a variety of tasks such as answering questions, generating text, translating languages, summarizing text, and more. Agents in LangChain use an LLM to determine which actions to take and in what order, which plays a role of central control unit In our model, we use Zero-shot ReAct agent which can create realistic contexts even without being trained on specific data and be used for various tasks such as generating creative text formats, language translation, and generating different types of creative content.

- Chain: Chains refer to sequences of calls - whether to an LLM, a tool, or a data preprocessing step. The primary supported way to do this is with LCEL(LangChain Expression Language). The most basic form of chain within this system is the LLM-Chain and its operation involves a structured arrangement, including a PromptTemplate, an OpenAI model (either a Large Language Model or a ChatModel), and an optional output parser. Within this setup, the LLMChain accepts various input pa-

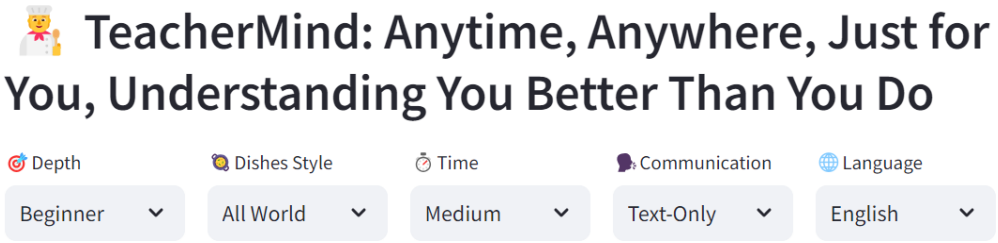

Figure 2: User configurations setting in the UI

rameters. It employs the PromptTemplate to transform these inputs into a coherent prompt. This polished prompt is then inputted into the model. After receiving the output, the LLMChain uses the OutputParser, if provided, to further refine and format the result into its ultimate usable form.

- Memory: An essential component of a conversation is being able to refer to information introduced earlier in the conversation. At bare minimum, a conversational system should be able to access some window of past messages directly. A more complex system will need to have a world model that it is constantly updating, which allows it to do things like maintain information about entities and their relationships. In langchain we use the ConversationSummaryBufferMemory to remember the conversation held before, so that it will not take up too many tokens.

2.3 Streamlit

For the UI we choose Streamlit as it is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps. Since we have no front-end (html, js, css) experience or knowledge to build web apps. Instead, we want a tool that is easier to learn and to use, as long as it can display data and collect needed parameters for modeling. Streamlit perfectly allows us to create a stunning-looking application with only a less lines of code. In this streamlit we mainly use **st.session_state** as our global variable memory to store data in database.

3 Workflow

The user interface of our program is shown in Figure 2. Through this interface, users can set their preferences. These personalized settings are not only read by the agent as part of the prompt but are also stored in the database (see Figure 3). Based on this information, our chatbot can retrieve relevant data from the database and provide appropriate responses accordingly.

In addition, there is a text input box on the interface where users can elaborate on their needs. Based on the user's input, the agent selects the most suitable tool to answer, with the selection process based on the descriptions we wrote when defining each tool. These descriptions detail the applicable scope of the tools, the required input format, and other key information.

In LangChain, tools are also regarded as executable functions, with their output being the return value of the function. The process of executing these functions is

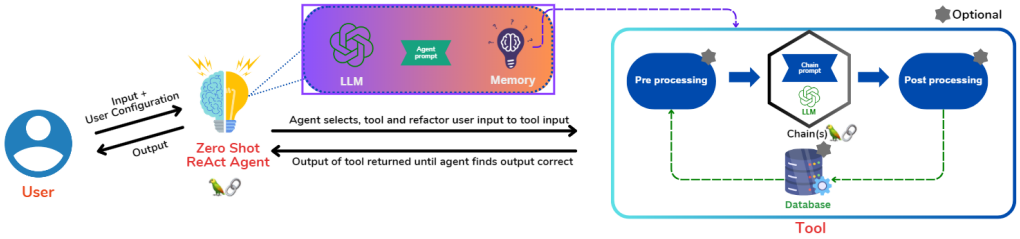

Figure 3: Workflow

mainly carried out through various chains. We also add some pre-processing and post-processing steps in the functions to optimize the output. For example, since the output format of the chains is limited to plain text without any formatting, we have to manually separate the titles from the main text. When a function (i.e., tool) completes execution, its return value is presented to the user as a reply through the agent.

4 Application structure

Our cooking assistant boasts 6 key functionalities, achievable through the selection of different tools:

- **Curriculum generating tool:** it can create personalized teaching courses based on the user's interest topics;
- **Teaching tool:** it generates detailed content for each course module for instructional purposes;
- **Evaluation tool:** it offers an assessment feature that creates multiple-choice questions based on the instructional content, allowing users to answer by clicking on the options;
- **Analyze tool:** based on the answers submitted by users, it analyzes the mastery level of each module and provides learning suggestions;
- **Q&A tool:** it answers user queries with easy-to-understand responses;
- **Chat tool:** when a user's intent is unclear, it proactively asks for more information to obtain clear directions.

4.1 Curriculum Generating Tool

This feature is primarily executed through the curriculum generating chain. By crafting prompt words, we instruct the large language model to generate a course preview according to specified requirements and formats, which includes titles and summaries for each module. These details are stored in the course records on the left side of the page, readily accessible for users to review at any time, as shown in Figure 4.

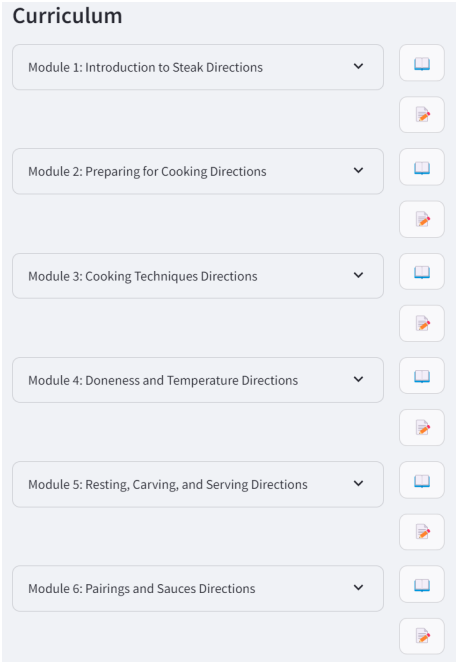

Figure 4: An example of curriculum overview

4.2 Teaching Tool

When users wish to delve deeper into the content of a specific module, they can directly click on the book icon next to the corresponding module in Figure 4. To accommodate different learning preferences, we have designed three learning modes, each implemented through three distinct chains. Firstly, the teaching chain is responsible for generating detailed content for each module, taking the module overview generated from the prompt words and user preferences as input variables. Secondly, to help reinforce memory, our flashcard chain can create flashcards for key information within the course content. Lastly, to aid in users' understanding of the course material, we utilize the DALL · E model provided by OpenAI to generate relevant images for the module content.

4.3 Evaluation Tool

School exams serve not only to assess students' learning outcomes but also to enhance their motivation to learn. Similarly, our program offers an assessment feature that can create multiple-choice questions based on module content, requiring users to simply click a specific icon next to the module in Figure 4. The large language model generates an appropriate number of test questions based on the extent of the module content, designed in HTML format for ease of answering by clicking options. This process is also implemented through a chain, for which we have crafted specific prompt words, using the content of the corresponding module *{module_content}* as the input variable. Outputs provided by the large language model are stored in the database and then converted into HTML format through some post-processing steps. Finally, we added a submit button to this feature, allowing the model to offer

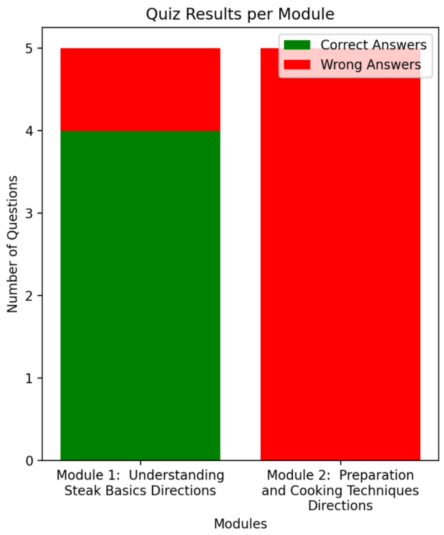

Figure 5: An example of chats in the analyze feature

feedback comparing the user's answers with the correct ones.

4.4 Analyze Tool

Following the end of the school exam season, receiving report cards often comes with teachers' comments. Similarly, our program offers this feature. After submitting test answers, users can click on the analysis button, and the large language model will analyze based on the accuracy of the answers, highlighting areas for improvement and offering learning suggestions. To visually represent each module's learning status, we have also created chart analyses, as shown in Figure 5. The implementation process is similar to before, starting with preprocessing the submitted answers to calculate the user's scores, then saving this data in global variables. Using prompt words based on the scores, a learning evaluation is generated. Finally, learning data is read from the database and processed to produce charts.

4.5 Q&A Tool

Given that our chosen GPT-4 model is a pre-trained model with inherent question-answering capabilities, it does not require any additional operations for this functionality. Therefore, within this tool, there is no complex processing involved; our contribution simply involves adding memory to the feature. This is because if the model lacks memory of previous conversations, it might struggle to understand the user's questions accurately, leading to irrelevant responses. For detailed instructions on how to add memory to the model using LangChain, refer to [1].

4.6 Chat Tool

Considering situations where the model may not understand user input or is uncertain about the user's requested action, we have specifically designed an inquiry feature for our program. For instance, if a user's input is empty, the model will politely inquire about the user's needs and guide them towards the next steps. This approach not only prevents the model from wasting time and resources on incorrect operations but also helps to reduce inaccuracies in feedback, thereby more accurately identifying the user's intentions.

5 Limitations

First of all, the problem of LLM hallucination still exists. Here the hallucinations refer to the generation of content that is irrelevant, made-up, or inconsistent with the input data. This problem leads to incorrect information, which will challenge the trust placed in models. the reason might arise from the training data's quality and this will affect the models' interpretative limits heavily.

Secondly, we do not use fine-tuning techniques to better adapt the LLM to our task. We only finish the whole sturcture of this applicaiton. It would be better if we try different parameters and prompts to evaluate different outcomes. Then we might get more trusted and suited version of this chatbot.

Besides, long response time is also a problem when generating the curriculum and answer rasied by the user. One possible way to solve this problem might be that we could store some generated answers in database, so that later it can directly retrive correlated answers instead of visiting OpenAI model by using API key each time.

Lastly, the context length, the number of tokens a language model can process at once, also limits the performance of our apllication. Hence the complexity of input and the meory coherence can be violated. In fact we have to restart this application when faced with too many tokens.

6 Conclusion

LLMs offer a transformative approach to education, enabling personalized learning experiences that reduce the reliance on traditional teaching methods and physical interactions. By facilitating self-directed learning, particularly in specialized areas such as culinary education, LLMs allow learners to explore subjects at their own pace and according to their interests. The potential for LLMs to provide a comprehensive learning experience, from technical skills to cultural appreciation, sets a promising direction for future research, especially in evaluating the effectiveness of such technology-assisted learning.

References

[1] Langchain documents. https://python.langchain.com/docs/modules/memory/.

[2] Langchain introduction. https://python.langchain.com/docs/get_started/introduction/.