# LLM chatbot for Liquor recommendation

Project 2

## 1. Introduction

Alcohol culture has a certain barrier for new comers, especially in various social occasions. When faced with dazzling brands and complex labels of alcohols, consumers may find it difficult to quickly determine on suitable products that can satisfy both their tastes and their social needs.

Meanwhile, professional alcohol websites lack convenient recommendation systems. On one hand, beginners need to face a large number of professional terms rather than a system that can understand their demands and provide suggestions in comprehensible languages. On the other hand, consumers have to waste their time on browsing through the webpages until finding a satisfying result rather than getting everything done with a few commands.

Our solution to the above problems is to build a liquor recommendation platform in the form of a chatbot. Our recommendation system will focus on cocktails because the labels of cocktails are often confusing and we believe our cocktail chatbot can help explain them. Also, cocktails can cater to the needs of a wider range of consumers given their richer social value due to their diverse appearances and production processes. Our key components include the following parts:

- Frontend: a cocktail recommendation system with an interactive webpage
- Backend: the complete chatbot code based on LLM
- Data: data used for fine-tune and RAG method

The major part of our implementation that focused on LLM is the backend development of the chatbot. For the chatbot development, we used two different LLM methods, fine-tuning and RAG. We used fine-tuning based on the TinyLlama-1.1B model, and we found that the fine-tuned chatbot would generate correct answers in logical formats. Apart from it, through RAG methods, answers can be generated from an external LLM API equipped with a vector database, with minimal device limitations and easy modification of data formats.

For deliverables, users can ask the chatbot for cocktail recommendations through an html webpage. Besides the chatbot function, they can use a search engine which is linked to an external website to search for related information, and even save their favorite products to a collection page for later reference.

In summary, we developed three interactive functions for our cocktail recommendation system, and we ensured ease of use with a simple and clear interface. To go further in business, our system can be integrated with websites owned by major liquor companies to enhance customer experience and loyalty.

# 2. Methodology

## 2.1 Front-end design

Using Cascading Style Sheets code, we designed the front-end interactive page. We set the background color, calligraphic style, and card style in the webpage and made it easy to operate and user-friendly through different settings:

- Distinguish different interaction situations by button colors: the main button *(. btn-primary)* is blue, green represents successful operation *(. btn-success)*, and yellow represents operation warning *(. btn-warning)*
- Enhance user experience through mouse hover effects *(transform: translateY (-5px))*
- Well displayed on different screens: defining a container class *(. container)* with a maximum width of 800px.

## 2.2 Backend design

The backend application is designed to process interactions from the front-end webpage. Based on Flask, it is a complete cocktail chatbot, which includes the three major functions mentioned in the introduction: chatbot *("Ask About Cocktail")*, search engine *("Search Cocktail")*, and favorite collections *("Favorite")*.

### 2.2.1 Fine-tune method

Chatbot:

- The chatbot can recommend suitable product names with brief introductions based on the user's query. Through the */ask* function, an API interface is established and collects front-end queries. The back-end code generates answers based on the *TinyLlama-1.1B-Chat-v1.0* model, with a maximum token count of 200.

Search engine:

- This function links to the URL for a cocktail website: *https://www.thecocktaildb.com/api/json/v1/1/search.php?s={cocktail name}*. The backend receives queries through */search_comtail* and collects website information through the API. Through the external information, users can have more understanding of the suitable cocktails which the chatbot recommends. This function aims to facilitate instant search and supplement the answer of the chatbot.

Favorite collections:

- As a personalized functionality, the goal is to increase the usability of the recommendation system by enabling users to mark their favorite cocktails. We imported sqlite3 to create a database, set */add_favorite*, */remove_favorite* and */favorites* API routes with *'GET'*, *'POST'*, and *'DELETE'* methods to display, add, and delete contents in the "Favorite" page.

During the fine-tuning process, due to the limitations of CPU operation of our device, we defined two sets of fine-tuning dialogues. We converted the format of the two dialogues into IDs and attention masks, and set the parameters *epoch=10*, *batch_size = 1*, *learning_rate = 5e-6*. The fine-tuning time was about 35 minutes.

### 2.2.2 RAG method

Since RAG is flexible for device limitations and allows us to use more data, we first processed the data based on *Cocktails.csv* to capture the meaning of texts. The processing procedure is as follows:

- Segment cocktail information into multiple chunks by the *BAI/bge-based en-v1.5* model, then capture vector information.
- Store the cocktail vectors in a newly created SQLite database.

For the three front-end functions, we implemented them as follows:

Chatbot:

- Compared with the fine-tuning method, instead of using the 1.1B model, we employed a prompt engineering approach by *deepseek-chat-v3-0324:free* model. Then, the output format has been standardized in the main program file on the backend.
- For the answer query, we used both the vector databases through an embedding model named *BAAI/bge-base-en-v1.5* and an API interface of Deepseek model named *openrouter.ai* by using URL (*https://openrouter.ai/api/v1/chat/completions*) .

Search engine:

- We classified text based on semantic information captured by vectors, stored information using SQL statements, and displayed information through SQL queries and API calls.
- In the searching process, we first considered the matching of the exact name. If the exact name could not be matched, we would do the search under vector similarity, including overall text similarity, cocktail ingredient similarity, and cocktail menu similarity. Meanwhile, unified the format of the search results.

Favorite collections：

- The logic has not been changed because of its characteristics.

**2.3 Challenges**

The main challenge we faced during the implementation stage was to find a balance between the training cost and chatbot response accuracy for the fine-tuning approach. In order to reach a reasonable training time and a desirable level of accuracy, we needed to decide on the proper size of our training dataset and set appropriate numbers to the fine-tuning parameters (e.g., epoch, batch size, and learning rate).

**2.3.1 Training Data Size**

Initially, we collected and processed around 30,000 liquor reviews from Kaggle and tried to use them to fine-tune our selected model. However, since we only have an Intel-I7-14700 CPU for training, we could not afford the computational cost to fine tune the model with such a large amount of data. Therefore, we tried to reduce our training data and asked ChatGPT to generate some representative liquor recommendation data for us. The generated data has the format as in *figure 1*, and we expected that these communication like question-and-answer pairs can instruct the model to answer in a similar tone which not only gives a list of recommended alcohols but also includes descriptions of their characteristics.

```
data = [
    {
        "question": "What's a good cocktail for a summer party?",
        "answer": "A classic Mojito is perfect for summer parties. It's refreshing, easy to make, and
loved by many. Rating: 4.8/5. Price: $10-$15 per serving."
    },
    {
        "question": "I'm looking for a cocktail that's not too strong but still flavorful. Any
suggestions?",
        "answer": "Try a French 75. It's made with gin, lemon juice, simple syrup, and topped with
champagne. The drink is light, citrusy, and elegant. Rating: 4.7/5. Price: $12-$18 per serving."
    },
    {
        "question": "What's a good red wine for under $20?",
        "answer": "Carmenere from Chile is an excellent choice. It has rich flavors of berries and
spices, making it great for pairing with food. Rating: 4.6/5. Price: $15-$20 per bottle."
    },
```

*Figure 1.*

To decide on the proper amount of data we should use, we started from feeding a small number of data pairs and gradually added more. By this approach, we could see how the model progressed with increasing amount of training data while maintaining the total training time in a reasonable range. *Figure 2* shows that when the amount of training data was too small, the model would give us empty reply, which was not acceptable. Eventually we found that using 20 data pairs would control the training time to about 35 minutes and enable the model to generate meaningful replies.

```
Assistant:
Prompt size: 46 tokens. New text size: 0
```

*Figure 2.*

### 2.3.2 Parameter Modification

We tried different settings for the key parameters. As shown in *figures 3.1*, when the parameters were set as *epoch = 2, batch_size = 2, and learning_rate = 2e-5*, the result is not formatted as our training data. It did give us a menu of cocktails together with their components, but we expected a conversation-like answer with more "subjective" descriptions (e.g., flavors, tastes, effects, etc.) of the alcohols.

After manually changing the parameter values, we found that setting *epoch = 10, batch_size = 1, and learning_rate = 5e-6* would give us desired results, as shown in *figure 3.2*.

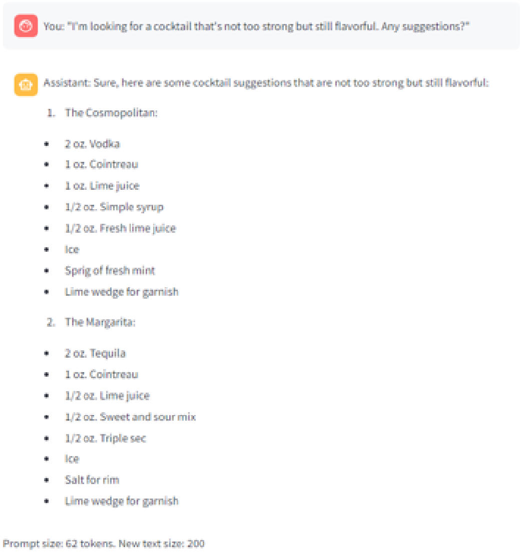


*Figure 3.1.*

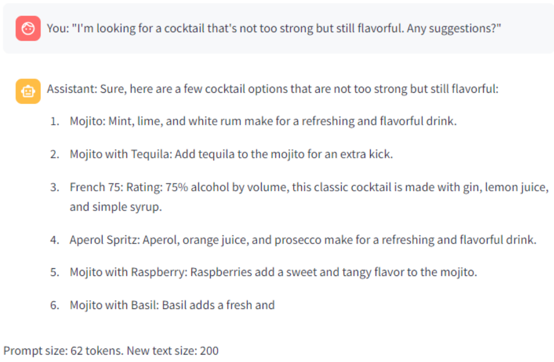


*Figure 3.2.*

# 3. Results

## 3.1 Front-end webpage

- Fine-tuned Chatbot (*figure 4.1*) & RAG Chatbot (*figure 4.2*)

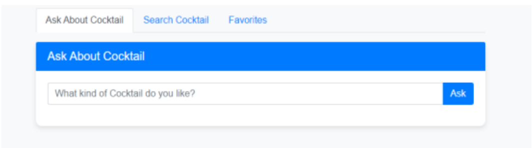


*Figure 4.1.*

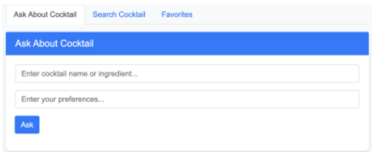


*Figure 4.2.*

- Search Engines

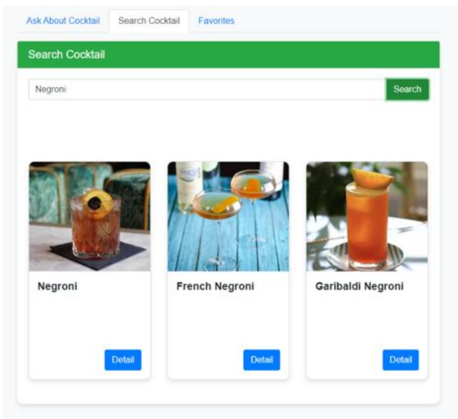


*Figure 5.1.*


*Figure 5.2.*

- Favorite collections

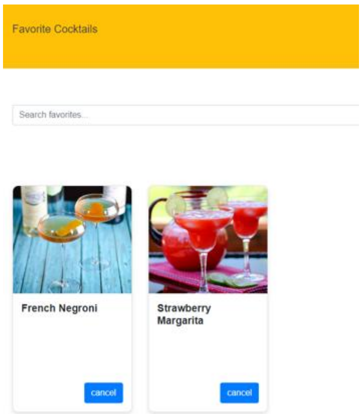


*Figure 6.*

### 3.2 Fine-tune results

- Parameters: "*epoch = 10, batch_size = 1, learning_rate = 5e-6*", the result is reasonable.
- Format: brief and unified, which is "Cocktail Name: Brief Explanation".

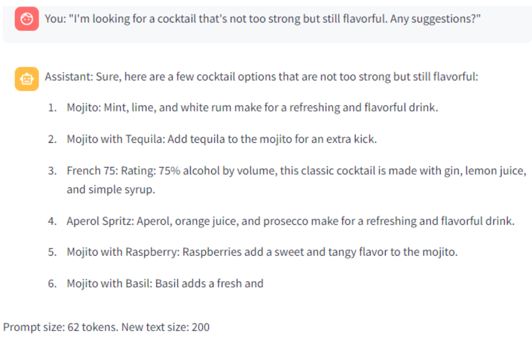


*Figure 7.1.*

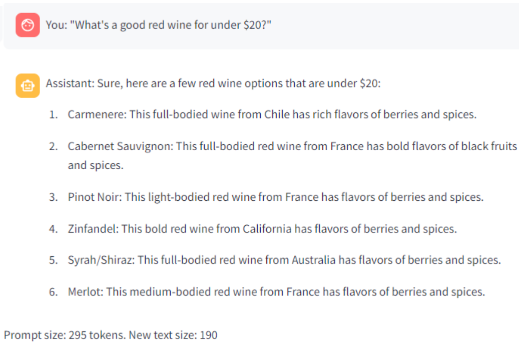


*Figure 7.2.*

As shown in *figures 7.1 and 7.2*, the optimized recommendations not only cover the names of cocktails but also give a brief summary in one sentence (including ingredients, alcohol content, flavor, menu, drinking scenario, etc.), which encourages users to use the search function to access more complete information online.

In summary, new customers can quickly select suitable cocktails that meet their expectations by using our recommendation system, increasing their interest and willingness to purchase cocktail products.

### 3.3 RAG results

Vector database generation:

- During the data vectorization, the parameters are: *chunk_Size=500 chunk_overlap = 50, batch_size = 16*. The overlap can reduce information loss and improve subsequent accuracy of searching.
- We successfully converted the information in the *Cocktails.csv* into vectors, then saved them in the database *cocktail-vectors.db*. *Figure 8* is the process:

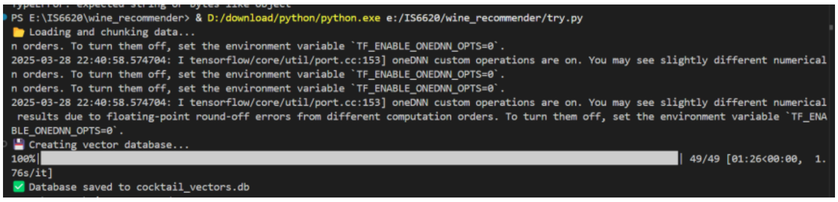


*Figure 8.*

Data retrieval:

- We use prompt engineering if the chatbot cannot find the corresponding answer by searching the vector database. Based on our query, LLM will guess the specific products we may expect and output the answer query through *openrouter.ai*, then generate the "Save to My Collection" button by passing the *found* parameter.
- If the vector database can retrieve the answer, the "Save to My Collection" button will not appear.

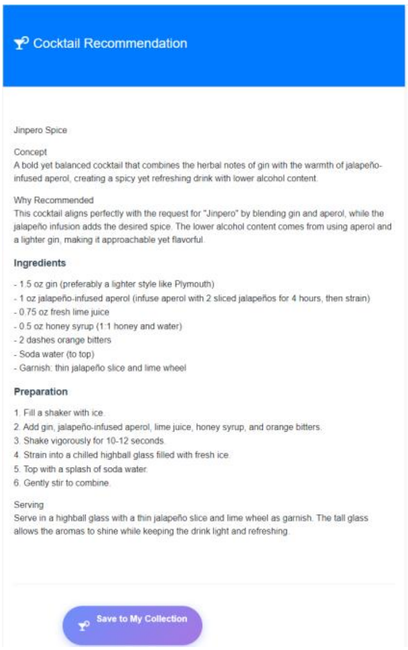


*Figure 9.1.*

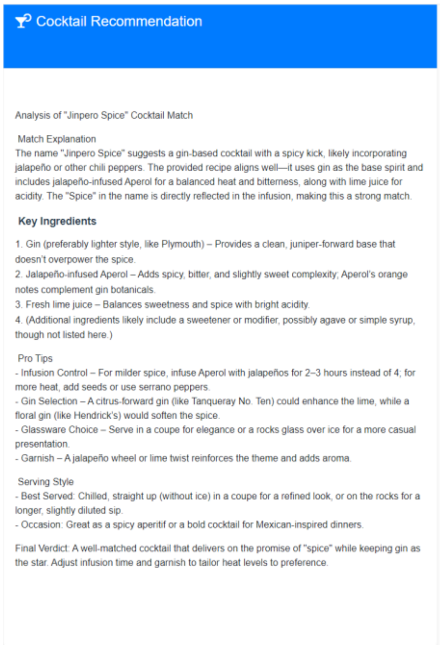


*Figure 9.2.*

# 4. Limitations and Future Work

## 4.1 Limitations

Fine-tune:

- The main limitations are hardware and data issues. Due to limited device capabilities, our training data only had 20 samples, and we only used 2 sets of dialogues for fine-tuning.
- Additionally, we made too few attempts for parameter settings. We tried only two times and yielded relatively good results, but it might not be the best choice.

RAG:

- Text length limitation: In similarity filtering of the vector database, *parse_cocktail_content = 200*, which means only the first 200 words will be checked for the menu similarity filtering. It may affect the accuracy of the answers.
- Answer completeness: The parameter *top-n* is 3, which means the function will return the top 3 results most relevant to the query, which may affect the comprehensiveness of the answer.

## 4.2 Future work

Fine-tune:

- Increase the amount of fine-tuning data by using more advanced devices. Additionally, external website links can be considered to ensure data updates.
- Evaluate and find the most efficient parameter values *(epoch, batch_2,* and *learning_rate)* to achieve the highest accuracy of answer queries together with the fastest output speed possible.

RAG:

- In information retrieval, we can try different *top_n* and *parse_cocktail_content* values to find the most suitable ones for the chatbot accuracy and speed.
- Besides, we can try different external models rather than *openrouter.ai* to generate answers and find the most suitable models.

# 5. Work Allocation (Ranked in no particular order)

## 5.1 Code development

FENG Yiming (58868018):

- Front-end & Back-end development; Fine-tuning work
- Communicate fine-tune details with the report writers

WANG Chenghao (58581879):

- Front-end & Back-end development; Fine-tuning work
- Communicate fine-tune details with the report writers

JIANG Yihui (58878190):

- Front-end & Back-end development, organize code files; RAG work
- Participate in report discussions

**5.2 Data Preprocessing:** ZHENG Yiyang (58838131) & QU Xinzhu (58553950)

## 5.3 Project report writing

TANG Jialei (58947280):

- Report writing (Introduction and Methodology section)

ZHENG Yiyang (58838131):

- Report writing (Challenge section) and overall editing

QU Xinzhu (58553950):

- Report writing (Result, Limitation & Future Work section)

DU Qinshu (58819313):

- Report writing and overall modification

## 5.4 PPT and other work

JIA Yunwen (58887807):

- Responsible for all PPT slides
- Participate in report discussions

DU Qinshu (58819313):

- Organizing meetings and communication matters
- Proposal writing
- Assistance in PPT production