Cocktail-Recommendation-QnA-Using-GPT4ALL

https://github.gatech.edu/Aviolette3/Cocktail-QnA-Using-GPT4

The cocktail recommendation application takes in a request for a cocktail recommendation based on any number of themes or ingredients or a question about the contents of a drink and outputs an answer to the query. The answer will be based on a backend storage of cocktail recipes, but will default to answering using the LLMs own knowledge if the information is insufficient.

Implementation Details

The implementation of this application follows a framework set by Chitti Ankith in his application Story-QA-using-GPT4All. First, util.py handles the reading of a file. If there is already a file, it uses that, and if not, it will get it from an online txt source. The file is parsed and divided into chunks, each of which is a full recipe (the txt separates recipes with at least 39 dashes in a row). Each recipe is then formatted into a single-line format and put into a new file, where it can be parsed one line (recipe) at a time.
Another thing to keep track of is the txt file's use of the '|' character to separate different versions of a cocktail in the same line, which had to be dealt with. I split up each line by that character and created as many lines as there were identicals of the recipe so there are multiples of many recipes, but they usually have a difference in ingredients.
Once the files are ready, the gpt4all model gets initialized and starts downloading. Then I create a table for the recipes, a table for the features of the recipe (to be extracted), and an index table. The cocktail_table gets filled with lines from the newly created file, each of which represents its own recipe, and then the EvaDB SentenceFeatureExtractor is called on that information to vectorize it and place it in the cocktail_feature_table. Finally, the index table is used to create a search index on the features so it can be parsed.
Then I input a string question which is unidecoded and vectorized to be compared to the cocktail feature table using the Similarity class to find the 5 recipes whose attributes match the question the best. Then, those 5 recipes, the question, and an order are all included in a query that is fed into the gpt4all model. The order is designed to force the LLM to use the context to the best of its ability and then default on its own knowledge if the context is not useful.

Inputs/Outputs

In each one of these, I italicize and underline a change that I made to the recipes in my own file which indicates to me that the LLM used the context instead of its own knowledge.
I'll include here the input query, the output from the LLM, and the context that was given to the LLM as a result of the Similarity query (I will only include the names of the drinks but the ingredients are also in the context).

Here I asked for a drink based on a type description and got an appropriate response. The context included mostly very tropical drinks.
Input: “Recommend to me a tropical drink?”
Output: “If you're looking for a tropical drink, I would suggest the Blue Hawaiian. It's made with coconut rum, *pink* curaçao, pineapple juice, and orange juice. You can also add some ice to make it more refreshing.”
Context: margarita, blue hawaiian, wild brew yonder, hurricane, cuba libre

Here I had a simple question of what is in a drink, so in return the LLM outputted the recipe for said drink. The ‘%’ behind the 15 was removed due to it having weird functionality with SQL.
Input: “Whats in a zipper cocktail?”
Output: “The ingredients for a zipper cocktail are 1 part 15 cream, 1 part *quadruple* sec, 1 part tequila, and served in a lowball glass.”
Context: zipper, tequila pink, kangaroo cocktail, bermuda rose cocktail, santiago cocktail

Here you’ll notice that the drink recommended was not from the context. This is because the Similarity query failed to get a group of relevant cocktails, which is an issue that persists.
Input: “Recommend me a cowboy drink”
Output: “How about a Whiskey Sour? It's a classic cocktail that's easy to make at home. Here's the recipe: Ingredients: 2 oz Bourbon, 1/2 oz Lemon juice, 1 tsp Simple syrup, 1 Egg white, Ice Combine Bourbon, Lemon juice, and Simple Syrup in a cocktail shaker. Add ice to the shaker. Shake until chilled. Strain into a rocks glass filled with ice. Garnish with a cherry or lemon twist.”
Context: mint julep, creamy bush, hawaiian cocktail, havana cocktail, sex on the beach

Here, the Similarity query did a very good job of finding strong drinks, even if that is a fairly abstract thing that does not come up in the recipes’ names or instructions. I ran this test to make sure that it didn’t always recommend tropical drinks. It passed the test but it did include one tropical drink, but a very strong one.
Another note is that the LLM pretty much just listed all the drinks in the context this time.
Input: “Recommend me a strong drink”
Output: “The hawaiian cocktail sounds like it might fit the bill with its triple sec and pineapple flavors. You could also try the presbytarian, which has similar ingredients but with a different ratio of spirits. Another option is the gentle ben, which blends vodka, gin, and tequila for a strong and complex taste. Finally, you could try the bosston cocktail, which uses brandy, cointeau, and light rum for a sweet and spicy flavor. Just be sure to add the ingredients in the correct order and let them sit for at least 30 minutes before shaking and straining into glasses.”
Context: hawaiian cocktail, presbyterian, gentle ben, boston cocktail, the brain

Time

Here are the recorded times for various portions of the application running. They are in the same order as the Inputs/Outputs above.
Insertion: 43.7s, 48.5s, 48.9s, 48.8s
Extracting features: 19.8s, 26.4s, 26.2s, 22.2s

Index Creation: 7.4s, 9.1s, 8.0s, 8.4s
LLM Response: 172.6s, 97.3s, 192.1, 206.5s
Total: 248.9s, 187.3s, 280.8s, 292.2s

Lessons Learned

The LLM is the most time-consuming and difficult-to-predict portion of the application. There needs to be a priority on improving the time efficiency of the LLM as well as making sure that the LLM behaves as expected. In order to do this, one needs to only give the LLM necessary tasks that cannot be done differently, the context that is fed to an LLM needs to be as concise as possible, and the instructions to it must be abundantly clear.
EvaDB is sometimes a little tricky to manage, so sometimes classes and functions need to be manually added. Additionally, some SQL functions can behave unexpectedly, but there is often a secondary way of performing a task, so that can be a good avenue to go down as far as troubleshooting goes.

Challenges

Dealing with the txt file was difficult because of the sporadic layout of some of the recipes. Some have steps to create a drink in the middle of the ingredients, sometimes there are multiple recipes in line with one another, and the dash separator between the recipes is of a variable length. In order to deal with this I had to Split lines based on the number of '|' characters in order to make multiple recipes that were inline have their own lines. I had to figure out how to account for a string of variable length and remove the whole thing no matter the length when I split the file into recipes. I also had to accept that the layout was not going to be perfect due to the sporadic nature of the recipe layouts so I separated everything by colons and placed it all inline so it could make some sense.

When trying to do cursor.execute for an insert query, I ran into constant NoneType errors with no obvious solution to deal with them. I discovered that instead, I could use cursor.query.df to achieve a similar result, and it worked perfectly.

Getting the orders to force the LLM to create consistent results was fairly difficult, and will need further attention when I expand on this project. Things need to be said very explicitly and the LLM needs to be aware of any format that it needs to understand or follow with its answers. Additionally, any way that the LLM can deviate from the expected behavior without directly going against the query, will happen without fail. That is why every detail needs to be stated.

I had a lot of issues with Python version control and keeping track of dependencies and making sure that they worked. I ran into many instances of having non-functional imports that were behaving strangely until I deleted them and reinstalled the correct version of them.