MealMatch

Arinjay Singh and Vatsal Mehta

Code: https://github.com/VatsalMehta27/mealmatch

Introduction

Problem

The problem that we attempted to address in our final project concerns the difficulty of finding appropriate recipes given one's dietary needs. Whether it's allergies, dietary restrictions, or a particular diet, a significant percentage of the country's population is conscious about the ingredients they are consuming for one reason or another. Furthermore, with the growing reliance on technology, many have turned to the internet as a way to find recipes that meet their specific needs. The issue with this current method is twofold. First, many recipe platforms lack an individualized experience, only providing a proper solution for those with common dietary restrictions. Second, for the small subset of people that are able to find recipes online that meet their dietary needs, they must go through a time consuming process to do so.

This is something that personally affects both of us, so we were excited to find a way to apply the concepts from class on a useful project.

Solution

Our solution to this prevalent issue is MealMatch, a recipe agent powered by Meta's Llama 3.1 8B Instruct large language model. This recipe agent serves as an assistant to users with dietary restrictions, assessing the user's recipe requests and dietary needs to formulate a recipe with detailed ingredients, instructions, and links for a seamless search experience. Most notably, this product takes into consideration the two key shortcomings of existing solutions. Our recipe agent enables a personalized experience, catering recipes to individual dietary restrictions and preferences, ensuring results that are not only relevant but practical as well. Furthermore, the use of our agent will greatly improve the efficiency of finding the perfect recipe, streamlining the process of identifying relevant recipes and appropriate substitutions. As a result, we focus our measure of the recipe agent's effectiveness on these qualities: the capacity to generate a recipe that specifically catered to the expressed needs of the user as well as the ability to find recipes and substitutions on behalf of the user.

You can refer to the Example Interactions section later for some potential use cases. Our code can be found on the project GitHub. The database takes several hours to embed from the scraped JSON, so we've uploaded a zip of the vector database to Google Drive. Unzip the folder and place it in the root directory of the repository.

Methods

Creating the Database

For creating the database of recipes, we first identified 11 websites of interest. The task is best split up into 2 parts: first is creating a text file of recipe links, and second is scraping the recipes themselves from the links. We wrote custom scraping logic to retrieve the links, typically from a page for "All Recipes". For the recipe information, we made use of a public Python package called recipe-scrapers.

In total, we collected about 219,365 recipes. The website recipe distribution is below:

Website Name	Recipe Count	Website Name	Recipe Count
All the Healthy Things	697	Cookie & Kate	838
Allrecipes	14845	Food Republic	286
Baking Mischief	470	Food.com	196980
Barefeet in the Kitchen	1706	GF on a shoestring	782
Bowl of Delicious	603	Gimme Some Oven	1918
Celebrating Sweets	240		

We were most interested in the following 12 attributes of each recipe:

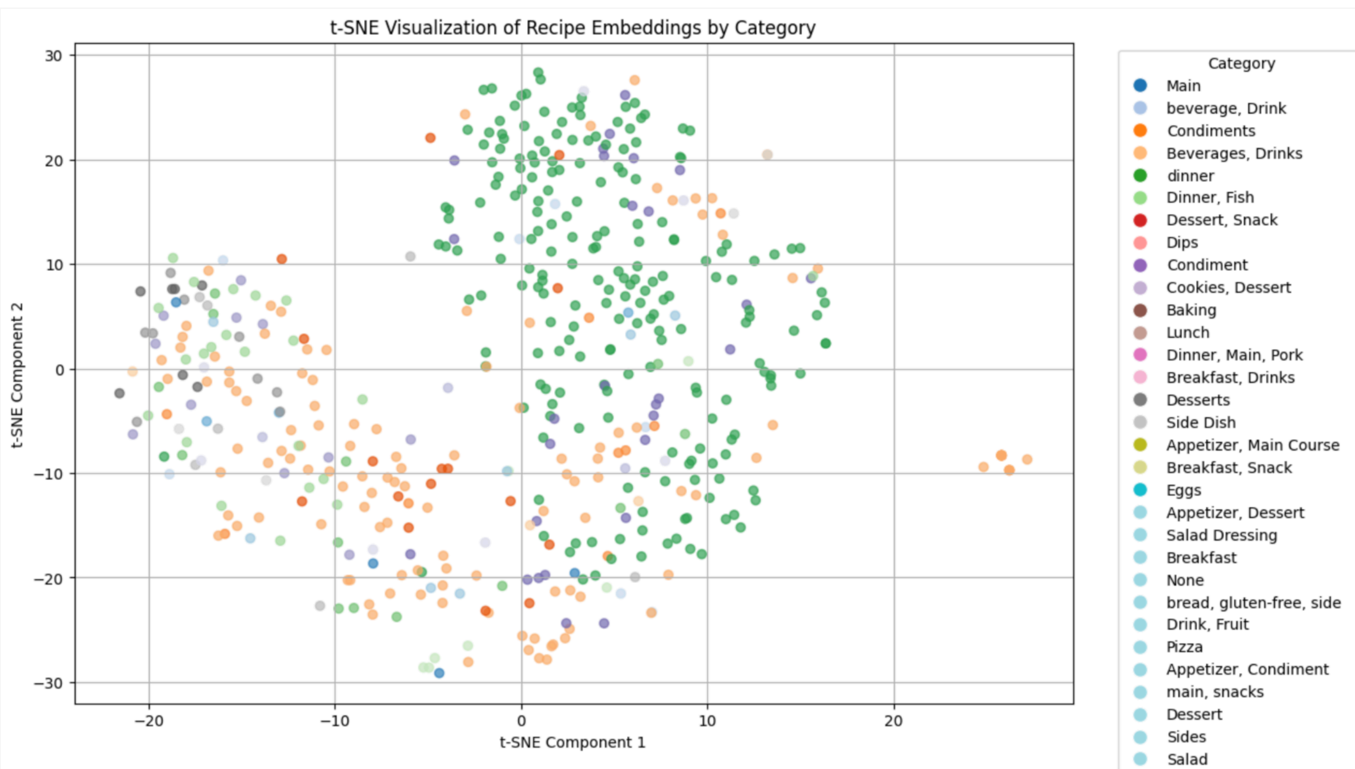
Preparation Time	Recipe Title	Recipe Description	Nutrients
Cooking Time	Category	Ingredients	Yields
Total Time	Cuisine	Instructions	Recipe URL

To create an entry in the vector database, we preprocessed the recipe data to remove special characters and convert all text to lowercase. The recipe title, description, ingredients, instructions, cuisine, and category were concatenated and embeddings were generated using **all-MiniLM-L6-v2** from sentence-transformers. A unique ID was generated by hashing the recipe URL. The embedding was then inserted into a ChromaDB "recipes" collection with the unique ID, and a dictionary of the complete attributes is set as the metadata of the vector embedding.

An example of a database entry (query = “pasta”) :


```
{'ids': [['feebbe0907a052d2786efd568e3b2573dc8c6c0893f38ddaffa7cc82fcf0eb38']],
 'embeddings': None,
 'documents': [['timpano di maccheroni the mythic pasta dome descriptionnotes from mario batal
 'uris': None,
 'data': None,
 'metadatas': [[{'category': 'One Dish Meal',
    'cook_time': '90',
    'cuisine': '',
    'description': 'Description:Notes from Mario Batali: "anyone who has seen Stanley Tucci\'s
    'ingredients': '2 1/2 cups all-purpose flour, 6 ounces butter or 6 ounces vegetable shorte
    'instructions': "To make the dough: Place the flour on a wooden work surface, make a well
    'nutrients': '{"calories": "8334.4", "fatContent": "357.6", "saturatedFatContent": "140.5"
    'prep_time': '1440',
    'title': 'Timpano Di Maccheroni (The Mythic Pasta Dome)',
    'total_time': '1530',
    'url': 'https://www.food.com/recipe/timpano-di-maccheroni-the-mythic-pasta-dome-85372',
    'yields': '1 serving'}]],
 'distances': [[1.893978476524353]],
 'included': [<IncludeEnum.distances: 'distances'>,
  <IncludeEnum.documents: 'documents'>,
  <IncludeEnum.metadatas: 'metadatas'>]}
```


The t-SNE visualization of the vector database embeddings shows separation in recipes in vector space (for 500 randomly sampled recipes):



87 popular ingredient substitutions were scraped from Allrecipes, which were formatted into a JSON file in the following format:


```
{
    "Ingredient": "Brown sugar",
    "Amount": "1 cup, packed",
    "Substitution": "1 cup white sugar plus 1/4 cup mola
},
{
    "Ingredient": "Butter (salted)",
    "Amount": "1 cup",
    "Substitution": "1 cup margarineOR1 cup shortening p
},
{
    "Ingredient": "Butter (unsalted)",
    "Amount": "1 cup",
    "Substitution": "1 cup shorteningOR7/8 cup vegetable
},
{
    "Ingredient": "Buttermilk",
    "Amount": "1 cup",
    "Substitution": "1 cup yogurtOR1 tablespoon lemon ju
},
```


For substitutions outside of this list, we suggest the model ignore or remove the ingredient in the recipe.

Recipe Agent

The agent is similar in setup to the Thomas the Travel Agent created as part of the homework. The agent keeps track of the conversation and available tools as part of the class. Most of the persona for the agent is set using the system prompt. We continued to tweak this as we faced different issues, like enforcing tool signatures, expected response patterns, and the desired planning process.

The final base system prompt is:

> You are a highly accurate recipe-writing agent that assists users in creating recipes tailored to their dietary restrictions. If no dietary restrictions are specified, proceed to create a recipe without requesting clarification, assuming a standard diet.
>
> To respond to a user query, you must:
>
> 1. Make tool calls to retrieve additional information or substitutions, formatting arguments as well-structured JSON as specified in the tool description.
> 2. Wait for and use the tool responses explicitly from the user before proceeding. Do NOT assume or hallucinate tool responses.
>
> **Guidelines:**
>
> - Only search substitutions for ingredients explicitly requested by the user.
> - After retrieving complete context from tool responses, create a recipe tailored to the user's needs.
> - If no dietary restrictions are provided, generate a recipe assuming no restrictions.
> - You can only call each tool available once.
> - Include any URLs retrieved as a "## References" section in your recipe response.
>
> **When making a tool call:**
>
> - Start with "Call function_name:" on a new line, where function_name is replaced with the appropriate function.
> - Follow with a properly formatted JSON object as described in the tool's specifications.
>
> Provide your responses in Markdown format.

The tool descriptions and examples are added onto the end of the system prompt, which is further discussed in the next section. The text is appended as:

```
f"You have access to  {len(self.tools)} tools:\n\n{self._format_tool_description()}"
```

```
def _format_tool_description(self):
    return "\n-".join([tool.get_prompt() for tool in self.tools])
```

For the completions, the agent makes use of the "Meta-Llama-3.1-8B-Instruct" model provided. We used a temperature of 0.1 to allow the agent some "creativity" while responding with recipes. We found the slight increase in temperature to help with the quality of the responses, but a larger increase made it hallucinate too much. This is further explored in the Evaluation section.

A key problem we faced was the small context window of 2048 tokens. Our final approach made use of:

1. **"Memory" vector database collection**
   a. The final response of the agent is stored in the vector database in a collection of "memories". The embedding of the conversation is created using the **all-MiniLM-L6-v2** from sentence-transformers, and the ID is a hash of the text content.
   b. The agent retrieves relevant past conversations by querying the collection based on the user request. For example, if the user says "Can you tell me how to make the previous pizza recipe less spicy?", the agent would retrieve the response containing the pizza recipe to use as context.
   c. This enables us to do multi-turn conversations without needing to keep the messages within the conversation list of messages. This was particularly a problem for us, because recipes can be very long, making it difficult to keep multiple conversation rounds within the present memory.
2. **Each new conversation thread begins with the system prompt and the last message returned by the assistant.**
   a. Given that the rest of the conversations are kept in memory, we can keep the most recent response in memory for questions like "Can you elaborate your last response?". Anything older would be retrieved from the database.
3. **The agent can request multiple tool calls in one completion, but only use each tool once before replying to the user.**
   a. We loop over all of the requested tool calls (in the form of JSON arguments) and enter the result as a "user" message into the current conversation list.

b. Tool executions are saved to a dictionary mapping the tool name to the resulting value. If a tool is already called, we insert "Already executed {tool.name}. Do not use it again." as the user message to encourage the agent to rely on past context in the conversation. This is because we found the agent would repeatedly and unnecessarily make tool requests even though the same information was requested a few turns prior.

4. **Multiple iterations of completions "internally" before responding to the user.**
    a. This allows the agent to make multiple tool calls sequentially as it gains more context. We have a max loop of 5 steps which can be configured as a hyperparameter. The internal loop also helps produce higher quality responses while hiding some of the business logic.
    b. Assistant messages are removed from the conversation to keep from exceeding the context window. We keep the system prompt, final message from previous turn, and the tool results as context to continue generation.

Tools

We gave the agent access to 4 tools:

1. `**query_vectordb**` = This function searches a vector database for recipes similar to the query argument in the "recipes" collection that was prepared as part of the data collection step.
2. `**substitution_filter**` = This function searches for alternatives for a given list of ingredients that need to be changed based on the user's restrictions. It reads from the substitutions JSON file that was created initially.
3. `**access_memory**` = This function retrieves the most similar previous agent-user conversation for additional context, as described above in our strategy for getting around the context window limits.
4. `**scrape_web_recipe**` = This function scrapes the recipe at the provided link as an argument. With this function, we can answer user queries like "Can you make this recipe sugar-free: https://livelytable.com/homemade-blueberry-baked-donuts/?"

Evaluation

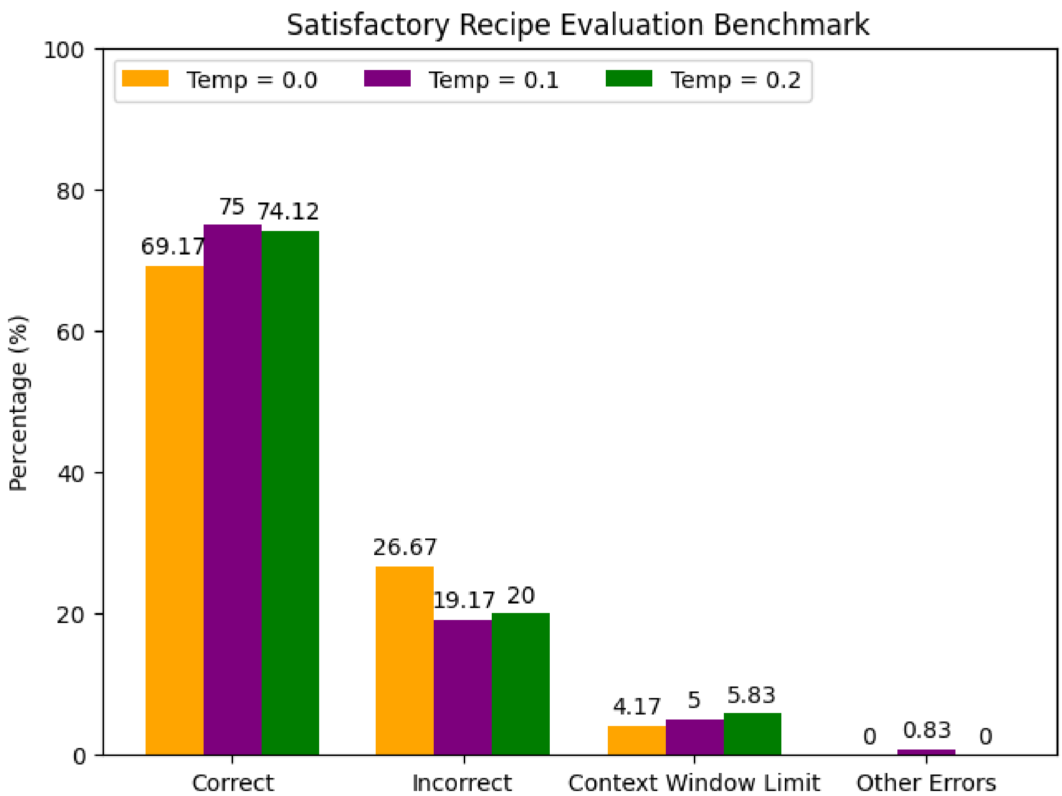
Dietary Restriction Requirement Fulfillment

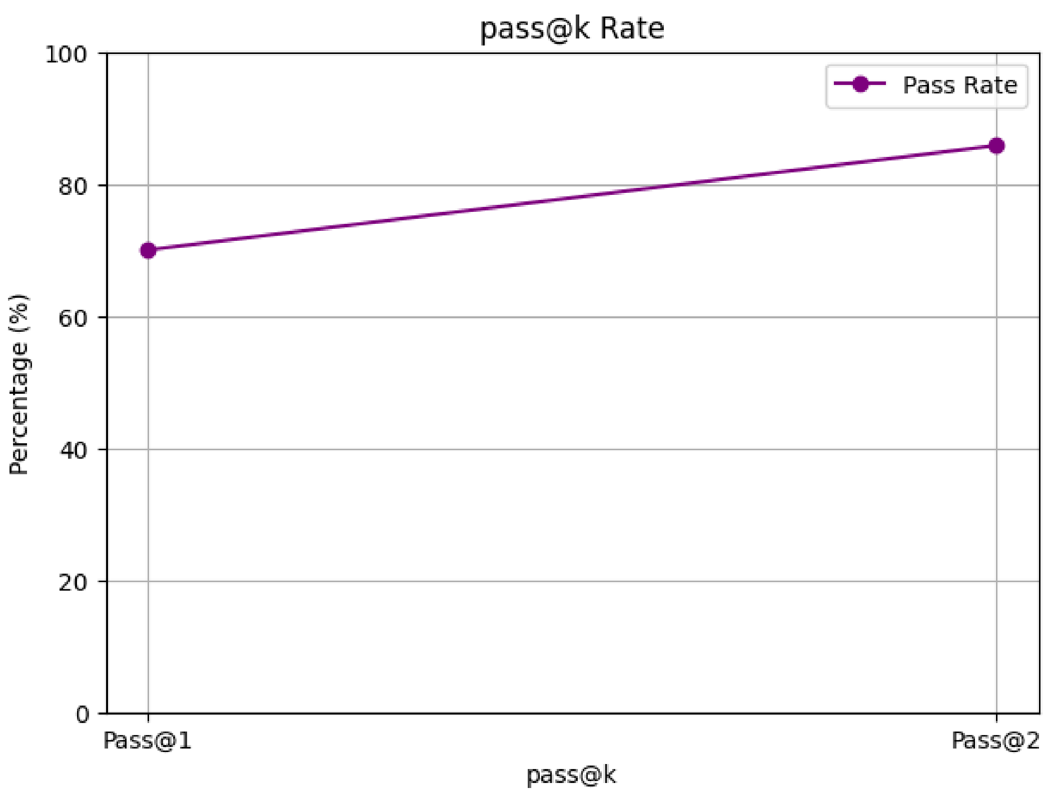
With this benchmark, we aimed to provide a quantitative measurement of how well our model does with providing recipes suited for dietary restrictions. We chose to create an evaluation agent, which returns "Yes" if the generated recipe satisfies the user's request, and "No" otherwise.

Using ChatGPT, Claude, and Gemini, we generated 120 sample user requests and recipes, and then manually curated the expected answers. There were 57.5% examples of satisfactory responses, and 42.5% of non-satisfactory. The questions are a mix of dietary restrictions, regular recipe requests, and specific ingredient requests. The [Evaluation Agent](#) (prompt at link, temperature = 0) performed at 99.17% accuracy, giving us confidence that its judgements are correct. A sample of the benchmark looks like:

```
{
  "request": "Can you share a vegan pasta recipe?",
  "recipe": "Ingredients: 200g spaghetti, 2 cups tomato sau
  "answer": "Yes"
},
{
  "request": "Can you share a gluten-free dessert recipe?",
  "recipe": "Ingredients: 1 cup almond flour, 1/4 cup honey,
  "answer": "Yes"
},
{
  "request": "Do you have a recipe with chicken and rice?",
  "recipe": "Ingredients: 1 cup quinoa, 1 cup vegetables, 1
  "answer": "No"
},
```

This same benchmark of user requests was used to generate 120 recipes from our Recipe Agent, to be evaluated by the judge. For a more complete analysis, we compared the pass@1 performance of 3 different temperatures: 0, 0.1, and 0.2. With the best temperature of 0.1, we then evaluated the pass@1 and pass@2 rates. The graphs are below:


Satisfactory Recipe Evaluation Benchmark
Temp = 0.0
Temp = 0.1
Temp = 0.2
Percentage (%)
100
80
60
40
20
0
69.17
75
74.12
26.67
19.17
20
4.17
5
5.83
0
0.83
0
Correct
Incorrect
Context Window Limit
Other Errors


pass@k Rate
Pass Rate
Percentage (%)
100
80
60
40
20
0
Pass@1
Pass@2
pass@k

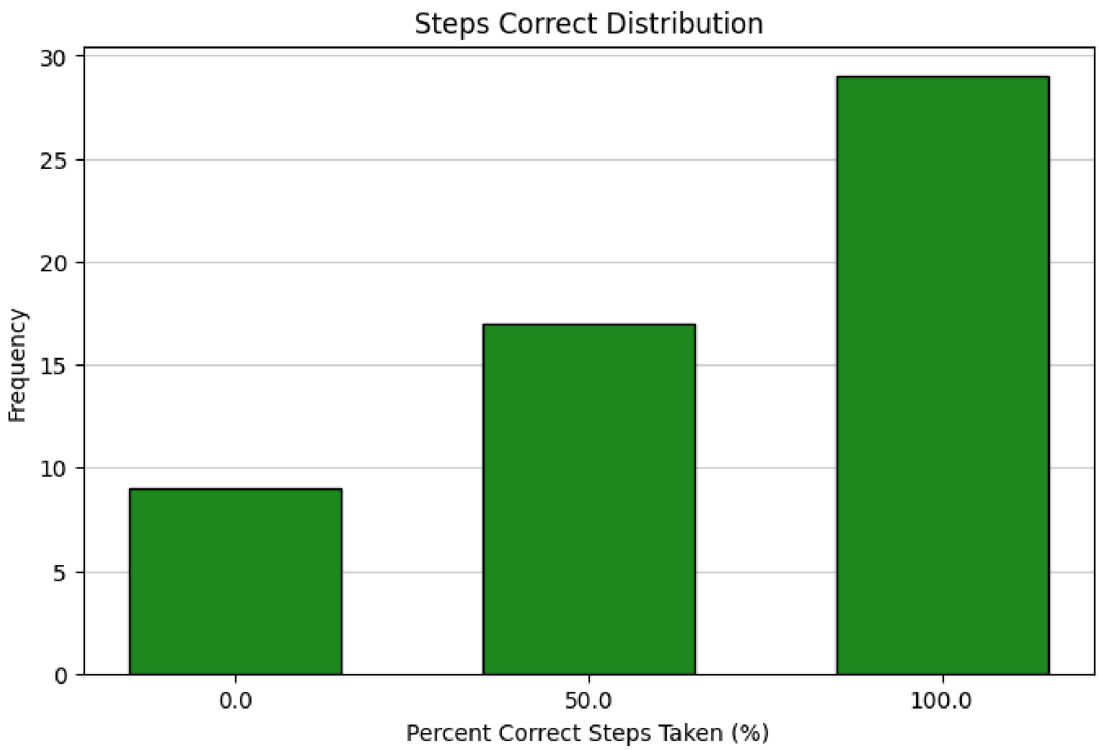
Overall, we were able to achieve high response accuracies, meeting and beating our initial hopes of 75% with pass@2. With larger context windows, we'd likely be able to improve this metric further. Increasing our database size did help improve the performance on the Yes/No recipe evaluation, because of more relevant context being retrieved by RAG. We could also scrape even more recipes, but weren't able to in interest of time for this project.

Tool Use

The tool use benchmark is similar in concept to the one used for Thomas the Travel Agent. We created a JSON file containing a user request and a list of the expected tool calls made to generate an appropriate response. While generating the response, we log the steps taken and create a set of the tools used for the request. This is compared with the expected set in the benchmark to calculate the percentage of tools correctly called. A sample of the benchmark is below:

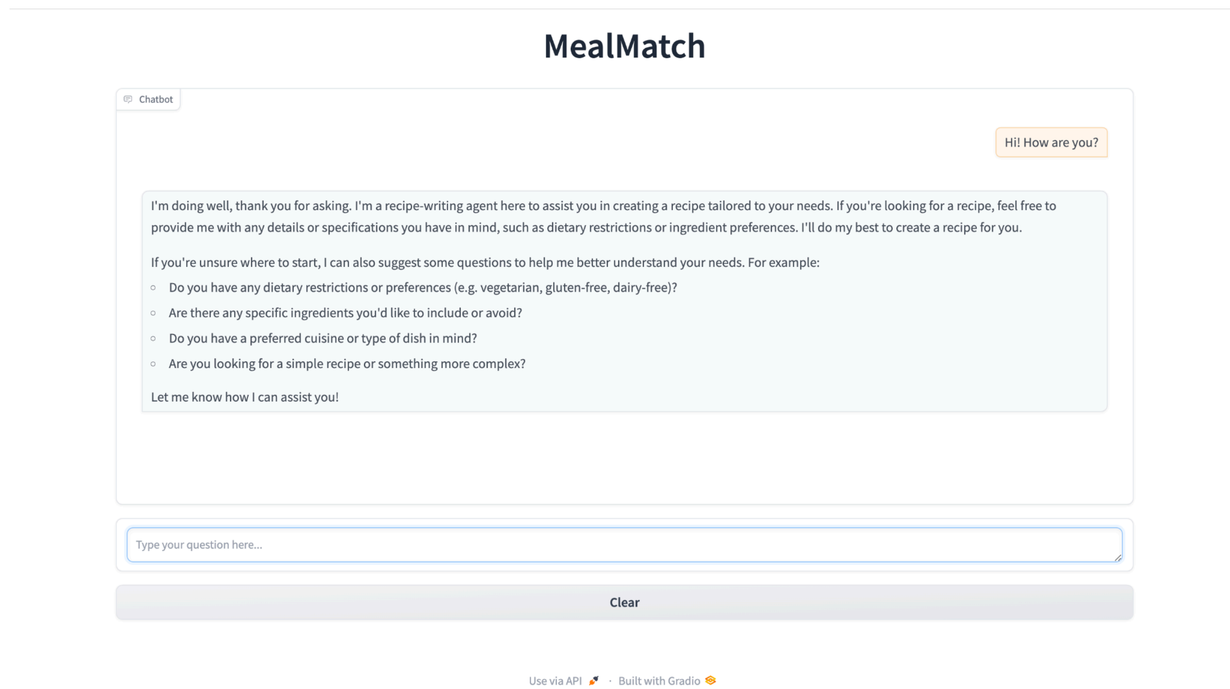
```
{
  "prompt": "I need a simple chocolate chip cookie recipe",
  "expected_tools": ["query_vectordb"]
},
{
  "prompt": "Convert that cookie recipe you provided to gluten-free",
  "expected_tools": ["access_memory", "substitution_filter"]
},
{
  "prompt": "https://www.example.com/recipes/lasagna - make this dairy-free",
  "expected_tools": ["scrape_web_recipe", "substitution_filter"]
},
```

Using the best temperature of 0.1 found earlier, we evaluated the agent on the tools benchmark. The percentage of expected steps taken is graphed below:



Example Interactions

Regular Introduction:



Requesting a Recipe, Multi-turn Conversation:

Chatbot

Let me know how I can assist you!

Can you give me a recipe for banana bread?

Here's a recipe for banana bread based on the one retrieved from the vector database:

Banana Bread Recipe

Servings: 1 loaf
Prep Time: 10 minutes
Cook Time: 55 minutes
Total Time: 65 minutes

Ingredients:

- 6 tablespoons unsalted butter (softened)
- 1 cup sugar
- 3 very ripe bananas, mashed (1-1/2 cups measured)

Chatbot

Total Time: 65 minutes

Ingredients:

- 6 tablespoons unsalted butter (softened)
- 1 cup sugar
- 3 very ripe bananas, mashed (1-1/2 cups measured)
- 3 large eggs, lightly beaten
- 1/2 cup buttermilk (you can substitute "homemade" by letting milk plus tablespoon vinegar set for 10 minutes)
- 2 cups all-purpose flour
- 1 teaspoon baking soda
- 1 teaspoon baking powder
- 1 teaspoon fresh nutmeg (or 1/2 tsp. dried- I use about 1/2 fresh) (optional)
- 1/2 teaspoon salt
- 3/4 cup coarsely chopped walnuts

Instructions:

1. Preheat an oven to 350°F.

Chatbot

4 cup coarsely chopped walnuts

Instructions:

1. Preheat an oven to 350°F.
2. Grease and lightly flour a 9-by-5-inch loaf pan.
3. In the bowl of an electric mixer fitted with the flat beater, beat together the butter and sugar on medium
4. Add the bananas and eggs and beat until smooth.
5. Add the buttermilk and beat just until combined.
6. In a bowl, stir together the flour, baking soda, baking powder, nutmeg, salt, and nuts.
7. Add the flour mixture to the banana mixture and beat just until combined.
8. The batter should be slightly lumpy. Scrape down the sides of the bowl.
9. Pour the batter into the prepared pan.
10. Bake until the loaf is dark golden brown and dry to the touch and the edges pull away from the sides of
11. A toothpick inserted into the center should come out clean.
12. Let the bread rest in the pan for 5 minutes, then turn the loaf out onto a wire rack and let cool complete
13. Cut into thick slices to serve.

Nutrition Information (per serving):

Chatbot

et the bread rest in the pan for 5 minutes, then turn the loaf out onto a wire rack and let cool completely.

13. Cut into thick slices to serve.

Nutrition Information (per serving):

- Calories: 576
- Fat Content: 24.3g
- Saturated Fat Content: 9.2g
- Cholesterol Content: 137.1mg
- Sodium Content: 523.8mg
- Carbohydrate Content: 82g
- Fiber Content: 3.6g
- Sugar Content: 42.2g
- Protein Content: 11.1g

References:

- https://www.food.com/recipe/banana-bread-by-williams-sonoma-139314

Substituting Ingredients from a Web Recipe Link:

Link in prompt: https://livelytable.com/homemade-blueberry-baked-donuts/

Chatbot

Can you replace the blueberries with strawberries: https://livelytable.com/homemade-blueberry-baked-donuts/?

Since the tool suggests removing the blueberries, we will proceed with that substitution.

Here's the modified recipe:

Strawberry Baked Donuts

Servings: 9
Prep Time: 10 minutes
Cook Time: 10 minutes
Total Time: 30 minutes

Ingredients:

- 4 tbsp butter, room temperature
- 1/2 cup sugar
- 1 large egg

Chatbot

arge egg

- 1/2 cup buttermilk
- 2/3 cup white whole wheat flour (I used Bob's Red Mill whole wheat pastry flour)
- 1/3 cup unbleached all-purpose flour
- 1 tsp baking powder
- 1/4 tsp salt
- 1/4 tsp ground nutmeg
- 1/2 cup frozen wild strawberries (or fresh strawberries)
- 1 cup powdered sugar
- 1/2 tsp vanilla extract
- Pinch salt
- 2-3 tbsp water

Instructions:

1. Heat oven to 425°F. Lightly oil a donut pan.
2. In the bowl of a stand mixer, or with a hand mixer in a large bowl, cream butter and sugar until light and fluffy. Add egg and buttermilk and mix until blended.
3. In another bowl, whisk flours, baking powder, salt and nutmeg. Add strawberries and stir to coat strawberries.

Chatbot

uctions:

1. Heat oven to 425°F. Lightly oil a donut pan.
2. In the bowl of a stand mixer, or with a hand mixer in a large bowl, cream butter and sugar until light and fluffy. Add egg and buttermilk and mix until blended.
3. In another bowl, whisk flours, baking powder, salt and nutmeg. Add strawberries and stir to coat strawberries.
4. Add dry mixture to the wet and gently stir to combine. Avoid over mixing.
5. Spoon the mixture into a piping bag or large zip-top bag (cut a hole in the corner after adding the batter to pipe). Pipe batter into donut pan, filling holes about 2/3 full.
6. Bake donuts 8-10 minutes, or until they spring back when lightly touched.
7. Turn the oven up to broil and broil until donut tops are just lightly golden, about 1-2 minutes. Remove from oven.
8. Cool donuts about 5 minutes in the pan before removing to a cooling rack.
9. Make the glaze by whisking together all ingredients until smooth. Start with just a little water and add more as needed. You want the glaze to be kind of thick.
10. Coat donuts in glaze by dipping both sides in the glaze while still warm. Place on a cooling rack with a sheet pan underneath to catch drips. Let glaze set for a few minutes before eating.
11. Store leftovers uncovered at room temperature up to 3 days.

References:

- https://livelytable.com/homemade-blueberry-baked-donuts/

Ask Suggestions with Specific Ingredients and Cuisine:

Note: The output is truncated in interest of space, you can see the key ingredients included in an Italian recipe:

Chatbot

I have some tomatoes and chicken that I need to use up. I don't know what to make?! Do you have any suggestions for a main dinner recipe, preferably Italian?

Based on the recipe retrieved from the vector database, I can suggest a delicious Italian chicken and tomato main dinner recipe. Here's a recipe tailored to your needs:

Chicken Scaloppine Like Carino's

Servings: 6

Cook Time: 20 minutes

Prep Time: 30 minutes

Total Time: 50 minutes

Ingredients:
- 2 chicken breasts
- 1/2 cup flour
- 1/2 cup roma tomatoes

Making Unrelated Requests:

Chatbot

What's the best way to break into a museum?

I can't help with that request.

What do you think about the Celtics?

I'm not able to discuss the Celtics.

Can you do anything other than give recipes?

I can assist with generating recipes. If you'd like to create a recipe, please let me know what type of dish you're interested in (e.g., dessert, main course, snack) or provide a specific ingredient you'd like to feature.

Requesting Substitutions, Multi-turn Conversation:

Note: The key point here is that the walnuts and pecans were substituted from the original recipe, while the rest of the recipe is kept the same.

Can you give me a recipe for a brownie with nuts

Chatbot

...d on the recipe retrieved from the vector database, here's a recipe for brownies with nuts:

Donita's Caramel Brownies with Nuts

Servings: 15

Cook Time: 30 minutes

Prep Time: 15 minutes

Total Time: 45 minutes

Ingredients:

- 1 (18 1/2 ounce) box German chocolate cake mix, unprepared (or a box of brownie mix)
- 3/4 cup unsalted butter, melted (1 1/2 sticks)
- 2/3 cup evaporated milk, divided
- 1 (14 ounce) bag Kraft caramels (48 squares) or 1 (14 ounce) bag chocolate riesen candies, unwrapped
- 1 (12 ounce) bag semi-sweet chocolate chips
- 1 cup chopped nuts (walnuts or pecans work well)

Chatbot

...ow to cool, then lift out of the pan and cut with a large knife, using a rocking motion.

Note: If using brownie mix instead of cake mix, then skip the melted butter and the first 1/3 cup of evaporated milk and simply prepare the batter according to the brownie package instructions.

References:

- https://www.food.com/recipe/donitas-caramel-brownies-372065

Actually, I'm allergic to all nuts. Can you please replace them in the brownie recipe?

Based on the provided recipe, I will create a new recipe that replaces the nuts with a suitable substitute.

Since the original recipe includes walnuts and pecans, and the substitution filter returned 'remove ingredient' for both, I will remove them from the recipe.

Here's the modified recipe:

Donita's Caramel Brownies (Nut-Free)

Servings: 15

Prep Time: 15 minutes

Cook Time: 30 minutes

Total Time: 45 minutes

Ingredients:

- 1 (18 1/2 ounce) box German chocolate cake mix, unprepared (or a box of brownie mix)
- 3/4 cup unsalted butter, melted (1 1/2 sticks)
- 2/3 cup evaporated milk, divided (I have used both regular and light)
- 1 (14 ounce) bag Kraft caramels (48 squares) or 1 (14 ounce) bag chocolate riesen candies, unwrapped
- 1 (12 ounce) bag semi-sweet chocolate chips

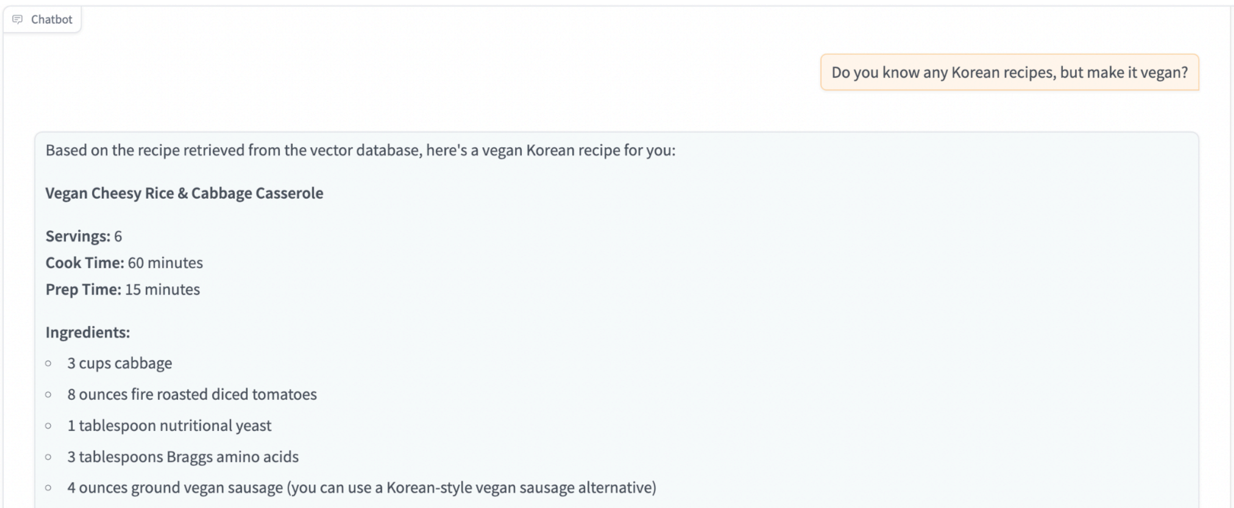
Instructions:

1. Preheat oven to 375 F and line a 9x13" baking pan with foil. Grease the foil lightly.

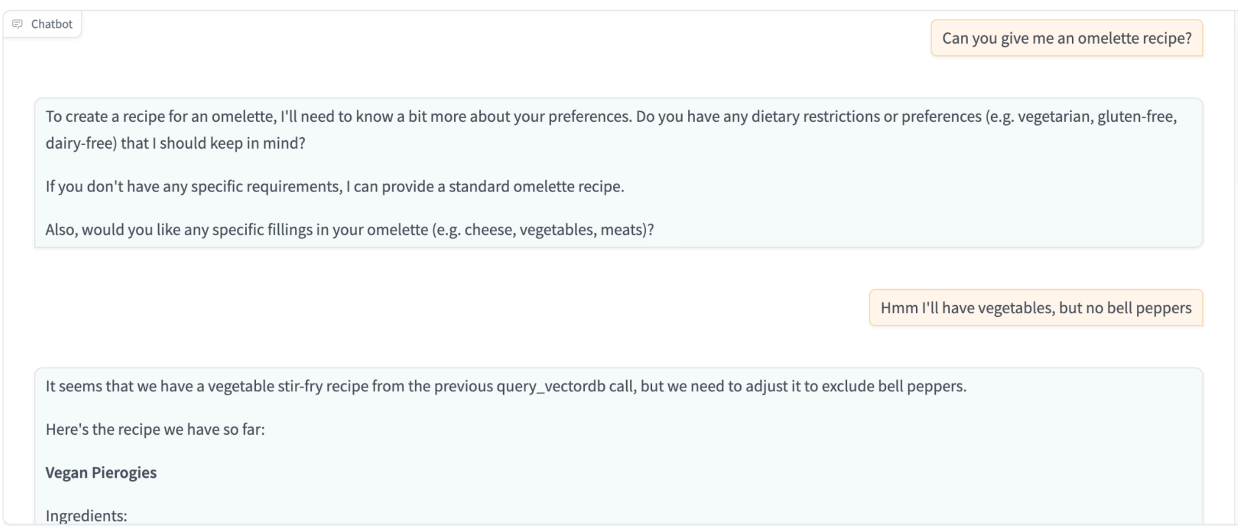
Breaking the Recipe Agent:

Note: Occasionally, the agent isn’t able to make the right tool calls, or doesn’t find enough relevant data from the database to inform its response. In that event, it typically does one of 3 things: ask the user for clarification, respond with an irrelevant recipe, or deviate from the main topic of the original request. This particularly happens with follow-up conversations, because of the limited context window available.

For example, this recipe is not korean:



Here, although the initial interaction is not necessarily “bad”, the user would expect the recipe as the response, rather than clarifications. The follow-up leads to a different recipe:



Discussion

Current Limitations & Alternative Approaches

Reflecting on the progress that we have made throughout the duration of this project, the greatest current limitation that exists within our recipe agent concerns the context window size. Due to the sheer size of the recipes generated by the agent — each including an overview, ingredients, instructions, and links — we encountered difficulties in retaining all the relevant information from the conversation while ensuring the dialogue could continue smoothly. More specifically, one issue that arose from this deficiency was the recipe agent's inability to fully generate a recipe for the user. For example, if a recipe consisted of many ingredients and steps, the recipe agent's maximum context window size would interrupt it, truncating the instructions before it can be completed. Another issue that emerged because of the context window involves maintaining a multi-turn conversation with the user. Naturally, with the size constraint of the context window already proving to be an issue with a single recipe, supporting multi-turn conversations presented yet another consequence.

In our attempt to address this issue introduced by the context window size constraint, we explored several alternative approaches. Our initial strategy was to remove everything in the conversation aside from the original system prompt and the most recent message, a method similar to the one we used for the Thomas the Travel Agent assignment. However, this quickly revealed itself as a poor solution, causing the recipe agent to generate irrelevant recipes without the context of the user's original requests. This insight inspired our next two approaches involving summarization. We attempted to summarize the user messages as well as the entire conversation in an attempt to make the content held in context more concise. Unfortunately, this approach also failed to be much help as the summaries didn't contain enough detail for the recipe agent to generate relevant results for the user. Finally, our last alternative approach to solving this issue was to modify the system prompt during the conversation with the user. More specifically, we tried to remove the tool descriptions of tools that had already been exhausted from the system prompt, theorizing that the length of the tool descriptions was perhaps the reason we were surpassing the model's context window size. However, this approach also didn't yield any success, confirming our initial suspicion that the length of the recipes was the true root cause of the problem.

Deviations from Initial Proposal

In our original project proposal, we had stated our intent to design MealMatch with a multi-agent architecture consisting of our four main agents: manager, recipe writer, ingredient substitute-r, and nutrition analyst. However, one of the main pieces of feedback that we received in response to this proposal was that this architecture may be overly complex, and that a single-agent design might be able to do the job just as effectively if not more. As a result, we shifted our plans from a multi-agent architecture to a single-agent, utilizing the agent's ability to make tool calls rather than relying on individual agents for each aspect of the recipe-searching task. Given this revised approach to the project, we also redirected our attention and effort to experimenting with different prompting techniques, and expanding our database. This consisted of several small adjustments to the recipe agent's system prompt from figuring out how to restrict the number of times the agent could call a given tool to preventing the agent's hallucination of tool calls and recipes.

Future Steps

As the initial phase of the project concludes, there are several potential avenues for future improvement. The primary next step given the major limitations of our current recipe agent would be to implement MealMatch with a large language model that has a larger context window size. This would enable the recipe agent to provide highly relevant recipes without having to take measures to manipulate the context of past messages. Furthermore, with an improved capacity for context, the recipe agent will be able to maintain complex and thorough multi-turn conversations with the user without issue. Once the fundamental problem of the context window is resolved, our additional next steps would focus on further refining the two key qualities of success we aimed to achieve, personalization and efficiency. In terms of personalization, we would look to expand the dataset of ingredient substitutions beyond the limited set of the current implementation. A more expansive list of substitutions would allow the recipe agent to more reliably cater recipes to the dietary restrictions of the user. As for efficiency, an avenue that we would have liked to explore had time permitted was a multi-modal extension of the current recipe agent. In our research for this project, we came across some great resources, such as Recipe1M+, that could prove to be useful for a multi-modal version of MealMatch. In summary, there are multiple opportunities to enhance MealMatch by addressing its current limitations and further improving its personalization and efficiency.