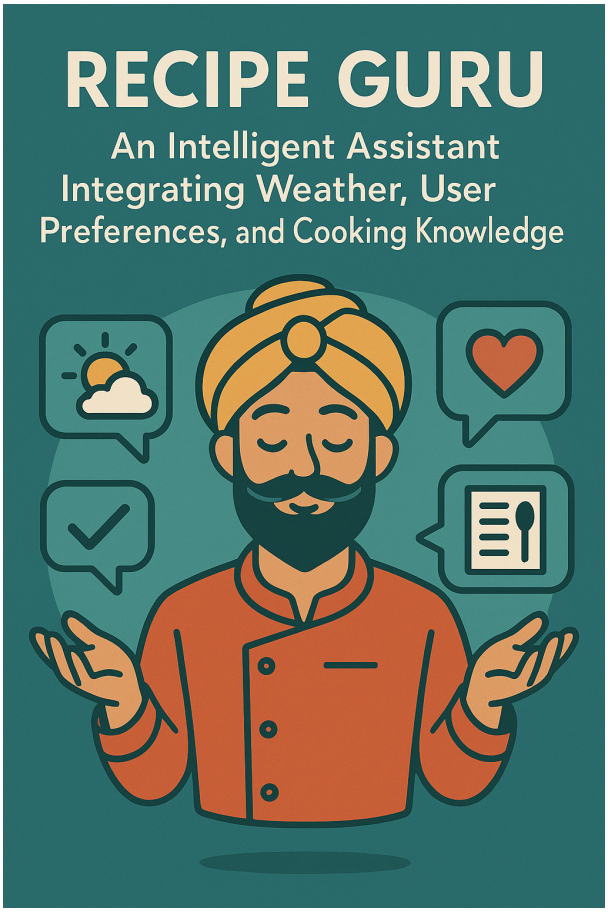
Recipe Guru: Weather-Aware Cooking Assistant

COMP47980: Generative AI and Language Models, Project Report, 2025

Student Name: **Arsenii Troitskii**
Student ID: **20204701**
Email: `arsenii.troitskii@gmail.com`



Contents

List of Figures

1. Introduction

This project details "Recipe Assistant", a personalized chatbot built using the OpenAI Assistants API [1]. It recommends recipes based on user context, including local weather, dietary needs, and preferences.

Standard LLMs lack real-time data (like weather) and user memory (preferences/allergies). Recipe Assistant addresses this by integrating APIs and a user database, offering dynamic, personalized, and relevant recipe suggestions superior to generic LLMs.

The assistant uses the OpenAI Assistants API [1] for core chat and Function Calling [2] to connect with external services. It queries OpenWeatherMap [5] for weather, Spoonacular [4] for recipes (based on weather, diet, etc.), and a local SQLite database [6] for user preferences and allergies.

2. Assistant Design: A Schematic View

The Recipe Assistant is designed as a modular, "kernel" system centred around the OpenAI Assistants API [1], augmented with specialised tools for accessing real-time data and user-specific information. The high-level architecture integrates several key components:

- **OpenAI Assistant Core:** This is the main mechanism, managing the conversation flow, interpreting user requests, and deciding when to utilize available tools. The system takes advantage of its natural language understanding and generation capabilities [1].
- **Function Calling Interface:** This important feature of the Assistants API [1] acts as the connection between the LLM core and external capabilities [2]. The assistant is provided with a defined set of callable functions.
- **User Profile Database:** A long-term storage layer implemented using SQLite [6]: *get_user_profile*, *save_user_preference*, *save_user_allergy* interact with this database to manage individual user preferences (likes/dislikes) and allergies, ensuring personalization across sessions.
- **User Interface (Enhanced Notebook Interaction):** Interaction occurs via Google Colab, leveraging its capabilities for richer output beyond simple text. Responses include formatted text (Markdown), embedded images (e.g., recipe photos via `IPython.display.Image`), and hyperlinks, providing a more visual and interactive user experience.
- **API Features Leveraged:** The primary feature utilized is *Function Calling* [2]. This allows the assistant to break free from the limitations of its static training

data and interact with the dynamic external world (e.g., weather services, extensive recipe database) and maintain stateful user information (such as user profiles).

- **Assistant Capabilities and Tool Usage:** The assistant employs a hybrid approach, combining custom function calls with OpenAI's built-in *File Search (RAG)* capability [3].
  - **Custom Functions:** Python functions connect to Spoonacular API [4], OpenWeatherMap API [5], and a local SQLite DB [6] for real-time recipe data, weather context, and user profile management (`find_recipes`, `get_weather`, etc.), enabling dynamic and personalized responses.
  - **File Search (RAG):** File Search is enabled via OpenAI Vector Stores [3], using an uploaded file with cooking tips. Setup involved a mix of SDK (for upload) and direct REST API calls (for Vector Store management, v2 beta) due to SDK limitations. This allows querying the file 's content, complementing API data.
  - **Code Interpreter:** The *Code Interpreter* tool remains unused in this implementation, as all necessary data processing, API interactions, database management, and RAG setup logic are handled externally within the Python environment and the defined custom functions.

  This combination allows the assistant to leverage both dynamic, real-time information from APIs and specific, static knowledge embedded via File Search [3].

External API Modules:

- **Weather Module:** Implemented with the OpenWeatherMap API via a Python function *get_weather* to get the current temperature and weather descriptions for a city, provided by the user.
- **Recipe Module:** Implemented with the Spoonacular API through two Python functions: *find_recipes* (for searching recipes based on complex criteria including weather, query, diet, intolerances, etc.) and *get_recipe_details* (for accessing detailed instructions and ingredients for a specific recipe ID).

Challenges in Implementing File Search (RAG)

File Search integration faced challenges due to SDK limitations (v1.76.0, see Figure 1) in supporting the required Assistants API v2 beta features like `tool_resources`. A hybrid solution using SDK for upload and REST API calls (with `OpenAI-Beta: assistants=v2` header) for vector store management was implemented.

```
import openai

print("OpenAI version:", openai.__version__)

try:
    from openai.resources import assistants_files
except ImportError as e:
    print(" ImportError:", e)
```

```
OpenAI version: 1.76.0
 ImportError: cannot import name 'assistants_files' from 'openai.resources' (/usr/local/lib/python3.11/dist-packages/openai/resources/__init__.py)
```

Figure 1: Illustration of an ImportError encountered with OpenAI SDK version

3. Added Value: More than Mere ChatGPT or basic LLM

Vanilla LLMs lack real-time context, persistent user memory, and domain-specific reliability crucial for tasks like personalized recipe advice. Recipe Guru overcomes this by integrating external tools (APIs for weather/recipes, SQLite DB for user profiles, RAG file for tips) via Function Calling. This combination allows the LLM core to dynamically access live data, recall user specifics (allergies/preferences), and use extra knowledge sources (RAG files), providing a significantly more relevant, personalized, and reliable experience than a standalone LLM.

3.1 Tool Usage

Recipe Guru utilizes several tools via the OpenAI Assistants API Function Calling mechanism to access external knowledge and capabilities. The general interaction flow between the user, the assistant, and the external tools is depicted in Figure 2.

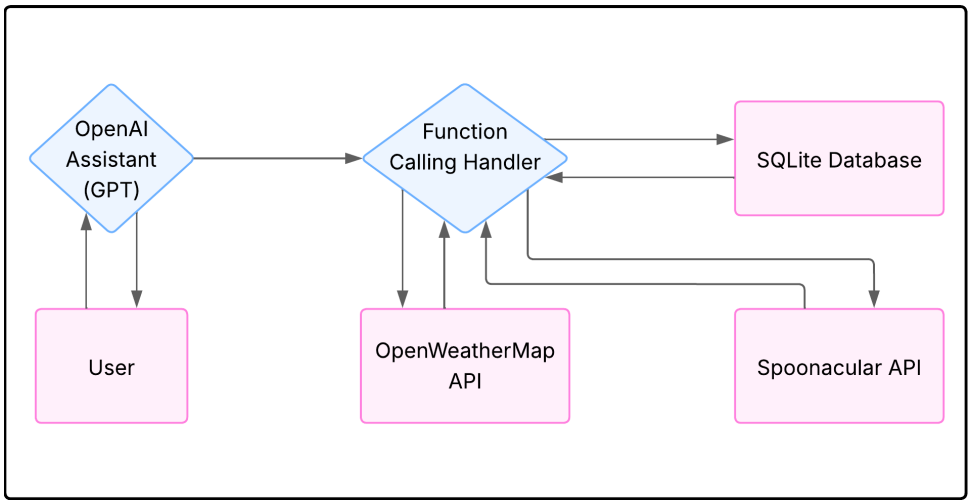


Figure 2: General workflow illustrating Assistant interaction with tools.

Custom Functions: Python functions were defined to bridge the LLM with specific external resources. A detailed example of the function call sequence for `find_recipes`, involving multiple API/DB interactions, is shown in Figure 3.

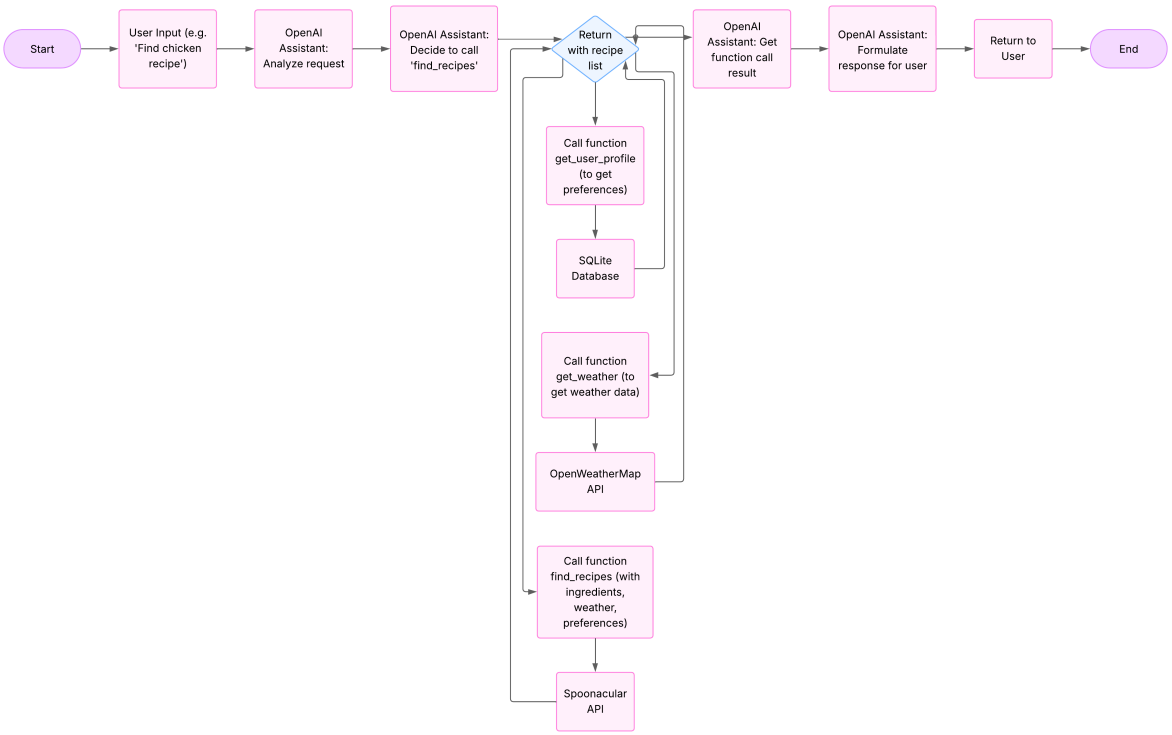


Figure 3: Detailed workflow for the `find_recipes` function call sequence.

The primary custom functions include:

- `get_weather`: Calls the OpenWeatherMap API for real-time weather context.
- `find_recipes` & `get_recipe_details`: Interact with the Spoonacular API for recipe searching (using context from weather/profile) and detail retrieval.
- `get_user_profile`, `save_user_preference`, `save_user_allergy`: Manage the local SQLite database (schema in Figure 4) for persistent user personalization.

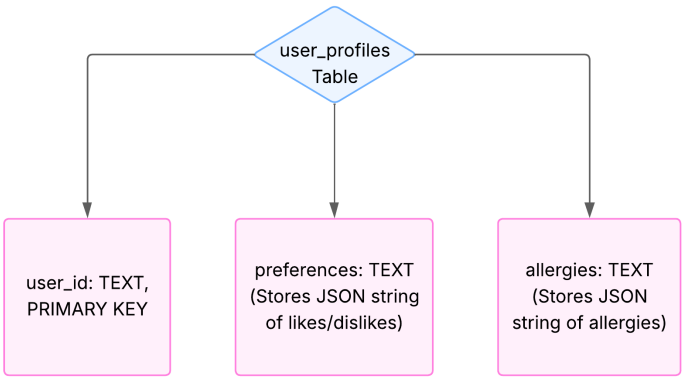


Figure 4: Schema of the `user_profiles` table in the SQLite database.

File Search (RAG): The assistant also uses File Search, leveraging an OpenAI Vector Store created from `cooking_tips_rag.txt`. This enables Retrieval-Augmented Generation for answering specific queries based on curated file content, complementing API knowledge.

These integrated tools allow Recipe Guru to provide dynamic, personalized, and context-aware responses, surpassing the capabilities of a standalone LLM.

4. Outside Knowledge: Curated Data Sources

Recipe Guru integrates several external knowledge sources:

4.1 OpenWeatherMap API

Provides real-time global weather data (e.g., temperature, conditions) via the `get_weather` function, offering environmental context. No preprocessing is applied. *Example Data:* `"temp": 18.5, "desc": "clear sky"`

4.2 Spoonacular API

Acts as the main culinary database (˜360k+ recipes). The `find_recipes` function filters based on context, and `get_recipe_details` retrieves structured recipe information. Minimal preprocessing; relies on API filtering. *Example Data:* `"id": 660306, "title": "Spinach Quiche"`

4.3 Local SQLite Database (User Profiles)

A local DB stores user preferences and allergies (in JSON format within the `user_profiles` table), enabling personalization and memory via functions like `get_user_profile`. *Example Data:* `("Arsenii", "Ïikes: [ïtalian]", "ällergies: [shellfish]")`

4.4 RAG File (Cooking Tips)

A small local text file (`cooking_tips_rag.txt`) containing significant tips was processed into an OpenAI Vector Store (using API v2 beta). This provides specific knowledge via File Search (RAG), complementing API data. *Example Data:* `Tip: Milk + vinegar for buttermilk.`

5. Worked Examples: Your Assistant in Action

This section illustrates the capabilities of Recipe Guru through specific interactions, comparing its performance to standard approaches where relevant.

5.1 Example 1: Personalization and Context Awareness vs. Generic Response

This example demonstrates how Recipe Guru utilizes user profile data and real-time weather information to provide a personalized interaction, contrasting with a generic response.

User Prompt: `"Suggest a healthy dinner suitable for the current weather in Dublin."`

Recipe Guru Interaction: Initially, the assistant gathers necessary context if not already known (Figure 5).

Figure 5: Recipe Guru requesting initial context (user name and location).

Once context is established (user "Arsenii" in "Dublin"), the assistant accesses the stored user profile and current weather via function calls (`get_user_profile`, `get_weather`). It then confirms understanding of preferences (likes Italian, avoids spicy, allergic to shellfish) and the weather (17.3°C, clear skies), tailoring the conversation accordingly (Figure 6).

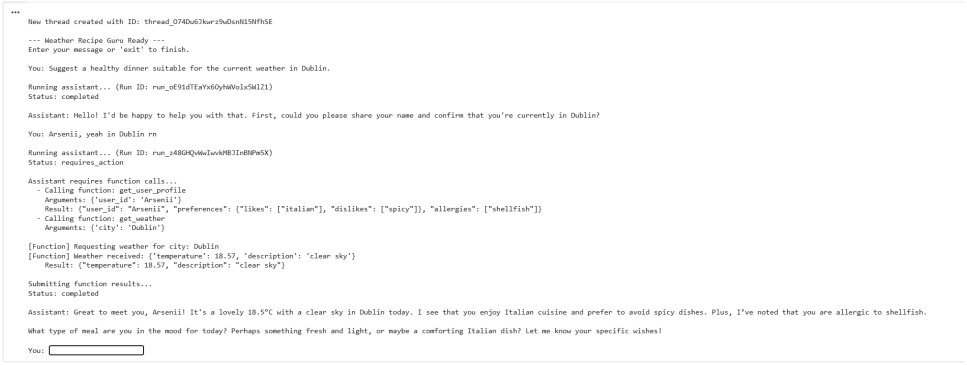


Figure 6: Recipe Guru using retrieved profile and weather data for personalized interaction.

Comparable Generic Response: In contrast, a standard LLM response (tested with ChatGPT 4o) to the same prompt provides a list of generally healthy recipes without acknowledging the user's specific location, current weather, or stored dietary profile (Figure 7).

Analysis: Recipe Guru's ability to integrate real-time data (weather via OpenWeatherMap API) and persistent user data (profile via SQLite DB) through Function Calling

Figure 7: Generic recipe list from ChatGPT 4o, lacking personalization and context.

results in a significantly more relevant and personalized interaction compared to the generic, context-unaware approach. This is particularly crucial for handling dietary restrictions like allergies, demonstrating clear added value.

5.2 Example 2: Weather-Based Request and Recipe Detail Retrieval

This example showcases the assistant's ability to handle a weather-related request and retrieve detailed recipe information, including visuals.

User Prompt: `"Yeah suggest me something for the weather around me. I wanna go for picknic"`

Recipe Guru Interaction: The assistant correctly interprets the request for picnic-appropriate food suitable for the current weather. It utilizes the Spoonacular API (`find_recipes`) to find suitable options and presents them, including an image for the first suggestion (Figure 8).

Upon user selection (`"i like the 2nd one, i like spinach"`), the assistant uses

Figure 8: Recipe Guru suggesting picnic recipes based on weather, including an image.

the recipe ID to call the Spoonacular API again (`get_recipe_details`) to fetch and display the ingredients (Figure 9) and instructions (Figure 10) for the chosen Spinach & Ham Quiche.

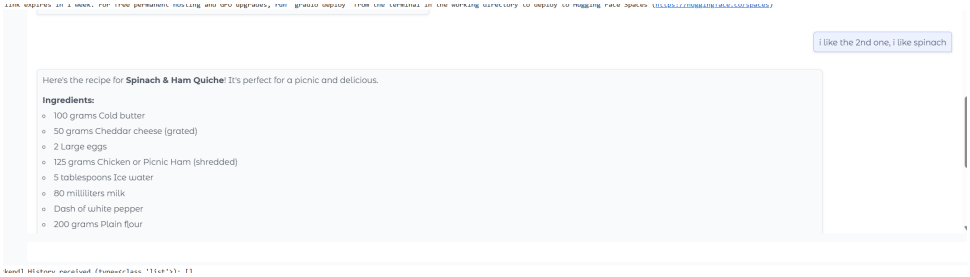


Figure 9: Recipe Guru displaying ingredients for the selected recipe.

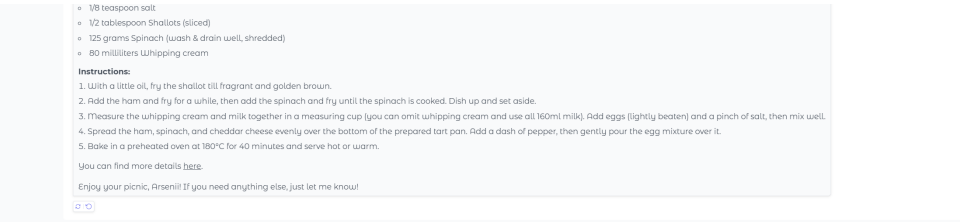


Figure 10: Recipe Guru displaying instructions for the selected recipe.

Result of comparison: This implementation highlights the effective use of the Spoonacular API for both searching recipes based on context (picnic, weather) and retrieving detailed information (ingredients, instructions, images). The assistant successfully manages the multi-turn conversation, guiding the user from the initial request to detailed recipes.

6. Summary and Conclusions

This project successfully developed "Recipe Assistant", a personalized conversational agent using the OpenAI Assistants API to suggest recipes based on real-time context like weather (via OpenWeatherMap), user preferences/allergies (via SQLite), and extensive recipe data (via Spoonacular). The integration, powered by Function Calling, effectively

addressed the limitations of standard LLMs, providing dynamic and relevant culinary guidance as demonstrated in the worked examples.

The assistant showcases the power of augmenting LLMs with external tools for specialized applications. Future evolution could significantly enhance its utility. For example:

- **Smart Home Integration:** Connecting Recipe Assistant to smart home ecosystems. Imagine the assistant preheating an oven based on a chosen recipe, adding ingredients to a smart fridge's shopping list, or adjusting kitchen lighting for cooking ambience. This would create a truly seamless kitchen experience.

Acknowledgements

Thanks to ChatGPT/SORA[10][9] for assistance with visualization and the logo.

I hereby acknowledge that this project and the accompanying report represent my own original work. The work is entirely my own, and every sentence in this report was written by me and me alone.

References

[1] OpenAI. Assistants API Overview. `https://platform.openai.com/docs/assistants/overview`

[2] OpenAI. Function Calling. `https://platform.openai.com/docs/guides/function-calling`

[3] OpenAI. Retrieval Augmented Generation (RAG) and File Search. `https://platform.openai.com/docs/assistants/tools/file-search`

[4] Spoonacular. Spoonacular Food API Documentation. `https://spoonacular.com/food-api/docs`

[5] OpenWeatherMap. OpenWeatherMap API Documentation. `https://openweathermap.org/api`

[6] SQLite. SQLite Home Page. `https://www.sqlite.org/index.html`

[7] Gradio. Gradio Documentation. `https://www.gradio.app/docs/`

[8] LaTeX Project. LaTeX - A document preparation system. `https://www.latex-project.org/`

[9] OpenAI. ChatGPT. `https://chat.openai.com/`

[10] OpenAI. Sora. `https://openai.com/sora`