By: Kenny Nguyen, Jackson Le, Joanne Lee, Beverley Appiatse

Data Science Project 2 Reflection Paper

For our Data Science Project 2, my group decided to implement a conversational chatbot application powered by generative AI that is designed to respond to user prompts and provide suggestions related to recipes. The project integrated Google's Gemini API and used Flask to develop a server to create a functioning backend to respond according to the input from the user. Our objective was to engage with a variety of technologies and data sets to enhance the user experience and show the processing of natural language and data retrieval. We implemented features for food recipes from a local SQLite database and drink recipes using the public Cocktail API to extend the features of the chatbot. The applications communicate and connect using a client-server architecture, with the server being deployed on Google Cloud Console as a virtual machine.

For the first feature of food recipes, we used a cleaned dataset of recipes and loaded them into the local SQLite3 database. We selected the global food recipe dataset because we thought that our bot should cover a variety of recipes, not just a one-dimensional cuisine. In choosing the dataset, we aimed to ensure the data was consistent and well structured, especially for the elements that were related to ingredients, which sometimes had irregular formatting. One of the challenges we encountered was managing the inconsistencies in the data itself. The naming of the column was super inefficient and unclear, so we renamed the column so that it is more clear what we are trying to retrieve. We then load the recipe onto an SQL database so that we don't have to keep extracting from the dataset, and we return a query of whatever food they want based on the query. We added a few error-handling features just in case the naming convention is off. If a response is valid, we return a JSON format text of the recipe and its elements to the client via the API.

For the second feature, we extracted the response from the cocktail API for whatever drink people feel like getting the recipe for. It works pretty similarly to the food recipe part in that it takes

the name of the specific drink that we want and returns the recipe on how to make it. We added some error handling too, just in case the API returns code 500 or something similar.

For the third feature, we implemented gemini chatbot where you can just submit a token of what question you want gemini to answer and it will return an answer, this is to generally allow more flexibility on the users to ask what they want, and not be limited to just preset recipes or learn more about nutrition informations.

This project taught us how many different technologies can be integrated to build intelligent systems. We were able to improve our skills with server-side programming using Flask, handling user input on the client side, database management, and using third-party APIs. A key lesson we learned was the importance of maintaining conversation context properly and ensuring that each request was interpreted properly based on the input type. In the future, we could enhance the chatbot by adding more advanced features, such as filters to allow more intents to be covered, so that many iterations of dialogues could be achieved. Like, say a user request for Korean food, we can pull out a list of all Korean recipes. Another feature we would have liked to implement is the Discord chatbot. While the current Python client works well, integrating the chatbot into Discord would make it more interactive and accessible for users on a daily basis.

Overall, this was a valuable learning experience that deepened our understanding of API integration, data processing, and backend development. By building a chatbot that combines generative AI with structured and unstructured data sources, we gained practical experience in designing scalable systems, handling data inconsistencies, and deploying real-world applications. This gives us more insight into how software chatbots are potentially made, and how it retrieves data to answer us efficiently.