Wine recommendation assistant Emily

Jin Giacomo

May 10, 2025

Contents

Chapter 1

Introduction

1.1 Emily: Specialized AI Sommelier for Everyday Wine Buyers

Buying a bottle of wine can be a very complex task to manage, especially when staring at a crowded supermarket shelf with no clue on which bottle suits you the most. To compensate to this problem, i have developed Emily, an AI assistant designed to simplify this experience. Unlike generic chatbots, Emily operates with a narrow but practical focus: recommending wines available in Irish supermarkets, using Tesco Irelands online inventory as its primary reference. Think of it as a sommelier who knows exactly whats on your local shelf and can explain options with a friendly, knowledgeable and witty personality.

Emily takes in input a photo of the wines displayed on the supermarket shelf the user is currently shopping in, and the users preferences to deliver a personalized suggestion of the best wine available. Users provide a URL of the shelf image and specify details like budget, preferred wine color, occasion, or flavor profile. The system then identifies visible bottles, cross-references them with curated data (RAG file that is webscraped from the tesco ireland website) and function calling of web search functions, and filters matches before generating a recommendation. To make interactions engaging, the assistant adopts the persona of Emy, a fictional but relatable French sommelier with a friendly, slightly witty tone, she is the kind of expert whod suggest "a bright, cherry-noted Pinot Noir for your pasta night" rather than lecturing you about different type of wine cultivations.

1.2 Why Emily and not Chatgpt?

Why build a specialized tool when ChatGPT exists? General-purpose language models fall short for this task in many reason. First, their training data lacks hyperlocal specificity; they might know Bordeaux blends in general but wont recognize a 12 bottle of Château Reynon currently stocked at Tesco. Second, they operate in a knowledge vacuum, unable to fetch real-time details like recent critic scores or user reviews. Emily tackles these gaps by integrating multiple technologies. OpenAIs vision capabilities allow it to "see" shelf images, while a custom database (scraped from Tescos site) grounds recommendations in actual inventory. For missing details, like a wines tasting notes and score, it quietly searches the web, ensuring suggestions arent just plausible but fact-checked.

1.3 What is Emily made of:

Emily runs as a Python application in Google Colab, built on OpenAIs Assistants API. The process begins when the user uploads a shelf image. The system uses GPT-4o vision model to extract wine labels and convert them into a structured json list. This list, combined with the users stated preferences, gets passed to the assistant to be evaluated. Emy first consults a curated dataseta JSON file of Tesco wines to fill in gaps about ABV, grape varieties, and prices. If crucial details like expert ratings are still missing, the assistant calls a Python function that scrapes the web for reviews, using the duckduckgo-search library to find credible sources. Once all data is assembled, Emily weighs options against the users criteria, selects the top match, and drafts a response in her signature style: helpful, concise, and lightly humorous. Where ChatGPT might hallucinate a wines price or availability, Emily's recommendations are tied to real stock and live data.

Chapter 2

Assistant Design: A Schematic View

This section provides a high-level overview of the Emily assistant's design, especially focusing on the interaction flow and the use of the OpenAI Assistants API features.

The main architecture use several different AI capabilities within the Assistants API framework and a initial vision processing using Chat gpt 4o API.

The workflow can be visualized as follows: Emily receives a wine shelf image URL and user preferences. Using **GPT-4o** it initially analyzes the image to create a basic JSON list of identified wines. This list, along with the preferences, is passed to the **Assistant** in a single run. the thread is then initialized and the Assistant first uses **File Search (RAG)** on a curated Tesco wine file to enrich the list with details like ABV and price. If needed, it triggers a **Function Call** to a Python function that performs a **web search** (via DuckDuckGo) for missing scores and tasting notes. After processing any web results, the Assistant analyzes the fully enriched wine data against user criteria, selects the best match, and generates the final **text-based sommelier recommendation**.

The primary API feature doesn't utilized a Code Interpreter because the tasks involved are about information retrieval (from image, file, web), data enrichment (adding details to the wine list), and text generation (recommendation)., and it

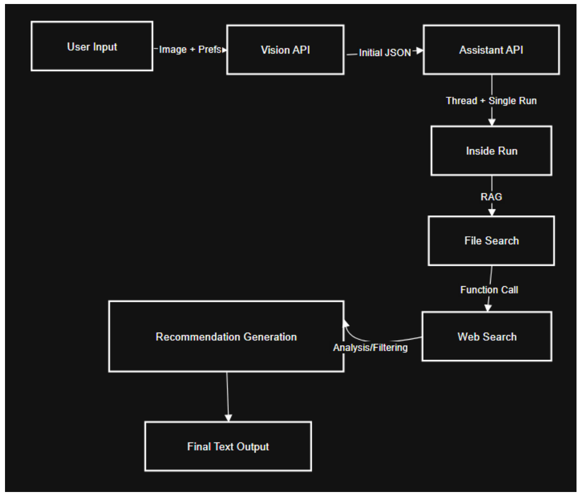


Figure 2.1: Workflow diagram

doesn't require runtime execution of Python code for complex calculations, data visualization, or analysis of user-uploaded files during the conversation.

After careful testing, a design choice was to implement everything within a single Assistant Run, because a two-run approach (one for enrichment of the json list, one for text generation recommendation wine) appeared less stable or more prone to context loss within the specific constraints of this implementation. While a two-run approach offers theoretical benefits in separating data retrieval from creative generation, the single-run method, carefully guided by a detailed prompt using a chain-of-thought approach, resulted functional for this project.

Another discarded implementation choice was the direct usage of Vivino API to enrich the list of recognized wines in the thread, but due to limited budget (it costed around 150 euro a month) the idea was not implemented.

To report an anomaly while implementing the RAG file, the Vector Store couldn't be initialized with client.beta.vector_stores, but by using client.vector_stores the problem was then solved.

Chapter 3

Added Value and Tool Usage

3.1 Added Value: More than ChatGPT

If we were to compare our assistant to simply chatting with a standard LLM like ChatGPT, it's undeniable that Emily provides much more added value. The key points in which she distinguish herself are:

- **Curated, Localized Knowledge (RAG):**

  ChatGPT has a trained general knowledge about all topics, but lacks specific and updated details about the wine product inventory, pricing variations, or new tasting reviews to wines availables specifically in Tesco Ireland.

  Emily utilizes File Search (RAG) over the curated 'tescoRAG.json' webscraped dataset to provide a focused knowledge relevant to Ireland retail territory, location for which our assistant is specialized in. This enables more accurate enrichment of details like grape varieties, ABV(alcohol percentage), or typical local pricing which most probably is missing or is outdated in the LLM's knowledge.

- **External Data Integration (Function Calling):** Wine scores, user reviews, and detailed tasting notes are dynamic information that is contained inside specific websites (ex. Vivino). Base LLMs cannot browse the web. But Emily can uses Function Calling to trigger real-time web searches using the ‘get-wine-reviews-from-web‘ function. This allows it to fetch current up-to date information related to ratings and reviews from websites like Vivino and it adds a layer of external validation and detail that is not present in the base model.

- **Comfort and quick access:** The Main point of using Emily is how quickly and easily you can fix a problem that otherwise would require a lot of research or a professional opinion on the matter. Throught only a photo and few clicks to set the user preferences, you will get the result(your wine reccomendation for the night) within few seconds whenever you need it.

By combining these features, Emily offer a service significantly more targeted, fast, convenient and potentially more accurate than using a standard LLM.

3.2 Tool Usage

Emily skills rely on the smart use of specific OpenAI Assistants API tools: File Search and Function Calling.

File Search (Retrieval-Augmented Generation):

- **Motivation:** The primary reason for using File Search is to obtain a persistent, curated knowledge base specific to the target domain (wines found in Tesco Ireland). Data extracted by the vision model can be incomplete or lack details. RAG allows the assistant to fill these gaps using reliable and fact-checked information, rather than relying solely on the generic LLM’s, that could hallucinates from time to time.

- **Implementation:** By using a webscraping python script and then cleaning the webscraped data, the dataset ‘tescoRAG.json‘ was built (derived from scraping Tesco.ie) and it contains details for numerous wines(400+ wine bottles). After uploaded and then coded into the Vector Store, the Assistant was then configured by enabling the ‘file-search‘ tool and linking this specific Vector Store

ID via the ‘tool-resources‘ parameter . The Assistant’s instructions explicitly direct it to consult this resource first when enriching the wine list identified from the wine shelf image recognization step.

Function Calling:

- **Motivation:** To cover up for the static nature of the information inside the RAG file, Emily uses Function Calling to find rapidly changing information and it enables Emily to query external sources on demand. It’s specifically used when the RAG file does not contain sufficient detail for ‘tasting-notes‘ and ‘score‘.

- **Implementation:** A function schema named ‘get-wine-reviews-from-web‘ was defined, specifying ‘wine-query‘ (containing producer, name) as the required input parameter. A corresponding Python function, ‘get-wine-reviews-from-web-impl‘, was implemented using the ‘duckduckgo-search‘ library. This function executes several targeted web searches (e.g., for Vivino ratings, general tasting notes, critic scores) based on the ‘wine-query‘. It processes the top few search result snippets and formats them into a JSON string, and returns this string. The main application code includes a polling loop that monitors the Assistant Run status. When the status becomes ‘requires-action‘, the loop extracts the ‘tool-call‘ details, executes the local ‘get-wine-reviews-from-web-impl‘ function with the arguments provided by the Assistant, and finally submits the returned data back to the Run. The Assistant then summarize the important information (scores, notes) from the obtained data using a witty and smart tone by following the instructions given.

These two tools work together perfectly: File Search provides the foundational, curated knowledge, while Function Calling offers a way to dynamically seek external elaboration, allowing Emily to build a much richer profile of each wine than possible with just one method.

Chapter 4

Outside Knowledge: Curated Data Sources

Emily source of curated outside knowledge is the dataset used for RAG using the File Search tool.

Data Origin: The dataset‘tescoRAG.json‘ was created by web scraping the Tesco Ireland grocery website (‘https://www.tesco.ie/groceries/en-IE/resources‘) filtering only the wine category This approach was chosen because we wanted to get as much local data as possible about wines available in most of the Irish supermarket chains, to provide direct relevanced information to the targeted user community(the Irish one).

Data Size and Scope: After cleaning the identified wine products (excluding different sizes or non-wine items of irrelevant features) from the initial scrape, the final dataset contained approximately 380+ unique wine entries. This size provides Significant coverage of the probable wine catalog present in the majority of stores in Ireland.

Cleaning and Preprocessing: The raw scraped data needed these following processing steps:

- Extract relevant fields: ‘producer-brand‘, ‘wine-name‘, ‘price-eur‘, ‘region‘ (derived from shelf/aisle name), ‘size‘/‘unitOfMeasure‘, ‘tesco-id‘, ‘image-url‘.

- Formatting the cleaned data into the JSON Lines format, and each line represents a single wine object.
- Adding placeholder keys (like ‘grape-type‘, ‘abv-percent‘, ‘tasting-notes‘, ‘score‘) set to ‘null‘ in the initial scraped data structure(needed so that the function calling would fill up afterwards).

Expert Data: This data allows Emily to confirm if a wine recognized visually is listed by Tesco, retrieve its current listed price, and potentially access basic categorization or other informations. It grounds the assistant's knowledge in a specific fact-checked retail environment.

Example Data Entry (from tescoRAG.json):

Listing 4.1: Example entry from the curated RAG file.


```
{
    "producer_brand": "A GABB FAMILY WINE",
    "wine_name": "A Gabb Family Wine Sheep Hill Sauvignon Blanc",
    "region": "South Africa",
    "price_eur": 18,
    "image_url": "https://digitalcontent.api.tesco.com/v2/media/ghs/84098e
    "size": "75cl",
    "type_grape": null,
    "abv_percent": null,
    "bottler_importer": null,
    "tasting_notes": null
  },
```

Chapter 5

Worked Examples: Your Assistant in Action

This section presents sample interactions to demonstrate its functionality.

Example 1: Basic White Wine Request

Input:

- Image URL: 'https://i.imgur.com/PHyvoLt.jpeg' (containing Viña Sol, Don Simon, Aresti, etc.)
- User Preferences (Code):

Listing 5.1: User Preferences for Example 1

```
price ="1" # Cheapest (<10)
color="2" # White
occasion="1" # Casual Chilling
body="2" # Medium-bodied
dryness="1" # Dry
flavour="1" # Fruity
food-pairing=""
```

Figure 5.1: Wine shelf 1

```
region="␣"
dislikes=""
```

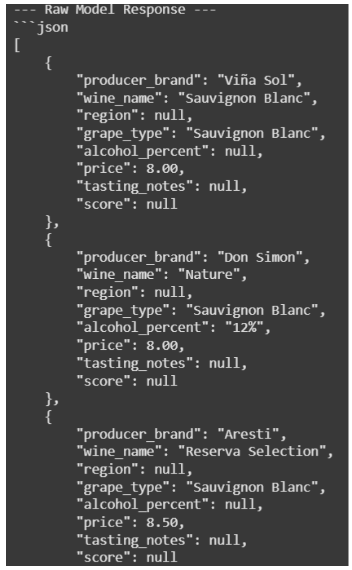
Interaction Flow and Analysis:

1. **Vision Step:** The ‘gpt-4o‘ vision model processes ‘img-link8‘ and identifies multiple bottles, including Viña Sol, Don Simon, Aresti, Santo Novo, Valentino, Cardos, etc., and return initial JSON list with mostly null values

2. **Assistant Run (Enrichment and Recommendation):** Once decided the user preferences, the single run is initiated with the initial JSON and the user preferences.

3. **RAG/File Search:** The Assistant consults the ‘tescoRAG.json‘ Vector Store. Whenever it finds a match, it announce it.

4. **Function Calling:** Because ‘tasting-notes‘ and ‘score‘ were missing, the Assistant calls the ‘get-wine-reviews-from-web‘ function multiple times. The Python

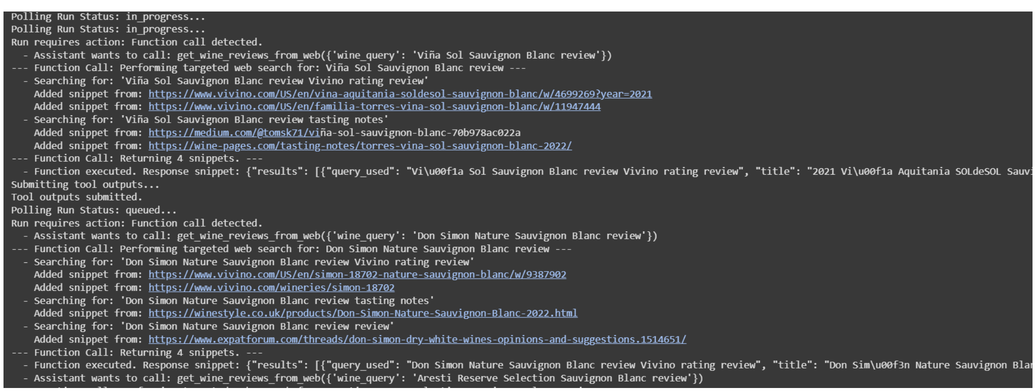


```
--- Raw Model Response ---
```json
[
    {
        "producer_brand": "Viña Sol",
        "wine_name": "Sauvignon Blanc",
        "region": null,
        "grape_type": "Sauvignon Blanc",
        "alcohol_percent": null,
        "price": 8.00,
        "tasting_notes": null,
        "score": null
    },
    {
        "producer_brand": "Don Simon",
        "wine_name": "Nature",
        "region": null,
        "grape_type": "Sauvignon Blanc",
        "alcohol_percent": "12%",
        "price": 8.00,
        "tasting_notes": null,
        "score": null
    },
    {
        "producer_brand": "Aresti",
        "wine_name": "Reserva Selection",
        "region": null,
        "grape_type": "Sauvignon Blanc",
        "alcohol_percent": null,
        "price": 8.50,
        "tasting_notes": null,
        "score": null
```

Figure 5.2: identified wines 1

function performs DDG searches and returns the results.

5. **Snippet Processing and Enrichment:** The Assistant processes the returned snippets. For Viña Sol, it extracts "Citrus notes, particularly elderflower and gooseberry" and a score of "3.5/5 on Vivino". It successfully populates many previously null fields.



```
Polling Run Status: in_progress...
Polling Run Status: in_progress...
Run requires action: Function call detected.
  - Assistant wants to call: get_wine_reviews_from_web({'wine_query': 'Viña Sol Sauvignon Blanc review'})
--- Function Call: Performing targeted web search for: Viña Sol Sauvignon Blanc review ---
  - Searching for: 'Viña Sol Sauvignon Blanc review Vivino rating review'
    Added snippet from: https://www.vivino.com/US/en/vina-aquitania-soldesol-sauvignon-blanc/w/4699269?year=2021
    Added snippet from: https://www.vivino.com/US/en/familia-torres-vina-sol-sauvignon-blanc/w/11947444
  - Searching for: 'Viña Sol Sauvignon Blanc review tasting notes'
    Added snippet from: https://medium.com/@tomsk71/viña-sol-sauvignon-blanc-70b978ac022a
    Added snippet from: https://wine-pages.com/tasting-notes/torres-vina-sol-sauvignon-blanc-2022/
--- Function Call: Returning 4 snippets. ---
  - Function executed. Response snippet: {"results": [{"query_used": "Vi\u00f1a Sol Sauvignon Blanc review Vivino rating review", "title": "2021 Vi\u00f1a Aquitania SOLdeSOL Sauvi
Submitting tool outputs...
Tool outputs submitted.
Polling Run Status: queued...
Run requires action: Function call detected.
  - Assistant wants to call: get_wine_reviews_from_web({'wine_query': 'Don Simon Nature Sauvignon Blanc review'})
--- Function Call: Performing targeted web search for: Don Simon Nature Sauvignon Blanc review ---
  - Searching for: 'Don Simon Nature Sauvignon Blanc review Vivino rating review'
    Added snippet from: https://www.vivino.com/US/en/simon-18702-nature-sauvignon-blanc/w/9387902
    Added snippet from: https://www.vivino.com/wineries/simon-18702
  - Searching for: 'Don Simon Nature Sauvignon Blanc review tasting notes'
    Added snippet from: https://winestyle.co.uk/products/Don-Simon-Nature-Sauvignon-Blanc-2022.html
  - Searching for: 'Don Simon Nature Sauvignon Blanc review review'
    Added snippet from: https://www.expatforum.com/threads/don-simon-dry-white-wines-opinions-and-suggestions.1514651/
--- Function Call: Returning 4 snippets. ---
  - Function executed. Response snippet: {"results": [{"query_used": "Don Simon Nature Sauvignon Blanc review Vivino rating review", "title": "Don Sim\u00f3n Nature Sauvignon Blar
  - Assistant wants to call: get_wine_reviews_from_web({'wine_query': 'Aresti Reserve Selection Sauvignon Blanc review'})
```

Figure 5.3: matched wines and function calls 1
```

6. **Filtering and Selection (Internal):** The Assistant analyzes the now-enriched list against the preferences (White, <10, Casual, Medium Body, Dry, Fruity).

````
### Summary of Enriched Wines

```json
[
    {
        "producer_brand": "Viña Sol",
        "wine_name": "Sauvignon Blanc",
        "region": "Penedès, Catalunya, Spain",
        "grape_type": "Sauvignon Blanc",
        "alcohol_percent": "13.00%",
        "price": 8.0,
        "tasting_notes": "Citrus notes, grapefruit, lemon, vibrant and refreshing.",
        "score": 3.9
    },
    {
        "producer_brand": "Don Simon",
        "wine_name": "Nature",
        "region": "Castilla La Mancha, Spain",
        "grape_type": "Sauvignon Blanc",
        "alcohol_percent": "12%",
        "price": 8.0,
        "tasting_notes": "Lemon, citrus, gooseberry, herbaceous.",
        "score": 3.1
    },
    {
        "producer_brand": "Aresti",
        "wine_name": "Reserve Selection",
        "region": "Curicó Valley, Central Valley, Chile",
        "grape_type": "Sauvignon Blanc",
        "alcohol_percent": "13.00%",
        "price": 8.5,
        "tasting_notes": "Bright citrus notes with a refreshing finish.",
        "score": 88
    },
````

Figure 5.4: enriched wines 1

7. **Final Output Generation:** The Assistant generates the final response, the *internal* enriched JSON list (which isnt shown in the front-end), and the final recommendation text for the selected wine (Don Simon).

```
Selection Based on Preferences

Given your preferences for a white wine that is
   medium-bodied, dry, fruity, and ideally priced under
   10, I recommend...
```
```

```
**Don Simon Nature Sauvignon Blanc**!

This delightful wine embodies a zesty vibrancy with
    juicy citrus notes of lemon and gooseberry. It's
    refreshingly easygoing, making it an ideal pairing for
    casual chilling or delightful summer picnics. At just
    8.0, its the perfect accompaniment to a light snack or
    vibrant salads, or simply savored on a lazy afternoon
    while basking in the sunshine. Grab a bottle and let
    the sunshine in!
```

Example 2: Red Wine for Party Request

Input:

- Image URL: "https://i.imgur.com/CMFJfaR.jpeg"
- User Preferences (Code):

Listing 5.2: User Preferences for Example 1

```
price ="2" # Mid Range
color="1" # Red
occasion="2" # Fancy Party
body="3" # Full-bodied
dryness="1" # Dry
flavour="5" # Complex
food-pairing="Cheese Platter"
region="Spain "
dislikes="Nothing too sweet"
```

Interaction Flow and Analysis:

1. **Vision Step:** The 'gpt-4o' vision model processes 'img-link6' and identifies multiple bottles and return initial JSON list with mostly null values

Figure 5.5: Wine shelf 2

2. **Assistant Run (Enrichment and Recommendation):** Once decided the user preferences, the single run is initiated with the initial JSON and the user preferences.

3. **RAG/File Search:** The Assistant consults the ‘tescoRAG.json‘ Vector Store. Whenever it finds a match, it announce it.

4. **Function Calling:** Because ‘tasting-notes‘ and ‘score‘ were missing, the Assistant calls the ‘get-wine-reviews-from-web‘ function multiple times. The Python function performs DDG searches and returns the results.

5. **Snippet Processing and Enrichment:** The Assistant processes the returned snippets. It successfully populates many previously null fields.

6. **Filtering and Selection (Internal):** The Assistant analyzes the now-enriched list against the preferences

7. **Final Output Generation:** The Assistant generates the final response, the *internal* enriched JSON list (which isnt shown in the front-end), and the final recommendation text for the selected wine (DADA Art Wine 2).

```
--- Raw Model Response ---
```json
[
    {
        "producer_brand": "DADA",
        "wine_name": "Art Wine #2",
        "region": null,
        "grape_type": null,
        "alcohol_percent": null,
        "price": "11.00",
        "tasting_notes": null,
        "score": null
    },
    {
        "producer_brand": "DADA",
        "wine_name": "Art Wine #3",
        "region": null,
        "grape_type": null,
        "alcohol_percent": null,
        "price": "11.00",
        "tasting_notes": null,
        "score": null
    },
    {
        "producer_brand": "DADA",
        "wine_name": "Art Wine",
        "region": null,
        "grape_type": null,
        "alcohol_percent": null,
        "price": null,
        "tasting_notes": null,
        "score": null
    },
```


Figure 5.6: identified wines 2

```
1. **DADA Art Wine #2**
   - **Producer:** DADA
   - **Region:** San Juan, Argentina
   - **Grape Type:** Merlot
   - **Alcohol Percent:** 13.5%
   - **Price:** €11.00
   - **Tasting Notes:** Notes of vanilla, oak, coffee; luscious and juicy.
   - **Score:** 91/100 .

2. **DADA Art Wine #3**
   - **Producer:** DADA
   - **Region:** San Juan, Argentina
   - **Grape Type:** Malbec
   - **Alcohol Percent:** 13.5%
   - **Price:** €11.00
   - **Tasting Notes:** Fruity aromas of ripe cherries and plums; oak notes of vanilla and chocolate.
   - **Score:** 3.6/5 .

3. **DADA Art Wine**
   - **Producer:** DADA
   - **Region:** San Juan, Argentina
   - **Grape Type:** Malbec
   - **Alcohol Percent:** 13.5%
   - **Price:** Approximately €11.00
   - **Tasting Notes:** Medium-strong oak with notes blending harmoniously with berry flavors.
   - **Score:** 3.6/5 .
```

Figure 5.7: enriched wines 2
```

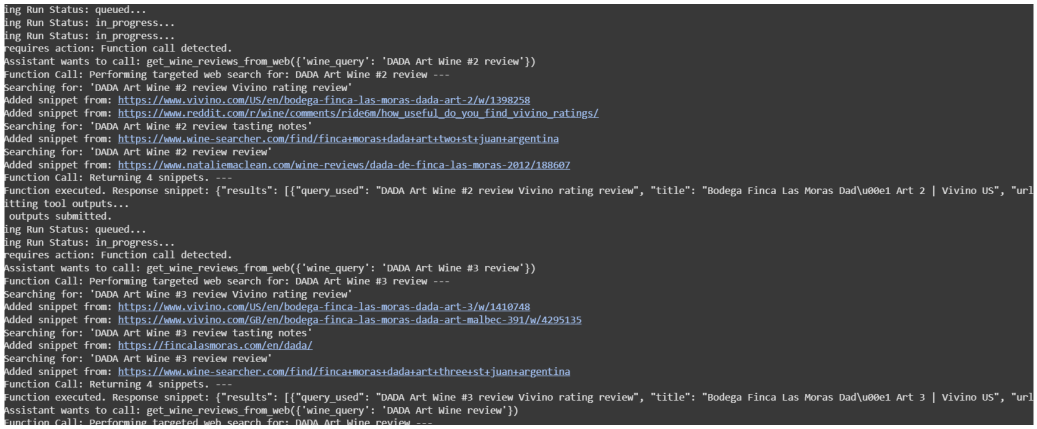


Figure 5.8: matched wines and function calls 2

```
After analyzing the fully enriched list, the **DADA Art
   Wine #2** stands out as an exceptional choice for your
   fanciful occasion. It is a full-bodied red wine with a
   combination of rich and bold flavors that would
   beautifully accompany a cheese platter, enhancing the
   experience with its complexity.

This wine's luscious profile and enticing notes of
   vanilla, oak, and coffee will surely impress your
   guests, making it a delightful piece for a fancy
   partycertainly not too sweet and absolutely within
   your preferred price range.

Here's your final recommendation:

**DADA Art Wine #2** is your go-to choice! Its rich
   Merlot character, harmonized with dark, decadent notes
   of vanilla and coffee, create a sumptuous experience
   worthy of your gathering. With each sip, guests will
   be enticed by the harmonious melding of oaky depth and
   juicy lushnessa true conversation starter and
```

```
   indulgent complement to a delightful cheese platter.
   Cheers to a splendid evening!
--------------------
```

Evaluation: Thess examples show the successful integration of wine recognization, RAG file search, Function Calling, and preference analysis within a single run. The main point that could lead to issues are the intermediate processing steps when the json list is passed between tasks. **ChatGPT Comparison:** Standard ChatGPT can give out generic advice but cant verify availability/price with a specific irish based catalog, see the original labels, or find real-time review information/scores.

Chapter 6

Summary and Conclusions

By analyzing the results, we can confirm that we developed an AI sommelier assistant, Emily, able to provide wine recommendations based on visual inputand user preferences. The assistant uses a combination of OpenAI's ‘gpt-4o‘ image recognition, File Search (RAG) with a curated dataset and Function Calling for external web search integration.

The Main accomplishment reached is the added value of the system that surpasses the capabilities of standard ChatGPT in this specific task.

The main problems encountered were for example to need to ensure that all the steps (RAG, Function Calling, Analysis, Formatting) were performed correctly within a single run. Because testing revealed frequent failures with a two-run approach, we chose to adopt the single run strategy. Another problem was the anomaly encountered with accessing the Vector Store API (‘client.vector-stores‘ vs. ‘client.beta.vector-stores‘), that highlights potential environment issues or library problems. A final problem observed was that the Assistant occasionally would include intermediate summaries and it wouldn't strictly conform to the requested final output format (returning only JSON or only text).

Future work could focus on several areas. Refining the RAG dataset by increasing its size, ensuring data accuracy. Revisiting the two-run architecture (Run 1 for enrichment/JSON output, Run 2 for recommendation/text output) could results in a more robust and easy to read system. More Function Calls could be used (if the budget permits it, for example Vivino API). Finally, developing a simple front-end graphical user interface for usability beyond the current notebook implementation.

In conclusion, the project successfully implemented an AI assistant able to create added value compared to basic LLM, specialized in a niche fields and it solves a concrete every day problem.

Acknowledgements and References

I would like to thank Professor Tony Veale for his teachings and interesting module slides which provided me the knowledge for this project. The work presented in this report, including all text and code unless otherwise cited, is entirely my own.

Bibliography

[1] OpenAI. (2024-2025). OpenAI API Documentation, Assistants API, Chat Completions API. Accessed [Specify Date Range, e.g., April-May 2025]. https://platform.openai.com/docs

[2] deedy5. duckduckgo-search Python Library. GitHub Repository. Accessed [Specify Date Range, e.g., May 2025]. https://github.com/deedy5/duckduckgo-search

[3] Python Software Foundation. Python Language Reference, version 3.11. Accessed [Specify Date Range, e.g., April-May 2025]. https://docs.python.org/3.11/reference/

[4] Google Research. Colaboratory. Accessed [Specify Date Range, e.g., April-May 2025]. https://colab.research.google.com/