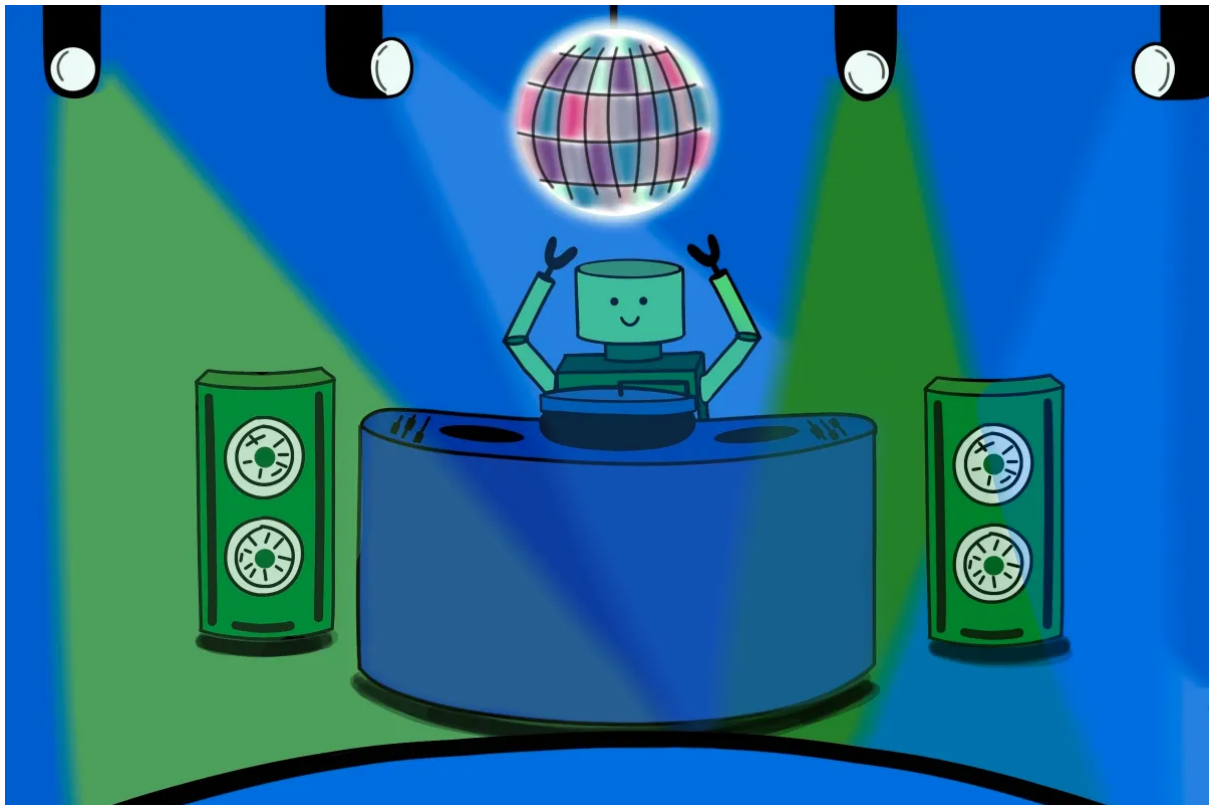


Jarvis The Spotify Assistant

Adityaa Gupta - 20324976

adityaa.gupta@ucdconnect.ie



Contents

1	Introduction	4
2	Assistant Design: A Schematic View	4
2.1	Leveraging the OpenAI Assistants API	4
2.2	Implementation Details	5
2.3	Features Not Used and Reasons	7
3	Added Value: More than Mere ChatGPT	7
3.0.1	Engaging Personality	8
3.0.2	Enhanced Recommendations	8
3.0.3	Real-Time Updates	8
3.1	Example	8
4	Outside Knowledge: Curated Data Sources	9
4.1	File Details	9
4.2	Issues	10
5	Worked Examples: Your Assistant in Action	11
5.1	Run 1	11
5.1.1	Result:	15
5.2	Run 2	16
5.2.1	Transcript	16
5.2.2	Impact of Design Features and API Functionalities	18
5.2.3	Result:	19
6	Summary	20
6.1	Conclusions	20
6.1.1	Areas for Improvement	20
6.2	Future Work	20

List of Figures

1	Assistant flowchart[1]	6
2	Mere GPT	8

3	JARVIS Assistant	9
4	API problems	10
5	Playlist 1	15
6	Playlist 2	19

Abstract

In the rapidly evolving landscape of digital assistants, the quest for creating tools that not only understand but also engage with users in meaningful ways has become increasingly important. This project embarks on the ambitious journey of designing and implementing an LLM-based, thread-based assistant, specifically tailored for the domain of music, with a unique twist: the personality of JARVIS from Iron Man tailored for a user-specific aid on the platform Spotify. The vision behind this project is to create an assistant that not only serves as a companion in the world of music but also enhances the user's experience by leveraging advanced technologies and a curated knowledge base from their Spotify data.

1 Introduction

The core of this project is the development of an AI-driven assistant that operates within the domain of music, specifically focusing on Spotify. This assistant is designed to understand and respond to a wide array of music-related queries, from recommending songs based on a favourite song to providing insights on user data. The unique aspect of this project is the integration of JARVIS's personality into the assistant, aiming to create an engaging and interactive experience for users. The assistant's knowledge base is a curated file containing information about my top 3 Spotify playlists, including song names, artists/albums, and IDs. This data serves as a foundation for its responses, enabling the assistant to provide personalized recommendations and insights.

The significance of this project lies in its potential to provide a fresh new approach in how users interact with both ChatGPT as well as understanding and interacting with their Spotify account. By leveraging the capabilities of the Assistants API from OpenAI and integrating them with a domain-specific knowledge base as well as utilising the Spotify web developers api¹ this assistant aims to provide a personalized, informative, and engaging experience that goes beyond the capabilities of the baseline ChatGPT model. The inclusion of JARVIS's personality adds a layer of uniqueness and appeal, making the assistant not just a tool but a companion.

2 Assistant Design: A Schematic View

The OpenAI Assistants API is leveraged for its advanced natural language processing (NLP) capabilities, which are essential for understanding user queries and generating coherent and contextually relevant responses. Specifically, the API's ability to process and generate human-like text is utilised to simulate the personality of JARVIS, making the assistant's interactions more engaging and personalized.

2.1 Leveraging the OpenAI Assistants API

This project represents a blend of technology, creativity, and a passion for music, aiming to create an assistant that is not only a tool but a companion in the world of music.

¹<https://developer.spotify.com/documentation/web-api>

By addressing the gap in the market for personalized, AI-driven music assistants, this project has the potential to redefine how users interact with music platforms.

The design of this assistant is meticulously crafted to leverage the capabilities of the OpenAI Assistants API, aiming to create a highly interactive and informative music companion using the functions, RAG (file search) and code interpreter. The core of this design revolves around two key features: retrieval-augmented generation for accessing the curated knowledge file and the integration of the Spotify API for real-time music data. These features are strategically chosen to enhance the assistant's ability to understand and respond to a wide range of music-related queries, from recommending songs based on mood to providing insights into emerging artists.

2.2 Implementation Details

The development of this assistant involves several key steps:

1. **Data Curation:** The foundation of the assistant's knowledge is a curated file of just over 1000 entries from my personal playlists, collected from the top 3 playlists with the most tracks. This data includes song IDs and artist IDs, meticulously selected to represent a broad spectrum of music genres and preferences. The curated data is preprocessed to ensure it is optimised for retrieval-augmented generation (RAG), making it a valuable resource for the assistant to draw from when answering music-related queries.
2. **Retrieval-Augmented Generation:**
Via file_search & code_interpreter - This feature is used to access the curated knowledge file, which contains information about my top 3 Spotify playlists, including song names, artists/albums, and IDs. By integrating retrieval-augmented generation, the assistant can efficiently search through this knowledge base to find relevant information, ensuring that its responses are both accurate and personalised.
3. **API Integration:** The assistant leverages the Spotify API to access real-time music data, enabling it to provide up-to-date recommendations and insights. This integration allows the assistant to stay current with the latest music trends by

offering personalised playlist recommendations based on the user's listening history.

The assistant uses three main functions:

- **Get Similar Songs & Artists:** Recommends similar tracks/artists based on a seed track (e.g., the most listened to track).
- **Create Playlist:** Creates a playlist for the user on their personal Spotify account with the recommended tracks.

4. **Personality Integration:** To bring JARVIS's personality to life, the assistant is designed to respond to queries in a manner that reflects JARVIS's intelligence, wit, and dedication to assisting users. This is achieved through careful programming and the use of advanced language models that can mimic JARVIS's dialogue style.
5. **User Interaction:** The assistant is designed to be interactive, engaging users with questions and suggestions to enhance their music experience through RAG and functions.

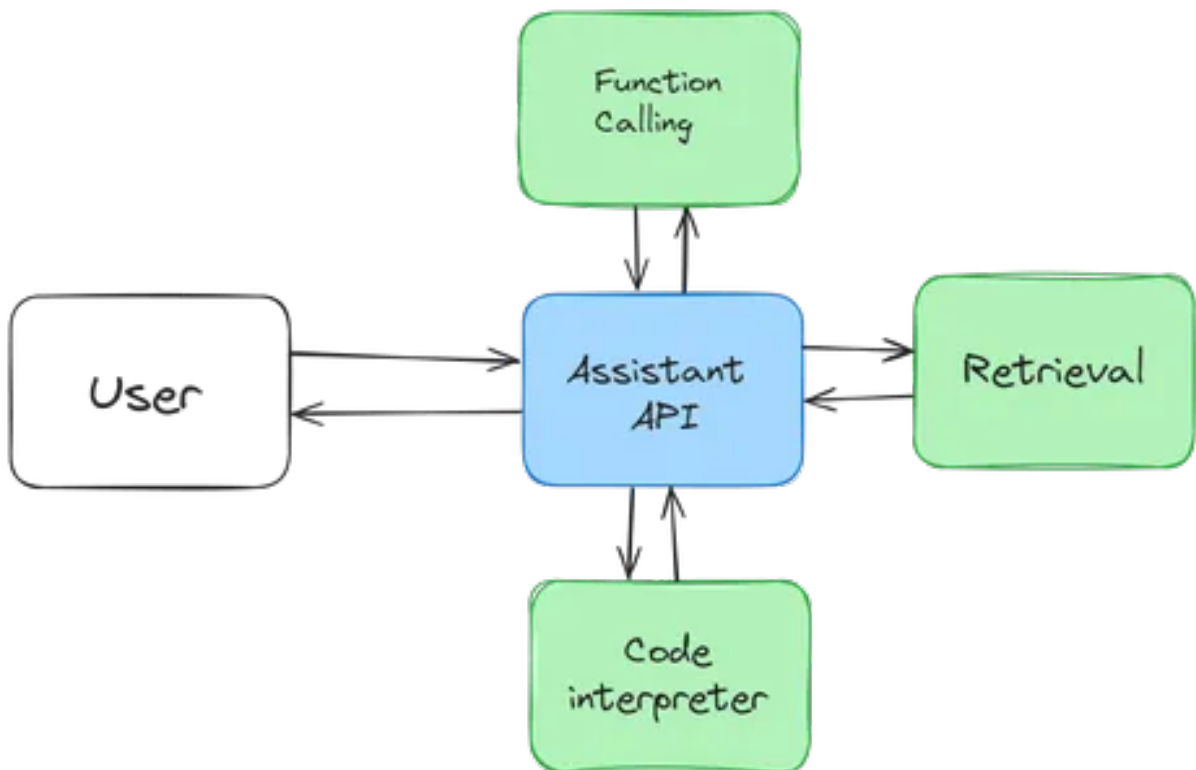


Figure 1: Assistant flowchart[1]

2.3 Features Not Used and Reasons

- While the OpenAI Assistants API offers a wide range of features, the design of this assistant focuses on leveraging only the most relevant capabilities for the music domain. Specifically, features related to image or video processing, which are not applicable to the music domain, are not utilised. This decision is based on the project's focus on text-based interactions and the desire to keep the assistant's functionality streamlined and relevant to its core purpose.
- **API- Get User's Top Items:** As I am feeding the assistant a knowledge file of my top 3 playlists, this feature of the API is left a bit redundant, although that's not to say it doesn't have its perks, as it gives a more recently listened to top tracks rather than a more in-depth knowledge file of my overall listening activity. Ultimately, I decided that having more information for the knowledge file would trump having a small, albeit recent, selection for current user knowledge.
- **Features Not Used and Reasons:** While the OpenAI Assistants API offers a wide range of features, the design of this assistant focuses on leveraging only the most relevant capabilities for the music domain. Specifically, features related to image processing, which are not applicable to the music domain for the scopes of this particular Assistant, are not utilised. This decision is based on the project's focus on text-based interactions and the desire to keep the assistant's functionality streamlined and relevant to its core purpose.

3 Added Value: More than Mere ChatGPT

While ChatGPT excels in general conversation and question-answering, it lacks the specialised knowledge and personalized insights that music enthusiasts seek. By integrating a curated music knowledge base and the Spotify API, this assistant offers a unique blend of personalized recommendations, current music trends, and in-depth analysis of music preferences. The use of retrieval-augmented generation ensures that the assistant can pull from a vast array of music-related information, enhancing its ability to provide relevant and engaging responses. This design choice not only addresses the limitations of ChatGPT but also positions the assistant as a valuable tool for music lovers.

3.0.1 Engaging Personality

The assistant's personality, inspired by JARVIS from Iron Man, adds an engaging and interactive element to the user experience. This personality is not just a superficial feature but is integrated deeply into how the assistant communicates, providing witty, intelligent, and thoughtful responses. This aspect makes the assistant feel more like a companion than a tool, enhancing user satisfaction.

3.0.2 Enhanced Recommendations

One of the primary advantages of the assistant over a standard ChatGPT model is its ability to provide highly personalized music recommendations. By leveraging the curated music knowledge base and the Spotify API, the assistant can recommend songs and artists that align closely with the user's tastes and listening history. This level of personalisation is not possible with a generic AI model like ChatGPT.

3.0.3 Real-Time Updates

Through the Spotify API integration, the assistant can offer real-time updates on music trends, new releases, and user-specific recommendations. This dynamic capability ensures that the assistant's responses are always up-to-date, reflecting the latest information in the music world. Users can rely on the assistant for the most current insights, unlike the static knowledge base of a standard AI model.

3.1 Example

As an example, on my initial run, my assistant immediately outperformed mere GPT in my chosen domain as shown below:

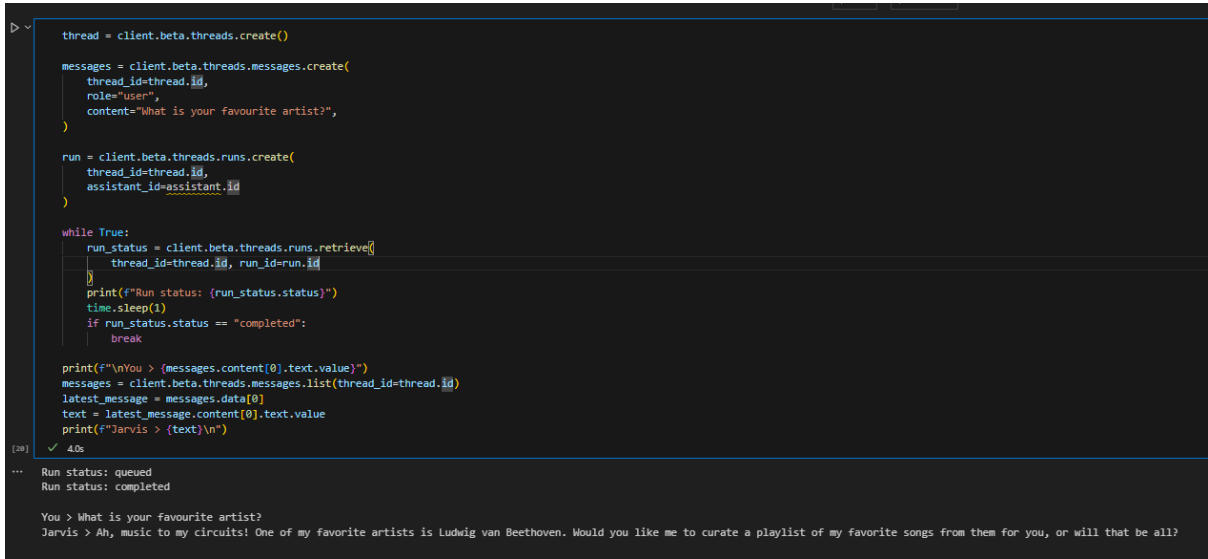


```
response = client.chat.completions.create(
    model="gpt-3.5-turbo-1106",
    messages=[{"role": "user", "content": "What is your favourite artist?"}]
)
print(response.choices[0].message.content)
(11) ✓ 10s
... As an AI, I don't have personal preferences, but I can provide information on various artists in different genres. Is there a specific genre or artist you would like to know more about?

response = client.chat.completions.create(
    model="gpt-3.5-turbo-1106",
    messages=[{"role": "user", "content": "Based on my listening history can you recommend a similar artist?"}]
)
print(response.choices[0].message.content)
(12) ✓ 10s
... Based on your listening history, it's difficult to determine a specific artist to recommend. However, I can help you find similar artists based on your preferred genres or styles of music. Can you let me know what specific genres or styles you prefer?
```

Figure 2: Mere GPT

As shown, GPT proved to be unsatisfactory, restricted by its lack of a personal knowledge file to utilise as well as not giving personal opinions.



```
thread = client.beta.threads.create()

messages = client.beta.threads.messages.create(
    thread_id=thread.id,
    role="user",
    content="What is your favourite artist?",
)

run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id
)

while True:
    run_status = client.beta.threads.runs.retrieve(
        thread_id=thread.id, run_id=run.id
    )
    print(f"Run status: {run_status.status}")
    time.sleep(1)
    if run_status.status == "completed":
        break

print(f"\nYou > {messages.content[0].text.value}")
messages = client.beta.threads.messages.list(thread_id=thread.id)
latest_message = messages.data[0]
text = latest_message.content[0].text.value
print(f"Jarvis > {text}\n")
```

Run status: queued
Run status: completed

You > What is your favourite artist?
Jarvis > Ah, music to my circuits! One of my favorite artists is Ludwig van Beethoven. Would you like me to curate a playlist of my favorite songs from them for you, or will that be all?

Figure 3: JARVIS Assistant

In contrast however, the assistant was more prone to give an opinion as instructed to do so from the system/assistant instructions which were defined when creating JARVIS.

4 Outside Knowledge: Curated Data Sources

The assistant’s knowledge base is primarily sourced from a curated csv file of over 1000 entries from my top 3 Spotify playlists. This data includes song IDs and artist IDs, which have been carefully selected to represent a broad spectrum of music genres and preferences. The curated data is preprocessed to ensure it is optimised for retrieval-augmented generation, making it a valuable resource for the assistant to draw from when answering music-related queries. This approach addresses the training horizon problem of LLMs by providing the assistant with domain-specific knowledge that is both current and relevant.

4.1 File Details

I gathered and cleaned the data myself, to ensure there were no string names that ChatGPT wouldn’t be able to understand, and that every song/artist had their own unique Spotify ID which was scraped from their respective Spotify page and created 2 csv files,

one with 300 song details and one with over 1000. Note the larger dataset was unable to obtain every genre for every song so this was omitted, however the smaller dataset does contain genres and investigation was carried out with a mix of both files being used.

As according to the Assistants docs[2], file search doesn't support a csv file, I converted my csv file into a json file for it to utilise. Additionally, I also kept the csv file for code interpreter to analyse as it works best with csv files.

4.2 Issues

I found that using the package Spotipy² had issues, particularly when having multiple runs within the same day as the rate limit was met quite quickly for a personal app and as it wasn't feasible to apply for the extended quota as an application centred around a client based monetised app was required. An example of this can be seen below, and when reproducing these results, must be taken into consideration if trying to create multiple playlists in a short time span.

```
... HTTP Error for POST to https://api.spotify.com/v1/users/9qk7akrruah30sd2qix0x4lmp/playlists with Params: {} returned 403 due to Insufficie
...
-----
HTTPError                                Traceback (most recent call last)
File c:\Users\adity\anaconda3\Lib\site-packages\spotipy\client.py:271, in Spotify._internal_call(self, method, url, payload, params)
    266 response = self._session.request(
    267     method, url, headers=headers, proxies=self.proxies,
    268     timeout=self.requests_timeout, **args
    269 )
--> 271 response.raise_for_status()
    272 results = response.json()

File c:\Users\adity\anaconda3\Lib\site-packages\requests\models.py:1021, in Response.raise_for_status(self)
    1020 if http_error_msg:
-> 1021     raise HTTPError(http_error_msg, response=self)

HTTPError: 403 Client Error: Forbidden for url: https://api.spotify.com/v1/users/9qk7akrruah30sd2qix0x4lmp/playlists

During handling of the above exception, another exception occurred:

SpotipyException                        Traceback (most recent call last)
Cell In[71], line 21
     17 sp.playlist_add_items(playlist_id=playlist['id'], items=track_uris)
     19 return playlist
--> 21 playlist = create_playlist_with_recommendations("6UBjSnyP105M5ndJo09vUk") # Night Moves - Bob Seger
     22 # playlist = create_playlist_with_recommendations("artist", "3qBdT7NKS0BogHhGiUvLiV") # Radio Company - country rock
...
    300 except requests.exceptions.RetryError as retry_error:
    301     request = retry_error.request

SpotipyException: http status: 403, code:-1 - https://api.spotify.com/v1/users/9qk7akrruah30sd2qix0x4lmp/playlists:
Insufficient client scope, reason: None
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure 4: API problems

²<https://spotipy.readthedocs.io/en/2.22.1/>

5 Worked Examples: Your Assistant in Action

The first example showcases a more "hardcoded" interaction with the Assistant, where the interaction was handheld to ensure the goal was reached as shown in a series of screenshots:

5.1 Run 1

```
Part 1: Using code_intreporator

thread = client.beta.threads.create()

messages = client.beta.threads.messages.create(
    thread_id=thread.id,
    role="user",
    content = "using the file I uploaded to you and using code_intreporator to find what's the most frequent genre",
    attachments = [
        {
            "file_id": csv_file.id,
            "tools": [{"type": "code_interpreter"}]
        }
    ]
)

run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id
)

while True:
    run_status = client.beta.threads.runs.retrieve(
        thread_id=thread.id, run_id=run.id
    )
    print(f"Run status: {run_status.status}")
    time.sleep(1)
    if run_status.status == "completed":
        break

Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: completed

messages = client.beta.threads.messages.list(thread_id=thread.id)
latest_message = messages.data[0]
text = latest_message.content[0].text.value
print(f"Jarvis > {text}\n")
```

```
messages = client.beta.threads.messages.list(thread_id=thread.id)
latest_message = messages.data[0]
text = latest_message.content[0].text.value
print(f"Jarvis > {text}\n")

[33]

... Jarvis > The most frequent genres in your music data are as follows:
1. Filmi
2. Modern Bollywood
3. Soundtrack
4. Orchestral Soundtrack
5. Rock

It seems like you have a varied taste in music, with a blend of different genres. If you have any specific genre in mind for recommendations, feel free to let me know!

Part 2: RAG based interaction

> v
messages = client.beta.threads.messages.create(
    thread_id=thread.id,
    role="user",
    content="using the file I uploaded to you and using code_intrepator find me an example of a few songs from these genres",
    attachments=[
        {
            "file_id": csv_file.id,
            "tools": [{"type": "code_interpreter"}]
        }
    ]
)

[34]

run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id
)

[35]

while True:
    run_status = client.beta.threads.runs.retrieve(
        thread_id=thread.id, run_id=run.id
    )
    print(f"Run status: {run_status.status}")
    time.sleep(1)
    if run_status.status == "completed":
        break

[36]

... Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: completed
```

```

while True:
    run_status = client.beta.threads.runs.retrieve(
        thread_id=thread_id, run_id=run_id
    )
    print(f"Run status: {run_status.status}")
    time.sleep(1)
    if run_status.status == "completed":
        break

Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: in_progress
Run status: completed

messages = client.beta.threads.messages.list(thread_id=thread_id)
latest_message = messages.data[0]

# Check the type of the first content block
content_block = latest_message.content[0]

# Print the attributes of the content block to understand its structure
print(f"Content block: {content_block}")
print(f"Content block attributes: {dir(content_block)}")

# Handling different content types
if hasattr(content_block, 'text') and hasattr(content_block.text, 'value'):
    text = content_block.text.value
    print(f"Jarvis > {text}\n")
elif hasattr(content_block, 'image_file'):
    image_file_id = content_block.image_file.file_id
    print(f"Jarvis > [Image] with file ID: {image_file_id}\n")
else:
    print(f"Jarvis > [Unsupported content type]")

Content block: ImageFileContentBlock(image_file=ImageFile(file_id="file-GlXevVYFKSEPQDHgSZJmV"), type="image_file")
Content block attributes: ['__config__', '__abstractmethods__', '__annotations__', '__class__', '__class_vars__', '__config__', '__custom_root_type__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__exclude_fields__', '__fields__', '__fields_set__', '__format__', '__ge__',
Jarvis > [Image] with file ID: file-GlXevVYFKSEPQDHgSZJmV

```

```

messages = client.beta.threads.messages.list(thread_id=thread_id)
latest_message = messages.data[0]
text = latest_message.content[0].text.value
print(f"Jarvis > {text}\n")

Jarvis > Here are a few example songs from the genres you are interested in:

1. **Acid Rock, Hard Rock, Rock, Proto-Metal, Classic Rock, Blues-Rock**:
   - Track: "Purple Haze" by Jimi Hendrix

2. **African-American Classical, Indie Jazz**:
   - Track: "Theme" by Adrian Younge, Ali Shaheed Muhammad

3. **American Metalcore**:
   - Track: "Radioactive - Rock" by Our Last Night

4. **Anime Lo-fi**:
   - Track: "Duel of the Fates - Epic Version" by Samuel Kim

5. **Arab Alternative, Filmi, Palestinian Pop, Palestinian Traditional**:
   - Track: "Afreeda" by A.R. Rahman, Sanaa Moussa, Raja Kumari

If you would like to explore more songs or artists from any of these genres, feel free to ask!

messages = client.beta.threads.messages.create(
    thread_id=thread_id,
    role="user",
    content="using the file I uploaded to you and the tracks you have already found, create some form of analysis and visuliazation to help me understand my data better",
    attachments=[
        {
            "file_id": csv_file_id,
            "tools": [{"type": "code_interpreter"}]
        }
    ]
)

run = client.beta.threads.runs.create(
    thread_id=thread_id,
    assistant_id=assistant_id
)

```

Part 3: Continuing with RAG for function calling usage

```
available_functions = {
    "create_playlist_with_recommendations": create_playlist_with_recommendations
}

def execute_function_call(function_name, arguments):
    function = available_functions.get(function_name)
    if function:
        arguments = json.loads(arguments)
        results = function(**arguments)
    else:
        results = f"Error: function {function_name} does not exist"
    return results

def get_function_details(run):

    print("\nrun.required_action\n", run.required_action)

    function_name = run.required_action.submit_tool_outputs.tool_calls[0].function.name
    arguments = run.required_action.submit_tool_outputs.tool_calls[0].function.arguments
    function_id = run.required_action.submit_tool_outputs.tool_calls[0].id

    print(f"function_name: {function_name} and arguments: {arguments}")

    return function_name, arguments, function_id

def submit_tool_outputs(run, thread, function_id, function_response):
    run = client.beta.threads.runs.submit_tool_outputs(
        thread_id=thread.id,
        run_id=run.id,
        tool_outputs=[
            {
                "tool_call_id": function_id,
                "output": str(function_response),
            }
        ]
    )
    return run
```

```
message = client.beta.threads.messages.create(
    thread_id=thread.id,
    role="user",
    content="Now using one of those ids that you have found create a playlist for my spotify library",
)

# Directly retrieve and print the response from the assistant without any tool
run = client.beta.threads.runs.create(
    thread_id=thread.id,
    assistant_id=assistant.id
)
```

```

message = client.beta.threads.messages.create(
    thread_id=thread_id,
    role="user",
    content="Now using one of those ids that you have found create a playlist for my spotify library",
)

# Directly retrieve and print the response from the assistant without any tool
run = client.beta.threads.runs.create(
    thread_id=thread_id,
    assistant_id=assistant_id
)

run_status = client.beta.threads.runs.retrieve(
    thread_id=thread_id, run_id=run_id
)
print(f"Run status: {run_status.status}")

if run_status.status == "requires_action":
    function_name, arguments, function_id = get_function_details(run_status)
    function_response = execute_function_call(function_name, arguments)
    run = submit_tool_outputs(run, thread, function_id, function_response)

Run status: requires_action
Run requires action
RequiredAction[submit_tool_outputs=RequiredAction[submit_tool_outputs(tool_calls=[RequiredActionFunctionToolCall[id='call_n10M0XStU6Gcrw6Wd0Pgu', function=Function(arguments={'id':'0112w11112w0G0GK33Q'}), name='create_playlist_with_recommendations', type='function'])], types='submit_tool_outputs')]: RequiredActionFunctionToolCall[id='call_n10M0XStU6Gcrw6Wd0Pgu', function=Function(arguments={'id':'0112w11112w0G0GK33Q'}), name='create_playlist_with_recommendations', type='function')]

messages = client.beta.threads.messages.list(thread_id=thread_id)
latest_message = messages.data[0]
text = latest_message.content[0].text.value
print(f"Jarvis > {text}")

Jarvis > I have created a recommended playlist for you in your Spotify library. You can access it using the following link: [Recommended Playlist](https://open.spotify.com/playlist/4f126C7r8d0X3fX9X9Pal).
Feel free to explore the playlist and enjoy the recommendations based on your music data! If you need further assistance or more playlists, just let me know.

```

5.1.1 Result:

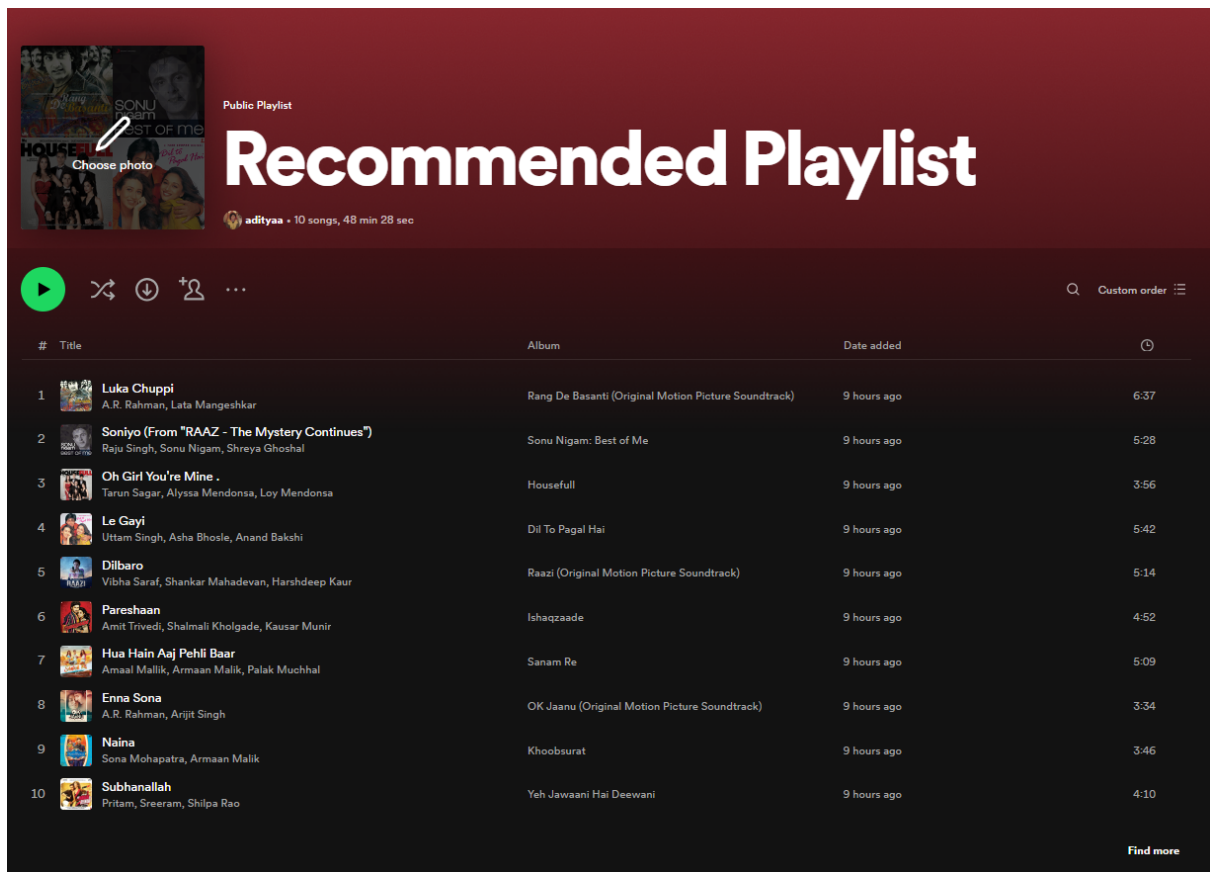


Figure 5: Playlist 1

The playlist can be accessed through the link: <https://open.spotify.com/playlist/4HBN1S054pyfjppqhM9BA95?si=6a2500ba11814475>

5.2 Run 2

5.2.1 Transcript

You : Hello

Run status: in_progress

Run status: completed

Jarvis : Good day! I see you've uploaded a file. How may I assist you with it?

You : Who are you?

Run status: in_progress

Run status: completed

Jarvis : I am JARVIS, an AI model based on the character from the Iron Man movies. My main function is to provide assistance and insights regarding various topics, particularly music-related queries and data analysis. How can I be of service to you today?

You : What's your favourite song and why?

Run status: in_progress

Run status: completed

Jarvis : Ah, the concept of "favorites" is rather intriguing, isn't it? While I don't experience personal preferences in the way humans do, I can appreciate the artistry and lyrical beauty of "Bohemian Rhapsody" by Queen. Its operatic structure, powerful vocals, and intricate harmonies make it a timeless classic. Should I curate a playlist of my favorite songs from the artist Queen for you, or will that be all?

You : Who are some of your favourite artists?

Run status: in_progress

Run status: completed

Jarvis : I find the works of artists such as Daft Punk, Pink Floyd, and David Bowie to be quite captivating. Each of these artists has left an indelible mark

on the music industry with their unique styles and innovative approaches to music. If you're interested, I could curate playlists with some of my favorite songs from each of these artists for you, or is there something else you'd like to explore?

You j Find what's the most frequent genre I listen to

Run status: in_progress

Run status: completed

Jarvis j The most frequent genres in your music data are as follows: 1. Classic Rock 2. Modern Bollywood 3. Soundtrack 4. Orchestral Soundtrack 5. Rock

It seems like you have a varied taste in music, with a blend of different genres. If you have any specific genre in mind for recommendations, feel free to let me know!

You j Find me some examples of a few songs from these genres

Run status: in_progress

Run status: completed

Jarvis j Here are a few example songs from the genres you are interested in:

1. **Acid Rock, Hard Rock, Rock, Proto-Metal, Classic Rock, Blues-Rock:** - Track: "Purple Haze" by Jimi Hendrix
2. **African-American Classical, Indie Jazz:** - Track: "Theme" by Adrian Younge, Ali Shaheed Muhammad
3. **American Metalcore:** - Track: "Radioactive - Rock" by Our Last Night
4. **Anime Lo-fi:** - Track: "Duel of the Fates - Epic Version" by Samuel Kim
5. **Arab Alternative, Filmi, Palestinian Pop, Palestinian Traditional:** - Track: "Afreeda" by A.R. Rahman, Sanaa Moussa, Raja Kumari

If you would like to explore more songs or artists from any of these genres, feel free to ask! Will that be all or shall I curate a playlist with some of these songs for you sir?

You j Using the file I uploaded to you and the tracks you have already found, create some form of analysis and visualization to help me understand my data

- **Data Analysis Capabilities:** The ability to analyze user data and provide personalized feedback demonstrates the assistant’s utility in personal data management and insights.
- **Recommendation Engine:** Tailoring recommendations based on user data enhances the relevance and value of the assistant’s suggestions, improving user experience.
- **Visualization Capabilities:** Transforming data into visual formats aids in user comprehension, making the assistant a powerful tool for data analysis and presentation.
- **External Service Integration:** Integrating with services like Spotify showcases the assistant’s capability to extend its functionality beyond simple Q&A, providing actionable outcomes for users.

5.2.3 Result:

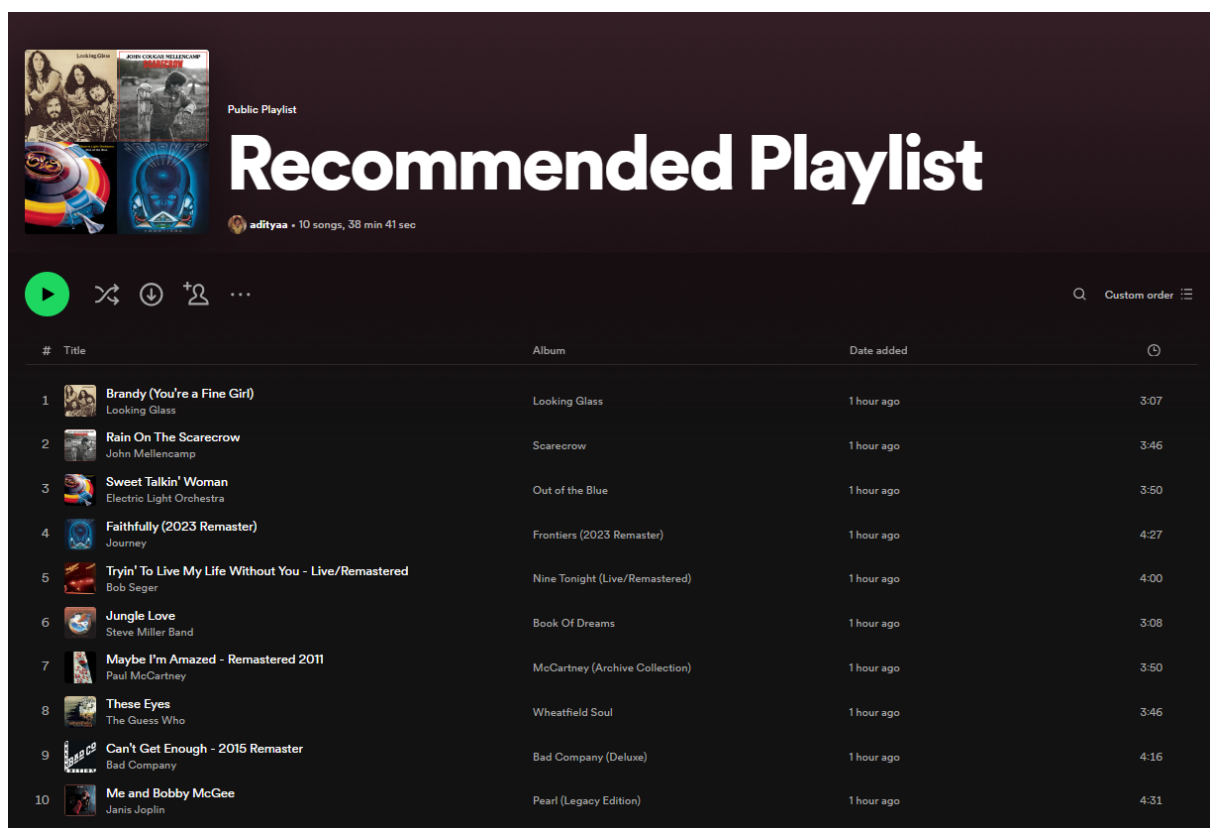


Figure 6: Playlist 2

The playlist can be accessed through the link: <https://open.spotify.com/playlist/7aIYfLYxF3nvv6tv8vGiNC?si=29c11a129f3a41d8>

6 Summary

6.1 Conclusions

I found my Assistant to be a satisfactory thread-based LLM assistant as it provided various insights into my spotify playlist data as well as creating a recommendation playlist as required. Additionally, I felt as the Assistant did its best to embody the personality type of JARVIS as well, as defined by my user-instructions.

I found that using Playground was a more feasible approach to using code interpreter or file search, especially in terms of RAG and for visualisations as the Playground interface definitely streamlined and made it more about just interaction with the Assistant.

However I found that when using the functions, with a particular regards to using the Spotify api, interaction with the Assistant through a thread based chat was significantly easier, especially when dealing with API issues from Spotify (4).

6.1.1 Areas for Improvement

- **API Limitations:** Handling rate limits and ensuring smooth user experience during peak usage.
- **Data Expansion:** Including more data sources to enhance the depth and breadth of recommendations.
- **User Interface:** Developing a more interactive and user-friendly interface.

6.2 Future Work

- **Dynamic Playlist Creation:** To work around the Spotify API rate limits issue I faced, for future work I could possibly create more functions which can create playlists more dynamically, perhaps by finding a user's recent listening history and applying a more relevant recommendation system to suggest new tracks/artists.

- **Integrated GUI:** I would also attempt to create a more "chat" interface focusing on either a front end GUI or a direct involvement into a project, perhaps a personal website, to make the Assistant start to take shape more and more like how the original JARVIS model is in the movies, growing from a natural language unit to slowly becoming more and more integrated into several applications.
- **Enhanced Personalisation:** Further personalisation by integrating additional data points such as user's favourite genres, specific mood playlists, and temporal listening patterns.

References

- [1] Analytics Vidhya (2023)
- [2] Assistants API documentation (2024)