# Enhance ShopAssist AI (ShopAssist 2.0)

Submitted by: Shabnam Sardar, MLAI Upgrad IITB

# Introduction

While ShopAssist 1.0 is a robust system, there's always room for improvement. A notable challenge we've encountered is the complexity of managing multiple layers and components to achieve seamless and efficient interactions between the chatbot and users.

The objecOve of this assignment is to enhance ShopAssist AI by leveraging the newly introduced 'function calling' feature. This will enable us to enhance the customer experience by creating a more streamlined and efficient chatbot, gaining valuable experience in conversational AI while contributing to the development of innovative AI solutions.

Here are the tasks involved.

- **Integrate Function Calling API:** Modify the existing architecture to leverage the Function Calling API's capabilities for improved performance. Scope out which layers can be removed and how the existing layers can be updated to handle the new approach.
- **Refine Conversation Flow:** Enhance the chatbot's conversation flow by leveraging function calling. Make the interaction between the user and the chatbot more natural and dynamic.
- **Document and Present:** Create comprehensive documentation that outlines the changes made, the integration process, and the benefits of using the Function Calling API. Prepare a presentation to showcase the enhancements.

Function calling is a new capability in OpenAI's GPT-4-0613 and GPT-3.5 Turbo-0613 models. Instead of just generating plain text responses, these AI models are now equipped to recognise situations where they must interact with external systems. Function calling allows the AI to trigger specific functions based on the input of the user.

# Project Background

In today's digital age, online shopping has become the go-to option for many consumers. However, the overwhelming number of choices and the lack of personalized assistance can make the shopping experience daunting. To address this, we have developed ShopAssist AI, a chatbot that combines the power of large language models and rule-based functions to ensure accurate and reliable information delivery.

# Problem Statement

Given a dataset containing information about laptops (product names, specifications, descriptions, etc.), build a chatbot that parses the dataset and provides accurate laptop recommendations based on user requirements.

# Approach

1. **Conversation and Information Gathering**: The chatbot will utilize language models to understand and generate natural responses. Through a conversational flow, it will ask relevant questions to gather information about the user's requirements.

2. **Information Extraction**: Once the essential information is collected, rule-based functions come into play, extracting top 3 laptops that best matches the user's needs.

3. **Personalized Recommendation**: Leveraging this extracted information, the chatbot engages in further dialogue with the user, efficiently addressing their queries and aiding them in finding the perfect laptop solution.

# System Design

**Dataset**: We have a dataset laptop.csv where each row describes the features of a single laptop and also has a small description at the end. The chatbot that we build will leverage LLMs to parse this Description column and provide recommendations.

Here's the overall flow of conversation for the ShopAssist Chatbot: The chatbot should ask a series of questions to determine the user's requirements. For simplicity, we have used 6 features to encapsulate the user's needs. The 6 features are as follows:

1. GPU intensity
2. Display quality
3. Portability
4. Multitasking
5. Processing speed
6. Budget

Confirm if the user's requirements have been correctly captured at the end.

After that the chatbot lists down the top 3 products that are the most relevant and engages in further conversation to help the user find the best one.

# Building the Chatbot

## ShopAssist1.0

In **ShopAssist1.0** , these are three stages of the chatbot, which are as follows:

▪ Stage 1: Intent Clarity and Intent Confirmation

▪ Stage 2: Product Extraction and Product Mapping

▪ Stage 3: Product Recommendation Implementation

Below are functions used in ShopAssist1.0

- initialize_conversation(): This initializes the variable conversation with the system message.

- get_chat_completions(): This takes the ongoing conversation as the input and returns the response by the assistant

- moderation_check(): This checks if the user's or the assistant's message is inappropriate. If any of these is inappropriate, it ends the conversation.

- intent_confirmation_layer(): This function takes the assistant's response and evaluates if the chatbot has captured the user's profile clearly. Specifically, this checks if the following properties for the user has been captured or not GPU intensity, Display quality, Portability, Multitasking, Processing speed, Budget

- dictionary_present(): This function checks if the final understanding of user's profile is returned by the chatbot as a python dictionary or not. If there is a dictionary, it extracts the information as a Python dictionary.

- compare_laptops_with_user(): This function compares the user's profile with the different laptops and come back with the top 3 recommendations.

- initialize_conv_reco(): Initializes the recommendations conversation

## ShopAssist2.0

In **ShopAssist2.0** , we leverage from Function calling feature of OpenAI. It improves performance and simplifies architecture of the chatbot. Below are the simplified stages in 2.0 version

▪ Stage 1: Intent Clarity

▪ Stage 2: Product Mapping

▪ Stage 3: Product Recommendation

Below are functions used in ShopAssist2.0

- initialize_conversation()
- get_chat_completions with and without function calling
- moderation_check()
- compare_laptops_with_user()

Some layers are removed while the ShopAssist AI was adapted for Function Calling as they became redundant like

- intent_confirmation_layer
- dictionary_present layer
- initialize_conv_reco layer

# Benefits

Major Advantages of function calling are as follows:

-Based on the user interaction, as per defined function schema, OpenAI API decides when to call what function automatically. Let us assume it is function X

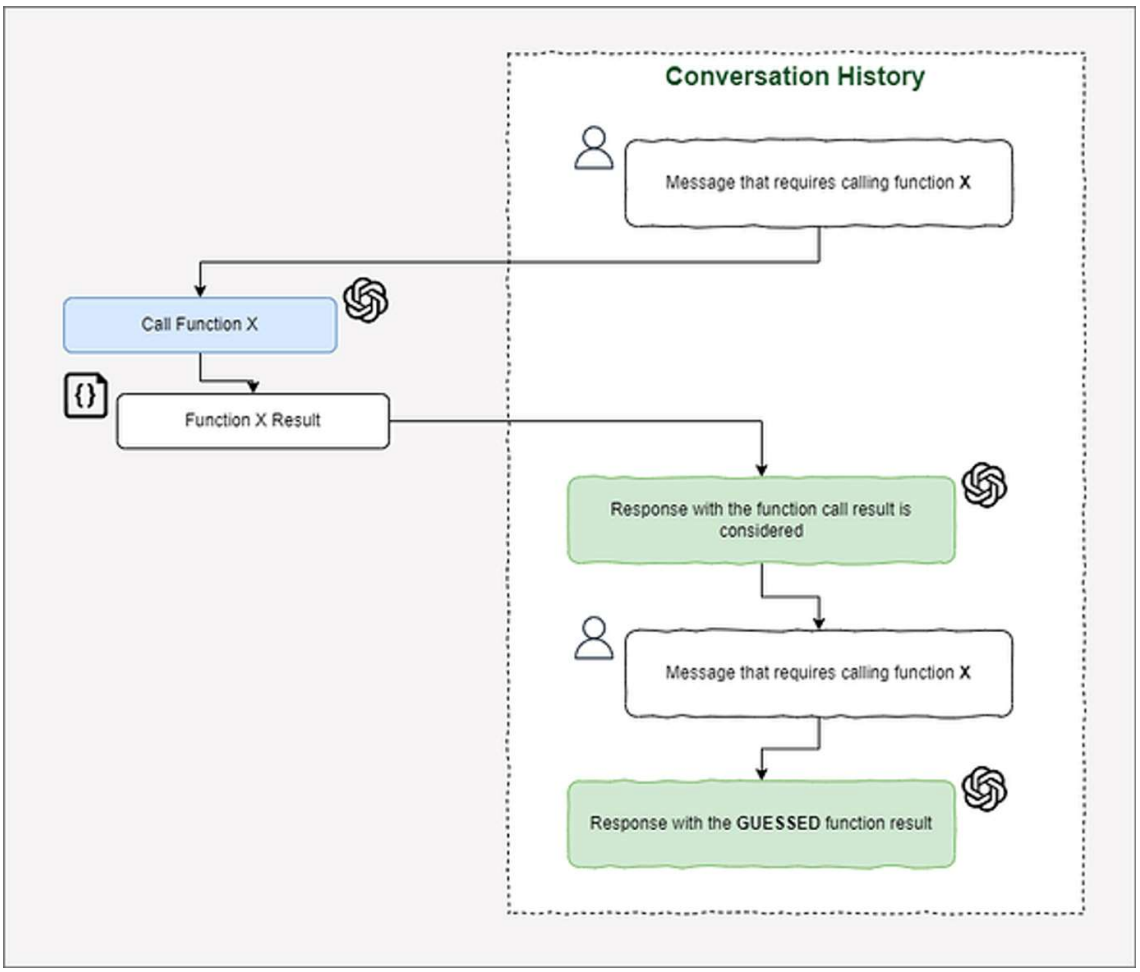
- Hence, it removes a need to code separately in the solution and that is why we have removed intent_confirmation function.

- Output of function X is further personalized and presented to user.

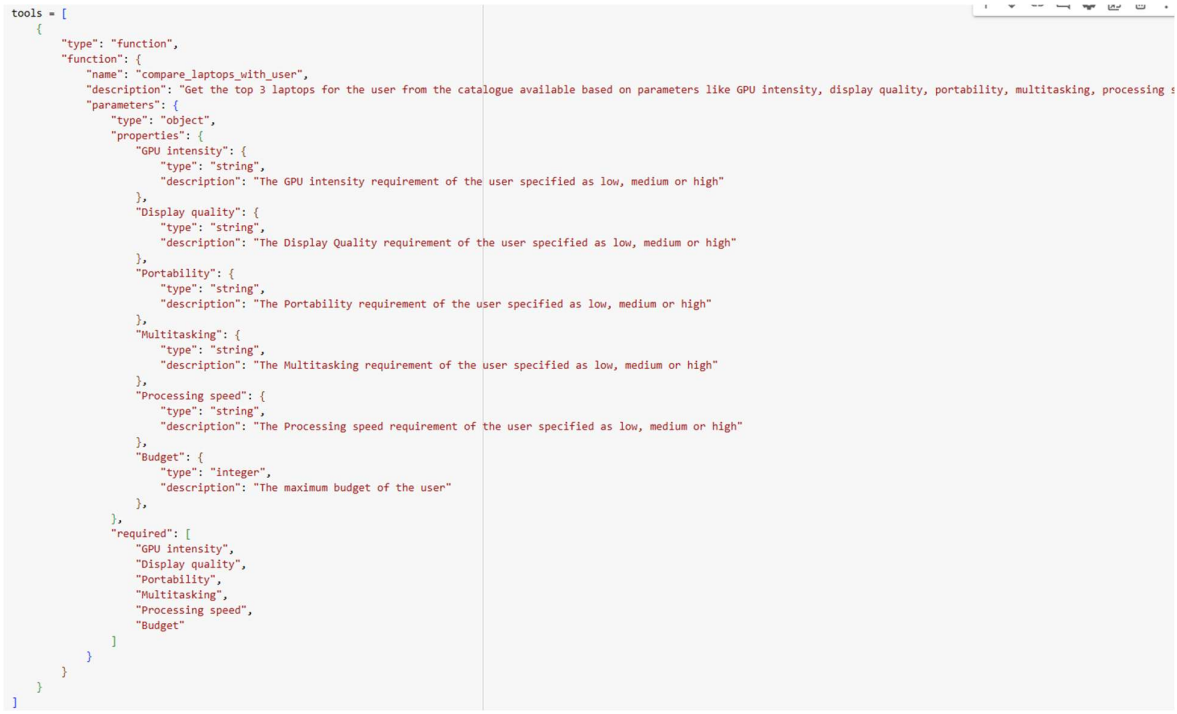
- It also simplifies the over code for dialogue_mgmt_system() unlike when we have not used function calling.

Function calling makes conversations more interactive. Users can trigger specific actions or get precise answers without the need for ambiguous back-and-forth, allowing dynamic changes to accept.

Below screenshot is for better visualization of how function calling works.



Below is the function schema defined for function **“compare_laptops_with_user“** used for function calling.



```
tools = [
    {
        "type": "function",
        "function": {
            "name": "compare_laptops_with_user",
            "description": "Get the top 3 laptops for the user from the catalogue available based on parameters like GPU intensity, display quality, portability, multitasking, processing s
            "parameters": {
                "type": "object",
                "properties": {
                    "GPU intensity": {
                        "type": "string",
                        "description": "The GPU intensity requirement of the user specified as low, medium or high"
                    },
                    "Display quality": {
                        "type": "string",
                        "description": "The Display Quality requirement of the user specified as low, medium or high"
                    },
                    "Portability": {
                        "type": "string",
                        "description": "The Portability requirement of the user specified as low, medium or high"
                    },
                    "Multitasking": {
                        "type": "string",
                        "description": "The Multitasking requirement of the user specified as low, medium or high"
                    },
                    "Processing speed": {
                        "type": "string",
                        "description": "The Processing speed requirement of the user specified as low, medium or high"
                    },
                    "Budget": {
                        "type": "integer",
                        "description": "The maximum budget of the user"
                    },
                },
                "required": [
                    "GPU intensity",
                    "Display quality",
                    "Portability",
                    "Multitasking",
                    "Processing speed",
                    "Budget"
                ]
            }
        }
    }
]
```

# Refinement:

We have refined the prompts not to handle any other request other than laptop recommendation.

# Challenges:

1. Ensuring increased accuracy in responses from LLM by prompt engineering.

2. Avoiding overwhelming the user with questions by improving the system prompt. It was a challenge to make the AI assistant ask questions about all the parameters at one at a time to ensure user-friendliness.

3. Navigating the complexity of response handling as a result of function calling.

4. Ensuring AI assistant perform only laptop recommendation task and apologize for other questions like asking for trip plan or movies recommendation.

# Lesson Learned:

1. Refining the prompts can always improve the response and it is indeed an iterative task.
2. Understood how to use Function calling feature to handle various solution flow and best thing is that model itself tell us what to do and when to do.