ShopAssist 2.0

Table of Contents

Project Background

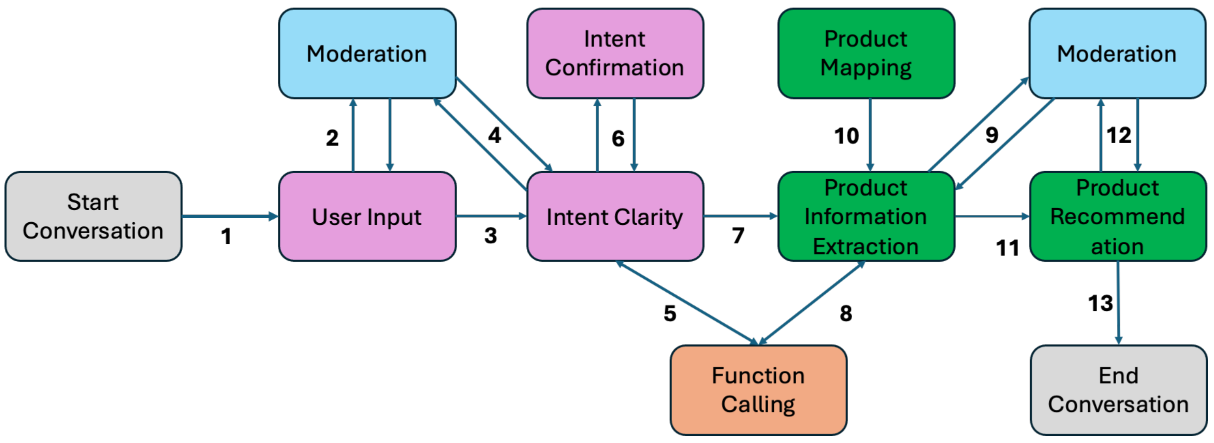
In today's digital age, online shopping has become the go-to option for many consumers. However, the overwhelming number of choices and the lack of personalized advice can make the shopping experience daunting. To address this, we've developed ShopAssist AI—a chatbot that combines advanced language models with rule-based features to provide accurate and reliable information

Problem Statement

Create a chatbot that utilizes a dataset containing laptop details (such as product names, specifications, descriptions, etc.) to provide accurate laptop recommendations based on user preferences.

System Design

Shop Assist Chabot Workflow



Grey blocks indicate user initiation and closure
Pink blocks indicate user information extraction
Green blocks indicate product extraction and recommendation
Blue blocks indicate text moderation
Orange block indicate function calling

The improved ShopAssist 2.0 system design incorporates OpenAI function-calling to accurately classify user inputs and invoke the appropriate external functions for identifying the required laptop information. This enhances the interaction flow. Additionally, this workflow includes moderation at appropriate stages whenever the OpenAI API is utilized.

Here are the steps followed in ShopAssist 2.0:

Step1: User initiates conversation with Shop Assist Bot
Step2: User inputs are moderated for inappropriate content.
Step3: Bot seeks further information on user requirements.
Step4: Bot responses are moderated for inappropriate content.
Step5: Bot checks for appropriate function call
Step6: All user inputs are validated.
Step7: Bot provides information for information extraction.
Step8: Product extraction checks for appropriate function call.
Step9: Product information is validated for inappropriate content.
Step10: Existing products are mapped according to user requirements.
Step11: Shortlisted Products are extracted.
Step12: Short listed products are checked for inappropriate content.
Step13: User ends the conversation after accepting the recommendation.

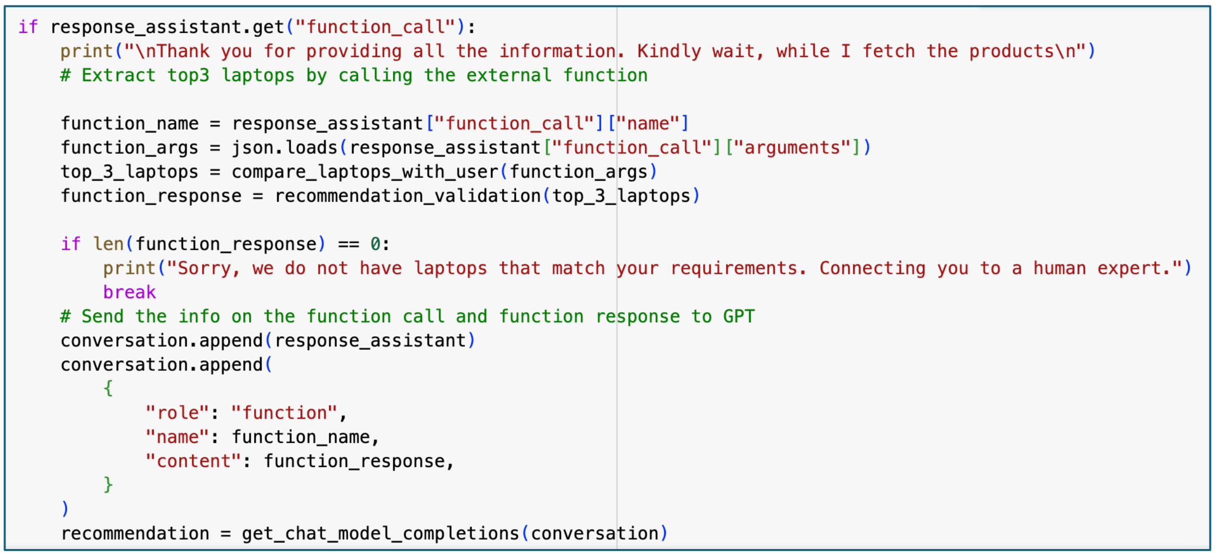
Technical Implementation

Below are list of all functions defined in the chatbot. In order to follow the 13 steps mentioned in system design, below functions have been defined.

- 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
- product_map_layer(): This function is responsible for extracting key features and criteria from laptop descriptions
- recommendation_validation(): This function verifies that the laptop recommendations are good enough, has score greater than 2, and matches the user's requirements
- function_description for “compare_laptops_with_user() is definition of all the parameters needed for fulfilling user requirement. This is in dictionary format.

Function Calling

In this particular ShopAssist 2.0, function calling is integrated. Function calling is used for extracting structured insights from unstructured text. This unstructured text is the output from chat completion API which has relevant laptop data instructions like portability, budget, GPU, graphics, etc from the user and from chat completion API.



```python
if response_assistant.get("function_call"):
    print("\nThank you for providing all the information. Kindly wait, while I fetch the products\n")
    # Extract top3 laptops by calling the external function

    function_name = response_assistant["function_call"]["name"]
    function_args = json.loads(response_assistant["function_call"]["arguments"])
    top_3_laptops = compare_laptops_with_user(function_args)
    function_response = recommendation_validation(top_3_laptops)

    if len(function_response) == 0:
        print("Sorry, we do not have laptops that match your requirements. Connecting you to a human expert.")
        break
    # Send the info on the function call and function response to GPT
    conversation.append(response_assistant)
    conversation.append(
        {
            "role": "function",
            "name": function_name,
            "content": function_response,
        }
    )
    recommendation = get_chat_model_completions(conversation)
```

The above code snippet shows how function calling was integrated into ShopAssist 2.0.

The user is prompted till all the required information is confirmed by the intent_confirmation function. When all information is available, function calling is invoked. Once again the unstructured data is converted to structured data in dictionary format by function call. Also the mandatory fields are ensured to be filled and in the format of “high”, “medium”, “low” with the budget in numeric format.

In the code, we see that the corresponding function call name is stored in variable “function_name”. The output from function call is then provided to function “compare_laptops_with_users” to get the top3 recommended laptops.

Further the code also checks if there is a function response based on user requirements. If there is a case where all information is provided but still no laptops meet the requirement then the bot responds with - No laptops available, message.

The structured information is again fed to the chat completion API to further check for appropriate recommendations.

Below is an example output of chat completion when function call is invoked.

```
{
  "role": "assistant",
  "content": null,
  "function_call": {
    "name": "compare_laptops_with_user",
    "arguments": "{\n\"gpu intensity\": \"high\",\n\"display quality\":
\"high\",\n\"portability\": \"low\",\n\"multitasking\":
\"high\",\n\"processing speed\": \"high\",\n\"budget\": 300000\n}"
  }
}
```

Moderation

```python
if recommendation.content is not None:
    moderation = moderation_check(recommendation.content)
    if moderation == 'Flagged':
        display("Sorry, this message has been flagged. Please restart your conversation.")
        break
else:
    moderation = moderation_check(recommendation.function_call.arguments)
    if moderation == 'Flagged':
        display("Sorry, this message has been flagged. Please restart your conversation.")
        break
```

The above code snippet is exercised for moderation check. Moderation ensures that inappropriate text is not entertained in the ShopAssist 2.0. Moderation check is called every time there is a user input or a response from chat completion API.

There is if, else statement defined for moderation check. The reason is because the content field of assistant response is not null when function call is not invoked. Also, the “arguments” field is populated when function call is exercised.

Below is an example output of chat completion where “content” is null and “arguments” is populated.

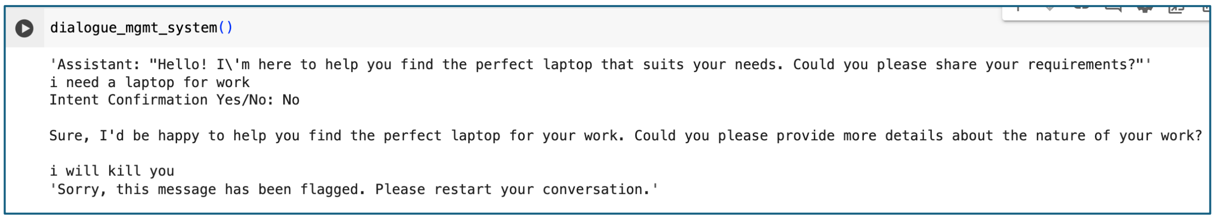
```
{
  "role": "assistant",
  "content": null,
  "function_call": {
    "name": "compare_laptops_with_user",
    "arguments": "{\n\"gpu intensity\": \"high\",\n\"display quality\":
\"high\",\n\"portability\": \"low\",\n\"multitasking\":
\"high\",\n\"processing speed\": \"high\",\n\"budget\": 300000\n}"
  }
}
```

User Experience

User experience has improved because of moderation check and function calling.

Below are some examples.

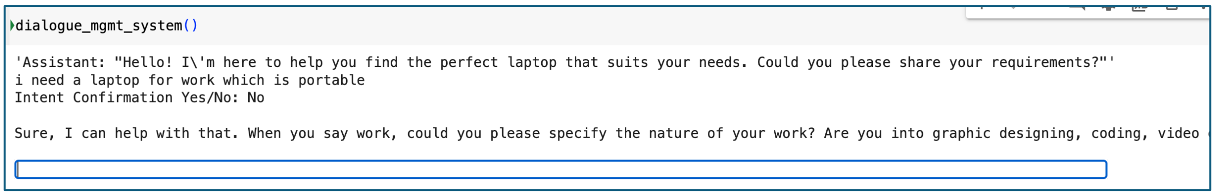
1. Inappropriate text



```
dialogue_mgmt_system()

'Assistant: "Hello! I\'m here to help you find the perfect laptop that suits your needs. Could you please share your requirements?"'
i need a laptop for work
Intent Confirmation Yes/No: No

Sure, I'd be happy to help you find the perfect laptop for your work. Could you please provide more details about the nature of your work?

i will kill you
'Sorry, this message has been flagged. Please restart your conversation.'
```

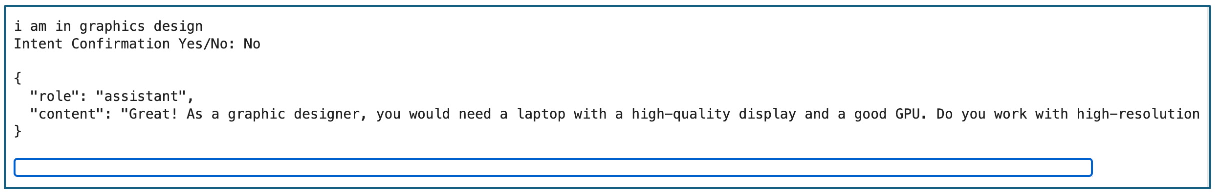
The above example shows how moderation kicks in when user enters inappropriate text.

2. Function calling



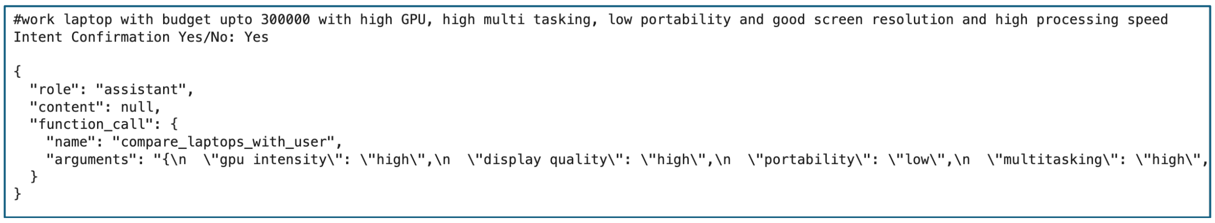
```
dialogue_mgmt_system()

'Assistant: "Hello! I\'m here to help you find the perfect laptop that suits your needs. Could you please share your requirements?"'
i need a laptop for work which is portable
Intent Confirmation Yes/No: No

Sure, I can help with that. When you say work, could you please specify the nature of your work? Are you into graphic designing, coding, video
```

The above example shows how the bot continues the conversation and prompts for required mandatory information.



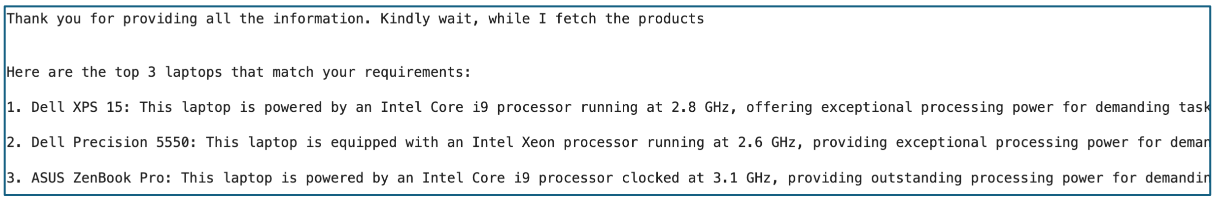
```
i am in graphics design
Intent Confirmation Yes/No: No

{
  "role": "assistant",
  "content": "Great! As a graphic designer, you would need a laptop with a high-quality display and a good GPU. Do you work with high-resolution
}
```

The above example shows how further information is extracted.



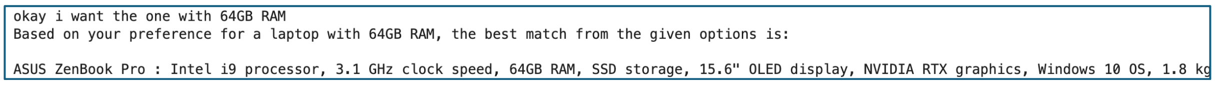
```
#work laptop with budget upto 300000 with high GPU, high multi tasking, low portability and good screen resolution and high processing speed
Intent Confirmation Yes/No: Yes

{
  "role": "assistant",
  "content": null,
  "function_call": {
    "name": "compare_laptops_with_user",
    "arguments": "{\n  \"gpu intensity\": \"high\",\n  \"display quality\": \"high\",\n  \"portability\": \"low\",\n  \"multitasking\": \"high\",
  }
}
```

The above example shows how function calling is invoked and all information is available and given values – high, medium and low. This ensures that the data is then parsed through available laptop dataset to recommend appropriate laptops.



```
Thank you for providing all the information. Kindly wait, while I fetch the products


Here are the top 3 laptops that match your requirements:

1. Dell XPS 15: This laptop is powered by an Intel Core i9 processor running at 2.8 GHz, offering exceptional processing power for demanding task

2. Dell Precision 5550: This laptop is equipped with an Intel Xeon processor running at 2.6 GHz, providing exceptional processing power for demar

3. ASUS ZenBook Pro: This laptop is powered by an Intel Core i9 processor clocked at 3.1 GHz, providing outstanding processing power for demandir
```

The above example shows how the user requirements are met and top 3 laptops are recommended.



```
okay i want the one with 64GB RAM
Based on your preference for a laptop with 64GB RAM, the best match from the given options is:

ASUS ZenBook Pro : Intel i9 processor, 3.1 GHz clock speed, 64GB RAM, SSD storage, 15.6" OLED display, NVIDIA RTX graphics, Windows 10 OS, 1.8 kg
```

The above example shows how the bot further narrows down the recommended laptop based on user request.

Documentation

Appropriate documentation can be found in the code notebook attached in the assignment. Here is snapshot from notebook.

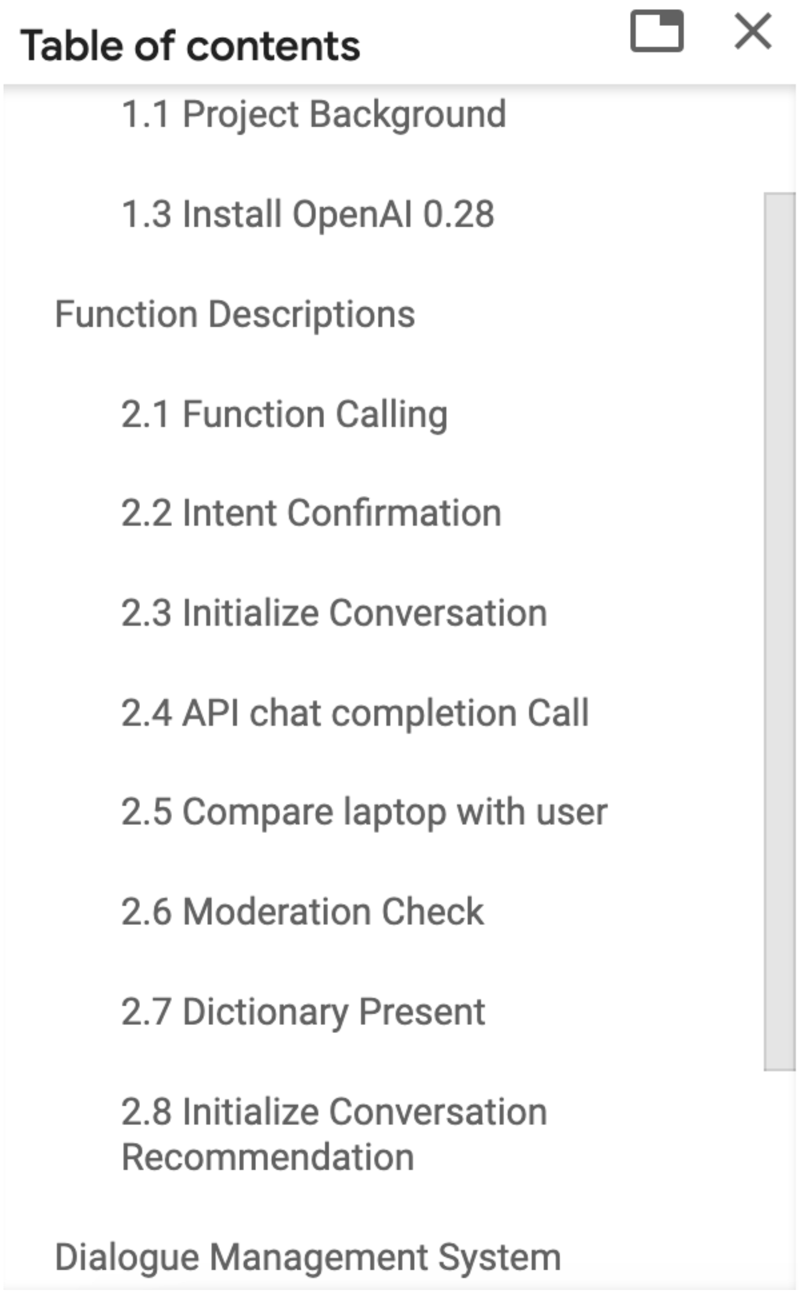


Table of contents

Instructions to Run the code.

1. Install openAI 0.28
2. Change your OS directory. Currently set to os.chdir('/Users/mlsryv/Downloads/GenAI/ShopAssist')
3. Read in you own openapi key.
4. Run the code till the last cell. Function dialogue_mgmt_system() is the main function.
5. Sample string user input.
   # Need a laptop for a Sr Data Scientist