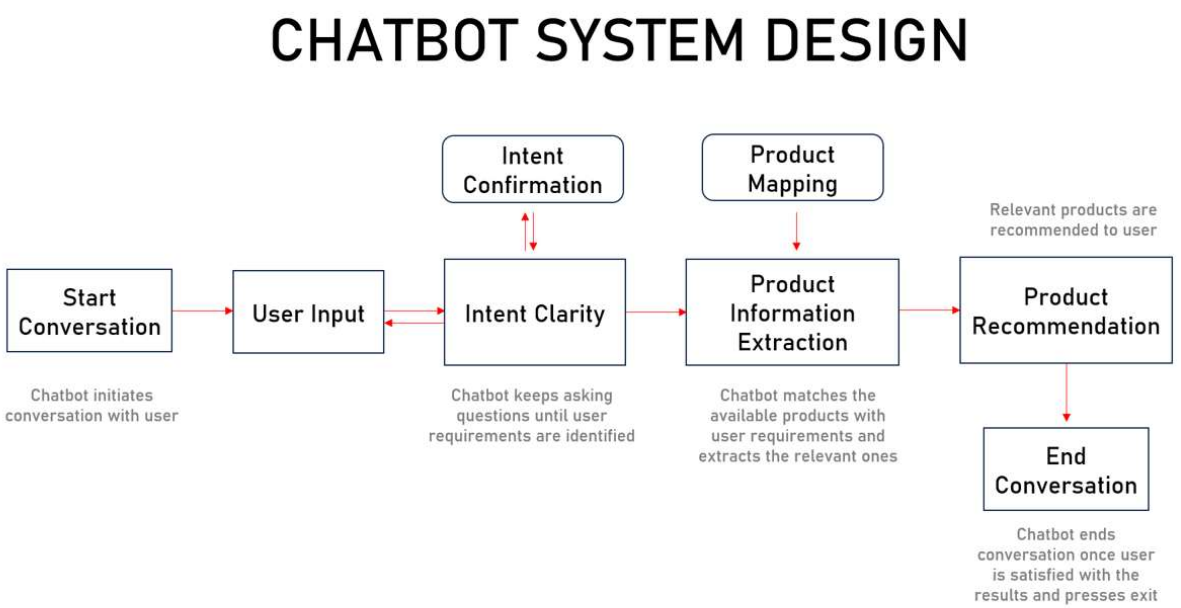
LAPTOP SHOP ASSISTANT AI_2.0

SUBTITLE : - INTEGRATION OF OPEN AI FUCTION OR TOOL CALLING

AUTHOR : -
ARNAB BERA (ML C63)

Table of Contents



Introduction

- **Background** : The evolution of AI-driven systems has transformed user interactions with technology. The Laptop Shop Assistant AI 2.0 builds upon the existing ShopAssistAI by integrating advanced OpenAI function calling capabilities to provide a more dynamic and personalized shopping experience for users.

- **Objective** : To demonstrate how integrating OpenAI's Function Calling API enhances the chatbot's ability to perform tasks beyond simple responses, enabling seamless and real-time personalized laptop recommendations based on user preferences.

- **Key Focus** : This presentation will focus on the process of integrating the OpenAI Function Calling API into ShopAssistAI, allowing for advanced interactions such as executing functions, accessing external data, and delivering tailored results based on complex user inputs.

- **End Goal** : Improve the user experience with more interactive and context-aware features, making the assistant a smarter tool for laptop shopping.

Problem Statement

- **The Challenge:**
  In **ShopAssistAI**, while the chatbot provided laptop recommendations, there were issues with maintaining **consistency** in user interactions and ensuring **modularity** in the underlying system architecture. As new features were added, the system became more difficult to scale and maintain, leading to inconsistent responses and limited adaptability to new user needs.
- **The Result:**
  These challenges led to a suboptimal user experience, with customers encountering occasional discrepancies in recommendations and the inability to seamlessly integrate new features or improve existing ones.
- **Solution (Laptop Shop Assistant AI 2.0):**
  Laptop-Shop-Assist-AI 2.0 addresses these issues by integrating OpenAI's **Function Calling API**, which allows the chatbot to access dynamic functionalities and handle complex tasks in a modular and consistent manner. This upgrade ensures that each function is designed as a modular unit, making the system scalable and easier to maintain, while delivering consistent, accurate, and personalized recommendations.

Approach

- **Modular Architecture** : Rebuilt the backend with a modular design for better scalability and easier maintenance.

- **OpenAI Function Calling Integration** : Integrated OpenAI's Function Calling API to enhance chatbot capabilities and provide personalized laptop recommendations.
- **Consistent and Seamless User Interaction** : Improved conversation flow for consistent, safe, and efficient user interactions.
- **Ensuring Safe Interactions** : Integrated OpenAI's Moderation API to filter and prevent inappropriate content during user interactions.
- **Real-Time Content Filtering** : Automatically scans user inputs to detect harmful or unsafe language, ensuring a secure environment for users.

- **User-Friendly Web Interface** : Developed the frontend using HTML, CSS, and JavaScript for a responsive and intuitive interface.

Changes Made - 1

Function Calling API Integration:

- Custom Functions : Added shopassist_custom_functions to define the structure for extracting user preferences regarding laptop attributes (e.g., GPU intensity, display quality, portability, etc.).
- Updated API Calls : Refined get_chat_completions_tool to leverage the Function Calling API for precise handling of user inputs and preferences extraction.
- Enhanced Moderation : Integrated moderation checks for both user inputs and assistant responses to maintain appropriate and safe conversations.
- Backend-Frontend Communication : Optimized interaction flow between frontend and backend by using structured JSON communication, ensuring smooth data exchange and recommendation display.

Changes Made - 2

- Code Modifications:

shopassist_schema.py

- Added shopassist_custom_functions to define custom functions for extracting user preferences.
- This code defines a custom function for leveraging OpenAI's function calling feature.
- The function, named 'extract_user_info', is designed to extract structured information from user input regarding their laptop preferences to guide personalized recommendations.

```
shopassist_custom_functions = [
    {
        'name': 'extract_user_info',
        'description': 'Get the user laptop information from the body of the input text',
        'parameters': {
            'type': 'object',
            'properties': {
                'GPU_intensity': {
                    'type': 'string',
                    'description': 'GPU intensity of the user requested laptop. The values  are ''low'', ''medium'', or ''high'' based on the importance of the corresponding keys, as stated by user'
                },
                'Display_quality': {
                    'type': 'string',
                    'description': 'Display quality of the user requested laptop. The values  are ''low'', ''medium'', or ''high'' based on the importance of the corresponding keys, as stated by user'
                },
                'Portability': {
                    'type': 'string',
                    'description': 'The portability of the user requested laptop. The values  are ''low'', ''medium'', or ''high'' based on the importance of the corresponding keys, as stated by user'
                },
                'Multitasking': {
                    'type': 'string',
                    'description': 'The multitasking abiliy of the user requested laptop. The values  are ''low'', ''medium'', or ''high'' based on the importance of the corresponding keys, as stated by user'
                },
                'Processing_speed': {
                    'type': 'string',
                    'description': 'The processing speed of the user requested laptop.  The values  are ''low'', ''medium'', or ''high'' based on the importance of the corresponding keys, as stated by user'
                },
                'Budget': {
                    'type': 'integer',
                    'description': 'The budget of the user requested laptop. The values are integers.'
                }
            }
        }
```

Changes Made - 3

- Code Modifications:

openai_api.py

- Integrated function calling logic to parse user input and extract preferences for more accurate recommendations.
- Retried API calls for better stability during execution.
- get_chat_completions_tool introduced to utilize the Function Calling API for better handling of user inputs.

```
try:
    chat_completion_json = openai.chat.completions.create(
        model=model,
        messages=final_message,
        n=1,
        seed=1234,
        max_completion_tokens=500,
        temperature=0.3,
        tools=tools,
        tool_choice="auto"
    )
    response = chat_completion_json.choices[0].message
    tool_call = response.tool_calls[0]
    tool_name = tool_call.function.name
    arguments_str = tool_call.function.arguments
    # Parse the arguments string into a dictionary
    arguments = json.loads(arguments_str)
    if tool_name == tool_name:
        # Pass the extracted arguments to the function dynamically
        output = extract_user_info(
            arguments.get('GPU_intensity', ''),
            arguments.get('Display_quality', ''),
            arguments.get('Portability', ''),
            arguments.get('Multitasking', ''),
            arguments.get('Processing_speed', ''),
            arguments.get('Budget', 0)
        )
        return output
    else:
        return {"error": f"Unexpected tool call: {tool_name}"}
```

Changes Made - 4

- Code Modifications:

dialogue.py:

- Enhanced the conversation flow to incorporate function calls, ensuring real-time user preferences extraction and feedback.

app.py:

- Introduced new API endpoints and integrated the function calling feature into the existing Flask application.
- Handled session management for persistent user interactions, including recommendations.

```
44
45
46 def confirm_intent(response_tool):
47     """
48     Confirms the user's intent based on the response tool's output.
49     """
50     confirmation = intent_confirmation_layer(response_tool)
51     return confirmation.get('result')
52
53
54 def fetch_recommendations(conversation_reco, response_tool):
55     """
56     Fetches laptop recommendations and updates the conversation.
57     """
58     logger.info("dialogue_info 1 - Fetching laptop recommendations. Response: %s and Type: %s", conversation_reco, type(conversation_reco))
59     response = get_chat_completions_tool(response_tool, func_name=shopassist_custom_functions)
60     logger.info("dialogue_info 2 - Fetching laptop recommendations. Response: %s and Type: %s", response, type(response))
61     top_3_laptops = compare_laptops_with_user(response)
62     validated_reco = recommendation_validation(top_3_laptops)
63     conversation_reco = initialize_conv_reco(validated_reco)
64     return top_3_laptops, conversation_reco
65
```

API Endpoints

Update Flask Application: Added new endpoints for the API integration.

API Endpoints:

1. GET /api/start_conversation: Initializes a new conversation.
2. POST /api/process_input: Processes user input and returns the assistant's response.
3. POST /api/end_conversation: Ends the current conversation.

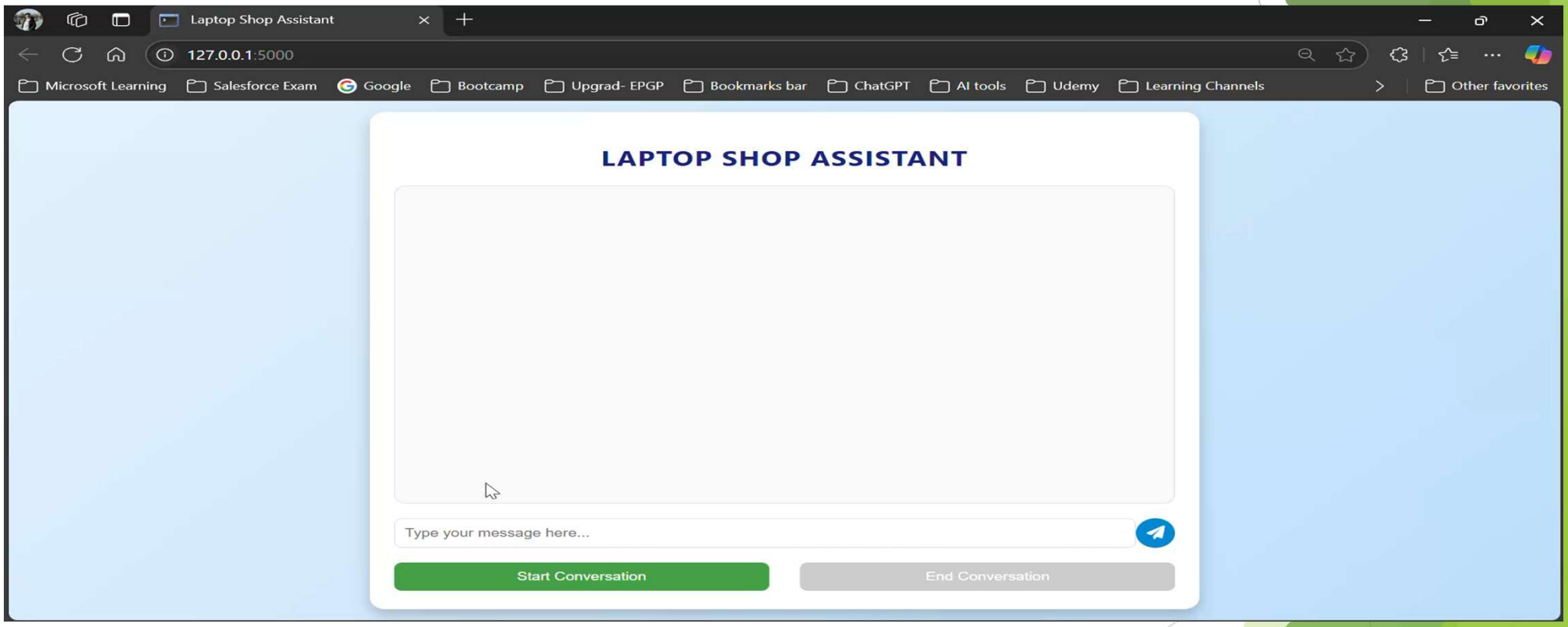
```python
@app.route('/api/start_conversation', methods=['GET'])
def start_conversation():
    logger.info("Starting a new conversation.")
    try:
        # Initialize conversation and introduction
        conversation, introduction = initialize_conversation_system()
        session['conversation'] = conversation
        logger.debug("Conversation initialized: %s", conversation)
        return jsonify({"introduction": introduction}), 200
    except Exception as e:
        logger.error("Error starting conversation: %s", str(e))
        return jsonify({"error": "Failed to start conversation."}), 500

@app.route('/api/process_input', methods=['POST'])
def process_input():
    user_input = request.json.get('message')
    logger.info("app_info 1 - Processing user input: %s", user_input)
```

```python
@app.route('/api/end_conversation', methods=['POST'])
def end_conversation():
    logger.info("Ending the conversation.")
    try:
        # Check if session exists before clearing
        if 'conversation' in session or 'top_3_laptops' in session:
            session.clear()
            logger.debug("Session data cleared successfully.")
        else:
            logger.debug("Session already empty.")

        return jsonify({"status": "conversation ended", "content": "Th
```

Demonstration



Conclusions

- Recap : The integration of the Function Calling API has significantly enhanced ShopAssistAI's ability to provide precise and personalized laptop recommendations.

- Key Benefits :
  1. Improved Accuracy : Precisely extracts user preferences for better recommendations.
  2. Enhanced Personalization : Tailors responses based on specific user inputs (e.g., GPU, budget).
  3. Streamlined User Experience : Eliminates manual parsing, allowing for quicker, more relevant interactions.
  4. Scalability : Handles a wide range of diverse user inputs and requirements efficiently.
  5. Increased Efficiency : Optimizes the conversation flow by automating preference extraction and decision-making.

- Future Work : Explore additional AI integrations for further personalization and recommendation optimization.

Glossary

- AI (Artificial Intelligence): Machines programmed to simulate human intelligence, think, and learn.
- API (Application Programming Interface): A set of rules for software communication.
- Chatbot: Software for conducting online text or speech-based conversations.
- OpenAI: A research organization and company developing AI technologies, including the GPT models.
- Pandas: A Python library for data manipulation and analysis.
- Flask: A Python-based micro web framework.
- GPT (Generative Pre-trained Transformer): A deep learning model by OpenAI that generates human-like text.
- HTML (HyperText Markup Language): Standard language for web page structure.
- CSS (Cascading Style Sheets): Language for styling HTML and XML documents.
- JavaScript: Programming language for creating interactive web elements

Acknowledgements

- The project references course materials from upGrad's curriculum.
- The project references presentations in upGrad’s recorded module given by Kshitij Jain.
- The project references presentations in upGrad’s recorded module given by Kaustubh Singh.
- The project references presentations in upGrad's live class given by Sheshanth AS.
- The project references insights and inferences from presentations in upGrad’s doubt clear session given by Shridhar Galande.

THANK YOU