ShopAssist AI 2.0

Function Calling Enhancement Report

Prepared by: Huy Vo Duc

upGrad and IIITB Machine Learning & AI Program – November 2024

1. Executive Summary

ShopAssist AI 2.0 is an enhanced version of the original ShopAssist chatbot, redesigned to leverage the **OpenAI Function Calling API**. This upgrade transforms the chatbot from a static, rule-based system into **an intelligent conversational agent** capable of understanding natural language and invoking backend logic to fetch real product data. The project focuses on **integrating Function Calling, refining the conversation flow, and improving user experience through contextual interactions.**

2. Objectives

The main objectives of this enhancement were to:

- Integrate the Function Calling API to enable the chatbot to trigger backend logic dynamically.
- Redesign the conversation flow for improved natural language understanding.
- Optimize performance, modularity, and maintainability of the system architecture.

3. System Design

The redesigned architecture introduces modular service layers (OpenAI Service, Conversation Service, Product Service) and a Flask-based backend for efficient handling of chat requests. The Function Calling API acts as a bridge between the AI model and backend functions, allowing seamless execution of data-driven operations such as filtering products.

Key architectural changes include:

- Removal of notebook-style procedural flow in favor of Flask MVC structure.
- Integration of Function Calling schema to define actions (e.g., recommend_product).
- Session-based context management to track conversation state.
- Dynamic configuration using environment variables (.env + config.py).

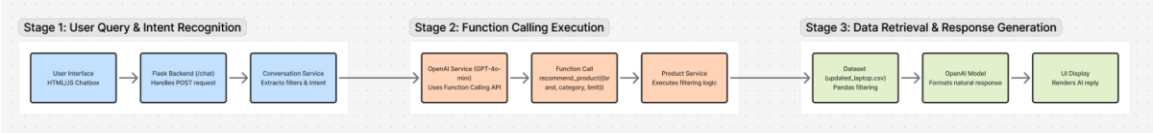

Figure 1: ShopAssist AI 2.0 System Architecture

4. Solution Overview — Three-Stage Function Calling Workflow

ShopAssist AI 2.0 operates through a streamlined three-stage architecture, combining natural-language understanding, intelligent function execution, and data-driven response generation.

This design makes the chatbot both conversational and action-oriented.

Stage 1 – User Query & Intent Recognition

The interaction begins when the user enters a query through the chat interface (HTML/JS).

The Flask backend (/chat) receives the message and passes it to the Conversation Service, which:

- Analyzes the text and extracts filters (e.g., brand, category, RAM, price range).
- Prepares a structured context for the AI model.

Outcome: The system understands what the user wants — not just keywords but intent (e.g., "Find top 3 Dell gaming laptops").

Stage 2 – Function Calling Execution

The OpenAI Service (GPT-4o-mini) interprets the intent and decides whether to invoke a defined backend function.

Using the Function Calling API, it generates a structured call such as:

```
{
  "name": "recommend_product",
  "arguments": {
    "brand": "Dell",
    "category": "gaming",
    "limit": 3
  }
}
```

Flask then executes this function in the **Product Service**, which performs real data filtering on the dataset (updated_laptop.csv).

Outcome: The chatbot shifts from text prediction to real backend execution — fetching factual, context-aware results.

Stage 3 – Data Retrieval & Response Generation

The filtered data is returned to the AI model.
GPT formats this structured output into a natural, human-like message:

> *"**ShopAssist AI:** Here are top 7 laptops matching your criteria:*
>
> *1. Msi **Gl65** — I7 | 16GB | Nvidia Gtx 💲 $55,000*
>
> *2. Acer **Predator** — I7 | 16GB | Nvidia Gtx 💲 $80,000*
>
> *3. Asus **Rog Strix G** — I7 | 16GB | Nvidia Rtx ..."*

The message is rendered in the browser UI with typing effects and smooth transitions.

Outcome: The assistant delivers a fluent, data-backed response that feels conversational yet precise.

End-to-End Flow Summary

1. **User query → Flask backend → OpenAI model**
2. **Model calls backend function → Dataset query → Results returned**
3. **Model formats response → UI displays reply**

This three-stage workflow ensures ShopAssist AI 2.0 maintains **clarity, modularity, and real-time intelligence** — bridging human conversation with executable system logic.

4. Technical Implementation

The technical upgrade centers around the integration of the OpenAI Function Calling API. The chatbot identifies user intent and invokes a defined function schema—such as 'recommend_product'—to perform structured backend operations.

Implementation details:

- **Function Calling Integration:** GPT-4o-mini dynamically triggers defined backend functions such as recommend_product() using structured JSON arguments, improving accuracy and reducing prompt complexity.
- **Modular Design:** Clear separation between openai_service, conversation_service, and product_service ensures clean logic, easier debugging, and scalability for future functions.

- **Flask Workflow:** /chat receives user queries → OpenAI determines intent → backend executes the mapped Python function → structured data returned → GPT formats a natural response.
- **Dataset Integration:** Product data (from updated_laptop.csv) is filtered via Pandas and served as real recommendations rather than hard-coded examples.
- **Session Management:** Context is reset automatically on invalid filters to maintain coherent, natural conversation flow.
- **Frontend UX:** Lightweight HTML/JS interface with typing indicator, enter-to-send, and realistic AI response timing.
- **Configuration & Security:** .env-based configuration for API keys and model names; structured logging across all layers.

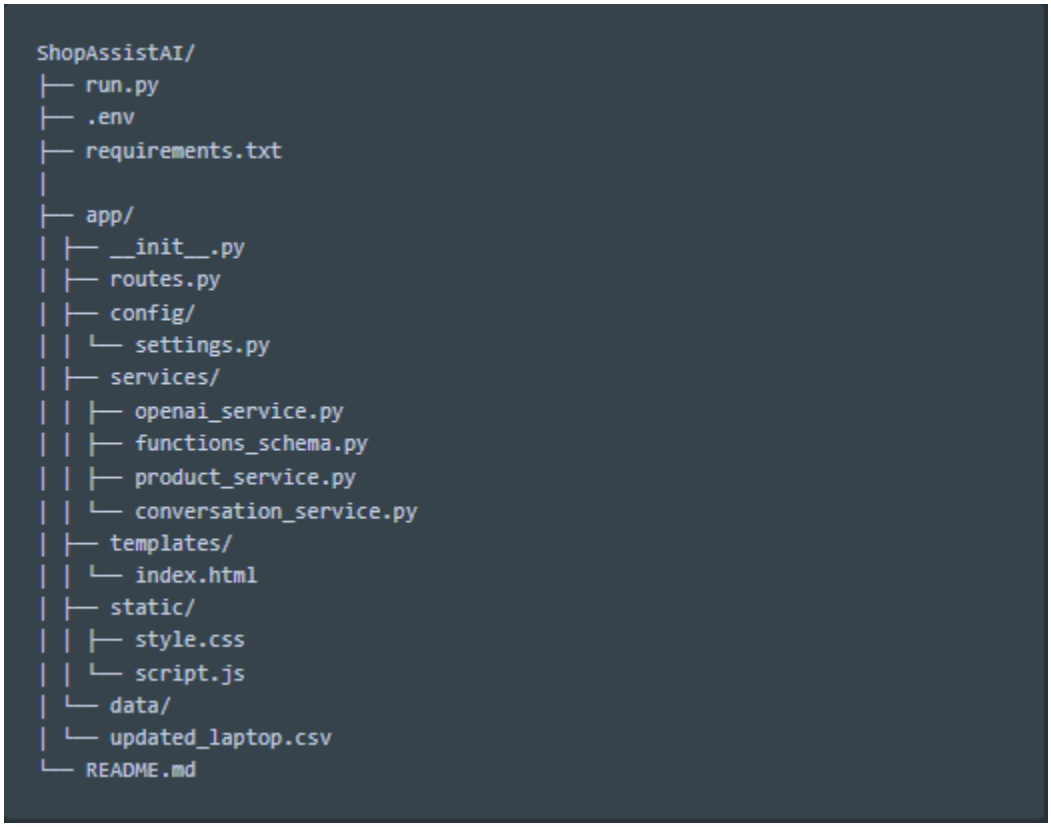

Figure 2: Product Structure

5. User Experience

User experience was significantly improved with responsive and intuitive chat design. Enhancements include a typing indicator, realistic response delay, send and clear buttons, and automatic welcome messages. The chatbot resets its context when no products match, ensuring the next interaction starts fresh.

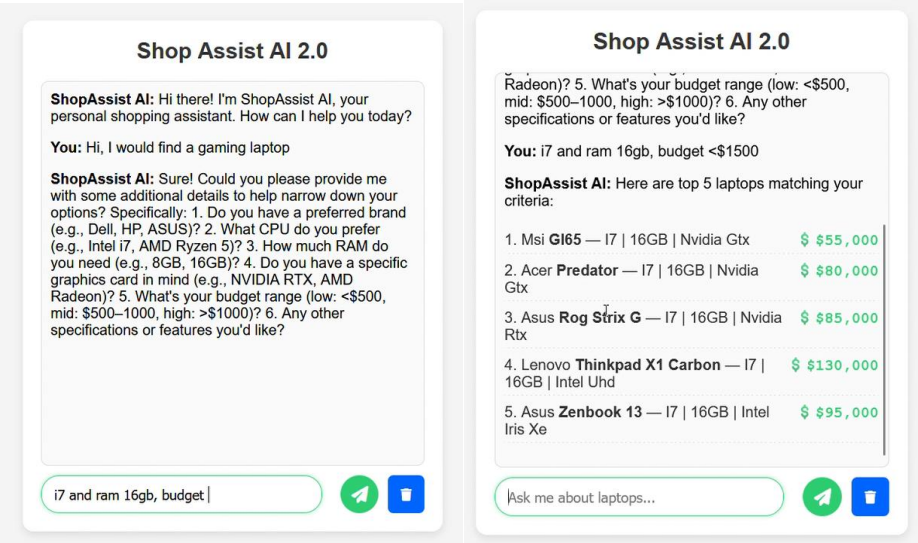

Figure 3: Chatbot Interface

6. Key Features

Category	Feature	Benefit
AI Intelligence	Function Calling API	Enables structured, reliable backend execution.
Conversational Flow	Context reset & natural replies	Keeps dialogue consistent and human-like.
Data-Driven Logic	Real product filtering via CSV	Produces factual, dynamic answers.
User Interface	Minimalist web chat	Enhances user engagement and clarity.
Configurable Setup	Environment-based settings	Ensures security and portability.

7. Challenges

Getting Started with Flask:
This was the first time building a full web-based AI chatbot using Flask.
Setting up routes, templates, and services required understanding request flow and modular organization.

OpenAI API Limitations
Encountered API quota and version compatibility issues during Function Calling integration.
Resolved by updating client versions and managing usage efficiently.

Context Management
Managing conversation flow was challenging.
The system now resets context automatically when no matching products are found, preventing stale filters and improving response accuracy.

8. Lessons Learned

Flask Simplicity, Big Power: Even a lightweight framework like Flask can support modular, production-ready AI systems when structured correctly.
Structured AI Is Reliable: Function Calling provides more control and stability than prompt-only responses.
Context Control Matters: Managing and resetting chat context is key to maintaining logical, consistent conversations.
Practical Limits Teach Design Discipline: Working within OpenAI API usage limits encouraged cleaner, more efficient architecture.

9. Conclusion

ShopAssist AI 2.0 successfully demonstrates how modern AI capabilities - particularly the **OpenAI Function Calling API** - can be applied to build a truly intelligent and data-driven conversational assistant.
Through this enhancement, the project evolved a **modular, scalable web application** capable of understanding natural language, executing real backend logic, and delivering human-like, contextually accurate responses.

The integration of **Function Calling**, **modular service layers**, and a **real product dataset** transformed the system's design and functionality, creating a bridge between conversational reasoning and structured data operations.
The result is an assistant that not only converses but also **acts intelligently** - executing precise tasks such as filtering, recommending, and explaining laptop options.

Beyond its technical implementation, this project reinforced key engineering principles:

- The power of **clean architecture** and separation of concerns.
- The importance of **context management** in conversational AI.
- The effectiveness of grounding AI responses in **real-world data**.

With its strong foundation, ShopAssist AI 2.0 stands as a flexible platform for further innovation—ready to integrate comparison functions, semantic search, and persistent conversational memory in future versions.

In essence, this project showcases how Function Calling elevates a chatbot from text generation to **true AI-driven interaction**, marking an important step toward building intelligent assistants that combine natural communication with actionable system capabilities.