## **Project Documentation: ShopAssist 2.0 - Upgrading a Chatbot with Function Calling**

### **1. Introduction**

This document outlines the successful enhancement of **ShopAssist AI**, a conversational agent designed to recommend laptops, into its next-generation version, **ShopAssist 2.0**.

The primary goal of this project was to refactor the original architecture by integrating the **Function Calling** capabilities of modern Large Language Models (LLMs). This upgrade moves the chatbot from a complex, prompt-driven system to a more robust, reliable, and streamlined **tool-augmented agent**.

The original chatbot relied on intricate prompting and string parsing to connect the AI's conversational output to the application's code. ShopAssist 2.0 bridges this gap directly, resulting in a simpler codebase and a more efficient user experience.

### **2. The Original Architecture: Limitations**

The first version of ShopAssist AI operated on a multi-stage, prompt-engineering-heavy model:

1. **Intent Clarity:** The AI was given a complex system prompt, instructing it to have a conversation with the sole purpose of generating a Python dictionary formatted as a string.
2. **String Parsing:** The Python application would then receive this string and use helper functions (intent\_confirmation\_layer, dictionary\_present) to validate and parse it, hoping the AI's output matched the required format.
3. **Code Execution:** Only after a valid dictionary was successfully extracted would the application's core logic (compare\_laptops\_with\_user) be executed.

This approach had several key limitations:

* **Brittleness:** The entire system depended on the AI perfectly formatting a string. Any slight deviation in the AI's output could cause the parsing to fail.
* **Complexity:** It required verbose "Chain-of-Thought" prompts and extra Python code dedicated solely to validating and cleaning the AI's output.
* **Disconnection:** The AI had no real "awareness" of the application's capabilities. It was simply generating text based on a pattern, not acting with a goal.

### **3. Architectural Changes: Before and After**

The integration of function calling fundamentally changed the workflow and the division of responsibility between the AI and the Python application.

#### **Before: The Disconnected Approach**

The application would instruct the AI to generate a string, then the application's code would parse that string and decide to run a local function. The AI was unaware of the final action.

#### **After: The Integrated Tool-Using Approach**

The application now tells the AI about the tools it has. The AI's job is to have a conversation and decide when to ask the application to run a tool. The communication is direct and structured.

This table summarizes the core improvements:

| **Aspect** | **ShopAssist 1.0 (Before)** | **ShopAssist 2.0 (After)** |
| --- | --- | --- |
| AI's Role | A text generator following a strict pattern. | An intelligent agent that decides when to use tools. |
| Communication | Unstructured text output that needs parsing. | Structured, reliable tool call requests. |
| Reliability | Brittle. Fails if the AI's text format changes. | Robust. The data exchange is guaranteed to be correct. |
| Code Complexity | Higher. Required extra functions (intent\_confirmation\_layer, dictionary\_present) just for parsing and validation. | Lower. We removed all the parsing logic and simplified the main prompt. |

### **4. Core Components of the New Architecture 🛠️**

ShopAssist 2.0 is built on three key components that work together:

#### **4.1 The Python Tool: find\_laptops()**

This is the application's core logic, refactored into a clean Python function. It acts as the "search engine" for our data. It accepts user preferences (like budget, portability, etc.) as direct arguments, filters the updated\_laptop.csv file, and uses a scoring system to find and return the top 3 best-matching laptops.

#### **4.2 The Tool Declaration**

This is a structured schema that serves as a "menu" or "user manual" for the AI. It describes the find\_laptops tool, explaining its purpose, the parameters it requires (e.g., budget is a required integer), and the valid options for each parameter (e.g., portability must be 'low', 'medium', or 'high'). This declaration is what makes the AI "aware" of the tool.

#### **4.3 The Interactive Engine: run\_chatbot()**

This is the main application loop that manages the conversation. Its logic is designed to:

1. Wait for user input.
2. Send the message to the AI.
3. Intelligently check the AI's response.
4. If the response is a **tool call request**, it executes the find\_laptops function and sends the results back.
5. If the response is a regular **text message**, it simply prints the reply.

### **5. Summary of Benefits :**

The upgrade to the Function Calling architecture provides significant advantages:

* **Simplicity & Readability:** The code is cleaner and easier to maintain because we removed all the complex parsing and validation logic.
* **Reliability:** Structured communication via tool calls eliminates errors that would occur if the AI formatted its text output incorrectly.
* **Extensibility:** The system is now much easier to expand. To add a new capability (e.g., "check inventory"), one would simply need to write a new Python function and its corresponding tool declaration.
* **Better AI Logic:** The AI can now focus on understanding the user's intent and the flow of the conversation, rather than on the difficult task of perfectly formatting a specific string output.

### **6. How to Run the Application**

1. **Prerequisites:** Ensure you have Python installed, along with the laptop\_data.csv and updated\_laptop.csv files in your project directory.
2. **Installation:** Install the necessary libraries by running: pip install google-generativeai pandas
3. **API Key:** Obtain a free API key from Google AI Studio and configure it in the script.
4. **Execution:** Place all the code (imports, tool definition, tool declaration, and the run\_chatbot function) into a single Python script or notebook. Execute the run\_chatbot() function to start the interactive session.