ShopAssist 2.0 Report

1. Introduction

The ShopAssist 2.0 is an intelligent assistant that helps users find the most suitable laptop based on their preferences. It integrates with Google's Gemini API (OpenAI-compatible) and uses advanced function calling to perform structured extraction, validation, comparison, and summarization of laptop recommendations.

2. System Overview

The system operates through a structured conversation between the user and the Gemini model. It dynamically extracts user preferences, validates the completeness of the information, compares available laptops from a dataset, and generates clear and user-friendly recommendations.

3. User Flow

The step-by-step flow of user interaction is as follows:

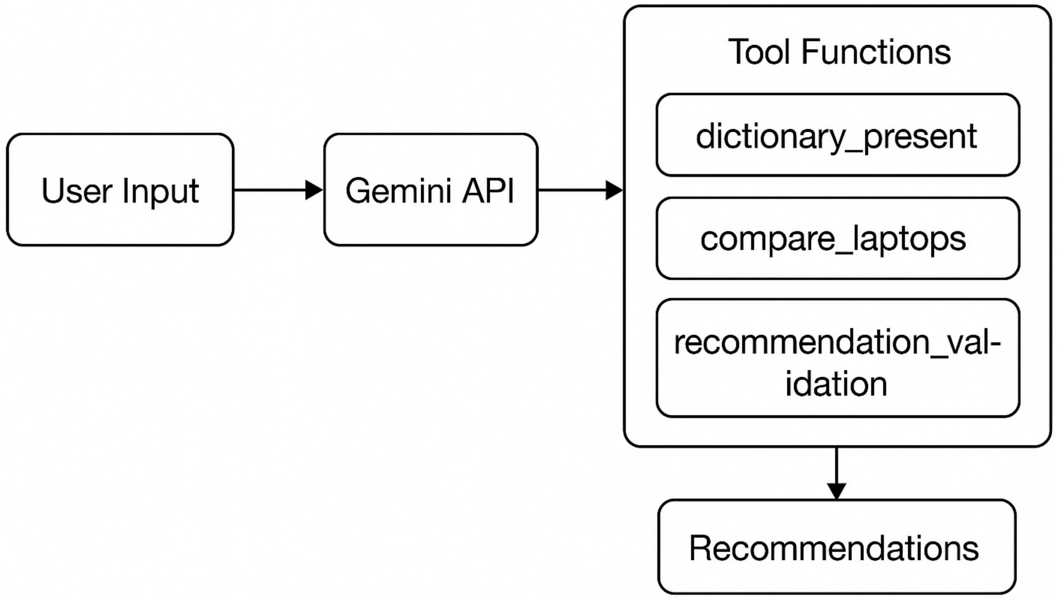
1. Step 1: User initiates the chat with a greeting .
2. Step 2: Gemini prompts the user to describe their laptop requirements (use case, budget, etc.).
3. Step 3: The model calls the `dictionary_present` tool to extract a structured preference dictionary.
4. Step 4: The extracted preferences are validated using the `validate_requirements` tool.
5. Step 5: If the budget is in a foreign currency, the `convert_currency` tool converts it to INR.
6. Step 6: The model then calls `compare_laptops_with_user` to find matching laptops from `updated_laptop.csv`.
7. Step 7: The resulting laptop recommendations are filtered using the `recommendation_validation` tool.
8. Step 8: Finally, the top laptops are summarized using the `summarize_recommendations` tool and shown to the user.

4. Function and Tool Flow

The backend logic is organized around a set of modular functions and Gemini tool calls that ensure structured processing.

- initialize_conversation: Sets up the initial system prompt for Gemini, defining the assistant's behavior and tone.
- get_chat_completions: Main communication handler. Sends messages to Gemini and manages function calling logic.
- dictionary_present: Extracts structured key-value preferences from user text (GPU intensity, Budget, etc.).

- validate_requirements: Ensures all required fields are present and values conform to constraints (e.g., Budget ≥ 25,000).
- convert_currency: Converts user-provided budget from foreign currency to INR.
- compare_laptops_with_user: Matches user preferences against laptops in the CSV file and computes scores.
- recommendation_validation: Filters the comparison output to include only laptops with sufficient score (>2).
- summarize_recommendations: Generates a clear, human-readable summary of top laptop matches.



5. Function(Tool Call) Integration in Gemini

Each tool is defined with a JSON schema and integrated into the `get_chat_completions` function under the `tools` list. When Gemini identifies a need for structured data, it automatically triggers the appropriate function call. The returned result is appended to the conversation, and the assistant continues the dialogue with enriched context.

6. Data Handling and Scoring Logic

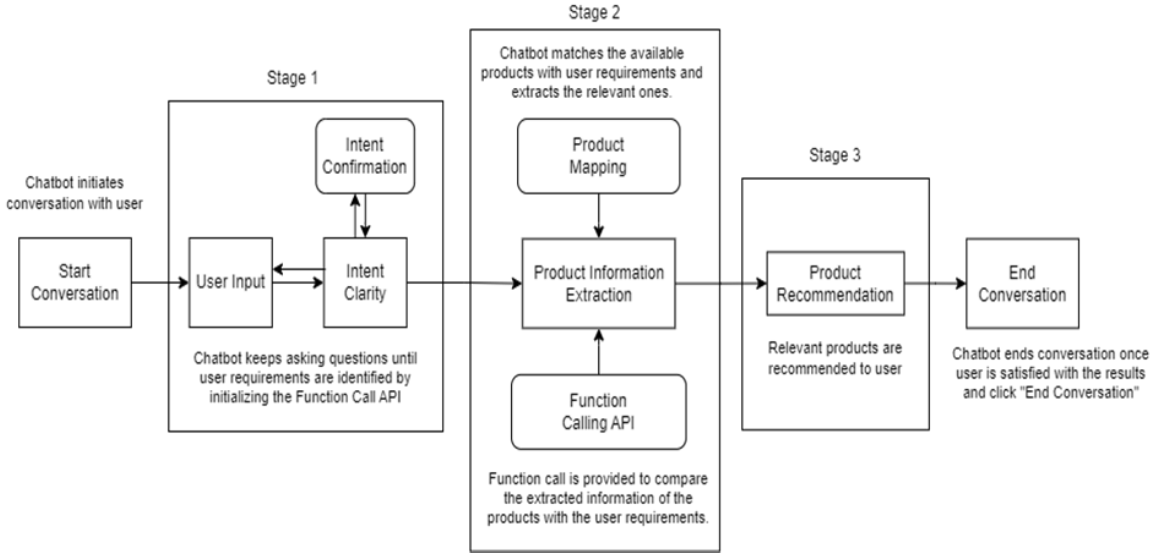
Laptop data is stored in a CSV file (`updated_laptop.csv`). During comparison, each laptop entry is scored based on how well it meets the user's specified preferences. Scoring uses ordinal mapping ('low' = 0, 'medium' = 1, 'high' = 2). Only laptops within the user's budget are considered.

7. Example Tool Call Flow

An example sequence of function calls:

- User: 'I want a high-performance laptop for video editing, around $1500.'
- → `dictionary_present` extracts structured preferences.
- → `convert_currency` converts $1500 to INR.
- → `validate_requirements` checks completeness of keys.
- → `compare_laptops_with_user` finds top matches within budget.
- → `recommendation_validation` filters low-score options.
- → `summarize_recommendations` formats and presents results.

8. System Architecture


Chatbot initiates conversation with user
Start Conversation
Stage 1
User Input
Intent Confirmation
Intent Clarity
Chatbot keeps asking questions until user requirements are identified by initializing the Function Call API
Stage 2
Chatbot matches the available products with user requirements and extracts the relevant ones.
Product Mapping
Product Information Extraction
Function Calling API
Function call is provided to compare the extracted information of the products with the user requirements.
Stage 3
Product Recommendation
Relevant products are recommended to user
End Conversation
Chatbot ends conversation once user is satisfied with the results and click "End Conversation"

Summary of Changes Made

a. Core Enhancements

- **Function Calling Integration:** Implemented structured API calls via Gemini-compatible tools.
- **Additional Tools Introduced:**
    1. `validate_requirements` – Validates completeness of user preferences.
    2. `recommendation_validation` – Filters laptops based on score thresholds.

3. `summarize_recommendations` – Converts JSON results into a user-friendly summary.
4. `convert_currency` – Converts non-INR budgets into INR using fixed exchange rates.

b. Core Logic Update

- The `get_chat_completions()` function was expanded to:
    - Define all tool schemas.
    - Dynamically detect and execute function calls returned by Gemini.
    - Append function results to the conversation for continued reasoning.

c. Data Validation and Scoring

- Integrated budget validation and numeric conversion.
- Added `recommendation_validation()` to ensure only strong matches are displayed.
- Maintained mapping for preference scoring (low = 0, medium = 1, high = 2).

d. Conversation and Output Management

- System message improved for better onboarding.
- Added structured follow-up handling to support tool chaining.

---

3. Integration Process

Step 1: Define Tool Schemas

Each function (tool) was defined with a JSON schema specifying:

- `name`
- `description`
- `parameters` (with `type`, `properties`, and `required` fields)

Step 2: Register Tools in `get_chat_completions()`

All tools were registered inside the `tools` list, and the API call to Gemini includes them dynamically:

```
response = client.chat.completions.create(

model=model,
messages=messages,

tools=tools if not json_format else None,

response_format={"type": "json_object"} if json_format else None

)
```

Step 3: Detect and Execute Function Calls

When Gemini identifies the need for structured execution, it triggers a tool call. The handler then:

1. Parses `fn_name` and `fn_args`.
2. Matches the function from a local mapping (`available_functions`).
3. Executes it and appends the result to the message chain.

Step 4: Continue the Conversation

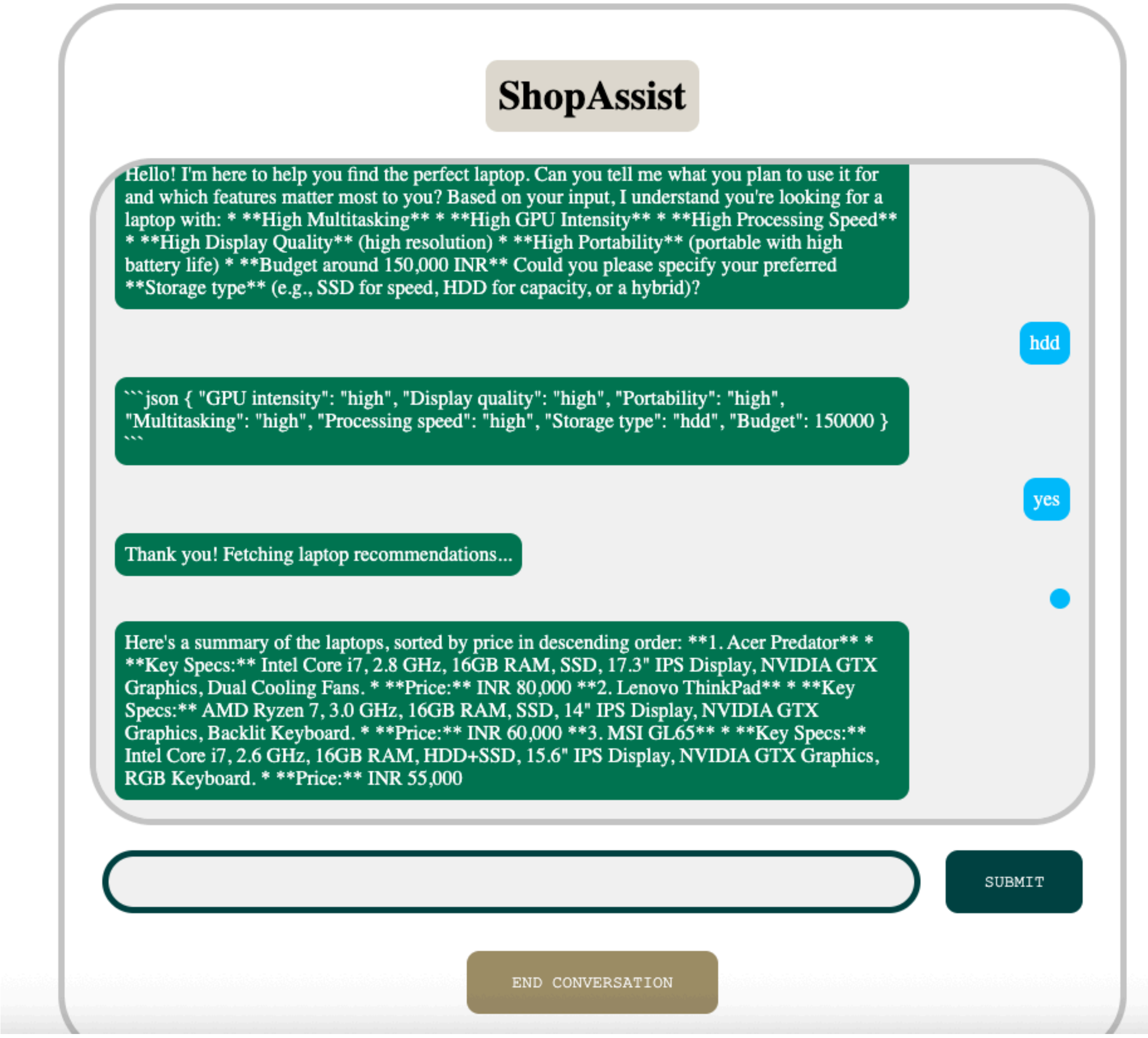
After executing a tool, Gemini receives the result and continues reasoning—allowing sequential multi-tool calls in a single flow.

---

4. Function Call Chain Example

User → describe needs → dictionary_present → validate_requirements → convert_currency → compare_laptops_with_user → recommendation_validation → summarize_recommendations → Gemini → Output summary

This pipeline ensures that every user input is structured, validated, processed, and summarized before returning the final output.



Hello! I'm here to help you find the perfect laptop. Can you tell me what you plan to use it for and which features matter most to you? Based on your input, I understand you're looking for a laptop with: * **High Multitasking** * **High GPU Intensity** * **High Processing Speed** * **High Display Quality** (high resolution) * **High Portability** (portable with high battery life) * **Budget around 150,000 INR** Could you please specify your preferred **Storage type** (e.g., SSD for speed, HDD for capacity, or a hybrid)?

hdd

~~~
```json { "GPU intensity": "high", "Display quality": "high", "Portability": "high", "Multitasking": "high", "Processing speed": "high", "Storage type": "hdd", "Budget": 150000 } ```
~~~

yes

Thank you! Fetching laptop recommendations...

Here's a summary of the laptops, sorted by price in descending order: **1. Acer Predator** * **Key Specs:** Intel Core i7, 2.8 GHz, 16GB RAM, SSD, 17.3" IPS Display, NVIDIA GTX Graphics, Dual Cooling Fans. * **Price:** INR 80,000 **2. Lenovo ThinkPad** * **Key Specs:** AMD Ryzen 7, 3.0 GHz, 16GB RAM, SSD, 14" IPS Display, NVIDIA GTX Graphics, Backlit Keyboard. * **Price:** INR 60,000 **3. MSI GL65** * **Key Specs:** Intel Core i7, 2.6 GHz, 16GB RAM, HDD+SSD, 15.6" IPS Display, NVIDIA GTX Graphics, RGB Keyboard. * **Price:** INR 55,000