# **Overview**

## **Objective: Migrate and refactor a laptop recommendation chatbot to use a more robust, maintainable design that leverages advanced prompt engineering and, if desired, the new Function Calling API for LLMs.**

# **File level Comparison**

## [**app.py**](http://app.py) **Changes**

### **Old Version:**

#### **Manages conversation\_bot, conversation, top\_3\_laptops mostly as mutable global variables and shares state across all endpoints.**

#### **Several layers for conversation reset, confirmation checking, and special routing logic for recommendations or clarification questions.**

#### **Handles confirmation steps and “Did I get all your requirements?” with complicated checks and context management.**

### **Enhanced Version:**

#### **Separation of dialog flow, relying more on clear intent abstraction (“intent\_confirmation\_layer” and “dictionary\_present” functions).**

#### **More modular—confirmation, recommendation, and regular queries are handled with less duplication .****[[1]](#fn1)**

### **User Input and Moderation:**

#### **Both versions use user-input moderation before processing LLM queries or presenting output. The enhanced version is more consistent (applies moderation after every model output as well).**

### **Conversation Flow:**

#### **Old Version:**

##### Logic for handling “yes/no” confirmation, recommendations, and fallback messages interleaved within a single large view/post handler.

#### **Enhanced Version:**

##### Clearly separates cases:

###### If top\_3\_laptops is None, gathers user requirements and proceeds to LLM completion for recommendations.

###### Confirmation layer validates dictionary structure with explicit “intent\_confirmation\_layer”.

###### If information is missing, loops until requirements are full, otherwise continues to recommendations and follow-on conversations based on state.

###### Adds debugging printouts ("Intent Confirmation Yes/No") for easier troubleshooting.

###### Ensures that error and fallback handling are consistently applied at each step (e.g., not found cases, flagged moderation, etc.) .[[1]](#fn1)

## **functions.py Changes**

### **Modularity and Chain-of-Thought Prompting**

#### **Old Version:**

##### Contains function schemas and mappings to support an explicit Function Calling API scheme (though implementation of actual LLM function calling is partial and may require further completion).

##### Functions are individually wrapped and registered, intending to map user requirement parsing to a function call.

#### **Enhanced Version:**

##### Simplifies the conversational object lifecycle, focusing on initialize\_conversation, get\_chat\_completions, intent\_confirmation\_layer, and dictionary\_present.

##### Uses detailed chain-of-thought prompts in initialize\_conversation to guide the LLM to ask for all required user profile keys before making a recommendation.

##### “dictionary\_present” uses explicit example-based prompting and validation via OpenAI to robustly extract user requirement dictionaries from model output in standardized format .[[2]](#fn2)

### **Recommendation Calculation & Validation**

#### Both versions use compare\_laptops\_with\_user to calculate a score for laptops based on feature-user need mapping.

#### Enhanced version always checks and parses feature strings into dictionaries before comparison (using dictionary\_present), improving accuracy in matching and enabling more robust, model-agnostic comparison.

#### The function for result validation (“recommendation\_validation”) ensures that only laptops with a meaningful score are forwarded.

### **Function Calling API Pattern**

#### **Old Version:**

##### Contains explicit function schema and function\_map for integrating with OpenAI’s “function calling” mechanism (though some scaffolding and error checks may still be required).

##### The LLM can decide to invoke functions on demand by returning a tool call, and those are dispatched to actual backend Python handlers, making the application extensible for adding new capability endpoints such as product lookup, external API calls, etc. .

#### **Enhanced Version:**

##### Focuses on robust prompt design and modular, reusable logic for handling intent, requirement extraction, and output formatting.

##### Can be easily extended to plug in function calling by using the structure from the old version.

### **Documentation of Function Calling API Integration**

#### **How Integration Works**

##### **Function Schema Declaration**

###### Define function(s) in a schema that the LLM understands, including: function name, parameters, and their types.

##### **Backend Mapping**

###### Register actual backend implementations in a Python dictionary for dispatch when an LLM response requests the function.

##### **Prompt and LLM Chat Completion**

###### Pass schema and messages to the LLM; when a tool call is required, extract function name and arguments from the response, then invoke the mapped backend logic.

##### **Return and Render Output**

###### Pass output from the backend Python function as the “assistant” reply for seamless multi-turn conversation.

#### **Benefits**

##### **Scalability & Maintainability:**

###### Adding new functionality (i.e., product categories, order tracking, product reviews) only requires registering new schema+function in the backend, without major refactoring.

##### **Consistency:**

###### Enforces a consistent format for LLM-to-backend communication, preventing prompt hacking and error-prone string parsing.

##### **Extensibility:**

###### Clear separation between LLM message handling and business logic/services means the app can grow to support a wide range of commerce workflows.

##### **Security:**

###### Fewer chances for unwanted prompt injection as logic is routed by schema rather than ad-hoc prompt parsing.

### **Technical Benefits of the Enhanced Approach**

#### **Resilience & Error Handling:** Every model output—user input and bot completion—is vetted for moderation flags, reducing risk of unwanted content.

#### **Chain-of-Thought Reliability:** By forcing the LLM to iteratively gather all requirements and confirm profile accuracy, the probability of relevant, personalized recommendations is maximized without logic errors.

#### **Debuggability:** Added console log printouts at every step help diagnose issues quickly during development or deployment.

#### **Standardized Output Formatting:** Recommendation formatting and output HTML generation are consolidated, ensuring UI uniformity.

### **Summary Table: Core Differences**

|  |  |  |
| --- | --- | --- |
| Area | Old Version ([function.py/app.py](http://function.py/app.py)) | Enhanced Version ([function.py/app.py](http://function.py/app.py)) |
| Function Calling Support | Explicit schemas & partial mapping | Modular design, easily extendable |
| Input Validation | Manual + prompt/regex | Chain-of-thought + OpenAI validation |
| Recommendation Handling | Inline logic, less modular | Modular, re-usable, standardized |
| Conversation State | Overlapping global state | Isolated, clear per-request flow |
| Error Handling | Basic flags, ad hoc handling | Consistent, end-to-end moderation & logs |
| Prompt Engineering | Simple | Detailed with few-shot and CoT |
| Extensibility | Scalable but scattered | Plug-and-play for new function calls |