ShopAssist 2.0 - Project Summary

# 1. Objectives

* Primary Goal: Enhance the existing ShopAssist chatbot (ShopAssist 1.0) to build ShopAssist 2.0, a more intelligent system capable of understanding user needs and recommending laptops.
* Problem Statement: Given a dataset of laptops (specs, names, descriptions), create a chatbot that interacts with users to identify their needs and provide the top 3 most suitable laptops.
* Key Parameters for Matching: GPU Intensity, Display Quality, Portability, Multitasking, Processing Speed, Budget.

# 2. Design

System Architecture:

* Stage 1: Intent Clarity Layer - Captures user requirements using LLMs with Few-Shot prompting and conversational probing.
* Stage 2: Product Mapping Layer - Parses laptop descriptions and classifies features into high/medium/low using LLM prompts.
* Stage 3: Product Recommendation Layer - Matches user requirements with laptop features and recommends the top 3 options.

Enhancements in ShopAssist 2.0:

* Utilizes OpenAI’s Function Calling feature.
* Eliminates redundant layers from ShopAssist 1.0: intent\_confirmation\_layer(), dictionary\_present(), initialize\_conv\_reco().

# 3. Implementation

Key Components:

* Data Preparation: A dataset (laptop\_data.csv) containing 20 laptops and their descriptions. Extracted features stored in laptop\_dataset\_with\_features.csv.
* Major Functions:
* - initialize\_conversation(): Sets up system prompt and conversation behavior.
* - get\_chat\_completions(): Retrieves assistant responses via OpenAI API.
* - moderation\_check(): Ensures conversation stays appropriate.
* - product\_map\_layer(): Extracts structured laptop features using prompt rules.
* - compare\_laptops\_with\_user(): Scores laptops against user needs and returns top 3.
* - get\_chat\_completions\_function\_calling(): Integrates OpenAI Function Calling.
* - dialogue\_mgmt\_system(): Main orchestration logic for the chatbot.

# 4. Challenges Faced

* Parameter Extraction Accuracy: Ensuring LLMs extract exactly the six required user parameters and enforcing value constraints.
* Classification from Natural Language: Mapping vague descriptions to defined feature levels.
* Function Integration Complexity: Handling OpenAI Function Calling, API retries, and structured outputs.
* User Interaction Handling: Preventing irrelevant queries and ensuring smooth fallbacks.

# 5. Lessons Learned

* LLM Prompting is Powerful: Chain-of-thought and few-shot examples significantly improved chatbot behavior.
* Rule-Based + LLM Hybrid Works Best: Rule-based scoring with LLM-derived features created more precise recommendations.
* Function Calling Simplifies Architecture: Reduced the need for multiple legacy functions and streamlined the process.
* Data Structuring is Critical: Well-labeled datasets enhanced recommendation quality.
* Human-Centric Design Matters: Focusing on user clarity and iterative queries made the chatbot more useful.