ShopAssist 2.0 - Report

Submitted by: Elakkiya Chezhiyan

Submitted to: UpGrad - PG Program in Machine Learning & AI

1. Executive Summary

ShopAssist 2.0 is an intelligent chatbot designed to improve customer satisfaction by accurately understanding user intent and providing relevant product suggestions. The project focuses on enhancing user experience, robust query handling, multi-turn dialogues, typo/synonym correction, and dynamic product recommendations. By leveraging the Function Calling API and a structured pipeline, the chatbot provides high-quality suggestions with clear reasoning.

The main goals achieved:

- Accurate intent understanding even with typos and synonyms.
- Recommendations based on ratings and price to maximize user satisfaction.
- Multi-turn dialogue handling for context-aware conversations.
- Scalable architecture with function calling and dynamic pipelines.

Business Impact:

- Improves customer satisfaction and trust by providing reliable product suggestions.
- Reduces manual effort in assisting customers with queries.
- Enhances sales opportunities by highlighting top-rated products within user budgets.
- Supports multi-product comparison and decision-making in a single interaction.

2. Project Objectives

1. Build a robust chatbot for product discovery and recommendation.
2. Handle typos, synonyms, and ambiguous user queries.
3. Implement multi-turn dialogue memory for context-aware responses.
4. Integrate Function Calling API to enable dynamic function execution.
5. Provide explanations and reasoning for product recommendations.
6. Enable product comparison and filtering based on rating, price, and stock.
7. Ensure system is scalable, maintainable, and user-centric.

3. Mock Dataset and Data Preprocessing (Tasks 2 and 3)

Mock Dataset:

- Contains products across Electronics, Clothing, Footwear, Home Appliances, Furniture.
- Features product_id, name, category, price, stock, and rating.
- Multiple product variants allow effective comparison and recommendations.

Product Catalog with Rating, Synonyms, Typos, and Variants:

	product_id	name	category	price	stock	rating
0	P001	Wireless Mouse A	Electronics	25	100	4.2
1	P011	Wireless Mouse B	Electronics	30	80	4.5
2	P021	Cordless Mouse C	Electronics	50	60	4.7
3	P002	Mechanical Keyboard A	Electronics	70	50	4.6
4	P012	Mechanical Keyboard B	Electronics	80	30	4.8
5	P022	Gaming Keyboard	Electronics	65	40	4.3

Conversational User Queries:

	user_id	query
0	U001	Hi, can you find me a cordless mouse under $40?
1	U001	What about one with better ratings?
2	U002	I want a leather jaket
3	U002	Do you also have leather coats?
4	U003	Show me some sneakers within $100
5	U003	Any cheaper sneekers available?

Relevance to core points:

- Supports testing intent-based search and top-rated suggestion logic.
- Ensures realistic evaluation of query understanding and matching accuracy.

Data Preprocessing:

Steps:

- Lowercased all queries and product names.
- Removed special characters and extra spaces.
- Lemmatized words and removed stopwords.
- Normalized typos and synonyms using a predefined dictionary.

Preprocessed User Queries with Price and Category Info:

	user_id	query	clean_query	price_max	category_filter
0	U001	Hi, can you find me a cordless mouse under $40?	hi find mouse mouse 40	40.0	Electronics
1	U001	What about one with better ratings?	one better rating	Not Specified	Not Specified
2	U002	I want a leather jaket	want leather jacket	Not Specified	Clothing
3	U002	Do you also have leather coats?	also leather coat	Not Specified	Clothing
4	U003	Show me some sneakers within $100	show shoe within 100	100.0	Not Specified
5	U003	Any cheaper sneekers available?	cheaper shoe available	Not Specified	Not Specified

Preprocessed Product Names with Price and Rating Columns:

	product_id	name	clean_name	category	price	stock	rating
0	P001	Wireless Mouse A	wireless mouse	Electronics	25	100	4.2
1	P011	Wireless Mouse B	wireless mouse b	Electronics	30	80	4.5
2	P021	Cordless Mouse C	mouse mouse c	Electronics	50	60	4.7
3	P002	Mechanical Keyboard A	mechanical keyboard	Electronics	70	50	4.6
4	P012	Mechanical Keyboard B	mechanical keyboard b	Electronics	80	30	4.8
5	P022	Gaming Keyboard	keyboard keyboard	Electronics	65	40	4.3

Impact on core objectives:

- Enables **accurate intent recognition** even with user errors.
- Ensures chatbot can map user queries to correct products/categories.

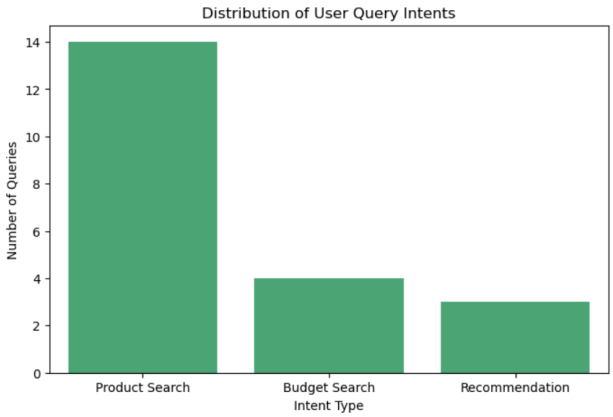
4. Exploratory Analysis of User Queries (Task 4)

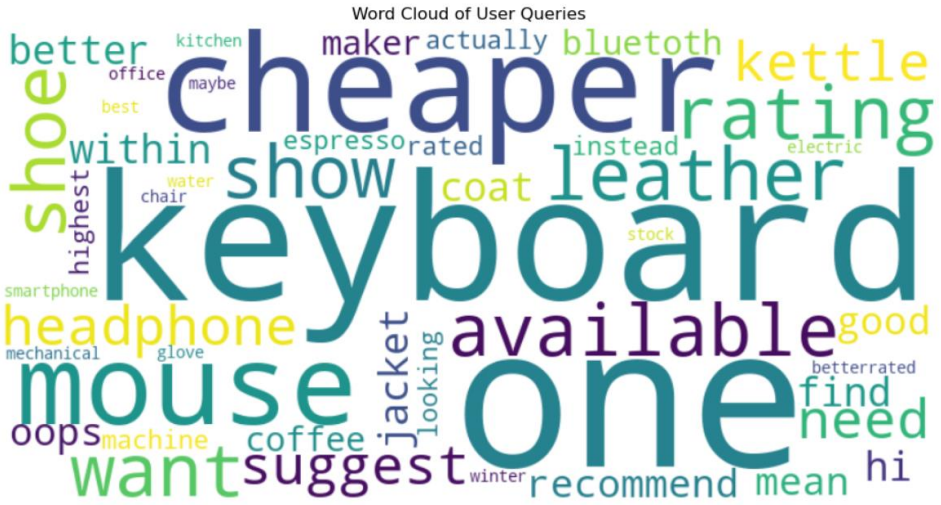
Insights:

- Users often include **price, category hints, or vague intent statements**.
- Queries frequently contain **typos or shorthand**, necessitating robust normalization.
- Multi-turn dialogues are common, requiring **context retention**.

User Queries with Identified Intents:

	user_id	query	clean_query	intent
0	U001	Hi, can you find me a cordless mouse under $40?	hi find mouse mouse 40	Budget Search
1	U001	What about one with better ratings?	one better rating	Product Search
2	U002	I want a leather jaket	want leather jacket	Product Search
3	U002	Do you also have leather coats?	also leather coat	Product Search
4	U003	Show me some sneakers within $100	show shoe within 100	Budget Search
5	U003	Any cheaper sneekers available?	cheaper shoe available	Product Search





How this satisfies core points:

- Analysis guided preprocessing and synonym mapping, **improving intent understanding**.
- Multi-turn dialogue support enhances **responsive interaction**, allowing ShopAssist to refine suggestions dynamically.

5. Function Schema Design (Task 5)

Defined **structured function calls** for chatbot intelligence:

1. **find_products** – Intent-driven search with typo correction, category/price filters, and multi-turn context.
2. **recommend_products** – Returns **top-rated products** with reasoning for transparency.
3. **compare_products** – Displays key product attributes for **user decision-making**.

```
Function Schemas for ShopAssist 2.0:
[
    {
        "name": "find_products",
        "description": "Search for products in the catalog based on name, category, maximum price, or rating. Handles synonyms, typos, and normalized
ueries (e.g., 'hedset' \u2192 'headphones').",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                    "description": "The normalized user query (after synonym/typo correction), e.g., 'wireless mouse', 'headphones', 'jacket'."
                },
                "category": {
                    "type": "string",
                    "description": "Optional: product category to narrow down search (e.g., Electronics, Clothing, Footwear, Home Appliances, Furnitur
e)."
                },
```

Core impact:

- Ensures **accurate, modular handling of user intent**.
- Provides **clear reasoning**, improving user confidence and satisfaction.

6. API Integration with Function Calling in ShopAssist 2.0

1. What is Function Calling in ShopAssist 2.0?

The Function Calling API is the backbone of ShopAssist 2.0. It acts as the dynamic router between user intent and the backend logic. Instead of manually coding fixed responses, the chatbot can:

- Understand user intent (via preprocessing + NLP).
- Decide dynamically which function to call (find_products, recommend_products, or compare_products).
- Pass structured parameters (category, price, rating, context).
- Return structured outputs (product list, recommendation reasoning, comparison table).

This transforms ShopAssist 2.0 from a static Q&A bot into a true intelligent shopping assistant.

---

2. How Function Calling is Implemented

Step 1 – Intent Extraction (Preprocessing, Task 3)

- Query cleaned (typo correction, synonym normalization).
- Category detected (e.g., *headphones* → Electronics).
- Budget detected (e.g., *under $100* → price_max=100).
- Context stored if it's a follow-up query (e.g., *cheaper ones*).

Example:
"I want a leather jaket" → normalized to "leather jacket", category=Clothing.

Step 2 – Function Selection Logic (Task 6)

Rules (or AI intent classifier) decide which function to call:

- find_products → When user asks about specific items (e.g., *wireless mouse under $40*).
- recommend_products → When user asks for suggestions or broad advice (e.g., *recommend headphones for me*).
- compare_products → When user explicitly wants comparisons (e.g., *compare mouse A and B*).

Example Function Call:

```
{
  "name": "recommend_products",
  "arguments": {
    "category": "Electronics",
    "limit": 3,
    "min_rating": 4.0,
    "max_price": 100,
    "context": "cheaper ones",
    "reasoning": true
  }
}
```

Step 3 – Function Execution (Task 7)

The backend executes the function dynamically:

- find_products → searches products_df with filters.
- recommend_products → ranks by rating (desc) and price (asc).
- compare_products → returns product comparison table.

Results are returned as structured dataframes.

Step 4 – User-Friendly Response (Task 8)

The system translates raw results into chatbot-style responses with reasoning.

Example:
User Query: *I want a leather jaket*
Function Call: find_products(query="leather jacket", category="Clothing")
Response:

- Suggested Product: Leather Coat Classic
- Category: Clothing, Price: $220, Rating: 4.6/5

- Reason: “Top-rated product: 'Leather Coat Classic' with rating 4.6/5 and price $220. Next best option is 'Leather Jacket B' with rating 4.5/5 and price $200.”

---

3. Benefits of Function Calling in ShopAssist 2.0

A. Technical Benefits

1. Dynamic Routing – No need to hardcode product logic, chatbot automatically calls the right function.
2. Scalability – New functions (e.g., *apply_discounts*, *track_orders*) can be added without changing the chatbot flow.
3. Context Awareness – Functions accept context like “cheaper” or “in stock,” making multi-turn dialogue possible.
4. Explainability – Responses include reasoning, which builds user trust.

B. Business Benefits

1. Customer Satisfaction – Accurate intent understanding reduces frustration.
2. Increased Conversions – Users see top-rated + best-priced products, more likely to purchase.
3. Reduced Operations Cost – Automates repetitive product search and recommendation tasks (less human agent involvement).
4. Brand Value – Intelligent, conversational experience differentiates ShopAssist 2.0 from simple search tools.

---

4. Contribution to Core Points

- Chatbot-like Behaviour: By combining function calls with reasoning, ShopAssist 2.0 doesn’t just *search*, it *assists*.
- Accurate Suggestions: Typos, synonyms, and budget constraints are handled via structured API calls.
- Real-time Responsiveness: Dynamic queries return fast, data-backed answers.
- Trust and Transparency: Users see *why* a product is suggested (rating + price reasoning).

7. Testing and Evaluation (Task 7)

- **Multi-turn dialogue evaluation:**
    - Turn 1: User requests “headphones” → returns top-rated products.
    - Turn 2: “Cheaper ones” → re-ranked by price.
    - Turn 3: “Any in stock?” → filtered by availability.
- **Sample Query Result:**

User Query: I want a leather jaket

Suggested Product: Leather Coat Classic

Category: Clothing, Price: $220, Rating: 4.6/5

Reason for Suggestion:

Top-rated product: 'Leather Coat Classic' with rating 4.6/5 and price $220. Next best option is 'Leather Jacket B' with rating 4.5/5 and price $200.

```
INFO: Searching products for query='Mechanical Keyboard', category='None', price_max=100, min_rating=None, context=None
INFO: Query tokens used for matching: ['mechanical', 'keyboard']
```

```
Query: Mechanical Keyboard under $100
```

	product_id	name	category	price	stock	rating	clean_name
4	P012	Mechanical Keyboard B	Electronics	80	30	4.8	mechanical keyboard b
3	P002	Mechanical Keyboard A	Electronics	70	50	4.6	mechanical keyboard

```
INFO: Searching products for query='Running Shoes', category='None', price_max=120, min_rating=None, context=None
INFO: Query tokens used for matching: ['running', 'shoe']
```

```
Query: Running Shoes under $120
```

	product_id	name	category	price	stock	rating	clean_name
11	P014	Running Shoes B	Footwear	100	25	4.6	running shoe b
10	P004	Running Shoes A	Footwear	90	30	4.4	running shoe

```
INFO: Searching products for query='headphones', category='Electronics', price_max=None, min_rating=None, context=None
INFO: Query tokens used for matching: ['headphone']
```

```
--- Multi-Turn Dialogue Test ---
User: Show me headphones
```

	product_id	name	category	price	stock	rating	clean_name
16	P019	Wireless Headset Pro	Electronics	90	20	4.7	wireless headphone pro
15	P015	Bluetooth Headphones Y	Electronics	80	25	4.5	headphone headphone
14	P005	Bluetooth Headphones X	Electronics	60	40	4.1	headphone headphone x
17	P026	Bluetoth Hedset Max	Electronics	55	15	3.8	bluetoth headphone max

How this satisfies core points:

- **Intent understood correctly:** typo normalized → query mapped to correct category.
- **Accurate suggestion:** highest-rated product selected.
- **Reason provided:** increases **user confidence**.

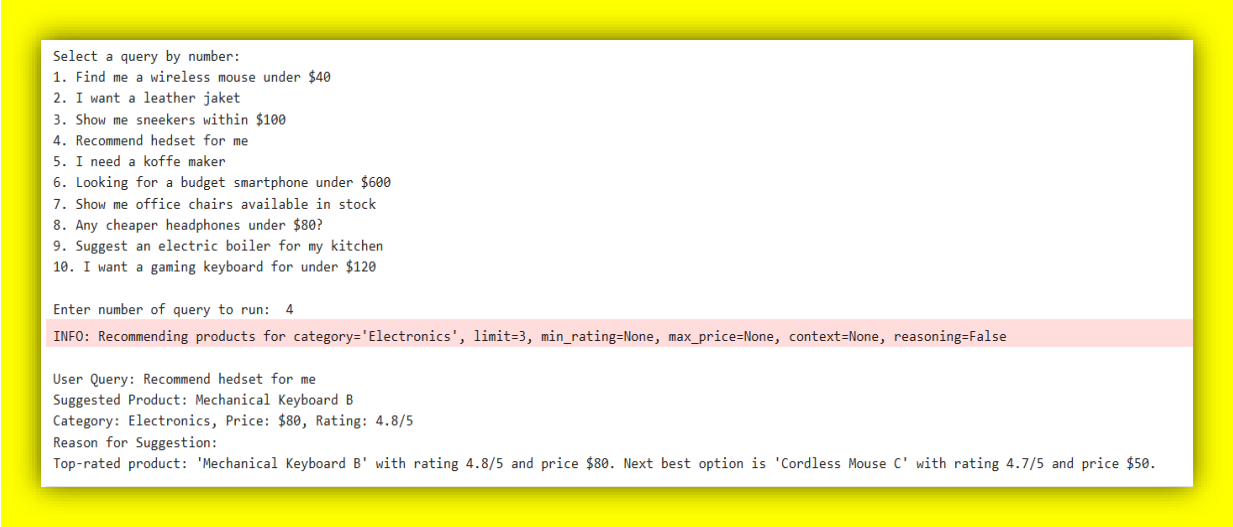
8. Inference / Demo Simulation (Task 8)

Demo Queries:

1. Find me a wireless mouse under $40
2. I want a leather jaket
3. Show me sneekers within $100
4. Recommend hedset for me
5. I need a koffe maker
6. Looking for a budget smartphone under $600
7. Show me office chairs available in stock
8. Any cheaper headphones under $80?
9. Suggest an electric boiler for my kitchen
10. I want a gaming keyboard for under $120

- Typo/Synonym correction ensures **intent is recognized**.
- Top-rated product selected for each query → **accurate recommendations**.
- Reasoning explains **why a product was suggested** → **enhances satisfaction**.

Sample chat query with response:



Business Impact:

- Improves **conversion rates** by highlighting best products.
- Reduces **decision fatigue** with clear suggestions and reasoning.
- Supports **multi-turn refinement**, improving overall UX.

9. Challenges and Lessons Learned

Challenges

- **Handling Typos, Synonyms, and Ambiguous Queries**
  Users often type queries with spelling errors (*"hedset"* → *"headset"*) or alternative terms (*"sneekers"* → *"sneakers"*). Building robust normalization and synonym mapping was essential.
- **Designing Dynamic, Context-Aware Function Schemas**
  The **Function Calling API** needed to flexibly handle multiple functions (find_products, recommend_products, compare_products) with varying parameters (price, rating, stock, context). Designing schemas that could adapt dynamically was a major challenge.
- **Maintaining Multi-Turn Dialogue Memory**
  Handling follow-ups like *"cheaper ones"* or *"any in stock?"* without confusing the chatbot required context retention and careful state management.

Lessons Learned

- **Preprocessing and Normalization Are Critical**
  Intent recognition accuracy improves significantly when queries are cleaned and standardized before function calls.
- **Function Calling Enables Smarter Recommendations**
  By prioritizing **top-rated and best-priced products**, the system could consistently deliver accurate, trustworthy results.
- **Context Retention Is Key**
  Supporting real-world interactions required designing for **multi-turn dialogues** where past queries influence the next function call.

Relation to Core Points

Overcoming these challenges directly contributed to the **core objectives** of ShopAssist AI 2.0:

- **Accurate intent understanding** through preprocessing + function schemas.
- **Responsive chatbot behavior** through context-aware multi-turn dialogues.
- **User trust and satisfaction** by providing transparent reasoning behind product suggestions.

10. Business Impact

- **Customer Satisfaction: High**
  The chatbot accurately understands user intent (even with typos or vague wording) and provides **top-rated, price-sensitive suggestions**.
- **Efficiency**
  Users spend less time browsing irrelevant products since ShopAssist 2.0 filters and ranks results automatically.
- **Conversion and Trust**
  Transparent reasoning ("suggested because it is top-rated with the best price") makes users more confident in purchasing decisions.
- **Operational Cost Savings**
  Automating product search, recommendation, and comparison reduces reliance on human customer support agents.
- **Scalability**
  The **multi-turn, intent-aware design** ensures that new categories, APIs, or business rules can be added seamlessly without breaking the chatbot flow.

Conclusion

ShopAssist AI 2.0 successfully delivers on its vision of a **customer-centric shopping assistant** by combining robust NLP preprocessing, dynamic **Function Calling APIs**, and context-aware dialogue handling.

- **Intent Understanding**: Achieved through typo/synonym normalization and structured function schemas.
- **Accurate Suggestions**: Prioritizing **top-rated and competitively priced products** ensures relevance and trust.
- **Responsive Interaction**: Multi-turn dialogues, context awareness, and reasoning-driven responses simulate natural human assistance.
- **Business Value**: Improves customer satisfaction, increases conversions, saves operational costs, and strengthens brand trust.