Online Shopping Assistant
Team 2

Amruth Amruth[2043415], Jahanvi Panchal[1939439], Marmee Pandya[1963521], Manasi Patil[2034414], and Rutuja Gaikwad[1960118]

University of Mannheim, Mannheim, Germany

Abstract. The online shopping experience has become increasingly complex, often overwhelming users with an abundance of choices, technical jargon, and the challenge of identifying the most suitable products. To address these issues, this project introduces a novel shopping assistant powered by Large Language Models (LLMs). Utilizing advanced natural language processing and a multi-agent architecture, the assistant enhances the shopping experience by retrieving and analyzing product data, verifying specifications, and generating intuitive, user-friendly summaries. A key feature of the system is its ability to deliver personalized product recommendations based on user preferences and contextual needs. By automating these tasks, the assistant reduces user effort, accelerates decision-making, and offers intelligent, data-driven guidance. The result is a more efficient, personalized, and satisfying online shopping experience.

Keywords: Large Language Models · Multi-Agent Systems · Natural Language Processing

Please find the GitHub repository here: GitHub Link

1 Introduction

Problem Statement

The modern online shopping landscape presents both an opportunity and a challenge for users. While there is unparalleled access to products across categories, shoppers are often overwhelmed by information overload, inconsistent specifications, and marketing jargon. Existing e-commerce platforms typically provide static filters and generic recommendations, lacking the depth to understand nuanced user preferences. Furthermore, the absence of interactive, intelligent systems means users must manually compare products across multiple tabs, often relying on incomplete or biased reviews. This leads to decision fatigue, suboptimal purchases, and reduced user satisfaction. A more adaptive, context-aware assistant is needed—one that can parse complex queries, extract meaningful product information, and guide users toward optimal decisions with confidence and transparency.

Motivation

This project is driven by the need to simplify and enrich the online shopping experience by applying recent advances in natural language processing and agent-based orchestration. Current recommendation engines rarely account for personalized feature-level preferences or user intent expressed in free-text queries. They operate largely on historical behavior and aggregated ratings, overlooking situational context such as budget, feature constraints, or brand expectations. By leveraging Large Language Models (LLMs), the assistant gains the ability to interpret complex queries, extract structured information from unstructured descriptions, and generate coherent, human-readable recommendations. When combined with LangGraph's multi-agent design, the assistant becomes a reliable partner for navigating e-commerce, mimicking expert reasoning to provide relevant and actionable guidance.

Objectives

The project aims to design and implement a smart shopping assistant that can intelligently handle user requests and provide high-quality product recommendations. The key objectives are:

- **Personalized Assistance:** Tailor product suggestions based on user-specified needs, such as price limits, feature preferences, and usage context.
- **Agentic Workflow Orchestration:** Use LangGraph to modularize and manage discrete agents responsible for query restructuring, product search, specification extraction, scoring, and recommendation generation.
- **LLM-Based Scoring and Justification:** Utilize the LLaMA 3.1 model to assess products on dimensions like performance, value for money, and alignment with user needs, and generate transparent explanations.
- **User-Centric Interface:** Offer an intuitive and interactive Streamlit-based frontend that allows users to view results in visually enriched cards and download structured comparisons.
- **Traceability and Extensibility:** Enable reproducible results and future enhancements by logging all outputs and maintaining a clean separation between data, logic, and presentation.

2 Methodology

This section describes how the shopping assistant was designed and implemented, covering the system architecture, workflow management, data handling, LLM integration, user interface, and core functionality. The system is built to simplify online shopping by accurately understanding user needs, retrieving relevant products, and delivering intelligent, personalized recommendations.

System Architecture

The system follows a modular, layered architecture, enabling clear separation between user input, data processing, retrieval, and output. Each layer is designed to be flexible and scalable:

- **Input Layer:** A user-facing interface built with Streamlit captures natural language queries, price limits, and additional requirements (e.g., technical features).
- **Processing Layer:** A workflow powered by LangGraph[1] manages the interaction between intelligent agents. This layer connects to data sources and LLMs to enrich information.
- **Data Layer:**
  - SERPAPI fetches real-time product listings.
  - Tavily extracts detailed descriptions, specifications, and reviews.
- **Output Layer:** Streamlit displays results with CSV export for transparency.

LangGraph-Based Agentic Approach

The system uses LangGraph to structure logic as a directed workflow, with each agent representing a specific task:

```
process_query → search_products → extract_specifications →
      rank_products → generate_recommendations → END
```

Each node is implemented as an asynchronous function in Python, orchestrated through the `ShoppingGraph` class. The workflow is executed using LangGraph's `ainvoke()` method, which processes a shared `ProductState` object throughout the graph. This state dictionary tracks query inputs, intermediate transformations, product lists, and module-wise status updates, ensuring modularity and transparency for debugging and evaluation.

Data Sources

- **SERPAPI:** Retrieves shopping results from Google, including product titles, prices, ratings, reviews, and links. This forms the primary dataset for the assistant.
- **TavilySearchResults:** Provides deeper context and specifications by performing targeted web searches. Keywords like "specifications," "features," "pros and cons" are included to retrieve meaningful content, which is later processed and structured.

All API keys are stored securely using environment variables.

LLM Integration

We initially used **Gemini 2.0 Flash** for its fast and capable generation. However, the free-tier limit of 15 requests per minute (RPM) became a bottleneck, especially since the system must process up to 20 product entries per query retrieved from SERPAPI. Additionally, Gemini frequently returned improperly formatted JSON outputs—even when given strict formatting instructions—which led to parsing failures during product analysis. Given that our workflow relies heavily on clean, structured outputs for summarization and ranking, these issues made Gemini unreliable for core components.

DeepSeek-Coder, a code-oriented language model designed primarily for programming and reasoning tasks, was also tested as a local alternative. While it offered decent performance on smaller, well-structured prompts, it struggled with tasks involving nuanced product analysis and natural language understanding. The model often produced rigid or shallow summaries, and had difficulty interpreting product specifications or extracting meaningful features from unstructured descriptions. Additionally, its outputs were more prone to generic phrasing and lacked the domain sensitivity required for personalized product recommendations. These limitations made it unsuitable for the multi-step, interpretive reasoning required by our assistant's ranking pipeline.

Although models such as **Claude** (Anthropic) and **ChatGPT (GPT-4)** were strong candidates due to their reasoning capabilities and established benchmarks, we opted not to implement them due to pricing constraints. Our system calls the LLM multiple times per query, and the cumulative cost of using commercial APIs would be significant, making them impractical for iterative local development.

In contrast, **LLaMA 3.1** performed reliably across all stages of our multi-agent pipeline with minimal intervention or exception handling. It consistently adhered to formatting instructions, preserved structural coherence, and delivered stable outputs across iterations. Its adaptability and reproducibility made it the most practical and cost-effective choice for our use case.

In summary, although several LLMs were tested or considered, LLaMA 3.1 offered the optimal balance between performance, reliability, cost-efficiency, and integration flexibility.

Based on this analysis, we selected **LLaMA 3.1** as our model of choice. Integrated via **Ollama** and hosted locally, it supports the full system pipeline and performs several key functions:

- **Query Restructuring:** The user's raw input, along with budget and optional features, is transformed into a structured query using a prompt-based approach implemented in the `process_query_node`. The prompt follows strict formatting rules with price and requirements enforcement. The LLM response is parsed using regular expressions to extract the restructured query, and validation steps ensure completeness before proceeding to product retrieval.

- **Specification Structuring:** Retrieved raw product descriptions from external sources are transformed into structured formats containing key features, pros, cons, and a concise summary. This aids downstream scoring and display.
- **Product Ranking:** Each product is evaluated along three core dimensions: performance, value for money, and alignment with the user's specific requirements. To ensure reliable model performance and prevent prompt overflows, products are processed in batches of five during the ranking phase. The LLM assigns numerical scores across these dimensions and supplements them with detailed, category-specific justifications. These justifications—covering performance analysis, value analysis, and requirement matching—are presented to the user to improve transparency and interpretability. Additionally, the scoring process accounts for the reliability of user reviews: products with fewer than 10 reviews are weighted conservatively, while those with over 50 reviews are considered more trustworthy and are given greater influence in the final ranking.
- **Recommendation Generation:** For the top-ranked products, the model generates clear, user-friendly justifications and an overall comparative summary to guide final decision-making.

To ensure consistency and avoid common issues with LLM outputs (e.g., formatting errors or incomplete JSON), additional cleaning, validation, and regex-based fixes are applied.

User Interaction and Workflow

From the user's perspective, the assistant operates through a simple, intuitive workflow:

- **Input:** The user provides a product query, sets a budget, and optionally adds specific needs (e.g., "16GB RAM").
- **Restructuring:** The query is rewritten to be more structured and suitable for search.
- **Product Retrieval:** SERPAPI fetches up to 20 matching products from Google Shopping.
- **Enrichment:** Each product is enriched with structured details using Tavily and the LLM.
- **Scoring & Ranking:** Products are evaluated and scored. The top 10 are retained.
- **Recommendation Generation:** A final summary is created for the top three products.
- **Output:** The results are presented visually in Streamlit, with scores, comparisons, and analysis.

This full pipeline typically completes within 30 seconds and runs asynchronously using LangGraph's `ainvoke()` method.

User Interface

The user interface is built using Streamlit, offering an interactive experience that works across desktop and mobile. Key UI features include:

- **Search Panel:** Users can type free-text queries and specify constraints such as price range and desired features.
- **Product Cards:** Each result is displayed in a visually styled card showing the product image, price, rating, key features, pros and cons, and a direct purchase link.
- **Score Highlights:** Products are scored across different metrics and visually represented using custom-styled elements.
- **Top Picks:** The top 3 recommended products are highlighted with LLM-generated justifications.
- **Downloadable CSV:** A .csv file is generated per session with both raw and ranked products, ensuring the user can revisit or compare results offline.

The frontend includes custom CSS for polished styling and clear, readable formatting.

Code Implementation

The backend system is engineered for modularity and resilience. All logic is encapsulated within the `ShoppingAssistant` class, which manages the complete end-to-end processing of user queries. Central to this logic is the use of LangGraph to define a stateful workflow, ensuring that each stage—query restructuring, product retrieval, enrichment, ranking, and recommendation—is handled asynchronously and independently. The backend operates on a shared, typed dictionary called `ProductState`, which carries information and status across the entire pipeline.

Graph Definition The `ShoppingGraph` class defines the workflow as a directed graph of asynchronous nodes:

- `process_query`: Parses and restructures user input into a precise, search-optimized format using prompt-based LLaMA inference.
- `search_products`: Connects to Google Shopping via SERPAPI and collects up to 20 products with metadata including price, ratings, image, and source URL.
- `extract_specifications`: Enriches each product using Tavily web search results and structured summarization via LLaMA. Extracted data includes key features, pros and cons, and summary.
- `rank_products`: Processes products in batches of five. Each product is scored by the LLM across multiple dimensions (performance, value, requirement match). A justification is generated for each score.
- `generate_recommendations`: Selects the top-ranked products and generates a natural language explanation for each, along with an overall comparative analysis.

Error Handling and Reliability Each agent node includes detailed exception handling:

- **API robustness:** All external calls (SERPAPI, Tavily) are wrapped in try-catch blocks with fallback logic to ensure the pipeline continues even if data is partially missing.
- **Output validation:** LLM responses are checked for structural conformity (especially JSON). Regex-based cleaning is applied when formatting inconsistencies are detected.
- **Status tracking:** A `status` dictionary is updated at each node, enabling easy inspection and debugging of the full execution lifecycle.

Session Logging and Export For transparency and reproducibility, a helper function `save_to_csv()` logs each session to disk. This includes:

- Raw product listings from SERPAPI with original query context
- Final ranked products with scores and analysis
- Selected recommendations with model-generated justifications

The resulting CSVs allow offline review, audit trails, and support future training or tuning of the assistant.

Overall, the codebase is designed to support both rapid experimentation and production-grade execution, with strong separation of concerns, centralized state management, and extensible module boundaries.

3 Results and Evaluation

Experimental Setup

To evaluate the system's recommendation quality, we curated a test set of 20 distinct product-related queries covering diverse categories such as electronics, kitchen appliances, personal gadgets, and home utilities. For each query, the system retrieved 20 product candidates using Google Shopping via SERPAPI. These queries were selected to reflect realistic shopping scenarios, incorporating both broad and narrowly defined product intents. The evaluation compared baseline and LLM-based recommendations over this fixed query set.

Evaluation Methodology

To rigorously assess the effectiveness of the AI Shopping Assistant, we designed a ranking-based evaluation framework grounded in manual annotation and standard information retrieval metrics. The setup was structured as follows:

- **Dataset Composition:** Each of the 20 evaluation instances involved 20 candidate products retrieved via SERPAPI. These covered a wide range of categories to ensure generalization and robustness.

- **Ground Truth Ranking:** A team of annotators manually evaluated the 20 retrieved products per query and selected the top 10 based on specifications, price-value ratio, and overall relevance. This human-curated set formed the reference ranking.
- **Models Compared:**
  - **Baseline:** Randomly selected 10 out of 20 candidates, representing a naïve, non-personalized approach.
  - **LLM-Based Assistant:** Used the full LangGraph-based multi-agent pipeline to restructure queries, extract structured details, compute relevance scores with the LLM, and rank products accordingly.
- **Evaluation Metrics:**
  - **Precision@10:** Measures the proportion of recommended products that are present in the ground truth top 10.
  - **MRR@10 (Mean Reciprocal Rank):** Captures how early the first correct match appears in the ranking.
  - **NDCG@10 (Normalized Discounted Cumulative Gain):** Evaluates ranking quality by assigning more weight to correctly ranked items appearing earlier in the list.

Table 1. Average Evaluation Metrics Across 20 Queries

Model	Precision@10	MRR@10	NDCG@10
Baseline (Random)	0.315	0.580	0.672
LLM-Based Assistant	**0.565**	**0.755**	**0.819**

As shown in Table 1, the LLM-based assistant outperformed the baseline across all metrics by a significant margin.

Interpretation of Results:

- **Precision@10:** The assistant recommended relevant products in 56.5% of cases, compared to 31.5% for the baseline, demonstrating better recall of user-relevant items.
- **MRR@10:** The higher score of 0.755 implies that relevant results were frequently surfaced near the top of the assistant's rankings.
- **NDCG@10:** With a score of 0.819, the assistant's output was well-aligned with the human-generated ground truth, emphasizing both relevance and ranking order.

To provide a concrete example, Table 2 presents the ranking results for the query *"Stand Mixer"*, showing how the baseline, LLM assistant, and human evaluators prioritized the products.

As illustrated in Table 2, the assistant correctly prioritized models like the Ninja BN750EU and Gourmetmaxx Standmixer, both of which were highly

Table 2. Sample Product Rankings for Query: *Stand Mixer*

Rank	Ground Truth	Baseline	LLM-Based
1	Ninja BN750EU 2-in-1 Standmixer	Ninja CB350EU Foodi 3-in-1	Ninja Foodi CB100EU
2	Ninja Foodi CB100EU	Ninja BN750EU 2-in-1	Ninja BN750EU 2-in-1
3	Gourmetmaxx Standmixer	Cecotec PowerTwist 500	Gourmetmaxx Standmixer
4	Ninja CB350EU Foodi 3-in-1	Domo DO710BL	H.Koenig Küchenmaschine KM80
5	SILVERCREST SKMP 1300 D3	Ninja Foodi CB100EU	Sencor SBL 7570SS
6	Morphy Richards Standmixer	Gourmetmaxx Standmixer	Severin SM 3737
7	Cecotec Cecomixer Merengue	Bosch MMB2111S VitaPower 2	Bosch MMB2111S VitaPower 2
8	Vospeed Küchenmaschine 1000W	Standmixer – Ambiano	Cecotec PowerTwist 500
9	Sencor SBL 7570SS	SILVERCREST SKMP 1300 D3	Domo DO710BL
10	Clatronic KM 3765	Blaupunkt BP4002 Mixer	Bella Mini Rührmaschine 1200W

ranked by human evaluators as well. Meanwhile, the baseline included less relevant or mismatched entries such as the Blaupunkt BP4002 Mixer.

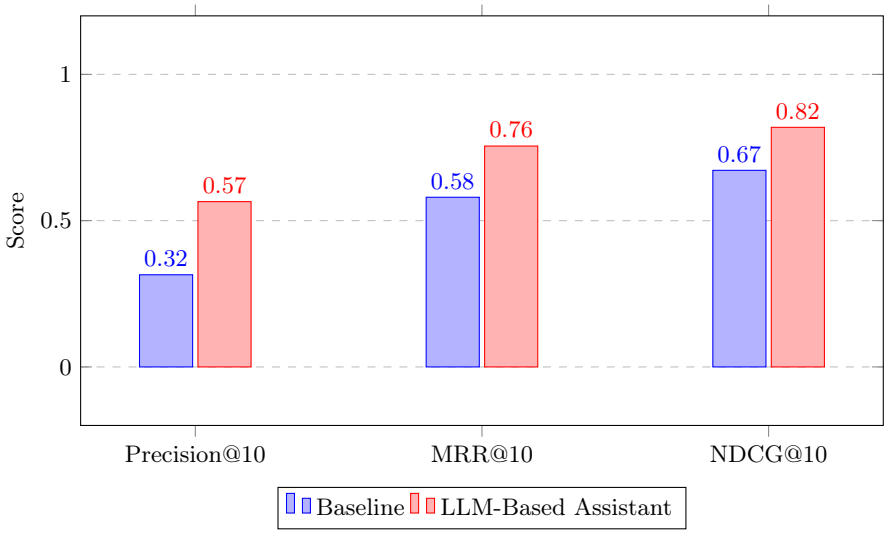

Fig. 1. Performance Comparison of Baseline vs. LLM-Based Assistant

Summary

The assistant's superior performance can be attributed to its ability to reason over structured specifications, interpret nuanced requirements, and filter products based on review quality and pricing. Unlike the baseline—which exhibited random and often repetitive outputs—the LLM-based approach consistently surfaced context-aware, high-value recommendations. Its structured justifications and transparent scoring further enhanced user trust and satisfaction.

Importantly, the assistant showed strong generalization across various product types, highlighting the robustness of LLM-driven ranking strategies. These findings validate that integrating structured prompt-based evaluation with modular agent workflows can significantly improve the quality and relevance of product recommendations in real-world scenarios.

4 Challenges and Limitations

While the assistant demonstrates promising capabilities, several limitations were encountered during implementation:

- **External API Dependencies:** The system relies on SERPAPI and Tavily for product retrieval and detail extraction. Any downtime, quota limits, or changes in API response formats can disrupt pipeline functionality.
- **LLM Output Instability:** Despite prompt engineering and regex-based validation, LLMs occasionally produce malformed or inconsistent outputs, especially for rare product categories or ambiguous inputs. This necessitates continual monitoring and format correction.
- **Latency Constraints:** The complete workflow—from query processing to product ranking—takes a long time. While manageable, this latency could be a bottleneck for users expecting real-time responsiveness.
- **Data Coverage Gaps:** For niche or region-specific products, SERPAPI may return incomplete or generic results, reducing the system's relevance and personalization quality.
- **Staleness of Results:** Since the assistant captures product data at the moment of query, there is no mechanism to track live inventory or price fluctuations. This may occasionally result in outdated recommendations.

5 Future Work

Building upon the current implementation, several avenues for enhancement and scalability have been identified:

- **Multi-Source Retrieval:** Integrate additional APIs (e.g., Amazon, eBay, BestBuy) or controlled scraping agents to enrich product listings, expand coverage, and mitigate dependency on a single provider.
- **Faster Model Alternatives:** Deploy lighter or quantized LLM variants to accelerate response times without significantly compromising output quality.

- **Persistent User Profiles:** Introduce a memory mechanism to retain user preferences, previous queries, and product interactions for hyper-personalized experiences.
- **Voice Interaction:** Integrate voice interfaces using models like Whisper to support hands-free interactions and improve accessibility.
- **Backend Monitoring Dashboard:** Create a real-time dashboard for developers to track user activity, API performance, and ranking accuracy, enabling continuous improvement and debugging.
- **Live Product Status:** Implement price and inventory validation checks at recommendation time to ensure product suggestions remain available and up to date.

6 Conclusion

This project demonstrates the feasibility and value of a Large Language Model-powered shopping assistant that enhances online retail experiences through intelligent query processing, specification analysis, and personalized recommendations. By leveraging LangGraph's agentic workflow, the system modularly orchestrates discrete tasks such as query restructuring, real-time product retrieval, structured specification extraction, scoring, and recommendation generation. The integration of **LLaMA 3.1** via *Ollama* proved to be both reliable and cost-effective, allowing the assistant to deliver consistent and explainable outputs across a diverse set of product categories.

Our evaluation across 20 real-world shopping queries showed significant performance gains over a baseline model, with improvements of over 20% in **Precision@10** and notable increases in both **MRR** and **NDCG**. These results affirm the assistant's capability to surface high-quality, user-relevant products by interpreting nuanced preferences and filtering noisy or suboptimal candidates.

Despite these strengths, the assistant is not without limitations. Its reliance on third-party APIs introduces fragility, and the inherent latency of LLM-based processing affects responsiveness. Additionally, live inventory tracking and persistent user preferences remain unaddressed in the current version.

Looking forward, the system offers numerous opportunities for enhancement, such as multi-source retrieval, faster model variants, and voice-enabled interactions. With continued development, the assistant can evolve into a comprehensive digital shopping companion, helping users make informed decisions with minimal effort and maximum confidence.

References

1. Building a Multi-Agent System with LangGraph and Gemini. *Medium*. https://medium.com/@ipeksahbazoglu/building-a-multi-agent-system-with-langgraph-and-gemini-1e7d7eab5c12
2. Shopping MMLU: A Massive Multi-Task Online Shopping Benchmark for Large Language Models *arXiv preprint arXiv:2410.20745*. https://arxiv.org/abs/2410.20745
3. LLaSA: Large Language and E-Commerce Shopping Assistant *arXiv preprint arXiv:2408.02006*. https://arxiv.org/abs/2408.02006

Declaration of Used Tools

Tool	Purpose	Where?	Useful?
ChatGPT	Paraphrasing	Throughout	+++
Claude AI	Code Generation	app.py, backend.py	++