Project Report

Jarir-NLP: Personalized AI Salesman

Almaan Khan[1], Baraa Zino[1], Muhammad Ammar Sohail[1], Mohammed Rashad[1]

Supervisor: Dr. Salman Ahmad Khan[2]

[1] *KAUST Academy*
[2] *Oxford Brookes University*

Contents

1 Project Objectives

The primary objective of this project is to create a personalized shopping experience for electronics buyers on Jarir's online platform, emulating the interactive and consultative role traditionally performed by showroom salesmen. Given the expansive range of PC products available through Jarir, customers face significant decision-making challenges. Our goal is to leverage artificial intelligence to streamline and personalize this decision-making process.

1.1 Specific Objectives

- **AI-Powered Salesman (Virtual Assistant):**
  - Create an intelligent conversational agent (AI Salesman) that simulates the interactive and informative role of an in-store salesperson.
  - Ensure that the AI Salesman can effectively handle customer inquiries, provide tailored advice, and assist customers in confidently selecting the ideal electronic product for their unique needs.
- **Business Impact:**
  - Enhance the online customer experience, leading to increased customer satisfaction, engagement, and ultimately higher conversion rates.
  - Address stagnation in the PC product category by capitalizing on current technological trends.
- **Project Delivery and Implementation:**
  - Clearly outline project phases, milestones, and timelines to ensure efficient progress.
  - Define and gather the necessary data requirements and resources to achieve project success.

2 Project Progress

2.1 Project Kick-off & Problem Analysis

- Received the exact problem statement on July 4, 2025.
- Held a meeting to align scope and define immediate next steps.
- Drafted clarifying questions for Jarir.

2.2 Method Selection

- Investigated multiple candidate approaches for conversational product recommendation:
  1. Retrieval-Augmented Generation (RAG) MVP
  2. Dynamic Questionnaire + Rule-Based Ranker
  3. Hybrid rule-based decision tree with ML-driven intent classification
  4. Agentic AI system with multi-step planning and tool usage
- **Decision:** Adopt an Agentic AI framework as our core architecture because:

1. Agentic systems support dynamic, multi-step reasoning and decision-making, allowing the assistant to plan and adapt based on user intent.
2. Tools such as retrieval, ranking, and structured querying can be orchestrated intelligently based on context.
3. The modular nature of agentic workflows enables scalability, debuggability, and easier integration of new capabilities over time.

2.3 Milestones Achieved

- Developed and deployed a robust product scraper to extract laptops, gaming PCs, tablets, and CPUs from Jarir.com, including price, images, and metadata.
- Cleaned and structured the product catalog into CSVs for downstream indexing and retrieval.
- Built a hybrid product search engine combining exact filtering with FAISS-based semantic retrieval using SentenceTransformer embeddings.
- Constructed a LangGraph-based AI agent with support for memory, tool usage, and adaptive decision-making.
- Integrated the agent with a structured JSON response schema suitable for frontend rendering.
- Created a natural and human-like LLM prompt flow to dynamically ask clarifying questions and explain product rankings.
- Began building a visual poster, including AI-generated illustrations and timeline graphics.
- Tested 10+ realistic use cases (e.g., “I need a laptop for video editing under 5000 SAR”) and refined outputs accordingly.

3 Methodology

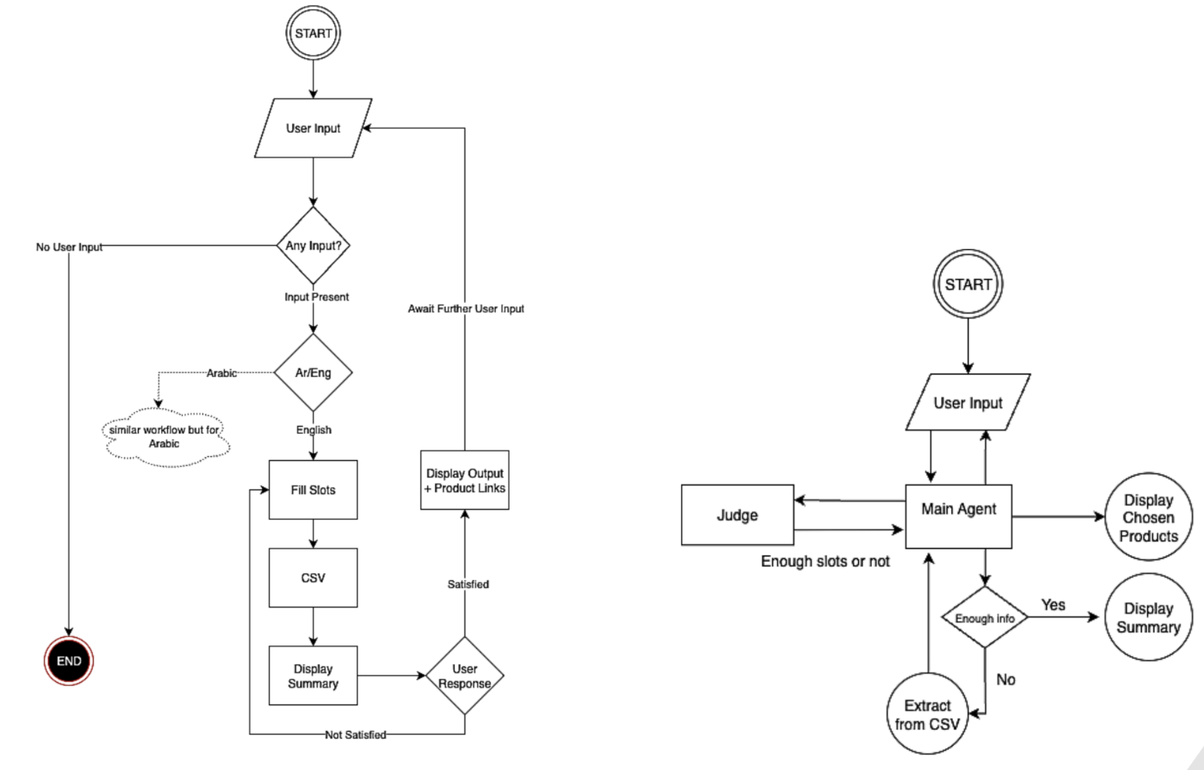
The Jarir-NLP project adopts an Agentic AI approach to build a dynamic, conversational assistant capable of replicating the decision-making and advisory skills of a human electronics salesman. Unlike traditional chatbots that rely on static intent-response pairs, our system is designed around autonomous reasoning and multi-step planning. At the core is a planning loop where the agent interprets user queries, formulates intermediate objectives, and selects the appropriate tools to fulfill them. These tools include product search, attribute filtering, specification comparison, price thresholding, and answer generation. For example, when a user asks, “What’s the best lightweight laptop for graphic design under 4000 SAR?”, the assistant identifies intent (creative work, portability, budget constraint), queries the database for matching products, filters results, and generates a human-like recommendation, often through multiple reasoning steps.

This agentic behavior is conducted using a modular framework where each tool acts as a callable function or service. We have implemented this using modern frameworks such as LangGraph, allowing for control over task routing and memory management. The architecture supports context tracking across multiple turns, meaning the assistant can handle follow-ups like “Which one has better battery life?” or “Show me something cheaper with similar specs”.

To power this pipeline with real-world data, the team has developed a robust web scraping system to extract product specifications directly from the Jarir website. This decision was made due to the absence of direct access to Jarir's internal APIs or datasets. The scraping pipeline leverages a combination of Python libraries including `requests` for HTTP requests, `BeautifulSoup` for parsing HTML content, `urllib.parse` for resolving relative URLs, `html` for unescaping HTML entities, `re` for regular expression-based extraction, `time` for throttling requests, `pandas` for data structuring, and `json5` for flexible data serialization. The fields include CPU model, GPU, RAM, storage type and size, screen size/resolution, OS, connectivity options, battery capacity, price information, etc. Extracted data is cleaned and normalized into a structured format (CSV and JSON), which is then ingested into a back-end store. The web scraping system is designed to be fault-tolerant, with scraping scripts capable of refreshing the product index at regular intervals to keep pace with updates on Jarir's site.

Together, these components allow the Jarir-NLP system to operate as a fully autonomous digital sales advisor, grounded in real data, with the capacity to evolve into a production-grade assistant aligned with modern e-commerce expectations.

We first prototyped two alternative architectural workflows (Workflow 1 and Workflow 2) to compare planning depth, tool hand-offs, and memory control; Workflow 2 consistently performed better, and we implemented it with a ReAct-style agent so we could explicitly expose our custom tools, instruct the agent on when and how to call them, and use their outputs to guide subsequent reasoning.



(a) LangGraph Workflow 1 (b) LangGraph Workflow 2

Fig. 1: Two alternative LangGraph workflows.

Following further testing, we consolidated the runtime into the simplified operational graph (Workflow 3) shown here:

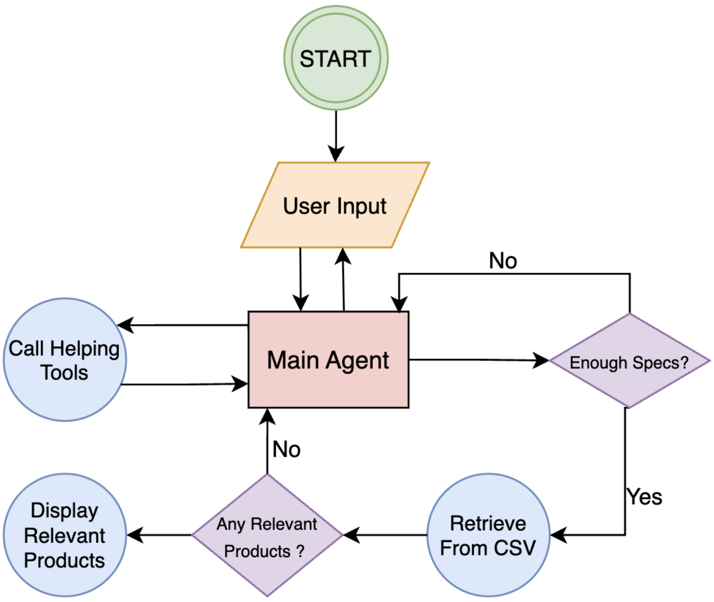


Fig. 2: Final LangGraph Workflow (Workflow 3)

User input flows into a main agent that runs ReAct think-act loops, checks whether the query contains enough specifications, retrieves and ranks candidates from the CSV catalog when it does, or loops back to elicit details and invoke helping tools when it does not; the agent then aggregates results, verifies that relevant products exist, and displays the recommendations.

```
================================== Ai Message ==================================

Hello! I'm Jarir's AI product advisor. How can I help you find what you're looking for today?


Update from node: agent
================================== Ai Message ==================================

I can definitely help you with that! To recommend the best laptop for you, could you tell me a bit more about what you'll be using it for? For example, will
it be for:

*   **Work or study?** (e.g., documents, presentations, browsing)
*   **Gaming?**
*   **Creative tasks?** (e.g., video editing, graphic design)
*   **Everyday use?** (e.g., watching movies, social media)

Also, do you have a particular budget in mind, or any preferences regarding brand, screen size, or operating system?


Update from node: agent
================================== Ai Message ==================================
Tool Calls:
  check_device (9a3f7f51-dbf2-4aec-a01b-511ffc85d7d2)
 Call ID: 9a3f7f51-dbf2-4aec-a01b-511ffc85d7d2
  Args:
    productName: MSI laptop


Update from node: tools
================================= Tool Message =================================
Name: check_device

Here are the top 5 MSI Gaming Laptops available:

1.  **Product Type:** Gaming Laptop
    **Series:** MSI Titan
    **CPU Model:** Intel Core Ultra 9 285HX
```

Fig. 3: Demo Agent Responses

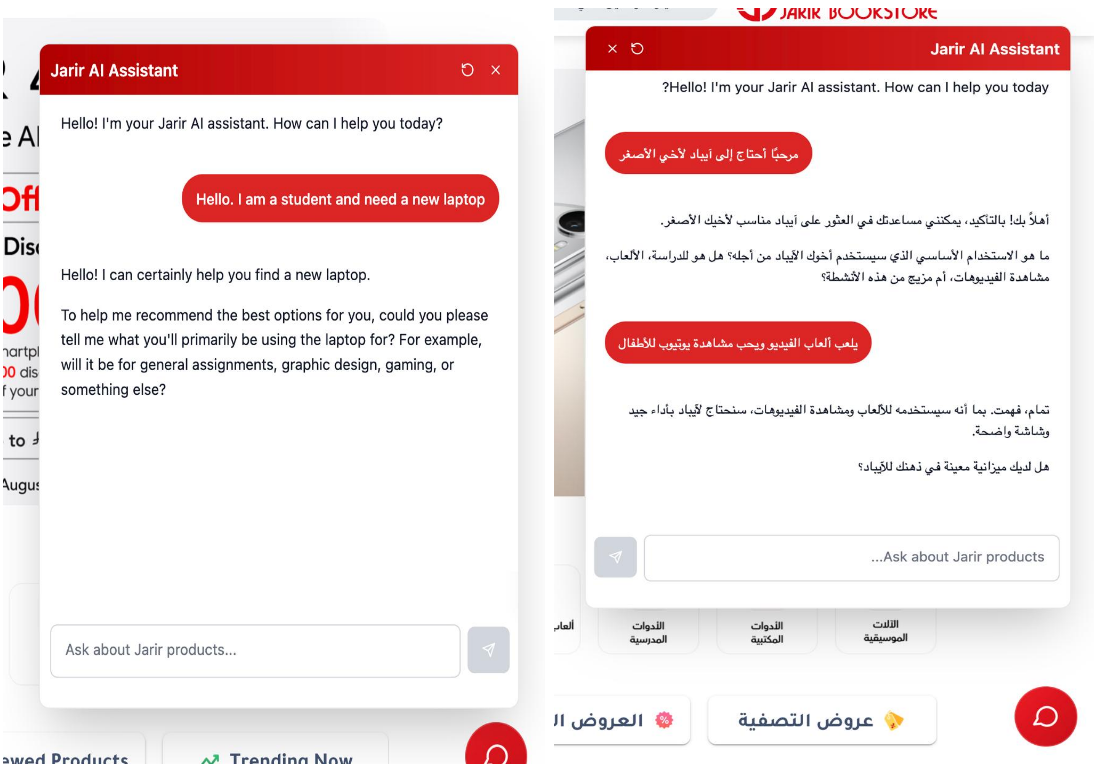
Our team developed and rigorously tested a suite of new retrieval and profiling tools to strengthen the agent's product-search capabilities. First, we built category-specific search modules that allow the assistant to quickly locate relevant items across Jarir's catalog. Next, we introduced two brand-and-model profiling utilities, giving the agent an at-a-glance view of which manufacturers and product lines Jarir carries so it can answer queries with greater precision. Finally, we ran a head-to-head comparison of two search strategies—one combining a vector-store similarity index with traditional query filters, and the other relying solely on direct queries without any vector store. This dual-track evaluation has provided valuable insights into price-range handling and helped us fine-tune our retrieval pipeline for more accurate, context-aware recommendations.

We also developed a slick frontend for our Jarir AI Salesman as a Firefox WebExtension. The chat widget itself is authored in React, styled with Tailwind CSS, and added seamlessly into any Jarir page. Under the hood, it communicates with our FastAPI backend over a persistent WebSocket connection, allowing near real-time exchanges without full-page reloads. The extension design handles authentication tokens, request batching, and automatic reconnection logic, so conversations never drop even if the user navigates between product pages.

The chat window itself has been engineered for maximum usability and minimal cognitive load. Users may drag the widget to any location within the viewport or minimize it to a compact icon when additional screen space is needed. Product recommendations are presented as horizontally scrollable cards, each card displaying key specifications, imagery, and pricing so that users can quickly scan options without wading through extensive text. Session-level memory preserves conversational context across visits. Smooth animations and responsive interactions further provide the impression of engaging with a live Jarir bookstore salesperson.

During the last week, our primary focus was on refining and enhancing the Firefox WebExtension for the Jarir AI Salesman to deliver a smoother, faster, and more visually appealing user experience. We improved the widget's design and responsiveness, optimized its load performance, and fine-tuned animations to make interactions feel more fluid. On the AI side, we revisited the system prompt and adjusted its structure to ensure the agent consistently follows our intended sales flow, responds in line with project objectives, and avoids unnecessary or off-topic information. These updates were complemented by backend stability improvements, ensuring the WebSocket connection remains persistent and uninterrupted during navigation across different product pages.

We also introduced significant enhancements to the product search functionality. A new price column was added to our CSV catalog, enabling the AI to factor in budget constraints when ranking and filtering recommendations. More importantly, we re-engineered our search algorithm into a full-queries progressive search method. This approach begins with an exact match using all provided specifications, then iteratively drops one specification at a time to broaden results while maintaining relevance. The system aggregates and deduplicates these results before returning the top 30 best-matching devices, balancing both specification fit and price suitability. Extensive testing confirmed that this new search method improves recall, delivers more relevant matches, and ensures consistent performance under varied user queries.



(a) English Chat Demo (b) Arabic Chat Demo

Fig. 4: Chat Extension UI

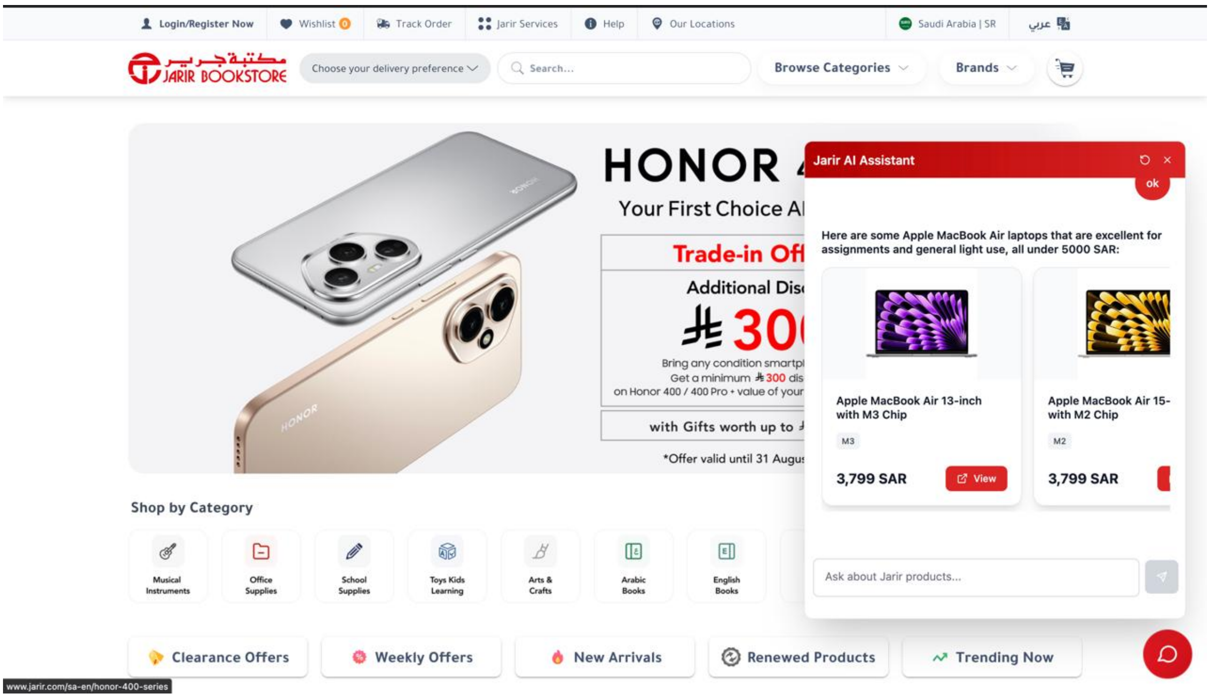


Fig. 5: Chat Extension Appearance in Jarir Website with card-style product recommendations

4 Project Timeline

Duration: June 29 – August 24

Week 1 (June 29 – July 5): Understanding & Setup

- Kickoff meeting with Dr. Salman to clarify project requirements
- Access and clean Jarir's product catalog (laptops, PCs)
- Define system architecture (LLM, tools, interface, vector DB)
- Assign individual responsibilities
- Setup environment: GitHub, virtual env, notebook, repo access

Week Deliverables:

- Requirements
- Initial dataset

Week 2 (July 6 – July 12): Data & Search Pipeline

- Develop project data scraper (products, URLs, categories, specs)
- Build hybrid search pipeline (FAISS + filters)
- Vectorize catalog descriptions using SentenceTransformer
- Test product matching with real user specs

Week Deliverables:

- Clean CSV product catalog
- Working hybrid search prototype
- Evaluation script for search accuracy

Week 3 (July 13 – July 19): LLM Agent Integration

- Integrate LangGraph or ReAct agent for product advice
- Refine prompt engineering for natural interactions
- Add product ranking explanation logic (LLM-generated)
- Run real query tests and iterate response structure

Week Deliverables:

- Functioning agent + hybrid tool
- Midway review and mentor feedback

Week 4 (July 20 – July 26): UI & Use Case Expansion

- Build a mockup frontend (Firefox extension)
- Add real-life use cases (e.g., gaming, business, student needs)
- Finalize personalization logic (preferences, budget, usage)

Week Deliverables:

- Interactive demo (UI or notebook)
- 10+ testable use cases
- Draft poster design

Week 5 (July 27 – August 3): Finetuning

- Finetune tools and system prompt for better agent responses in both English and Arabic
- Test and fix edge cases and errors
- Test full flow: input → recommendation → product link
- Refine AI agent tone and behavior
- Begin project poster designing

Week Deliverables:

- Improved working prototype

Week 6 (August 4 – August 10): WebExtension Optimization & Search Upgrade

- Enhance Firefox WebExtension design and responsiveness
- Optimize load performance and refine widget animations
- Improve backend stability with persistent WebSocket handling
- Revisit and restructure system prompt for consistent sales flow
- Introduce price-aware filtering in CSV catalog
- Implement full-queries progressive search with iterative specification relaxation

Week Deliverables:

- Optimized Firefox WebExtension with smoother UX
- Stable backend with uninterrupted session handling
- Advanced product search pipeline with budget integration

Week 7 (August 11 – August 17): Performance Optimization & Prompt Refinement

- Optimize codebase to improve speed and responsiveness of the extension
- Test multiple approaches to reduce latency and enhance chat fluidity
- Ensure smoother interactions within the chat interface
- Iteratively refine and test system prompts for higher response quality
- Improve agent's accuracy, relevance, and context-awareness in recommendations

Week Deliverables:

- Faster, more responsive extension
- Refined prompt engineering for improved agent reliability

Week 8 (August 17 – August 23): Project Poster Design Finalization

- Refined overall layout for clarity and visual balance
- Simplified text for readability at a glance
- Added visuals, icons, and workflow graphics
- Ensured consistent styling and formatting

Week Deliverables:

- Final project poster design

5 Challenges and Solutions

Throughout the project, the team successfully navigated several technical challenges by implementing targeted solutions. The primary obstacles and their resolutions are outlined below:

- **Data Accessibility**

  Challenge: The project's success depended on access to Jarir's comprehensive product catalog. In the absence of a direct API or official dataset, a method for reliable data acquisition was required.

  Solution: To overcome this, we developed a robust product scraper using Python libraries. This system was designed to be fault-tolerant and capable of extracting detailed product information, including specifications, prices, and images, which were then cleaned and structured into CSV catalogs for use by the AI agent.

- **Optimizing Search Relevance**

  Challenge: Early testing revealed the need for a search mechanism that could balance precision with recall, ensuring users receive relevant product options even when their queries are not a perfect match for a single product.

  Solution: We engineered a "full-queries progressive search method". This algorithm begins with a search for an exact match and then iteratively broadens the criteria by dropping one specification at a time. This ensures a wider range of relevant results, which are then aggregated and deduplicated to provide the most suitable options.

6 Resources Used

Technologies and Frameworks

- **LangGraph:** Used for building the agentic LLM workflow with memory and tool integration.
- **LangChain:** Provides the base for LLM orchestration and custom tool calls.
- **Google Generative AI (Gemini Pro):** Backend LLM used for generating responses and reasoning through product recommendations.
- **FastAPI:** Used in the backend service for serving model predictions and product responses.

Data Sources

- **Jarir Online Store:** A custom scraper was built to extract detailed product data (e.g., laptops, PCs, gaming devices), including price, URLs, images, categories, and specs.
- **CSV Catalogs:** Processed and cleaned versions of the scraped data were used to form the structured product catalog used in hybrid search.

Tools and Utilities

- `pandas, numpy, requests`: For data manipulation, processing, and scraping.
- `dotenv`: For environment variable management such as API KEYS.
- `json5, html, re`: For parsing and cleaning product descriptions.
- `matplotlib, altair` (optional): Used in plotting and visualization tasks if needed.
- `torch, scikit-learn`: Available for additional modeling or analysis (optional).

External References

- **SalesGPT GitHub Repository**: Used as reference for building the AI Salesman architecture.
- **KAUST Academy Materials**: The LangGraph enrichment session files were consulted for proper setup and state handling.

Collaboration Tools

- **GitHub**: Used for version control and team collaboration.
- **Overleaf**: Used for collaborative document writing and report writing.
- **PowerPoint/Canva**: Used for poster designing.