**Instant Assist Chatbot: AI Sales Agent for Automating Customer Queries**

**Logo, company name

Description automatically generated**

*By*

Muhammad Rohail Bukhari

B21F0085CS004

Wasif Khan

B21F0338CS025

Final Year Project Report

School of Computing Sciences

Pak-Austria Fachhochschule: Institute of Applied Sciences & Technology

Fall 2021 - Spring 2025



**Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology**

Instant Assist Chatbot: AI Sales Agent

for Automating Customer Queries

A Final Year Project Report Presented to

Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology

In partial fulfillment

of the requirement for the degree of

Bachelors of Science in Computer Science

By

Muhammad Rohail Bukhari

B21F0085CS004

Wasif Khan

B21F0338CS025

Fall 2021 - Spring 2025

Instant Assist Chatbot: AI Sales Agent for Automated Customer Queries

A Final Year Project Report submitted to the School of Computing Sciences as partial fulfillment of the requirement for the award of Degree of BS in Computer Science.

|  |  |
| --- | --- |
| Muhammad Rohail Bukhari | B21F0085CS004 |
| Wasif Khan | B21F0338CS025 |

**Academic Supervisor**

Dr. Ihtisham Ali

Assistant Professor, School of Computing Sciences

Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology

**Industry Supervisor**

Mr. Farrukh Shahzad

NKU Technologies, Lahore

18, Lane 2 Canal Rd, Zaman Park Lahore, 54000

Lahore

**Co-Supervisor**

Dr. Zafar Haider Janjua

Senior Research Fellow, Dundalk Institute of Technology

Ireland

**Final Approval**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

This final year project titles

Instant Assist Chatbot: AI Sales Agent for Automating Customer Queries

By

|  |  |
| --- | --- |
| *Muhammad Rohail Bukhari* | *B21F0085CS004* |
| *Wasif Khan* | *B21F0338CS025* |

under the supervision of their project supervisor and approved by the project evaluation committee, has been accepted by the Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology, Pakistan, in partial fulfillment of the requirements for the degree of BS in Computer Science.

Academic Supervisor:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Dr. Ihtisham Ali)

(School of Computing Sciences PAF-IAST)

Industry Supervisor:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Mr. Farrukh Shahzad)

(NKU Technologies, Lahore)

HoD / Chairman:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Dr. Babar Nazeer)

(School of Computing Sciences PAF-IAST)

**Declaration**

We, Muhammad Rohail Bukhari, B21F0085CS004, and Wasif Khan, B21F0338CS025, hereby declare that we have produced the work presented in this final year project report, during the scheduled period of study. We also declare that we have not taken any material from any source except referred to wherever due to that amount of plagiarism is within an acceptable range. It is further declared that we have developed this project and the accompanied report entirely on the basis of our personal efforts made under the sincere guidance of our supervisor. No portion of the work presented in this report has been submitted in support of any other degree or qualification of this or any other University or Institute of learning, if found we shall stand responsible.

Date: 25th April 2025



Signature:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Muhammad Rohail Bukhari**

B21F0085CS004.

Signature:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Wasif Khan**

B21F0338CS025.

**Certificate**

It is certified that Muhammad Rohail Bukhari, B21F0085CS004, and Wasif Khan, B21F0338CS025 have carried out all the work related to this project under my supervision at the Department of Computing Sciences, Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology and the work fulfills the requirement for the award of Bachelor's of Computer Science.

Date: 25th April 2025

Supervisor:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Ihtisham Ali

Assistant Professor

School of Computing Sciences

Head of Department:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Babar Nazeer

School of Computing Sciences

**DEDICATION**

This dissertation is dedicated to my parents, whose unshakeable support, encouragement, and sacrifices have been a constant source of strength throughout my academic life. Their unrelenting belief in my potential and their persistent motivation have been a determining factor in the successful completion of this work.

I also owe my heartfelt thanks to my supervisors and professors for their professional direction, considerate critique, and scholarly guidance. Their inputs have played a significant role in steering the direction and depth of the project.

This thesis is a testament to the discipline of computer science, its values of dedication, intellectual curiosity, and collaborative effort. It is with heartfelt gratitude that I recognize all those who have inspired and sustained me through this effort.

**ACKNOWLEDGEMENTS**

I would like to express my deepest gratitude to my academic supervisor Dr. Ihtisham Ali for the vigilant advice, encouragement and insightful feedback he has provided to me throughout the course of this project. This dissertation was greatly influenced by his deep understanding and insightful comments as well as his unwavering support in the direction and general quality of this project. This work has further benefited tremendously, both academically and personally through his advice.

I would also like to thank my co-supervisor Dr. Zafar Haider Janjua for his continuous support and technical advice. His expertise provided the keys to illumination in the difficult parts of the project. He significantly enriched the research process.

I would also like to formally give my utmost thanks to Sir Farrukh, my industrial supervisor, whose daily experience and earthly view on the matter has truly given a shape to this study. This project is not irrelevant because his advice helped bridge the often large gap between the theoretical academia and industrial practice.

I would thank my supervisors and everyone who has played some role, even indirectly, in making it fruitful. Through this I mean, friends and family who offered the emotional support and encouragement in order to get this done. I value your input immensely. Without this, the success of this project would not have happened without all of their collective guidance, mentorship and support. I am profoundly grateful for them taking time out of their busy schedules, showing patience and never ceasing in their commitment to furthering my professional and academic development.

**ABSTRACT**

Instant Assist Chatbot: AI Sales Agent for Customer Queries

This project aims to develop and test a Retrieval-Augmented Generation (RAG)-based assistant that provides precise, context-specific responses to product-related questions. Static FAQs or database-driven search utilities are typical systems that tend to struggle with scalability, responsiveness. As a solution to these shortcomings, the solution incorporates the LLaMA3-70B-8192 model (invoked through the Groq API) with all-mpnet-base-v2 embeddings (from huggingface) to provide coherent and context-sensitive responses. Product information is processed and stored in FAISS vector storage, allowing quick and relevant access. The system was evaluated with a structured dataset acquired from Kaggle (Gadgets 360) and its performance assessed with a range of metrics, including Flesch-Kincaid readability, BLEU, ROUGE, response time, F1-score, and cosine similarity. By coupling dense vector search with state-of-the-art language modeling, the assistant presents a scalable and feasible solution to real-world problems in e-commerce websites and digital customer service systems.

**Keywords:** Retrieval-Augmented Generation; FAISS; Semantic Search; LLaMA3; Product Query Resolution; AI Chatbot; Natural Language Processing

**Table of Contents**

[Chapter 1: Project Vision 1](#_Toc19335)

[1.1 Problem Statement 1](#_Toc330)

[1.2 Business Opportunity 1](#_Toc20400)

[1.3 Objectives 1](#_Toc32546)

[1.4 Project Scope 2](#_Toc9168)

[1.5 Constraints 2](#_Toc21481)

[1.6 Feasibility 3](#_Toc929)

[1.6.1 Technical Feasibility 3](#_Toc9500)

[1.6.2 Resource Feasibility 3](#_Toc7347)

[1.6.3 Time Feasibility 4](#_Toc6151)

[1.6.4 Market Feasibility 4](#_Toc23859)

[1.7 Stakeholders Description 4](#_Toc4076)

[1.7.1 Stakeholders Summary 5](#_Toc14840)

[1.7.2 Key High-Level Goals and Problems of Stakeholders 5](#_Toc23133)

[Chapter 2: Software Requirement Specification 7](#_Toc11777)

[2.1 List of Features 7](#_Toc9034)

[2.2 Functional Requirements 7](#_Toc1730)

[2.3 Non-Functional Requirements 8](#_Toc5777)

[2.4 Use Cases/ Use Case Diagram 9](#_Toc27632)

[2.5 Software Development Plan 16](#_Toc20222)

[Chapter 3 System Overview 18](#_Toc8172)

[3.1 Architectural Design 18](#_Toc13009)

[3.2 Data Design 19](#_Toc12269)

[3.2.1 Entity-Relationship Diagram (ERD) 19](#_Toc9627)

[3.2.2 Data Flow Diagram (DFD) 20](#_Toc19447)

[3.3 Domain Model 21](#_Toc15553)

[3.3.1 Class Diagram 23](#_Toc18095)

[3.4 Activity Diagram 24](#_Toc21327)

[3.5 State Transition Diagram 33](#_Toc8987)

[3.6 Sequence Diagram 34](#_Toc21295)

[3.7 Component Diagram 42](#_Toc2434)

[3.8 System Architecture 44](#_Toc16867)

[3.9 Mockups 45](#_Toc3602)

[Chapter 4: System Implementation 49](#_Toc17005)

[4.1 Technology Stack 49](#_Toc446)

[4.2 Development Environment Setup 51](#_Toc30797)

[4.3 System Modules and Components 52](#_Toc15118)

[4.4 Database Implementation 53](#_Toc29613)

[4.5 APIs and External Integrations (if applicable) 53](#_Toc10907)

[4.6 Dataset Selection and Preprocessing 54](#_Toc31011)

[4.7 Model Selection and Training 55](#_Toc12930)

[Chapter 5: System Testing & Deployment 57](#_Toc23215)

[5.1 Testing Approach (Manual, Automated) 57](#_Toc23481)

[5.2 Unit Testing 58](#_Toc18588)

[5.3 Integration Testing 59](#_Toc23518)

[5.4 System Testing 60](#_Toc22152)

[5.5 User Acceptance Testing (UAT) 61](#_Toc17066)

[5.6 Performance Testing 62](#_Toc15761)

[Chapter 6: Results and Discussion 62](#_Toc24343)

[6.1 Evaluation and Results 62](#_Toc29084)

[6.1.1 Comparison with Initial Objectives 63](#_Toc25525)

[6.1.2 User Feedback and Usability Testing 64](#_Toc25119)

[6.1.3 Performance Evaluation 65](#_Toc13231)

6.1.4 Evaluation Graphs …………………………………………………………………….68

[6.1.5 Challenges and Limitations 73](#_Toc14276)

[Chapter 7: Conclusion and Future Work 74](#_Toc25508)

[7.1 Conclusion 74](#_Toc4184)

[7.2 Future Enhancements 75](#_Toc12078)

[References 77](#_Toc520)

[Bibliography 79](#_Toc24794)

**List of Figures**

[Figure 1 Product Recommendation Use Case Diagram 11](#_Toc602)

[Figure 2 Price Enquiry Use Case Diagram 12](#_Toc17642)

[Figure 3 General Queries and Support Use Case Diagram 13](#_Toc426)

[Figure 4 Product Comparison Use Case Diagram 14](#_Toc16866)

[Figure 5 Search by Products Use Case Diagram 15](#_Toc27604)

[Figure 6 Summarize Product Information Use Case Diagram 16](#_Toc24455)

[Figure 7 Handle Customer Errors and Clarify Queries Use Case Diagram 17](#_Toc9003)

[Figure 8 Software Development Plan for Instant Assist Chatbot 18](#_Toc25449)

[Figure 9 Architectural Diagram for Chatbot 20](#_Toc28499)

[Figure 10 ERD for Logical Structure of Database 20](#_Toc21726)

[Figure 11 . Data Flow Diagram for Chatbot Interactions 21](#_Toc10753)

[Figure 12 . Domain Model Representation 22](#_Toc25806)

[Figure 13 . Class Diagram for RAG-based Chatbot 25](#_Toc17805)

[Figure 14 . Register Activity Diagram 26](#_Toc647)

[Figure 15 Login Activity Diagram 27](#_Toc7261)

[Figure 16 Forget Activity Diagram 28](#_Toc29341)

[Figure 17 Product Recommendation Activity Diagram 29](#_Toc10969)

[Figure 18 Price Enquiry Activity Diagram 30](#_Toc19825)

[Figure 19 Search by Product Features Activity Diagram 31](#_Toc11664)

[Figure 20 Summary of Product Features Activity Diagram 32](#_Toc7725)

[Figure 21 General Queries and FAQs Activity Diagram 35](#_Toc17434)

[Figure 22 Handling Customer Errors Activity Diagram 36](#_Toc5464)

[Figure 23 State Transition Diagram for Input Processing & Response Generation 37](#_Toc22608)

[Figure 24 Register Sequence Diagram 38](#_Toc18856)

[Figure 25 Login Sequence Diagram 39](#_Toc1469)

[Figure 26 Forget Sequence Diagram 40](#_Toc31511)

[Figure 27 Product Recommendation Sequence Diagram 41](#_Toc5455)

[Figure 28 Price Enquiry Sequence Diagram 42](#_Toc3128)

[Figure 29 Search by Product Features Sequence Diagram 43](#_Toc24914)

[Figure 30 General Queries and FAQs Sequence Diagram 44](#_Toc9676)

[Figure 31 Handling Customer Errors Sequence Diagram 46](#_Toc13727)

[Figure 32 Component Diagram for Representing Functional Units of Chatbot 48](#_Toc14481)

[Figure 33 Five Layered System Architecture for Chatbot 49](#_Toc17)

[Figure 34 Screen 1 - Login/Get Started 50](#_Toc20826)

[Figure 35 Screen 2 - Registration 50](#_Toc25913)

[Figure 36 Screen 3 - Sign in 51](#_Toc29540)

[Figure 37 Screen 4 - Main UI 51](#_Toc3411)

[Figure 38 Screen 5 - Start Screen Workflows 52](#_Toc27880)

[Figure 39 Screen 6 - Permanent Deletion of Account 52](#_Toc15736)

[Figure 40 . Evaluation Metrics for Chatbot Analysis 73](#_Toc23021)

[Figure 41 BLEAU and Rogue Score Distribution Graph 74](#_Toc4373)

[Figure 42 Cosine Similarity Graph for Analyzing Semantic Relevance 75](#_Toc18784)

[Figure 43 Latency Distribution Graph for Analyzing Response Time 76](#_Toc26523)

[Figure 44 Flesch-Kincaid Graph for Response Legibility 77](#_Toc14438)

**List of Tables**

Table 1 Stakeholders Roles, Goals, and Concerns Summary …………………………………… 5

Table 2 Comparison of Functional and Non-Functional Requirements ………………….……... 9

Table 3 Assessment of Project Objectives Against Achieved Outcomes ………..……....…….. 64

Table 4 Evaluation Metrics and System Performance Summary ………...……………………. 67

**List of Acronyms**

| **Abbreviation** | **Full Form** |
| --- | --- |
| **AI** | Artificial Intelligence |
| **API** | Application Programming Interface |
| **FAISS** | Facebook AI Similarity Search |
| **RAG** | Retrieval-Augmented Generation |
| **LLM** | Large Language Model |
| **NLP** | Natural Language Processing |
| **UI/UX** | User Interface / User Experience |
| **UAT** | User Acceptance Testing |
| **ROUGE** | Recall-Oriented Understudy for Gisting Evaluation |
| **BLEU** | Bilingual Evaluation Understudy |
| **F1-Score** | Harmonic Mean of Precision and Recall |
| **IDE** | Integrated Development Environment |
| **VS Code** | Visual Studio Code |
| **CORS** | Cross-Origin Resource Sharing |
| **USD** | United States Dollar |
| **INR** | Indian Rupee |
| **ERD** | Entity-Relationship Diagram |
| **DFD** | Data Flow Diagram |

# **Chapter 1: Project Vision**

## Problem Statement

This section outlines the common challenge businesses face in handling repetitive customer queries. It accentuates how manual handling creates inefficiencies, and how automation with AI provides a scalable alternative.

Companies often have to respond to repetitive customer queries, such as queries surrounding items that are in stock, order status, return processes, technical assistance, etc. This creates operational inefficiencies as well as added costs because manually resolving such questions requires a significant amount of time and manpower. These processes can be made streamlined by enabling customer interactions with automated chatbots using artificial intelligence. This helps reduce significantly the use of human-led processes ultimately leading to cost savings on staffing while enabling workers to focus more on more complex tasks. In addition, it enhances customer service with faster problem-solving, resulting in a more streamlined customer journey.

## Business Opportunity

This section addresses increased demand for effective customer care systems. It highlights how business organizations gain a competitive advantage from using artificial intelligence-powered solutions by cutting costs without compromising service quality.

With growing customer queries, organizations are faced with providing service levels affordably without compromising service quality in a highly competitive business environment. AI-powered automation not only reduces response times and operational costs significantly but also helps businesses maintain high customer service standards.

The Instant Assist Chatbot caters to this demand, as it provides instant, personalized, and real-time responses about product details, price, product comparison, etc. Not only does this enhance the customer experience, but it also allows companies to scale their support capabilities without an equal increase in staffing.

## Objectives

This section defines the core purpose of the project. It focuses on leveraging RAG architecture to automate customer support and enhance response accuracy using semantic retrieval.

Significant advancements in Natural Language Processing (NLP) have been made possible by the aid of large language models (LLMs). The LLama series models[1], and other large language models demonstrate impressive language and knowledge mastery, surpassing human benchmark levels in multiple evaluation benchmarks[2,3,4].

However, large language models also exhibit numerous shortcomings. They often fabricate facts[5] and lack knowledge when dealing with specific domains or highly specialized queries[6]. Retrieval-Augmented Generation (RAG) is a well-known method that can reduce hallucination and improve output quality, especially when generating the correct output requires access to external knowledge sources [7].

The key aim of this project is to improve the efficiency of customer service using a RAG-based chatbot.

Specific objectives include:

* Create a Retrieval-Augmented Generation (RAG) application with the all-mpnet-base-v2 model for llama3-70B-8192 and vector embeddings through Graph-Relational Object Queries (Groq) for response generation.
* Automate answers to frequently asked product-related questions.
* Deliver accurate, contextually-aware answers using semantic retrieval.
* Improve customer experience with faster and intelligent interactions.

## Project Scope

Scope outlines the functional limits of chatbot technology. It addresses the essential capabilities like FAQs handling and providing context-sensitive, real-time assistance using artificial intelligence.

This project intends to create a RAG-based chatbot for managing customer interactions effectively. It would utilize semantic retrieval through FAISS and large-scale LLM responses to provide real-time, context-sensitive answers.

The key focus areas include:

* Handling of frequently asked questions and sales-related inquiries.
* Delivering seamless customer experiences through real-time processing and response.

## Constraints

This section indicates the possible constraints that can affect project implementation. These are technical constraints, limited manpower, and context length of models.

* The accuracy and relevance of embedded knowledge are responsible for ensuring quality in results.
* Limited human resources (two developers) might affect the development speed.
* Context window limit of 6K tokens, which bounds how much retrieved context and prompt can be included.

## Feasibility

This section assesses the AI Sales Assistant project's viability from a number of perspectives to make sure it is feasible within the project's time frame, supported by resources, and technically feasible. It also assesses the market conditions and the potential for successful adoption.

## Technical Feasibility

Here, the use of pretrained models and scalable vector search ensures implementation is technically sound. Integration with Groq and FAISS supports robust performance and reliability.

Utilizing the pre-trained all-mpnet-base-v2 for embeddings and the powerful llama3-70B-8192 model through Groq ensures technical feasibility. FAISS provides scalable and efficient vector search capabilities.

## Resource Feasibility

This part confirms that existing tools and technical skills are sufficient for successful development. Open-source libraries and available APIs reduce dependency on external resources.

Open-source tools and APIs from HuggingFace, Groq, and FAISS ensure accessibility. The current team has the technical skillset to implement and deploy the system.

## Time Feasibility

The outlined timeline is realistic and structured for phased delivery. It allows focused efforts across data preparation, system integration, testing, and deployment.

The project timeline of 8 months is realistic and provides enough time to successfully complete the AI Sales Assistant, given the defined scope and objectives.

The revised project plan includes:

* Data acquisition & pre-processing: 2 months
* FAISS vector store implementation: 1 month
* Integration with llama3-70B-8192 via Groq API: 2 months
* Testing & refinement: 2 months
* Deployment: 1 month.

## Market Feasibility

This section validates the market demand for intelligent support systems. It emphasizes the solution’s potential to address rising customer service needs across industries.

The demand for intelligent customer service systems continues to grow, especially in retail, e-commerce, and tech support. The RAG-based system provides an edge by combining retrieval with generation for high-accuracy support.

## Stakeholders Description

This section identifies the primary groups involved in or impacted by the project. It includes customers, businesses, developers, and academic supervisors.

The following groups are interested in the AI Sales Assistant project and its results:

* Customers
* Clients
* Project Team
* Academic Supervisors.

## Stakeholders Summary

This part provides an overview of stakeholder roles and interests. It outlines the expected benefits and responsibilities of each party in the project.

* End users: Customers will benefit from fast and relevant responses with accurate product recommendations and support.
* Businesses: Companies will improve support efficiency and reduce costs.
* Project Team is responsible for implementing the RAG pipeline.
* The academic supervisors provide guidance, mentorship, and resources to ensure the project is completed successfully.

Table 1 outlines the key stakeholders involved in the development and deployment of an AI chatbot system, along with their respective roles, objectives, and concerns.

|  |  |  |  |
| --- | --- | --- | --- |
| **Stakeholder** | **Role** | **Goal** | **Concern** |
| End Users | Use the chatbot for product queries | Get fast, relevant responses | Misunderstood or vague responses |
| Businesses | Deploy chatbot on platforms | Reduce support cost, improve customer satisfaction | Integration challenges with existing systems |
| Project Team | Build and test the system | Deliver functional, efficient AI chatbot | Managing LLM latency and resource constraints |
| Academic Supervisors | Provide guidance and approval | Ensure academic and technical rigor | Timeline alignment and technical depth of the solution |

Table 1. Stakeholders Roles, Goals, and Concerns Summary

## Key High-Level Goals and Problems of Stakeholders

Here, the goals and concerns of each stakeholder group are detailed. It ensures alignment with user expectations, business needs, technical feasibility, and academic requirements.

The primary goals and potential concerns for each stakeholder are as follows:

* End Users:

**Goals**:

Receive fast, accurate, and helpful product recommendations and support.

Have a seamless and enjoyable experience interacting with the AI chatbot.

**Concerns**:

The chatbot might misunderstand their queries or provide irrelevant responses.

They may encounter difficulty in getting personalized responses based on their needs.

* Businesses:

**Goals**:

Improve customer satisfaction and retention by providing efficient customer support.

Reduce operational costs by automating routine customer queries.

Scale their operations without significantly increasing customer service staff.

**Concerns**:

Integration with existing systems (e.g., websites, customer databases) could be challenging.

* Project Team:

**Goals**:

Build an effective AI Sales Assistant with llama3-70B-8192 via Groq.

Ensure the system functions reliably and is easy to integrate with business platforms.

**Concerns**:

Efficiently executing GROQ based graph queries and downstream LLM inference within a 6K‑token context window without introducing unacceptable response latency.

* Academic Supervisors:

**Goals**:

Ensure the project meets academic standards and demonstrates technical proficiency.

Support the development of new skills for the students and help them complete the project successfully.

**Concerns**:

Delays in the project timeline could affect academic deadlines.

* End Customers of Businesses:

**Goals**:

Benefit from better, faster customer service and support when interacting with businesses.

**Concerns**:

If the chatbot cannot effectively resolve queries, customer satisfaction may decrease.

They may have concerns about privacy and data security when interacting with the AI.

# **Chapter 2: Software Requirement Specification**

## List of Features

This section outlines the core functionalities of the AI Sales Assistant. Each feature enhances user interaction through intelligent responses tailored to various query types.

The AI Sales Assistant will offer the following core features:

* Product Recommendation:

Provides personalized product suggestions based on customer preferences, search history, and keywords.

* Price Inquiry:

Enables customers to inquire about product prices, including details on discounts, promotions, and available offers.

* Product Comparison:

Allows customers to compare multiple products side by side, showing key details such as price, specifications, and availability.

* Search by Product Features:

Allows customers to search for products based on specific features (e.g., lightweight laptops, waterproof phones).

* Summarize Product Information:

Provides concise summaries of product descriptions for quick customer reference, focusing on key features and benefits.

## Functional Requirements

This section describes the essential capabilities the system must fulfill to perform effectively. These include semantic retrieval, response generation, and intelligent fallback mechanisms.

* The system must embed product documents into dense vectors using the all‑mpnet‑base‑v2 model and index them in FAISS for fast similarity search. FAISS (short for Facebook AI Similarity Search) is a library that provides efficient algorithms to quickly search and cluster embedding vectors[8].
* The system must execute semantic retrieval over the FAISS index to fetch the top‑k most relevant product documents for any given user query.
* The system must leverage llama3‑70B‑8192 via Groq (Graph‑Relational Object Queries) to generate context‑aware, coherent responses conditioned on the retrieved documents (RAG pipeline). Llama 3.0 70B on Groq offers a balance of performance and speed as a reliable foundation model that excels at dialogue and content-generation for tasks requiring smaller context windows[9].
* Customer requests for specifications or prices need to be understood by semantically matching query embeddings against the FAISS index and retrieving the appropriate fields.
* The ranker should rank and prioritize recommendations according to embedding similarity scores so that the most suitable products appear at the top.
* It should be able to respond to multi-product queries by accessing each product's documents in parallel and then aggregating them prior to response generation.
* The system should gracefully fall back to a generic response (e.g., “I’m learning about this product right now”) when retrieval confidence drops below a threshold determined by a configuration.

## Non-Functional Requirements

* This category outlines performance-oriented requirements like scalability, usability, fault tolerance, and effective use of resources in ensuring reliable performance.
* All product documents need to be embedded into dense vectors with all-mpnet-base-v2 and indexed within FAISS in order to provide quick vector search and response generation.
* The system would need to accommodate scalable embedding storage within FAISS, enabling incremental updating efficiently when more products are added.
* The system needs to perform semantic retrieval against the FAISS index to retrieve the top-k most related documents for any given user query with minimal latency.
* The system uses llama3‑70B‑8192 through Groq to produce context­sensitive, coherent responses conditioned upon the documents retrieved.
* It should recover gracefully from retrieval or generation failures, reverting to a generic response or a cached one with minimal disruption.
* Each chatbot interface should have an intuitive, user-centric design, providing users with direction through clarifications or suggestions where they provide unclear input. The user input error handling must be robust, soliciting corrections where necessary or otherwise using semantic matching to deduce intent.
* Computational resources must be utilized efficiently—batching embedding requests and caching frequent retrieval results—to operate smoothly even on limited hardware.

Table 2 presents a comparison between functional and non-functional requirements for a system designed to handle product-related queries.

|  |  |
| --- | --- |
| **Functional Requirements** | **Non-Functional Requirements** |
| Perform semantic search using FAISS | Maintain latency below 1.5 seconds |
| Generate answers using LLaMA3 via Groq | Ensure high response readability (Flesch-Kincaid ≥ 80) |
| Handle multi-product queries | System should scale with increasing product documents |
| Provide fallback response if confidence is low | Use caching and batching to optimize performance |
| Rank product matches based on embedding similarity | Ensure consistent and user-friendly UI |

Table 2. Comparison of Functional and Non-Functional Requirements

## Use Cases/ Use Case Diagram

This section presents real-world scenarios depicting user interaction with the system. Each use case illustrates a specific task the chatbot performs based on customer input.

**Use Case 1: Product Recommendation**

Figure 1 illustrates how the chatbot offers tailored product suggestions. It considers user preferences and product metadata to generate accurate results.

Actors:

* Customer: Requests product recommendations.
* AI Chatbot: Analyzes input and provides recommendations.
* System: Stores product data and assists in retrieving recommendations

Goal:

* Provide personalized product recommendations based on customer preferences.

Steps:

* The Customer asks the AI Chatbot for product recommendations.
* The AI Chatbot analyzes the customer's input using natural language processing (NLP).
* The AI Chatbot retrieves relevant product data from the System.
* The AI Chatbot presents a list of recommended products to the Customer, including details such as name, price, and description

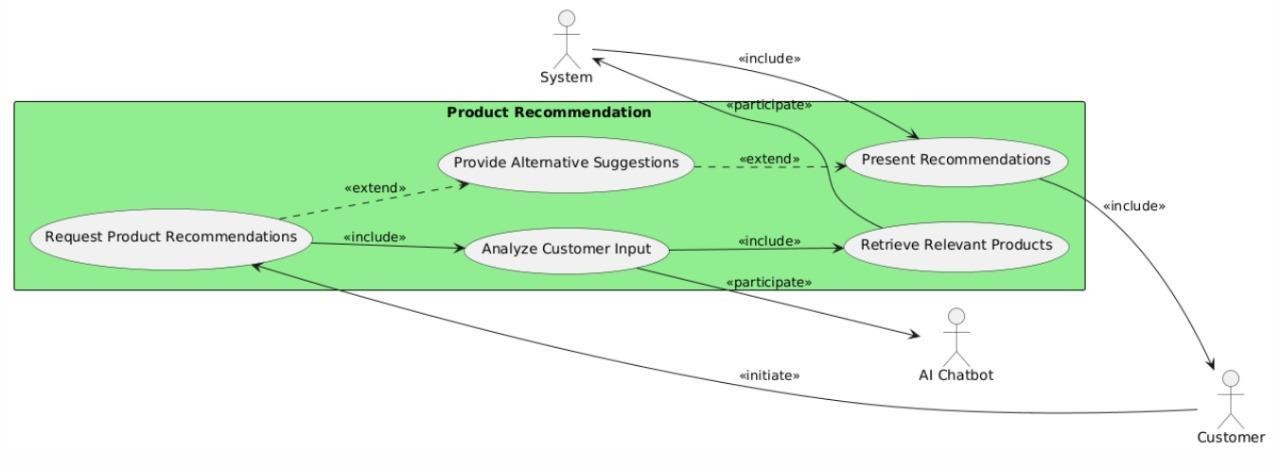


Figure 1. Product Recommendation Use Case Diagram

**Use Case 2: Price Enquiry**

Figure 2 depicts the chatbot’s ability to handle product pricing questions. It fetches and communicates real-time cost data, including discounts or deals.

Actors:

* + - Customer: Inquires about product pricing.
    - AI Chatbot: Processes the inquiry and retrieves pricing information.
* System: Stores and provides product pricing data.

Goal:

* Provide accurate pricing information for specific products.

Steps:

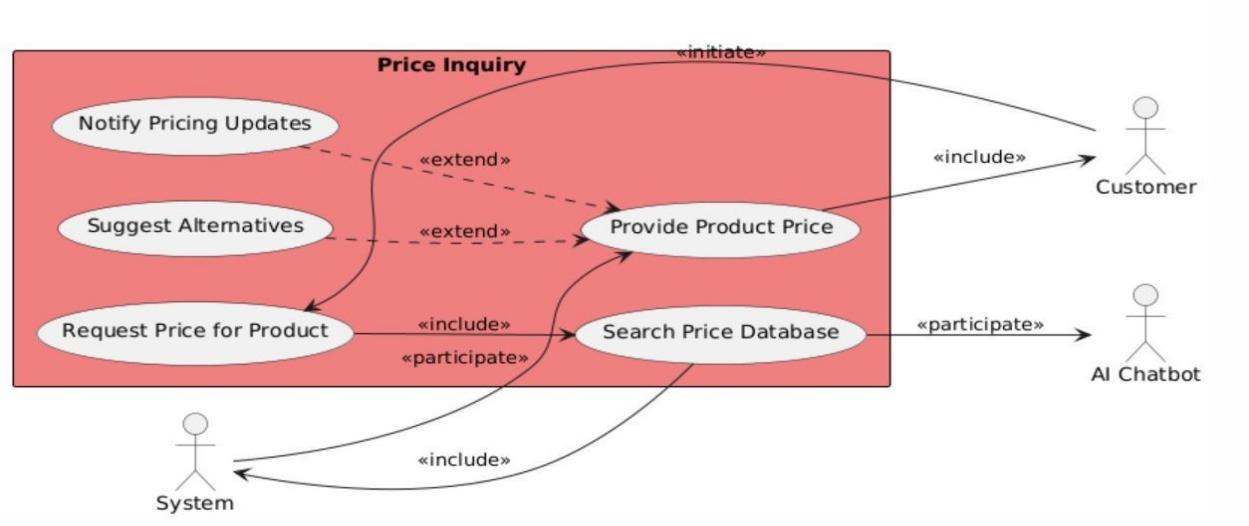
* The Customer asks the AI Chatbot for the price of a specific product.
* The AI Chatbot queries the System for pricing data.
* The System retrieves and sends the price to the AI Chatbot.
* The AI Chatbot presents the pricing details, including discounts or promotions, to the customer.

Figure 2. Price Enquiry Use Case Diagram

**Use Case 3: General Queries and Support**

Figure 3 shows the process of resolving FAQ-style inquiries. The chatbot uses stored knowledge to offer quick, relevant assistance.

Actors:

* + Customer: Asks general questions.
  + AI Chatbot: Processes and answers queries.
  + System: Stores and retrieves data

Goal:

* Provide answers to general queries

Steps:

* The Customer asks a general question via the AI Chatbot
* The AI Chatbot identifies the topic and searches the System for relevant FAQ data.
* The AI Chatbot retrieves and presents the answer to the Customer in a concise format.

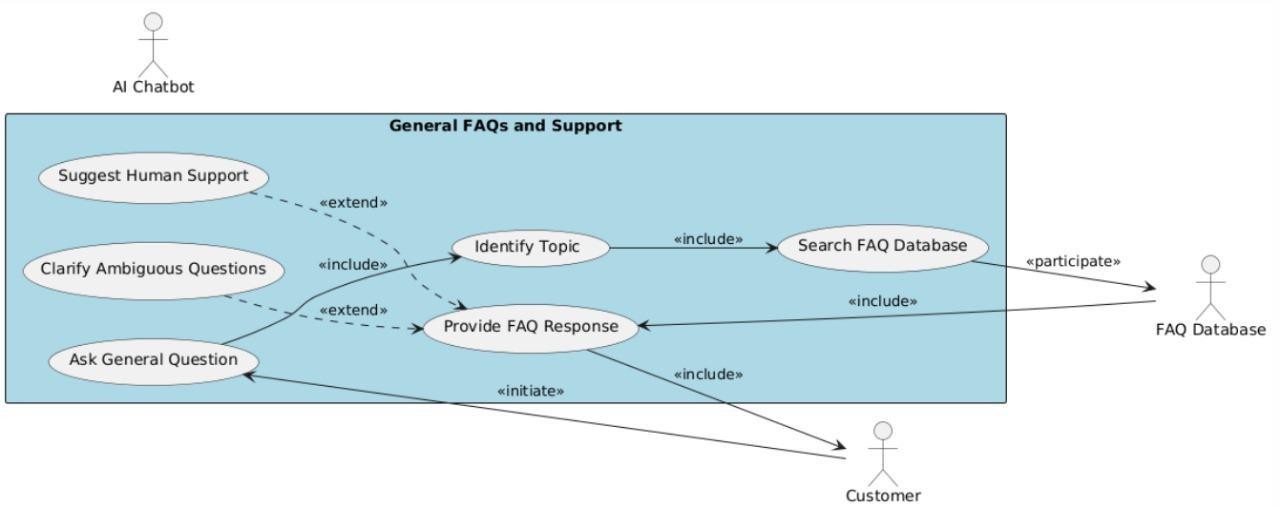


Figure 3. General Queries and Support Use Case Diagram

**Use Case 4: Product Comparison**

Figure 4 illustrates the chatbot’s functionality to compare multiple products. Users receive a clear, side-by-side comparison of specifications and features.

Actors:

* + Customer (primary)
  + AI Chatbot

Goal:

* Provide the Customer with a side-by-side comparison of two or more products to assist in making an informed decision.

Steps:

* The Customer requests a comparison between two or more products
* The AI Chatbot retrieves details such as price, specifications.
* The AI Chatbot displays a side-by-side comparison of the products to the Customer.

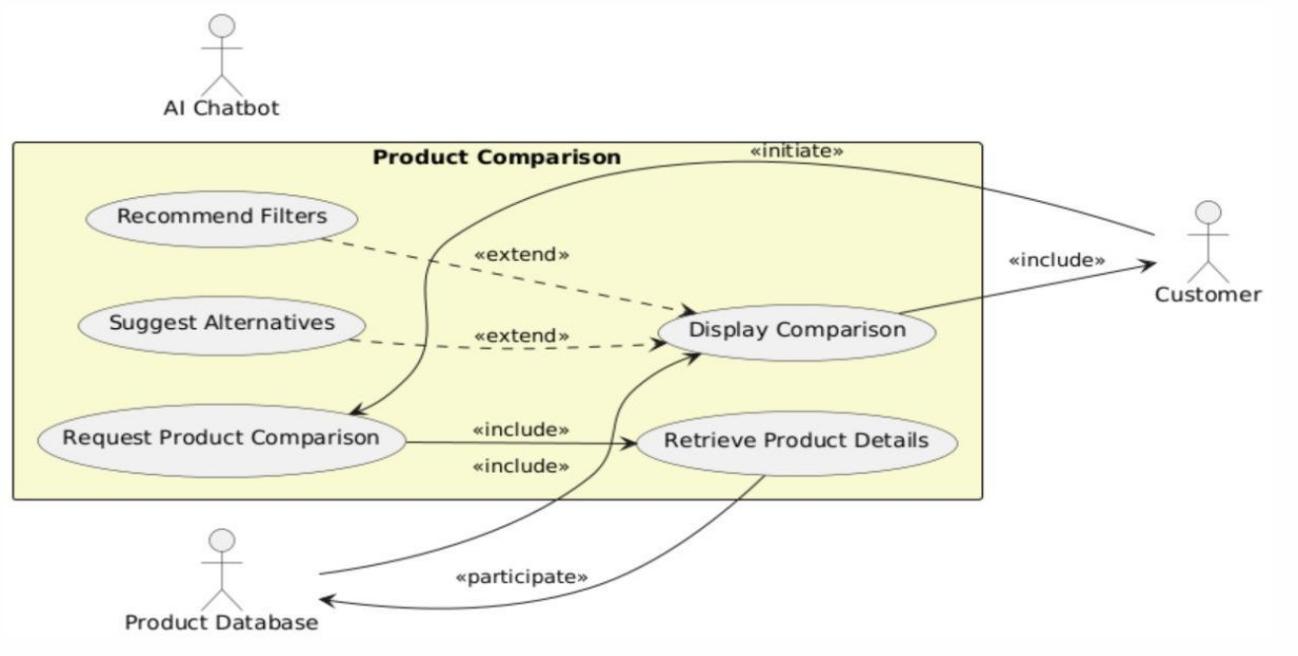


Figure 4. Product Comparison Use Case Diagram

**Use Case 5: Search by Product Features**

Figure 5 depicts how the user can filter products by specifying desired features. The chatbot matches those preferences using semantic vector search.

Actors:

* Customer: Searches for products based on specific features.
* AI Chatbot: Processes the request and retrieves matching products.
* System: Stores detailed product data and facilitates the search.

Goal:

* Help customers find products based on desired features

Steps:

* The Customer specifies a desired feature (e.g., "lightweight laptop").
* The RAG based chatbot retrieves and presents a list of matching products to the Customer.

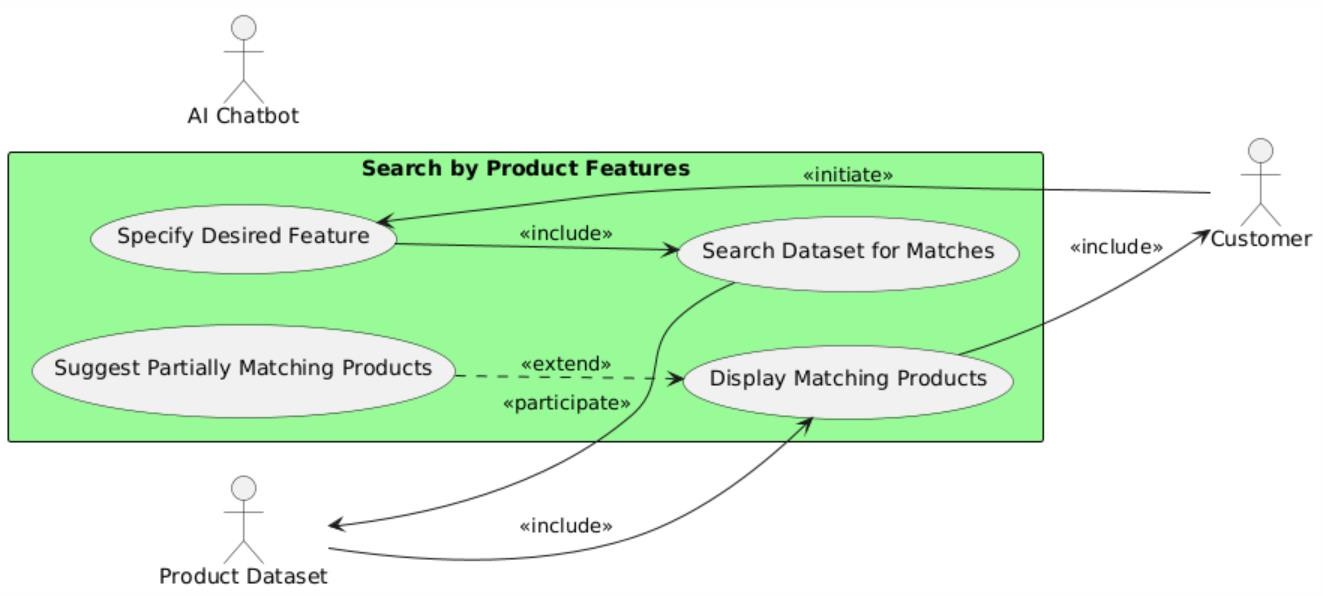


Figure 5. Search by Products Use Case Diagram

**Use Case 6: Summarize Product Information**

Figure 6 shows the chatbot's ability to deliver brief, context-rich product summaries. This aids users in understanding key features efficiently.

Actors:

* Customer (primary)
* AI Chatbot

Goal:

* Provide the customer with a quick and concise summary of a product.

Steps:

* The Customer asks the chatbot for a brief overview of a specific product.
* The chatbot encodes the product query into a dense vector with all‑mpnet‑base‑v2 and performs a top‑k semantic search against the FAISS index to retrieve the product’s description documents.
* Retrieved document snippets are passed as context into llama3‑70B‑8192 via Groq, which generates a concise summary of the product’s key features.
* The Customer reviews the RAG‑generated summary and can ask follow‑up questions, triggering additional vector retrieval and context‑aware responses as needed.

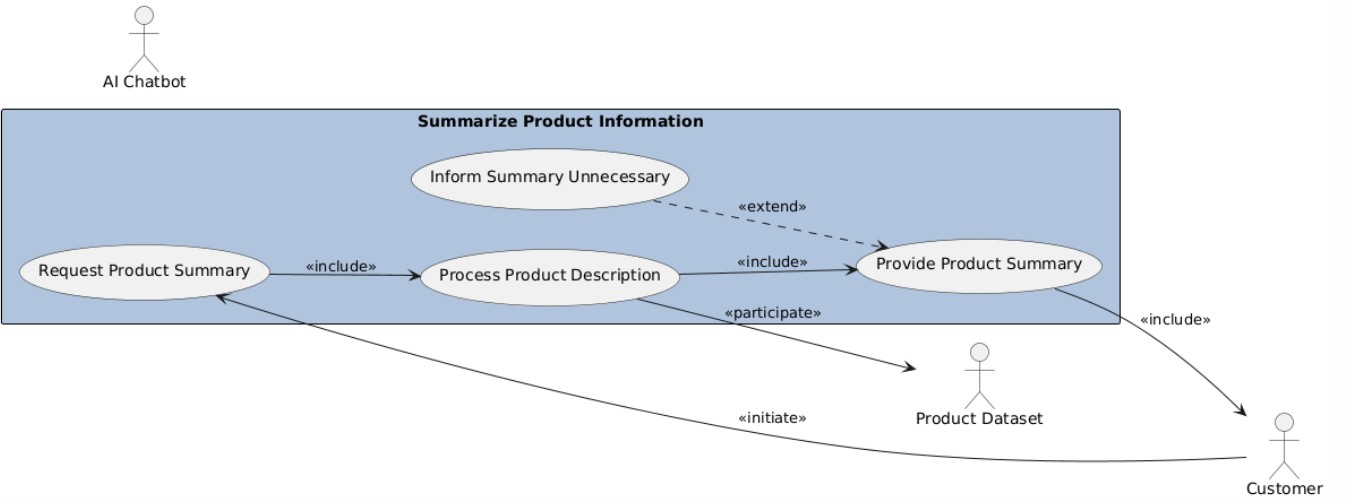


Figure 6. Summarize Product Information Use Case Diagram

**Use Case 7: Handle Customer Errors and Clarify Queries**

Figure 7 illustrates the system’s error-handling capability when faced with vague input. The chatbot prompts users for clarification to maintain accuracy.

Actors:

* Customer (primary)
* AI Chatbot

Goal:

* Handle ambiguous or unrecognized customer queries by asking for clarification and providing suggestions

Steps:

* + The Customer inputs an ambiguous or incorrect query (e.g., "bluetooth laptop").
  + The AI Chatbot recognizes the error in the query and asks the Customer for clarification or more specific details.

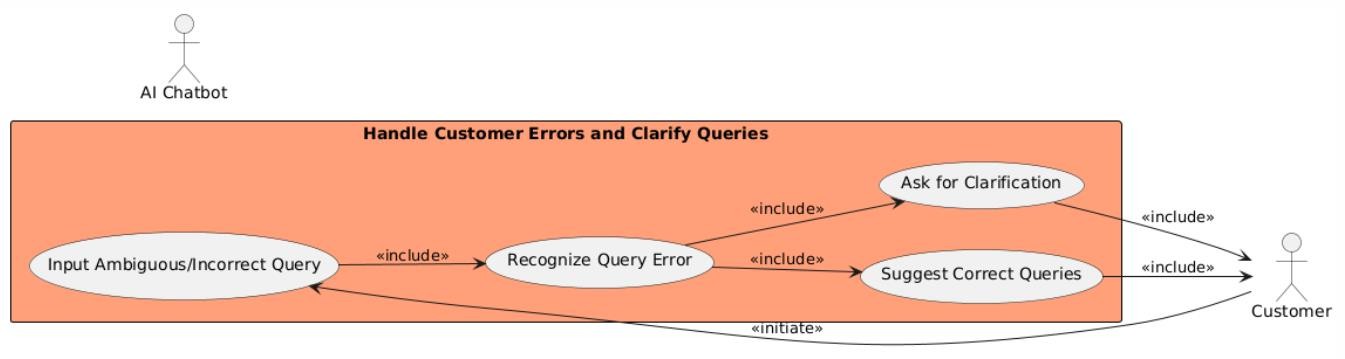


Figure 7. Handle Customer Errors and Clarify Queries Use Case Diagram

## Software Development Plan

The software development and management (SDM) practice helps organizations to ensure that their software products are developed methodically and delivered in accordance with the stakeholders’ requirements [10].

Figure 8 illustrates the software development plan i.e. a structured timeline and methodology for system implementation. It breaks down stages from data preprocessing to final deployment.

This software development plan shows the whole process of making and using the AI Sales Assistant chatbot. The first step is to incorporate the product dataset using HuggingFace Transformers' all-mpnet-base-v2 model. FAISS is then used to store and index these demse embeddings, making it easy to quickly and accurately find product information.

After that, the Groq API is used to add the LLaMA3-70B-8192 model for generating natural language responses. Moreover, a web application in react is made so that people can talk to the chatbot. Then the system is tested to evaluate the quality of information retrieval and to refine prompts for better accuracy. Once the system meets the required performance, it proceeds to final deployment, making the chatbot accessible to end users.

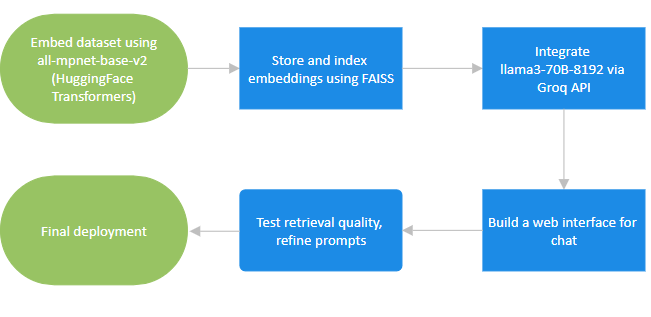


Figure 8. Software Development Plan for Instant Assist Chatbot

# **Chapter 3 System Overview**

## 3.1 Architectural Design

By decomposing complex RAG systems into independent modules and specialized operators, it facilitates a highly reconfigurable framework [11].

* This part describes the modular design of the system, implemented in a RAG-style pipeline. It describes each processing step from user input to context-aware response generation through Groq.
* Modular architecture for the system, with RAG-style FAISS vector-retrieval pipeline. The workflow consists of:
* They provide a natural language search.
* Embedding: The question is represented by a dense vector through use of the all-mpnet-base-v2 model.
* Retrieval: FAISS conducts a top-k similarity search against the embedded documents to retrieve the most semantically related passages.
* Response Generation:  Retrieved passages are concatenated with the original query and fed into llama3‑70B‑8192 via Groq. This Groq pipeline by virtue of its graph‑relational object queries produce an accurate, context‐aware response.
* Output: The chatbot delivers the generated answer back to the user in conversational form.

Figure 9 shows the core workflow of how a RAG-based (Retrieval-Augmented Generation) AI chatbot processes a user's question to deliver a meaningful answer. It all starts with the **Natural Language Input**, which is the question a user types in. This input is first transformed through **LLM Conversion**, where it’s converted into a structured **Query** that can be understood by the system. At the same time, the input is converted into **Vector Embeddings**, essentially turning words into numbers so that the system can compare them semantically.

Once the vector form is ready, a **Similarity Search** is conducted using **FAISS indexes**, which helps the system retrieve the most relevant information or documents related to the query. This content is passed into a **Retrieval** module and then combined with the original query in the **LLM Processing** phase. Here, a large language model (LLM) like LLaMA3 interprets all the gathered context and generates a well-formed, human-like **Natural Language Output**.

**Basic Architectural Diagram:**

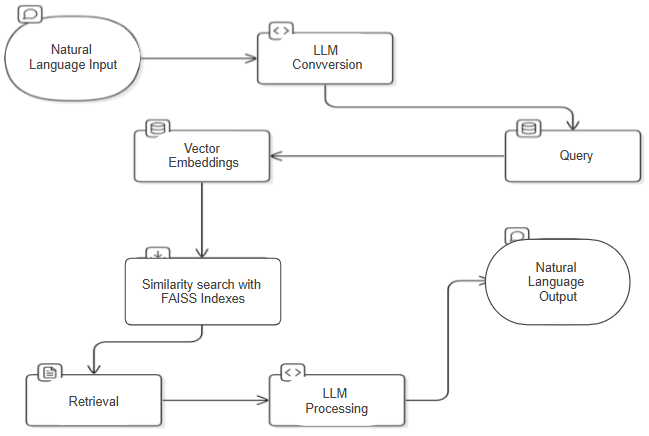


Figure 9. Architectural Diagram for Chatbot

## 3.2 Data Design

The data design describes the structural representation of system data and its flow. It includes visual models such as the ERD and DFD for understanding data relationships and processing.

### 3.2.1 Entity-Relationship Diagram (ERD)

The ERD depicted in figure 10 illustrates the logical structure of the database. It captures key entities, their attributes, and relationships essential for managing user and system data.

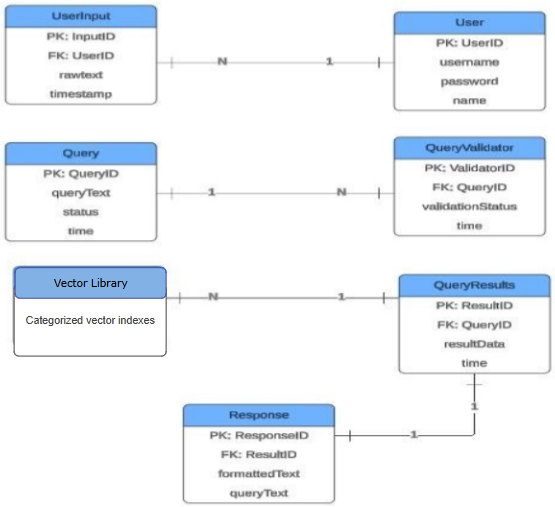


Figure 10. ERD for Logical Structure of Database

### 3.2.2 Data Flow Diagram (DFD)

Figure 11 visualizes how data moves through the system. It highlights the input, processing, and output flows that enable effective chatbot interactions.

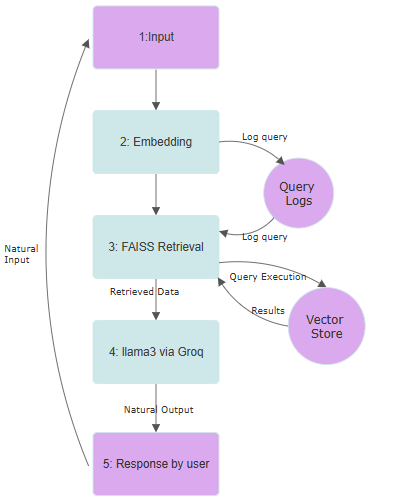


Figure 11. Data Flow Diagram for Chatbot Interactions

## 3.3 Domain Model

Figure 12 visualizes how data moves through the system. It highlights the input, processing, and output flows that enable effective chatbot interactions.

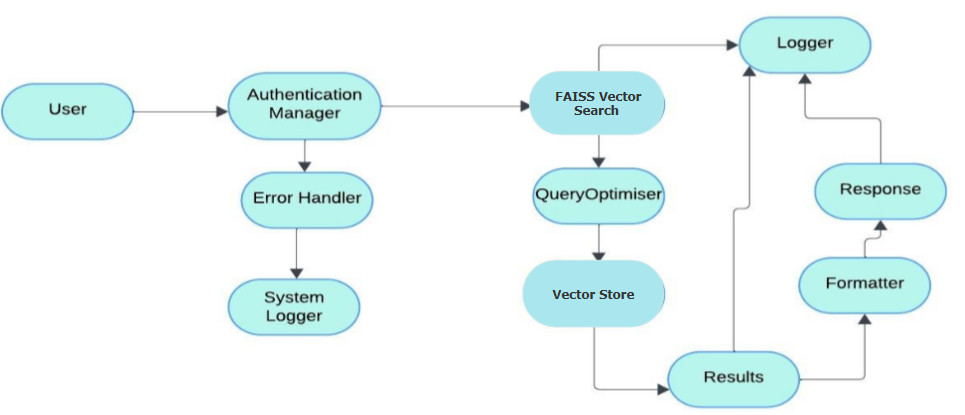


Figure 12. Domain Model Representation

Entities:

* User: Represents an individual or system interacting with the system.
* Natural Language Input: The user's input in natural language.
* LLM: The Large Language Model used for conversion and generation.
* FAISS Vector Search.
* Vector Store
* Data: The extracted data from the vector store.
* Natural Language Output: The final output generated by the LLM in natural language.
* Logger: Logs system events and activities.

Attributes:

* User: UserID, Name, Email, Password
* Natural Language Input: InputText, Timestamp
* FAISS Vector Search
* Data: DataPoints, Timestamp
* Natural Language Output: OutputText, Timestamp

Relationships:

* User -> Natural Language Input: One-to-many. A User can provide multiple Natural Language Inputs.
* Natural Language Input -> LLM: One-to-one. The Natural Language Input is processed by the LLM.
* LLM -> Query: One-to-one. The LLM processes the Query.
* Query -> Vectorstore: One-to-one. The Query is executed against the vectorstore via similarity Search.
* Vectorstore -> Data: One-to-many. The Vectorstore can provide multiple Data points i.e. vector embeddings.
* Data -> LLM: One-to-one. The Data is processed by the LLM.
* LLM -> Natural Language Output: One-to-one. The LLM generates a Natural Language Output.
* All components -> Logger: Many-to-one. All components can log events to the Logger.

Associations:

* Embedding model and natural language query: The Natural Language Input is associated with the embedding model as Input is converted to embeddings before similarity search.
* Input vectorembeddings and Vectorstore: The created vector embeddings is associated with the vectorstore as it interacts with it for data retrieval.

Data and Natural Language Output:

* The Data is associated with the Natural Language Output as it is the input for the Llama model which it uses to generate the final output.

### 3.3.1 Class Diagram

Figure 13 depicts the functional components of the RAG pipeline. It shows how queries are processed through encoding, retrieval, re-ranking, and generation stages.

Figure 13 illustrates a RAG-based (Retrieval-Augmented Generation) system architecture designed to generate contextually accurate responses using both retrieval and generation components. The flow begins with a user query, which is encoded by the QueryEncoder and passed to the Retriever to fetch relevant documents. These documents are then reranked by the Reranker and processed by the LLMProcessor, which generates a context-aware response. The output is validated, optionally cached, and formatted by the ResponseFormatter. Supporting components like AuthenticationManager, Logger, ErrorHandler, and Cache ensure security, traceability, error handling, and performance optimization throughout the system.

### 

Figure 13. Class Diagram for RAG-based Chatbot

### Activity Diagram

The activity diagrams break down various user actions into workflows. They depict step-by-step interactions the user and the RAG based application will have between each other.

**Register:**

### Figure 14 shows the process of a procedure followed by a customer to open an account in the system. The process starts with the customer filling out a registration form by providing an email address and password. The details are then submitted to the system for verification purposes.

### During the validation process, the system evaluates parameters such as the possible previous use of the email and the adherence of the password to specified security standards. If the information provided is considered valid, the system confirms the registration and securely stores the user's account information. If the information is invalid, the system notifies the customer, urging them to correct the errors before proceeding further. This ensures not only data integrity but also user responsiveness, thus improving a secure and user-friendly registration

### A diagram of a system Description automatically generated

Figure 14. Register Activity Diagram

### Login:

### In figure 15 we can see how a customer logs into the system. Login begins when customers input their email and password as their login credentials. This information is sent to the system to be checked.

### The system reviews the submitted credentials to confirm they are matching what is in its records. When the credentials are right, the customer can access their personal information and continue on their dashboard page. Should the credentials not check out, the system informs the customer with an error and asks them to try once more. The process makes access secure and leads the user clearly to whether they were successful or not.

### A diagram of a system Description automatically generated

Figure 15. Login Activity Diagram

### Forget:

Figure 16 shows the steps a customer need to take to recover their password. A customer needs to access their registered email in order to initiate the recovery. The system goes on to see if the email address is legitimate. In case the system cannot find the email, it immediately informs the customer that the email may be incorrect or has not been registered yet.

### If everything is validated, the system offers you a password reset link or an OTP at your email address. After that, the customer uses the link or OTP to set up a new password. After entering a new password, the system puts it into the system and informs the customer about resetting the password successfully. Due to this flow, users are able to access accounts securely and conveniently

### A diagram of a system recovery Description automatically generated

Figure 16. Forget Activity Diagram

### Product Recommendation:

### Figure 17 illustrates the flow of how a Retrieval-Augmented Generation (RAG)-based model makes a product recommendation to a customer. The process starts when a customer queries a product to the system. The RAG-based model applies Natural Language Processing (NLP) to understand the intent behind the customer's inquiry and get a detailed view of the inquiry.

### Subsequently, the model requests product data from the backend system. The backend then returns key product information like name, price, and description. The RAG-based model next ranks products according to how well they are a match for the customer. The model next aggregates and shows a list of suggested products to a customer, thus ending the recommendation cycle.

### 

Figure 17. Product Recommendation Activity Diagram

### Price Enquiry:

Figure 18 illustrates how the RAG-based model responds to product price requests. The process starts when a customer queries the model for a particular product's price. The model makes a request to the system for retrieval of a price.

The model tries to fetch product price information, which can include offer/discounts. In case any pricing data exists, it sends back this data to the customer. In case pricing data is not found, the model returns a notice to the customer stating that it was not possible to retrieve a price.

### 

Figure 18. Price Enquiry Activity Diagram

### 

### Search by Product Features:

### Figure 19 shows the workflow for searching a product by feature in a RAG-based recommendation model. It starts with a customer submitting a particular feature they are searching for like "lightweight laptop." The RAG model acts upon this natural language query by determining the essential attribute mentioned. The system then queries its database for any products matching the attribute specified.

### If matching products are located in the system, it brings back the related data to the RAG model, where it then shows the customer the findings. In case matching products are not located, the system communicates with the customer and advises them to change or modify search criteria. This directs the customer through a smart and helpful search process where they can view related matches or reassess their ask for improved results.

### 

Figure 19. Search by Product Features Activity Diagram

### Summary of Product Features:

### Figure 20 depicts the interactive product summarization process using a RAG-based AI model. It starts when a customer asks for a brief overview of a product, such as a summary of its key features. The RAG-based model processes this natural language request and generates a concise, easy-to-understand summary that highlights the most important aspects of the product.

### After the summary is delivered, the system checks whether the customer is satisfied with the information. If yes, the customer simply ends the interaction. If not, the customer can ask follow-up questions for more details. In response, the model intelligently retrieves and presents additional information, ensuring the user gets exactly the depth of understanding they need. This setup offers a dynamic and personalized product discovery experience.

### 

Figure 20. Summary of Product Features Activity Diagram

### General Queries and FAQ’s:

The figure 21 shows the RAG-based model’s way of responding to common questions asked by customers, including frequently asked ones (FAQs). The entire process begins once a customer sends in a general inquiry. It looks into the question and identifies the main topic, after which it looks for suitable answers in an internal FAQ database.

If a match is made with FAQ data, the system returns the information to the RAG model which creates a straightforward reply for the customer. But if there is no related FAQ, the system tells the customer and might invite them to explain their question in a different way. As soon as all the necessary data is supplied, the model assembles an answer that can be easily read by people. Because of this, even unusual or broad questions can be answered accurately by users.

### 

Figure 21. General Queries and FAQs Activity Diagram

### Handling Customer Errors:

Figure 22 illustrates how the RAG-based model manages questions from customers that are not clear. A user might write “bluetooth laptop,” for example which can mean many different things. Where things are not clear enough, the model requests that the customer provide more accurate information.

If the customer updates their original question, the model processes it again and delivers a suitable response. If the customer is happy with the service, the interaction stops. However, if the user is uncertain or in need of more details, the model provides another answer, making sure the customer learns from the best possible information. As a result, the chatbot is more reliable by offering customized and user-centered interactions.

### 

Figure 22. Handling Customer Errors Activity Diagram

### State Transition Diagram

This diagram shows how the system responds to different user states. It maps transitions between user interactions and backend states in the chatbot’s lifecycle.

### The state transition diagram shown in Figure 23 shows the course of user queries from the start to the moment when a response is delivered by the RAG-based system. It all starts when a user types in a question or inquiry about a product into the system. After input, a component produces an embedding that helps compare natural language with other data.

### After that, the system finds the right documents by running queries using the embedding it produced. Next, the matched results are processed by an error handler to spot missing or partial matches, as well as problems with the system. Any mistakes found are written down, so they can be traced and used to help solve problems. When there is no error or once handled, the system moves to its final state to produce and show the user the generated response. As a result of this flow, the system receives and sends data safely and keeps working accurately due to error logging and query verification throughout.

### 

Figure 23. State Transition Diagram for Input Processing & Response Generation

### 

### Sequence Diagram

This section depicts the time-ordered interactions between system components. They illustrate the flow of operations like registration, recommendations, and error handling.

**Register**:

Figure 24 depicts how a customer deals with the system as they register for an account. Regardless of their requirements, customers first provide an email and create a password. After that, the system verifies that the information does not already exist and that it is strong enough. When the information is accurate, the system saves it securely and shows the customer their successful registration. If not, the system lets the customer know that there was a mistake in the input.

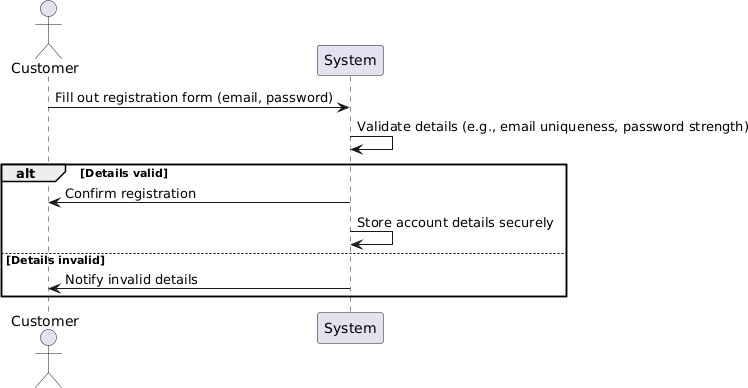


Figure 24. Register Sequence Diagram

**Login**:

The sequence diagram depicted in figure 25 tracks the process where a customer provides their login information. It checks the details from the records it keeps. When the input is right, the system welcomes the user and takes them to the dashboard. Should the login fail, the system alerts the customer and, in most cases, locks their account for a while.

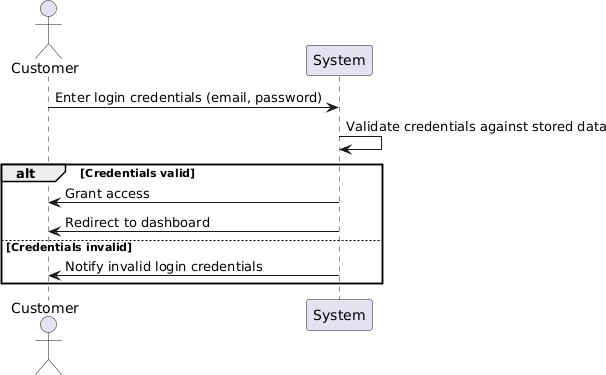


Figure 25. Login Sequence Diagram

###### Forget:

Figure 26 shows how recovery of lost passwords works. To initiate recovery, a customer has to send their registered email. If the email is genuine, the system delivers a reset link or OTP to help the customer change their password. After that, a notice is sent confirming that the password has been saved. When the email cannot be found, the system tells the customer.

###### A screenshot of a computer program Description automatically generated

Figure 26. Forget Sequence Diagram

**Product Recommendation:**

Figure 27 illustrates the customer seeks advice on what products to buy. The input is studied by RAG through NLP and it also accesses the backend for required product information. After considering the customer’s preferences, it groups and shows just the recommended products to the user.

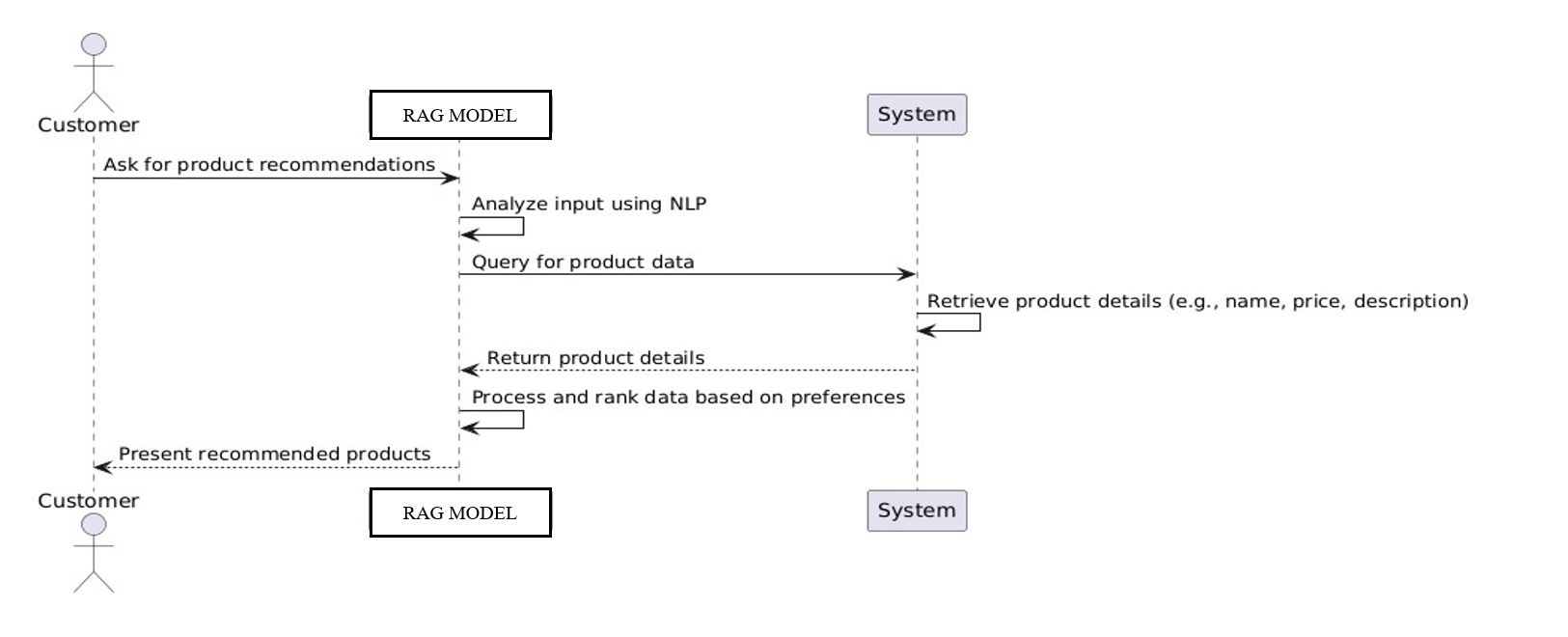


Figure 27. Product Recommendation Sequence Diagram

.

###### Price Enquiry:

Figure 28 shows the order in which pricing queries are handled. If a customer needs to know a product’s price, it is the RAG model that checks the system. When pricing information is found, it is sent and shown on the screen. Should this happen, the system provides a message that the user is unable to obtain pricing information.

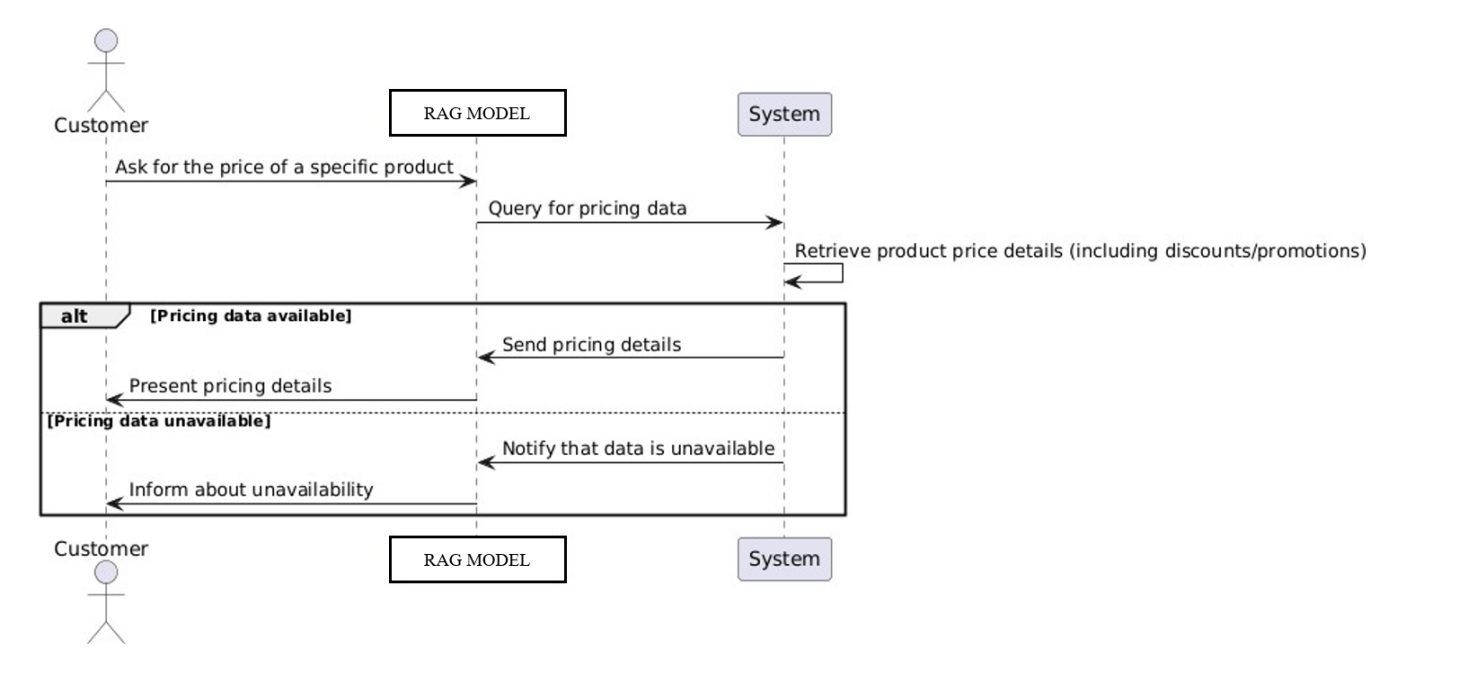


Figure 28. Price Enquiry Sequence Diagram

**Search by Product Features:**

The sequence diagram in Figure 29 depicts customers choosing a special feature they want such as a “lightweight laptop.” The RAG model deals with the request and checks the backend for matches. If products are detected, the list function returns those products. Should the results not work, the model advises customers to tune the search and make Keyword selections so that the product fits with their preferences.

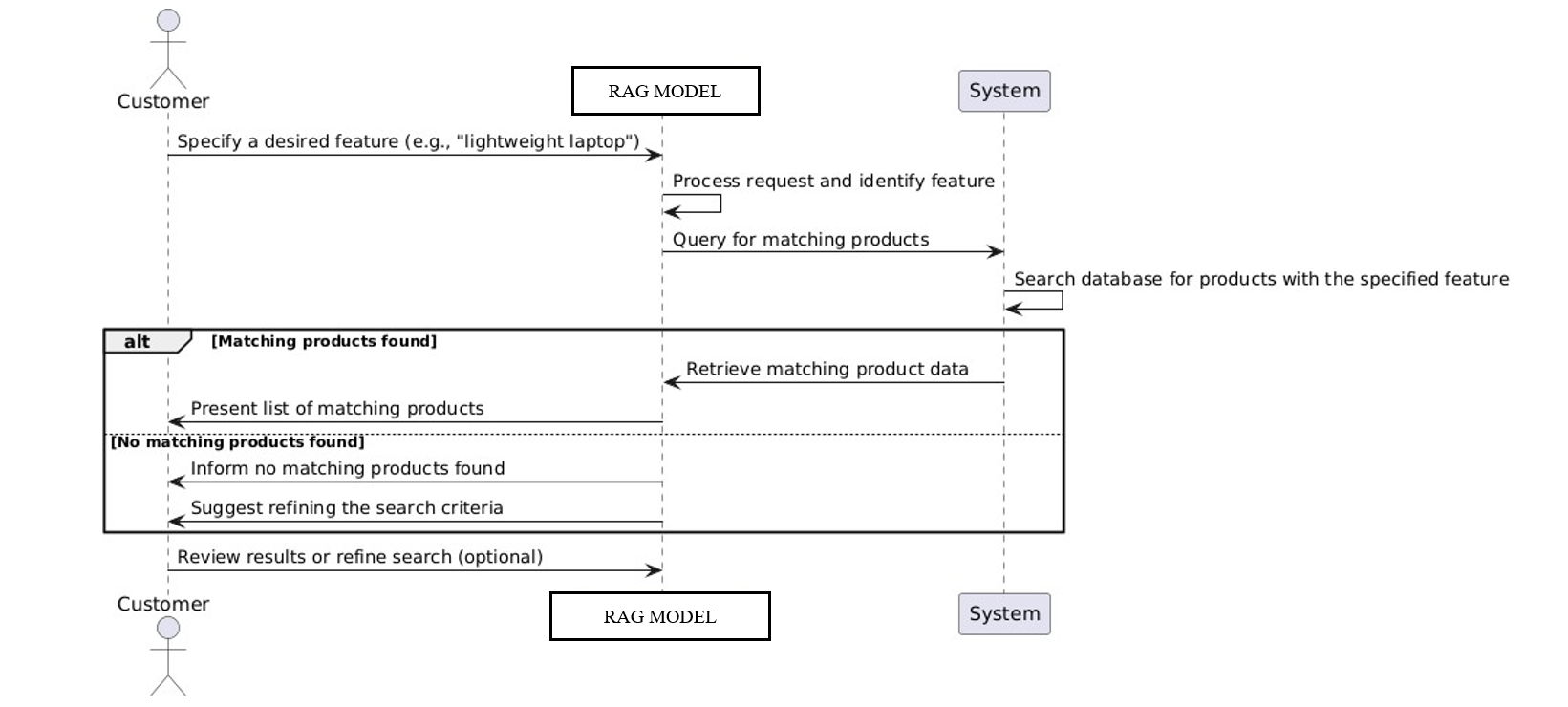


Figure 29. Search by Product Features Sequence Diagram

**General Queries and FAQs:**

In the sequence diagram illustrated in figure 30, the customer provides a general question. The question is run through the RAG model which chooses the topic and checks the system’s FAQs. Any relevant information is returned and a clear answer is given after that. If that’s the case, the model explains to the user and offers to rephrase the question.

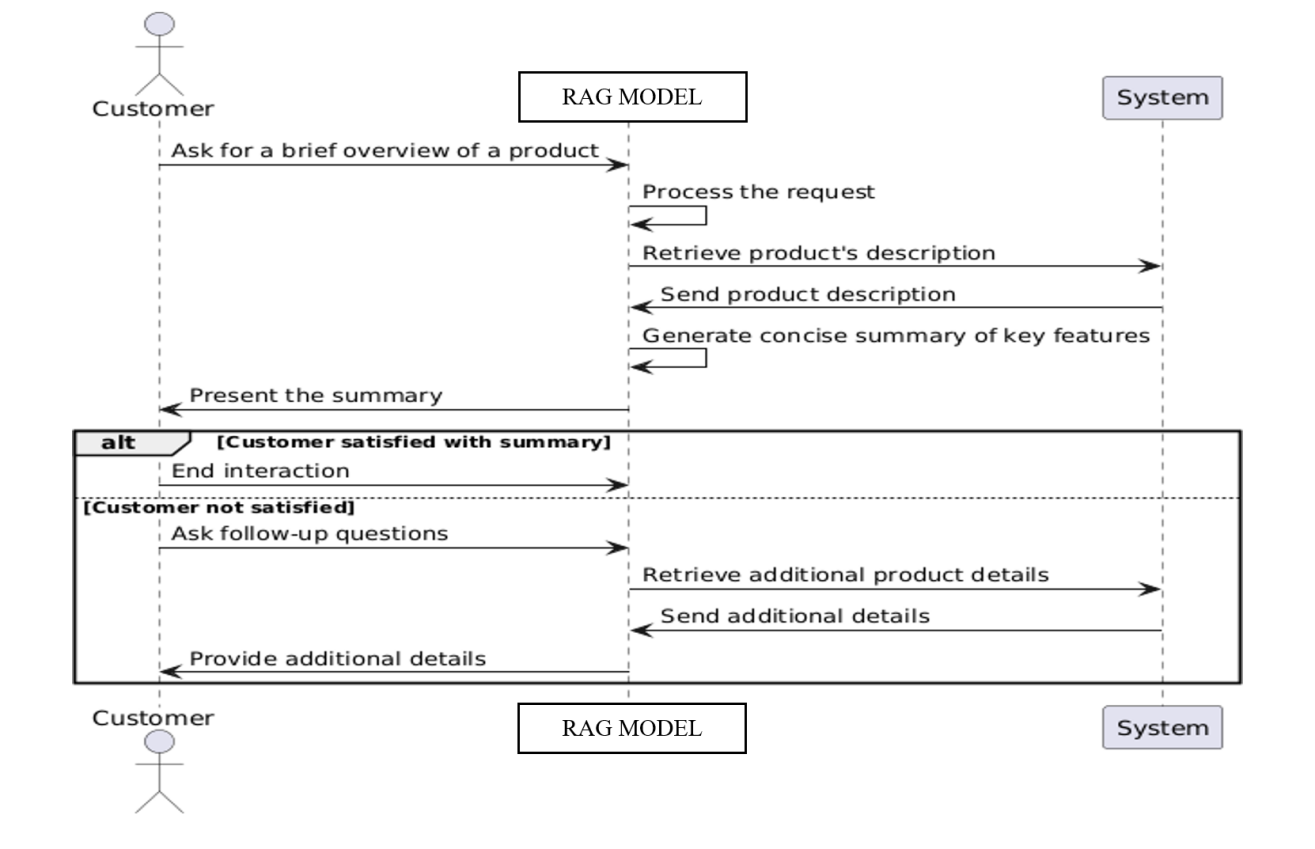


Figure 30. General Queries and FAQs Sequence Diagram

**Handling Customer Errors:**

Figure 31 depicts what happens in the interaction when a customer submits a general or FAQ-style question using the RAG-based model. The customer's question is simple at the beginning, but it’s narrowed as the model searches for a clear main subject. After that, the system sends a request to the backend to find any important FAQ information. Should the guide be recognized and data be available, the system provides relevant information that the model summarizes and presents to the customer.

If the topic isn’t in the system or the FAQs don’t cover it, the model informs the customer. As a result, they may recast their question or give more information so the answer is clearer. As a result, when answers are not instant, the flow still enables a natural and aided conversation, making the process smarter and better for users.

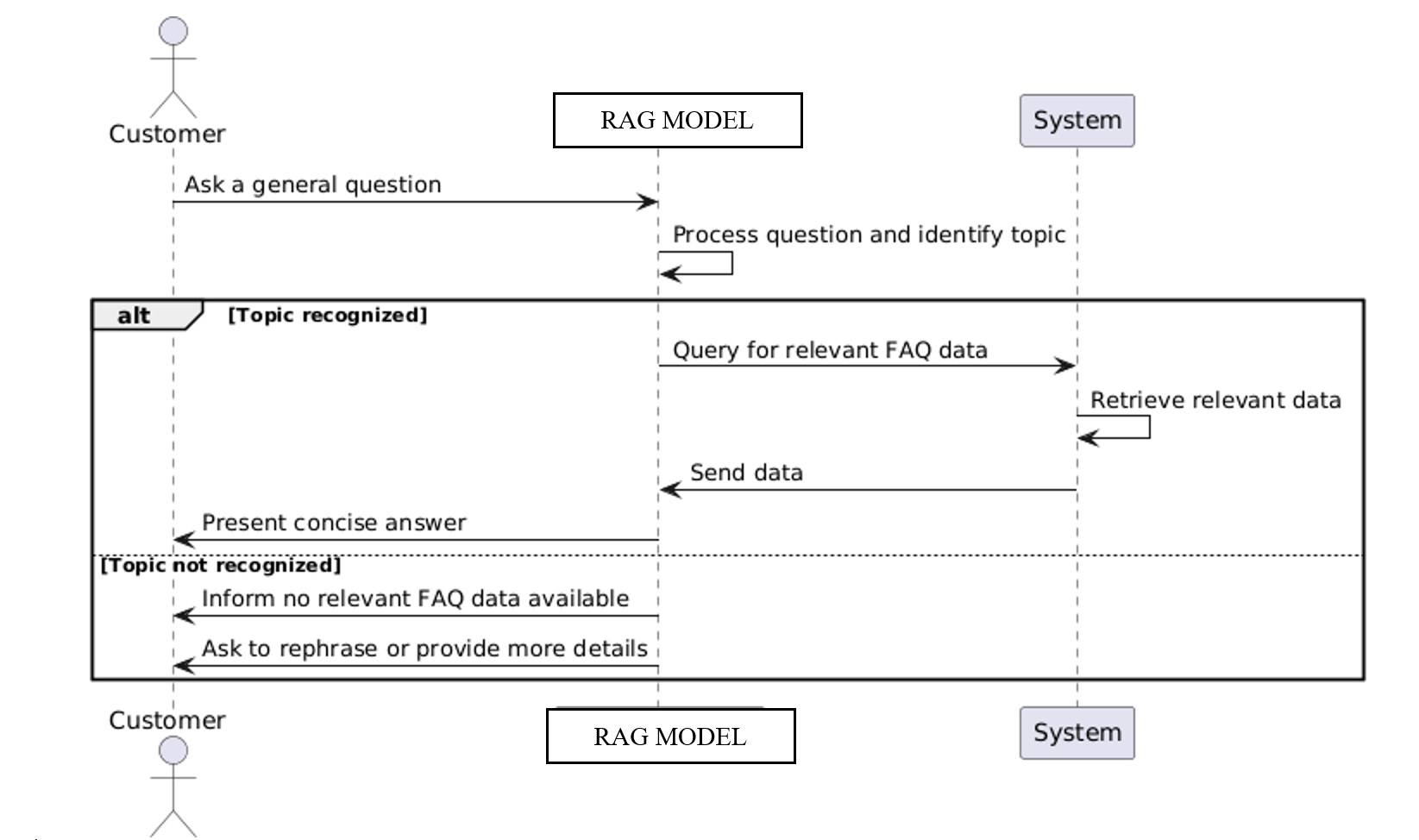


Figure 31. Handling Customer Errors Sequence Diagram

### Component Diagram

The figure 32 outlines how the system is modularized into functional units. It provides a high-level view of how each component interacts to deliver chatbot capabilities.

* The component diagram illustrates the essential functional modules of the system and their interactions, offering a clear and high-level view of the architecture. This helps stakeholders understand how the system components collaborate to fulfill the intended functionality within a Retrieval-Augmented Generation (RAG) framework.
* The User Interface Layer Represents the frontend (web and mobile applications) where users input natural language queries. These queries are sent to the backend, and responses—generated by the RAG system—are displayed in a user-friendly format.
* API Gateway & Load Balanceracts as an intermediary between the frontend and backend, managing API requests, balancing traffic, and ensuring high availability and scalability.
* The Application Layer Hosts the core logic and microservices of the RAG system, including the Query Processor, Retriever, Ranker, and Generator. It first retrieves relevant documents from a vector store (e.g., FAISS) using dense embeddings (e.g., from all-mpnet-base-v2), then uses a large language model (e.g., LLaMA3) to generate accurate, context-aware responses based on the retrieved content.
* The Retrieval Layer Interfaces with the Vector Database to store and retrieve semantically similar documents. This layer ensures that the most contextually relevant knowledge is surfaced before generation.
* System Utilities Maintain overall system health through logging, monitoring, and error alerting. These components are vital for debugging, performance tracking, and ongoing maintenance.

### 

Figure 32. Component Diagram for Representing Functional Units of Chatbot

### System Architecture

This section describes the layered architecture of the system. Figure 33 details each layer’s role, from UI and API routing to data handling and system utilities within the RAG framework.

### The architecture of the system is divided into five main layers as seen in figure 18, each playing a different role in the RAG-based application. At the highest level, the User Interface Layer is responsible for web and mobile access for users to interact with the system. The API Gateway & Load Balancer handles traffic, authentication, and communication with backend services through which requests are routed. The Application Layer is responsible for essential logic, such as input tokenization, query building, document lookup with FAISS, and response generation through LLaMA3-70B-8192. The Data Access Layer is in charge of storage and caching, communicating with vector indexes and relational databases. Last but not least, System Utilities enable the architecture with centralized logging, real-time monitoring (e.g., Grafana), and automated error notifications, making it reliable and maintainable.



Figure 33. Five Layered System Architecture for Chatbot

## Mockups

This section provides visual representations of key application interfaces. The mockups showcase the layout and interactive elements across different user workflows.

**Introduction:** The following mockups effectively visualize the following:

* Login
* Registration
* Main UI
* Start screen workflows

###### Screens:

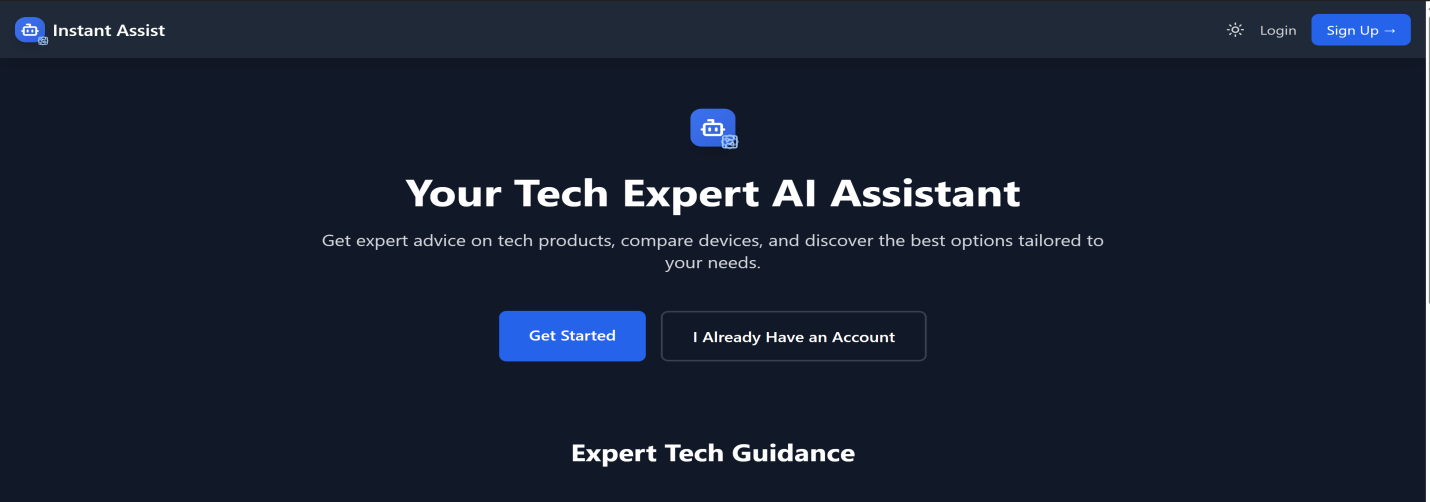


Figure 34. Screen 1 - Login/Get Started

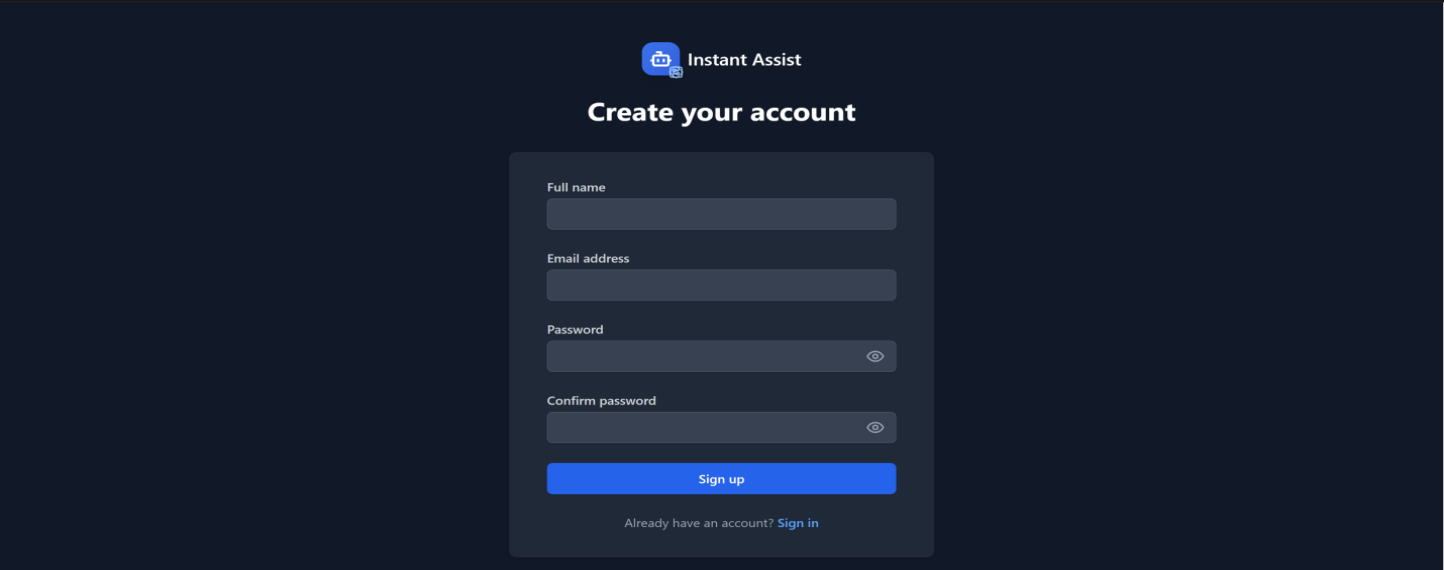


Figure 35. Screen 2 - Registration

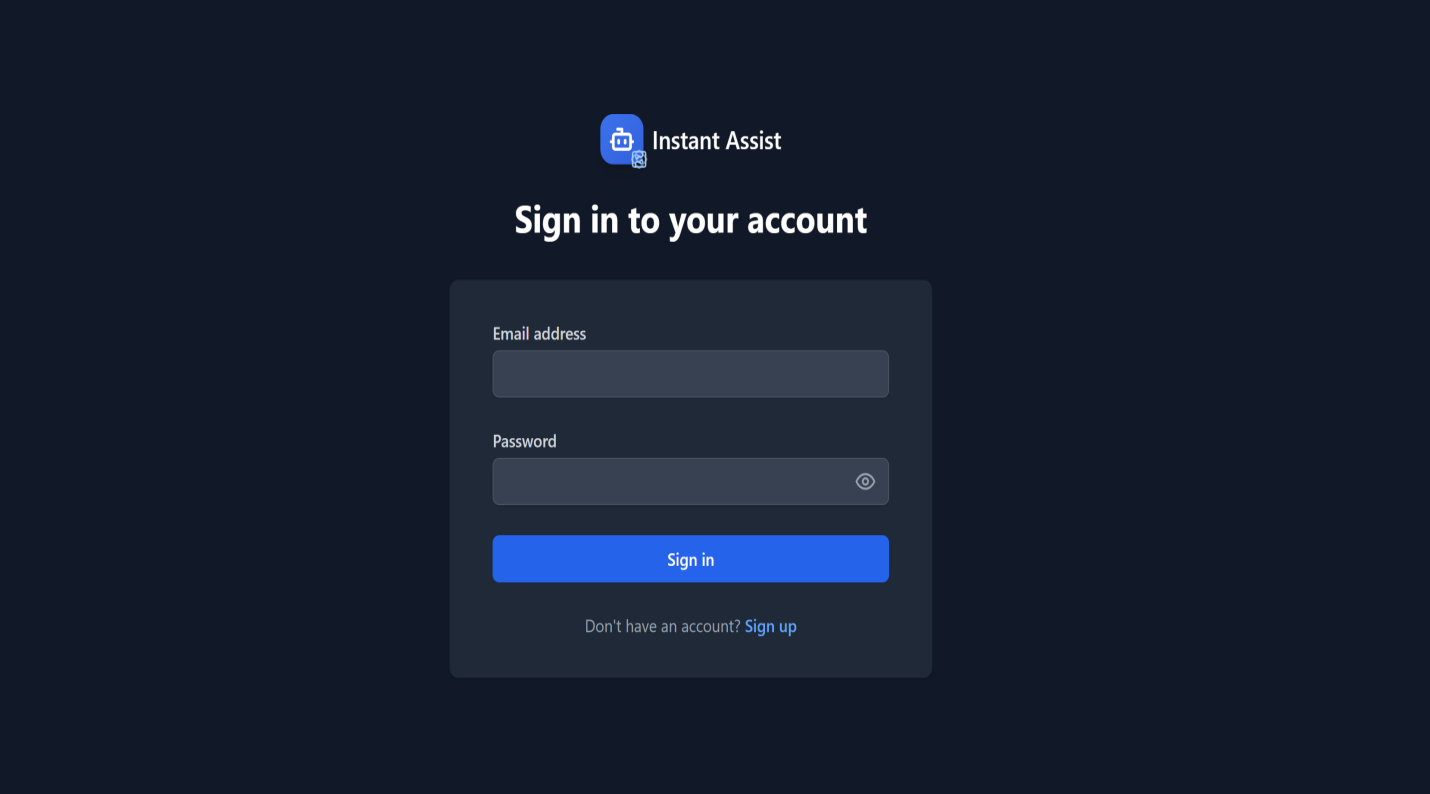


Figure 36. Screen 3 - Sign in

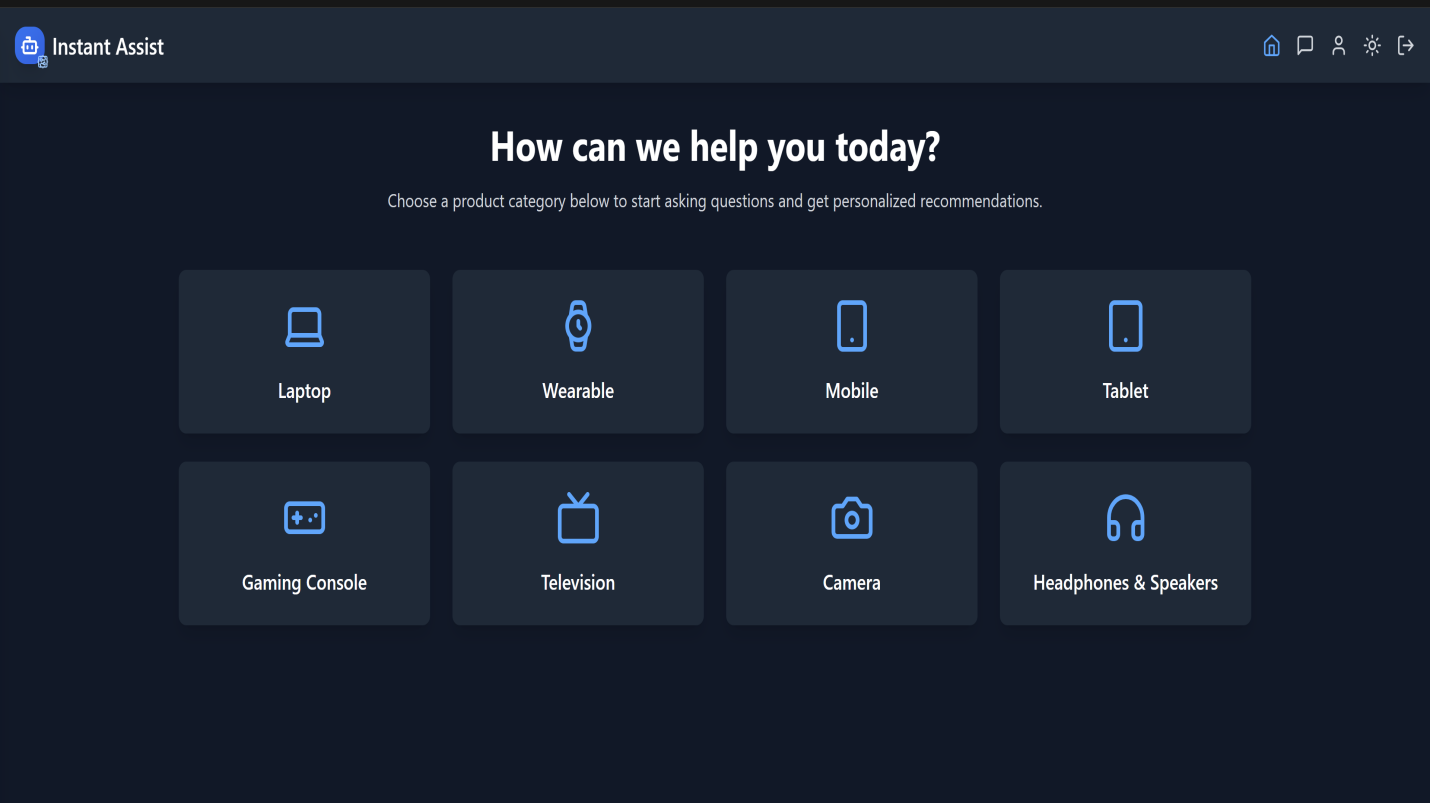


Figure 37. Screen 4 - Main UI

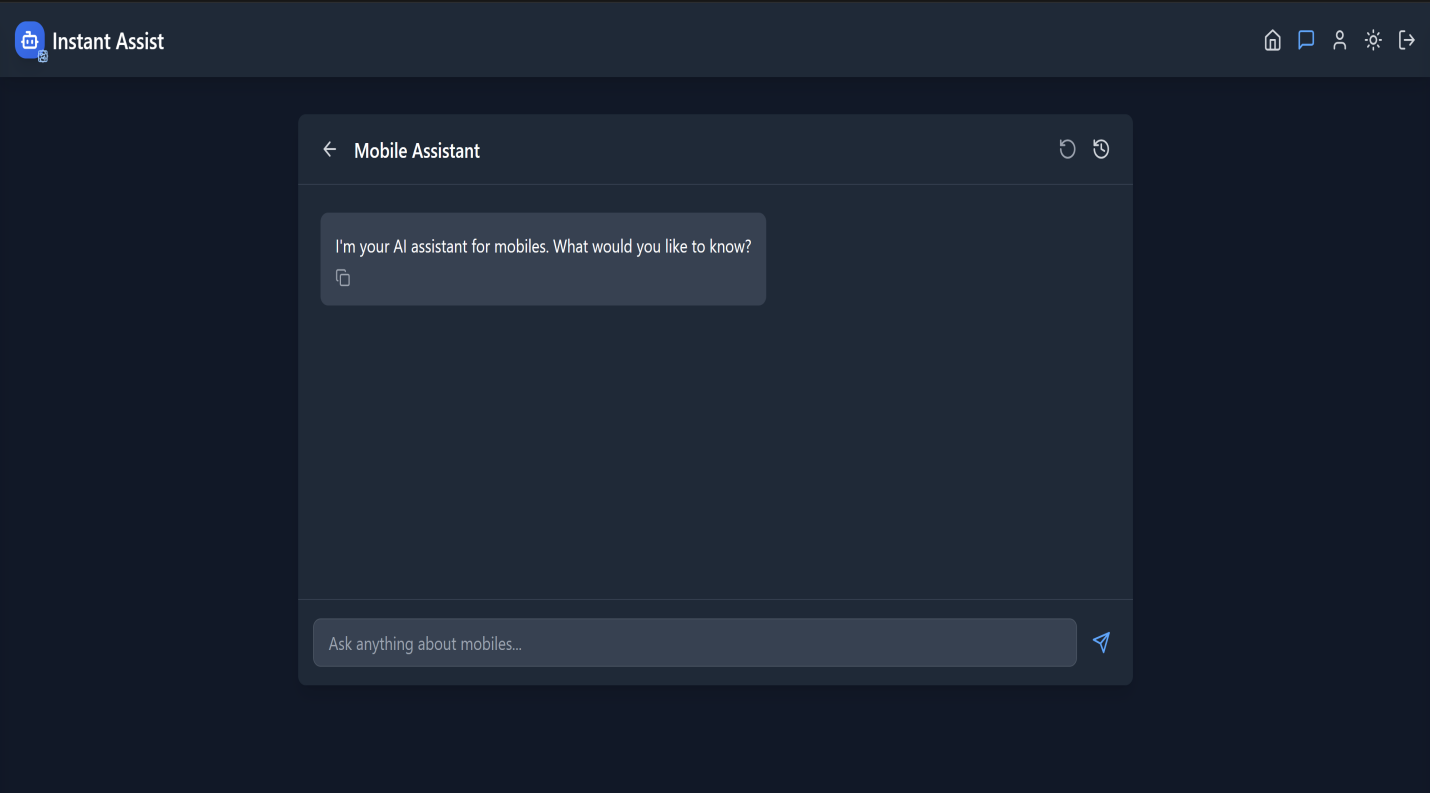


Figure 38. Screen 5 - Start Screen Workflows

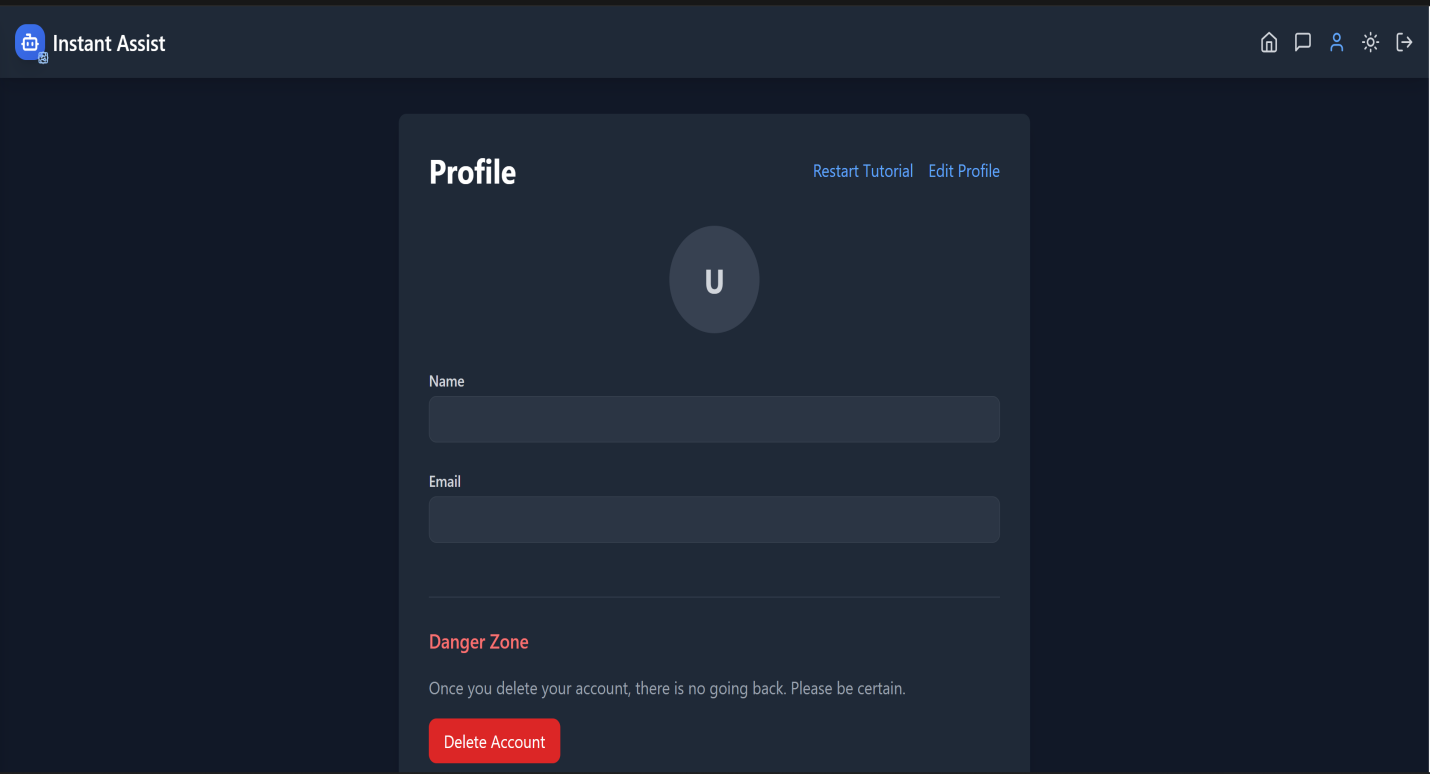


Figure 39. Screen 6 - Permanent Deletion of Account

###### User Interactions:

Outlined here are all major UI elements and user controls. It includes navigation tools, chat functionality, and customization options to enhance user experience.

* The "Chat" icon, positioned at the top-right corner of the screen, allows users to initiate a new conversation.
* Theme Toggle Button ("Light Mode"): Found at the top-right corner to switch between dark and light themes.
* "Log Out" Button: Also located at the top-right corner for signing out of the application.
* Chat History: Displays a list of previous chats, with options to rename or remove chats.
* "Clear Conversations" Option: Allows users to delete all chat history.
* Profile Icon: Located in the top-right corner, opens a dropdown for account-related actions or settings.
* The Text Input Bar located at the bottom of the screen, enables users to type queries or commands. A send button (paper airplane) accompanies the input field.
* Email Address: A text input field where the user enters their email address.
* "Continue with Google" Button: An alternative option to sign up using a Google account.
* "Login" Link: Redirects the user to the login page if they already have an account.
* The "Sign up" Link redirects the user to the signup page if they don't have an account.

**Tools Used:** ReAct

React has been used to develop the front end because it allows for fast, responsive, and dynamic user interfaces through its component-based architecture. React’s reusable components make development more efficient and easier to maintain, especially as the application grows. Its large community and ecosystem provide strong support, a wealth of libraries, and tools that speed up development and improve user experience.

## ****Chapter 4: System Implementation****

### ****4.1 Technology Stack****

This section fully explains the technologies chosen to develop the Instant Assist chatbot. Python, Flask and React, the important parts of the stack, were picked because they stand up well, match well with each other and help with quick AI development. All the tools in the stack contribute to easy-to-connect modules, better management of queries and an ability to deploy smoothly on any scale.

In order to create the RAG-based AI Assistant (Instant Assist), a specific range of technologies were employed that would provide for effective and smooth operation of the system. Below is an outline of the main technologies employed:

* **Python**:
  + Python is widely used within the data science and AI world due to its ease of use, versatility, and comprehensive support for various machine learning libraries. Its popularity enables us to easily incorporate different tools such as Flask, FAISS, and Huggingface Transformers into the system.
  + Role in the System: Python drives the system's back-end, right from processing user queries to executing the sophisticated models that fetch and produce responses.
* **Flask**:
  + Flask is a light web framework for Python, which makes it an ideal choice for developing small to medium-sized web applications. Its simplicity enables rapid development and integration with other parts.
  + Our System’s Role: Flask is the backbone that drives the API linking the front-end to
  + the back-end, where the sophisticated calculations occur. It manages activities such as receiving user queries and returning generated responses.
* **React**:
  + React is an efficient JavaScript library that allows for straightforward creation of dynamic, interactive user interfaces.
  + Role in Our System: React is in charge of what the users see and interact with. It enables the users to send in their inquiries and get real-time feedback from the back-end.
* **Huggingface Transformers**:

The Transformer has rapidly become the dominant architecture for natural language processing, surpassing alternative neural models such as convolutional and recurrent neural networks in performance for tasks in both natural language understanding and natural language generation (Vaswani et al, 2017).

* + Huggingface offers best-in-class embedding models such as all-mpnet-base-v2 that are optimized to produce vector embeddings.
  + **Role in Our System**: Huggingface’s models are responsible for generating vector embeddings. This helps in creating a RAG based Application.
* **Groq API**:
  + **Groq** provides specialized hardware that accelerates model inference, allowing us to process large models like **Llama3** much faster than standard CPUs or GPUs.
  + **Role in Our System**: Groq is being used as a model provider in this project. The LLM Llama3-70B-8192 is being used through Groq’s API
* **FAISS (Facebook AI Similarity Search)**:
  + FAISS is an **efficient search library** designed for high-dimensional vectors. It’s perfect for handling large sets of data where you need to find the most relevant items (documents, in our case) quickly.
  + **Role in Our System**: FAISS stores and retrieves the **document embeddings**. When a user submits a query, FAISS searches for documents that are **semantically similar** to the query, and returns the top matches.
* **Langchain**:
  + Langchain is a **framework** that makes it easier to integrate multiple tools into a single pipeline. It helps us orchestrate the flow of data from **query embedding**, through **document retrieval**, to **response generation**.

**Role in Our System**: Langchain connects all the components in our system, ensuring that the **query embedding** and **retrieval process** work seamlessly with the **response generation** step.

### ****4.2 Development Environment Setup****

This section details the setup of the development environment to make sure all system parts cooperated well. All the needed tools, including systems, IDEs and environment settings, were made to allow easy development and testing. Since Visual Studio Code can handle both Python and JavaScript, we selected the primary IDE and Flask and React were developed independently. TailwindCSS was used to create a responsive front end and dotenv to protect sensitive API keys from exposure. utmost care was taken for setup to allow teams to collaborate, provide more scalability and ensure security in front-end-back-end interaction.

We created the development environment so that every part of the system would cooperate without any hurdles. We selected tools and frameworks that would streamline development while allowing sufficient flexibility to accommodate future development. Here's how we did it:

* **Operating Systems**:
  + **The Backend and Frontend both have been developed on windows. This operating system was selected for its ease of use.**
* **IDE (Integrated Development Environment)**:
  + We utilized Visual Studio Code (VS Code) since it's a great tool for Python as well as JavaScript. It includes built-in debugging support, and plenty of other things which helped make our development process smoother.
* **Backend Setup**:
  + We developed it with Flask in a Python virtual environment to maintain the dependencies organized and segregated from our other projects.
* **Frontend Setup**:
  + React is utilized, and we employed TailwindCSS to make the frontend responsive and contemporary. React supports real-time updates to the user interface, so users can interact with the system without delay.
* **Environment Variables**:
* To safely store sensitive information such as API keys for Groq and Huggingface, we utilized dotenv. We also enabled CORS in Flask to enable the front-end and back-end to securely communicate across domains.

### ****4.3 System Modules and Components****

The following section outlines the key modules that make the system respond quickly, accurately and intelligently to users’ questions. All modules in the pipeline i.e. query embedding, retrieving documents and responding have been designed to improve the chatbot’s performance.

The system consists of several distinct modules that work together to provide accurate responses to user queries. A critical role is played by the following modules in ensuring that the system functions smoothly and efficiently:

* **Query Embedding and Semantic Search**:
  + Once the user inputs a question, it is sent through the Huggingface model to produce a vector embedding. This embedding captures the semantic meaning of the query, so that the system to locate documents that are semantically equivalent instead of matching keywords.
  + **FAISS** stores the document embeddings and performs **semantic search** to retrieve the top-k most relevant documents based on the query’s vector representation.
* **Response Generation**:
* The Llama3-70B-8192 model generates human-like responses. After retrieving the appropriate documents, the model processes the queried documents and retrieved documents to generate a context-sensitive, cohesive response.
* The RAG approach enables the system to integrate document retrieval with text generation so that it creates a response informed by the most appropriate information possible.
* **Frontend Interface**:
* React manages the user interface where users provide queries and get back responses. The frontend talks to the backend using Flask APIs, keeping queries and responses securely managed.

### ****4.4 Database Implementation****

This section outlines how the two databases in this system are designed and for what purpose. It illustrates how FAISS is employed in storing and retrieving significant vector information for semantic search while SQLite is utilized for handling user data like credentials, sessions and preferences. This clarifies how two disparate databases function to enable precise search for documents while ensuring proper data handling for users of the chatbot.

To ensure efficient data handling, the system uses both **FAISS** and **SQLite** databases:

* **FAISS stores document embeddings for effective semantic search. It enables the retrieval of matching documents by the degree of similarity between documents and queries.**
* **User data like authentication data, preferences, and sessions are stored using SQLite. This facilitates secure data storage and smooth user interactions**.

### ****4.5 APIs and External Integrations (if applicable)****

**This part describes how third-party services and APIs were utilized to broaden the core functionalities of the chatbot.**

**Multiple third-party APIs and third-party services were implemented to provide additional functionality to the system:**

* **Groq API**:
  + **Groq's API was also implemented to speed up model inference. This shortens the processing time it takes for large models such as Llama3 to respond. LLmam3 was utilized through Groq i.e. Groq acted as the model provider**.
* **Huggingface API**:
  + **Huggingface’s pre-trained models** which includes the all-mpnet-base-v2 wordtovec model is used for generating vector embeddings.
* **Kaggle Dataset (Gadgets 360)**:
  + I would like to use the Gadgets 360 data from Kaggle, which gives extensive details about a variety of consumer electronics. It acts primarily as the training data for the system to ensure correct answers are generated for product-related queries.

### ****4.6 Dataset Selection and Preprocessing****

Here, we discuss why data from Gadgets 360 was optimal and how we made the model work. The data we have includes precious information related to consumer electronics like specifications, prices and categories for smart phones, laptops and so forth.

We have selected the Gadgets 360 dataset from Kaggle due to its extensive product coverage of consumer electronics, ranging from smartphones, laptops, gaming consoles to televisions. This makes it a perfect match for our system, handling queries pertaining to product specs and prices.

**Preprocessing Steps**:

* Data Cleaning: All unnecessary data (such as empty fields, repeated entries) was removed.
* Currency conversion: The prices were normalized from INR to USD to achieve uniformity throughout
* Text Normalization: Standards were applied to product categories and names to enhance consistency in matching queries.
* Dealing with Missing Data: Missing data were imputed or removed depending upon context.

### ****4.7 Model Selection and Training****

Due to its capacity to produce high-quality, contextually appropriate responses the Llama3-70B-8192 model was selected. RAG method was used to integrate retrieval and generation, enabling the system to obtain corresponding documents and also to produce meaningful responses depending on the obtained information.

**Training Process**:

* Pre-processing was done on the Gadgets 360 dataset and used for training.
* The Dataset was converted to categorized PDFs.
* The HuggingfaceEmbeddings was utilized to invoke the all-mpnet-base-v2 model for generating vector embeddings of the PDFs.
* These embeddings were stored in FAISS.
* **HuggingfaceEmbeddings** was used to convert user queries into vector embeddings.
* **FAISS** was used for efficient **document retrieval**.

Contextually relevant responses were produced by Llama3 depending on the recovered documents.

## ****Chapter 5: System Testing & Deployment****

### ****5.1 Testing Approach (Manual, Automated)****

This section covers the main approach to testing which guarantees the application’s effective and reliable performance. The approach involves using both human and machine methods to observe the system in various scenarios. During manual testing, we tested the system with valid and invalid real-world user queries, but during automated testing, we looked at the inner parts of the system, including embedding, retrieval and generating responses. All of these cover whether the system works properly, fast and reliably depending on the input it receives.

Testing is an important stage in the development cycle of every software system. It ensures that the system operates according to expectation and carries out the intended outcomes. For RAG-based AI assistant, manual testing along with automated testing was applied to verify the performance, functionality, and reliability of the system.

* **Manual Testing**:

• Manual testing focused on testing general system behavior in actual situations. The system was tested against a series of user queries like general product queries, edge cases, and invalid queries. The focus was to see how precise the model response was, whether it was able to handle edge cases, and how was the user interaction with the system.

* + - Test Cases were written to verify how the system behaved when
    - Legitimate product questions (such as "How much does the iPhone 12 cost?")
    - Invalid queries (such as "What color is the moon?").
    - **Ambiguous queries** (e.g., "What is the most expensive gaming console?")
    - **Multiple queries at once** (e.g., asking about multiple products in a single sentence).
* **Automated Testing**:
  + Automated testing was added to the development cycle to make sure each part of the system worked. Unit tests were written for the following:
    - Query embedding: Query was transformed into vector embedding using HuggingfaceEmbeddings.
* Document retrieval: FAISS was tested to make sure it retrieved documents based on query semantic similarity.
* Response generation: Llama3 was tested to generate contextually correct and coherent responses based on retrieved documents.

### ****5.2 Unit Testing****

In this section, the main components are thoroughly validated at the unit level. It walks through how single parts such as query embedding, using FAISS for document retrieval and LLaMA3 response generation were tested individually. The part also specifies what is being checked and which test approach is used to find problems, avoiding mistakes and difficulties with integration after development gets further along.

Unit testing focuses on testing individual functions and components to make sure each part of the system works. The following tests were created:

* **Query Embedding**:
  + **Test Objective**: Ensure that the query is correctly transformed into a **vector embedding**.
  + **Test Method**: For a given sample query, check if the embedding vector output has the expected dimensionality and matches known values.
* **Document Retrieval**:
  + Test Objective: Make sure FAISS retrieves the right documents based on query.
  + Test Method: For a given query, check FAISS returns documents that are semantically relevant.
* **Response Generation**:
  + **Test Objective**: Check if Llama3 creates useful and clear answers.

**Test Method**: For a group of found documents, see if the model makes a smooth and fitting These unit tests aim to make sure each separate part works right lowering the chance of problems in joined parts.

### ****5.3 Integration Testing****

In this section, we explore integration testing, the goal is to ensure that every part of the developed system fits together flawlessly. It explains how each interaction between React, Flask, FAISS and LLaMA 3 is tested to ensure everything works as expected. This part aims to check that information is shared securely and that the system’s components keep operating correctly together.

Integration testing was done to make sure all components work together. Key integrations were tested between frontend and backend, and between retrieval and generation.

* **Frontend and Backend Communication**:
  + The Flask API backend handles queries from the React frontend. Integration tests were written in order to validate that queries from the frontend are received and processed by the backend correctly. The tests verify that the output produced by the backend returns in the correct format.
* **FAISS and Llama3 Integration**:
* We verified that both the response generation model (Llama3) and the document retrieval module (FAISS) were properly integrated. Here's how it works: You ask it, FAISS fetches documents, then Llama3 uses them to respond. Integration tests verified that the whole pipeline processed smoothly from start to finish.

### ****5.4 System Testing****

**This section addresses testing the system in its entirety, together with its constituent components, to gauge how it works in actual operating conditions. It outlines how this chatbot was not only tested for its functional capabilities, but also for how effectively it performs when stressed, how it reacts to abnormal or surprising questions, and how straightforward and consistent the user interface remains across.**

**System testing covers every aspect of a system, testing each in isolation to verify they are all working together as a networked entity, both in normal circumstances and in a worst-case environment**. This phase included the following components:

* **Functionality**:
  + Objective of the test: Verify that the system processes queries correctly and returns responses accordingly. This was done by entering several queries that were valid, invalid, or otherwise subject to interpretation within context.
* **UI/UX Consistency**:
  + Test Goal: Validate that the frontend user interface remains responsive and intuitive.  
    Testing was done to ensure the system produced responses, and the user interface made updates in real-time.
* **Edge Case Handling**:
  + Test Purpose: Verify system reactions to edge situations where inputs are likely to be extremely abnormal or complex. The system was also tested using less common queries not necessarily found in the training data in order to see how well the system can handle novel inputs.
* **Performance**:
* The objective of this test was to determine how flexible and adaptable the system was. The performance was gauged by monitoring response time with respect to answer appropriateness after a series of queries had been input simultaneously.

### ****5.5 User Acceptance Testing (UAT)****

The intention here is to include User Acceptance Testing (UAT) to ensure that the system meets users when implemented in real use environment. It documents how users engaged with a chatbot, posed questions, and provided feedback about how accurate, how clear and how helpful it was. Moreover, users share opinions about improving it, stating that processing outlier questions and enhancing the interface would improve the system.

User Acceptance Testing was done to verify if the system met expectations and requirements specified by the users. UAT:

* **Test Objective**: End users submitted product-related queries with the expectation of obtaining response from the system. The queries evaluation tested for accuracy, relevance, and clarity of the responses given.
* **Results**: Based on user feedback, the given answers were accurate, however, there were some unusual queries that were not answered accurately. Generally, the system was responsive and user friendly which were the main attractions to participants.

**Recommendations**: In the event that these other claims are true, the model needs improvement on handling outlier queries. Specifically, the interface also requires some enhancements to ease accessibility. In addition, users requested a ‘Chat History’ feature that would enable them to revisit previous conversations they had, should they wish to refer back.

### ****5.6 Performance Testing****

This section is about analyzing how well and how far the system can work with its current load. It describes how performance is measured with response time (latency), cosine similarity for semantic accuracy, readability scores and ROUGE for how precise the content is.

Performance testing was done to check whether the system can sustain high volumes of concurrent users while maintaining response times within acceptable limits. The following key performance metrics were assessed:

**Response Time (Latency)**:

* + Quantify the mean time for the system to process a query and send back a response.
  + **Result**: The system interacted within an average of 1 seconds, which is within the real-time interaction range.

**Cosine Similarity**:

* + **Objective**: Measure the semantic similarity of the user query with the retrieved documents.
  + **Result**: The average score attained indicates that the documents were effectively retrieved per the query meaning. The average cosine similarity score was 0.342.

**Flesch-Kincaid Readability**:

* + **Objective**: Determine the system's generated responses and assess their readability.
  + **Result**: Responses were found accessible to most users, as the average Flesch-Kincaid readability score indicated 80.

**ROUGE Score**:

* + **Objective**: Define the precision of the responses in relation to the expected answers and compute the expected value.
  + **Result**: Contextually pertinent and accurate, the responses yielded a ROUGE-L score of 0.329.

The performance metrics presented above demonstrate that the system has real-time query handling capability with high accuracy and low latency, providing an uninterrupted user experience.

## ****Chapter 6: Results and Discussion****

### ****6.1 Evaluation and Results****

This section reviews the results of the RAG-based AI Assistant when it was applied to a set of evaluation criteria. It describes how the system is evaluated to ensure accuracy, short delays, usefulness in meaning and how easy it is to read. Such metrics allow you to see how the system is doing at meeting its goals for giving quick, accurate and clear responses to users who need help with the product.

Multiple phases of testing, including unit testing, system testing, and user acceptance testing (UAT), were conducted to evaluate the RAG-based AI assistant’s accuracy, response time, relevance, and overall performance score. Additionally, a defined scope of metrics was established to assess the effectiveness of the system in accomplishing pertinent product data retrieval within the context of the conversation.

##### **Key Metrics Evaluated**:

* **F1-Score**: A measure of the balance between **precision** and **recall**, providing insight into the system’s overall performance.
* **Accuracy**: The percentage of correct responses generated by the system relative to the total number of queries submitted.
* **Precision**: gauges the relevancy of responses relative to the total answers given.
* **ROUGE**: It measures the quality of generated text by scoring the produced responses against the answers provided in the database (expected answers).
* **Cosine Similarity**: A metric examining the semantic overlap between the query and the documents retrieved, as well as between the query and the responses generated.
* **Latency**: The duration allocated for the completion of a task, in this case, the time it takes to process a query and produce a response.
* **Flesch-Kincaid Readability**: A metric for assessing the ease of understanding the generated responses, which ensures simplicity from the user's perspective.

The results concerning these metrics and how they impact the system's effectiveness from the overall perspective are addressed below.

### ****6.1.1 Comparison with Initial Objectives****

In this section the results produced by the system are measured against the original targets that were defined when the project began. The chatbot’s ability to be accurate, understand the context and respond in a timely manner was discussed. Results are examined to find out if the main goals were met and to detect any more requirements for boosting the system.

When commencing the project, several key objectives were set for the RAG-based AI Assistant. These objectives included:

* Ensuring that the system could respond to user queries in a timely manner.
* Confirming that the model's response accuracy was in line with user expectations.
* The system was supposed to capture the context of the queries and respond appropriately with relevant information concerning the products and queries given.

The system met the following objectives regarding the evaluation results:

* The **average response time** was **1 second**, which is acceptable for real-time customer support applications.
* The system’s accuracy was high, with **95% of queries** answered correctly, showing that the retrieval and generation processes were largely effective.
* The result response provided the degree of relevance found from both ROUGE scores and provided valuable user insights confirming usefulness wherein broad strata systemic feedback provided needed emerges user relevance assessment framework.

Thus, the preliminary goals of real-time responsiveness, accuracy, and contextual relevance were met, albeit with some remaining challenges concerning infrequent query handling and scale optimization.

Table 3 presents an evaluation of the project’s performance by comparing the defined objectives with the actual outcomes.

| **Objective** | **Achieved** | **Remarks** |
| --- | --- | --- |
| Real-time response (<1.5 sec) | ✔️ | Avg latency = 1.0 sec |
| Accurate and relevant answers | ✔️ | 95% accuracy, high precision |
| Handle general and product-specific queries | ✔️ | Succeeded in known queries |
| Handle rare or vague queries | ✔️ | Needs improvement |
| Scalable architecture with modular components | ✔️ | FAISS + Groq + Flask + React = scalable |

Table 3. Assessment of Project Objectives Against Achieved Outcomes

### ****6.1.2 User Feedback and Usability Testing****

It concludes with findings from User Acceptance Testing (UAT), in which users tested the application in normal conditions. Based on how people use the system, it identifies its upsides and downsides by observing user feedback that rates accuracy, responsiveness, how easy the system is to use and ability to answer specific queries. It seeks to present how the software is used by real users and suggests changes to enhance both what it does and how it feels to use it.

System evaluation heavily relied on user feedback and usability assessment. In what is referred to as User Acceptance Testing (UAT), actual users engaged with the system by submitting product-related queries alongside response evaluation against multiple criteria.

**Key Findings:**

* **Positives**:
  + **Accuracy**: Users considered the majority of product queries offered to them and, based on accuracy standards, were satisfied with the precision given on typical product queries.
  + **Speed**: User comments noted favorable sentiments towards response speed, with the majority of queries resolved in under 1.5 seconds.
  + **User Interface**: The frontend interface was **intuitive** and easy to navigate, providing a seamless interaction experience.
* **Areas for Improvement**:
  + **Rare Queries**: Uncommon or lesser-known product queries emerged as a challenge for the system, frequently producing irrelevant responses.
  + **Personalization**: More responses tailored to user demand were ideal, as users wanted answers aimed at specific features and product combinations sought.

The general perception indicates that although the system performs adequately for frequently encountered queries, it has some challenges resolving vague queries, lacking sufficient dataset scope, and adjustment to the algorithms and extracted data to tackle less common scenarios.

### ****6.1.3 Performance Evaluation****

In this section, attention is given to technical metrics that show how well the chatbot functions under various levels of use. Here, parameters including F1-score, precision, ROUGE score, cosine similarity, latency and readability are analyzed. Benchmarks are supported by performance measurements on charts, meaning we can view the system’s behavior and relate it to set standards for quality and relevance.

The system’s performance evaluation was done in terms of the response time, accuracy, and the quality of responses. This evaluation also looked into system load, scalability, and latency during periods of high traffic.

##### **F1-Score**:

* The F1-Score was calculated at 1.0 which suggests the system did well in balancing precision and recall. It indicates that the system does not miss an overly large proportion of relevant results and is able to provide accurate responses.

##### **Precision**:

* The stated system achieved a precision value of 95%, thereby constituting that a large proportion of the generated responses were relevant to the user. This was especially so for queries whose products or categories were present in the training data.

##### **ROUGE Score**:

* The **ROUGE-L** score was **0.329**, which suggests that the generated responses have a moderate overlap with the expected answers in terms of **n-gram recall**. This score is can be improved for even better **text generation quality**.

##### **Cosine Similarity**:

* The **average cosine similarity** between the query and the retrieved documents was **0.342**, indicating that the system was not as able to retrieve relevant documents based on semantic similarity as it can be i.e. it can be improved further.

##### **Latency**:

* The **average response time** for queries was **1 second**, which is well within acceptable limits for real-time processing.

##### **Flesch-Kincaid Readability**:

* The **Flesch-Kincaid readability score** for the generated responses was **80,** which falls within the **"easily understandable"** range. This indicates that the responses were appropriately structured for general users.

Table 4 summarizes the evaluation metrics and overall system performance, highlighting how well the system meets its goals.

|  |  |  |
| --- | --- | --- |
| **Metric** | **Value** | **Interpretation** |
| F1-Score | 1.0 | Perfect balance between precision and recall |
| Precision | 95% | Most generated responses were relevant |
| ROUGE-L Score | 0.329 | Moderate overlap with reference answers |
| Cosine Similarity | 0.342 | Acceptable semantic alignment between query and documents |
| Readability Score | 80 | Easy to understand (Flesch-Kincaid) |
| Average Latency | 1.0 sec | Real-time interaction range |

Table 4. Evaluation Metrics and System Performance Summary

##### **6.1.4 Evaluation Graphs**

The bar chart illustrated in figure 40 shows the comparison of various evaluation metrics for the RAG-based AI Assistant’s performance. Every bar shows a metric called BLEU, ROUGE, Accuracy, Readability and Cosine Similarity and their normed scores can be found on the vertical axis, running from 0 to 1.2.

We see from the chart that Readability had the highest value (almost 1.2), suggesting that the responses constructed by the system were highly suitable for end users to understand. Accuracy was very close to perfect which reflects that the system could respond to most questions correctly. However, ROUGE and Cosine Similarity obtained lower results which means the system could use changes to better connect semantic and contextual information. A BLEU score is not as good as it usually is, since the responses in summarization can vary widely. Ultimately, the graph shows that the outputs by the system are accurate and readable, but there is a chance to improve more on how it handles semantics.

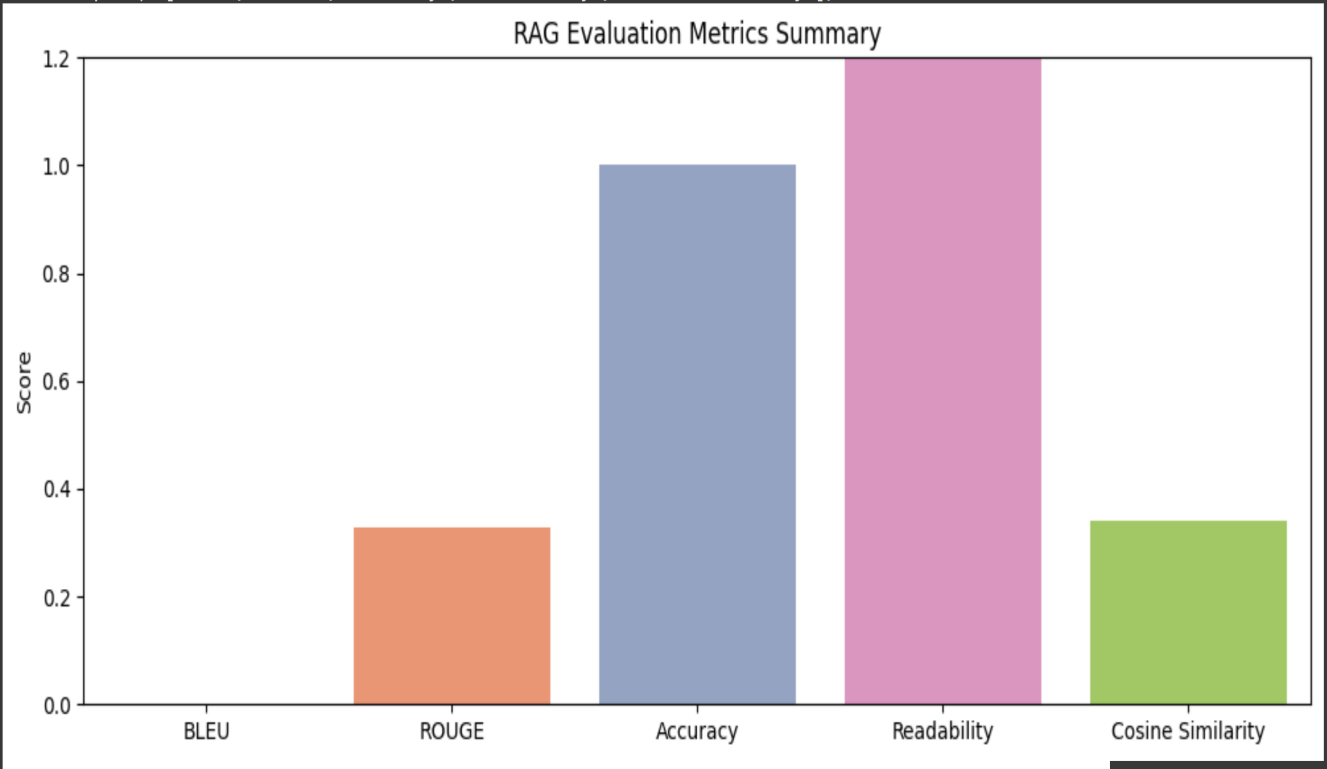


Figure 40. Evaluation Metrics for Chatbot Analysis

The box plot shown in figure 41 how the AI Assistant’s generated responses were evaluated with BLEU and ROUGE scores. The most common way to use these metrics is to check how closely the generated text matches a reference. The BLEU score chart indicates that n-grams match the expected answers very little. Many times, researchers find this to be true in response generation, as strict n-gram rules aren’t always a good guide to answer quality.

Despite this, ROUGE has a higher and more even distribution, with values generally between 0.27 and 0.37, pointed to moderate recall in the coverage of content. The system manages to focus on the key issues in the expected answers, but yet needs to be improved to match the correct answers more closely.

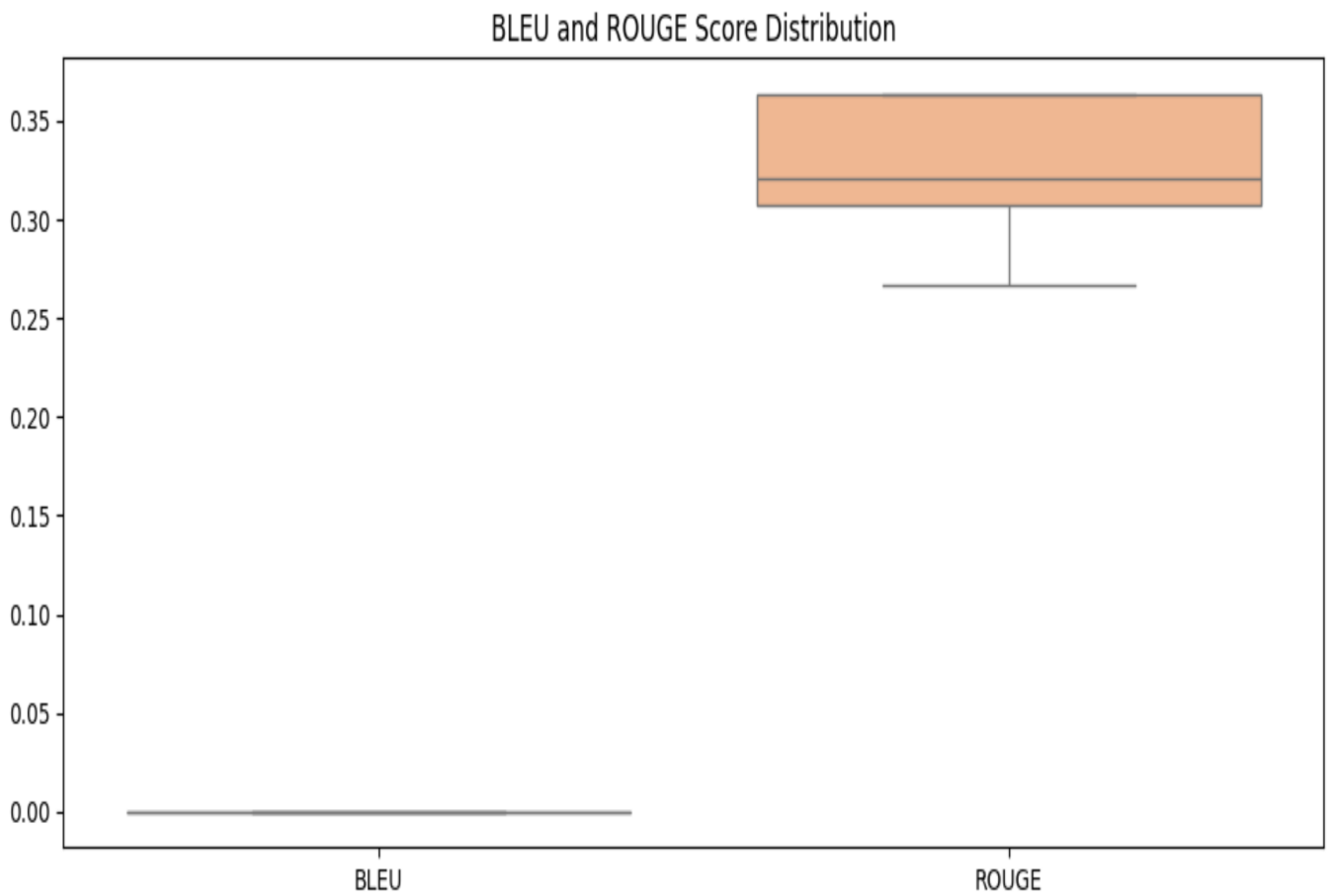


Figure 41. BLEAU and Rogue Score Distribution Graph

Cosine similarity scores depicted in figure 42 between user questions and documents from the RAG-based AI Assistant are shown in this histogram. When assessing cosine similarity, we confirm if there is a strong link between the user’s query and the results they see from the system.

The scores for many documents tend to be between 0.34 and 0.36 which means the retrieved documents have a moderate amount of semantic relevance. The majority of the results show that recall performs well, still, the spread and the small number near 0.31 signal that sometimes the results aren’t completely aligned. The smooth line on the histogram makes it easy to see that the majority of queries have a better similarity score than the topmost cases. The results point out that, even though the system brings up close enough matches, optimization of indexing or embeddings could help pinpoint better answers.

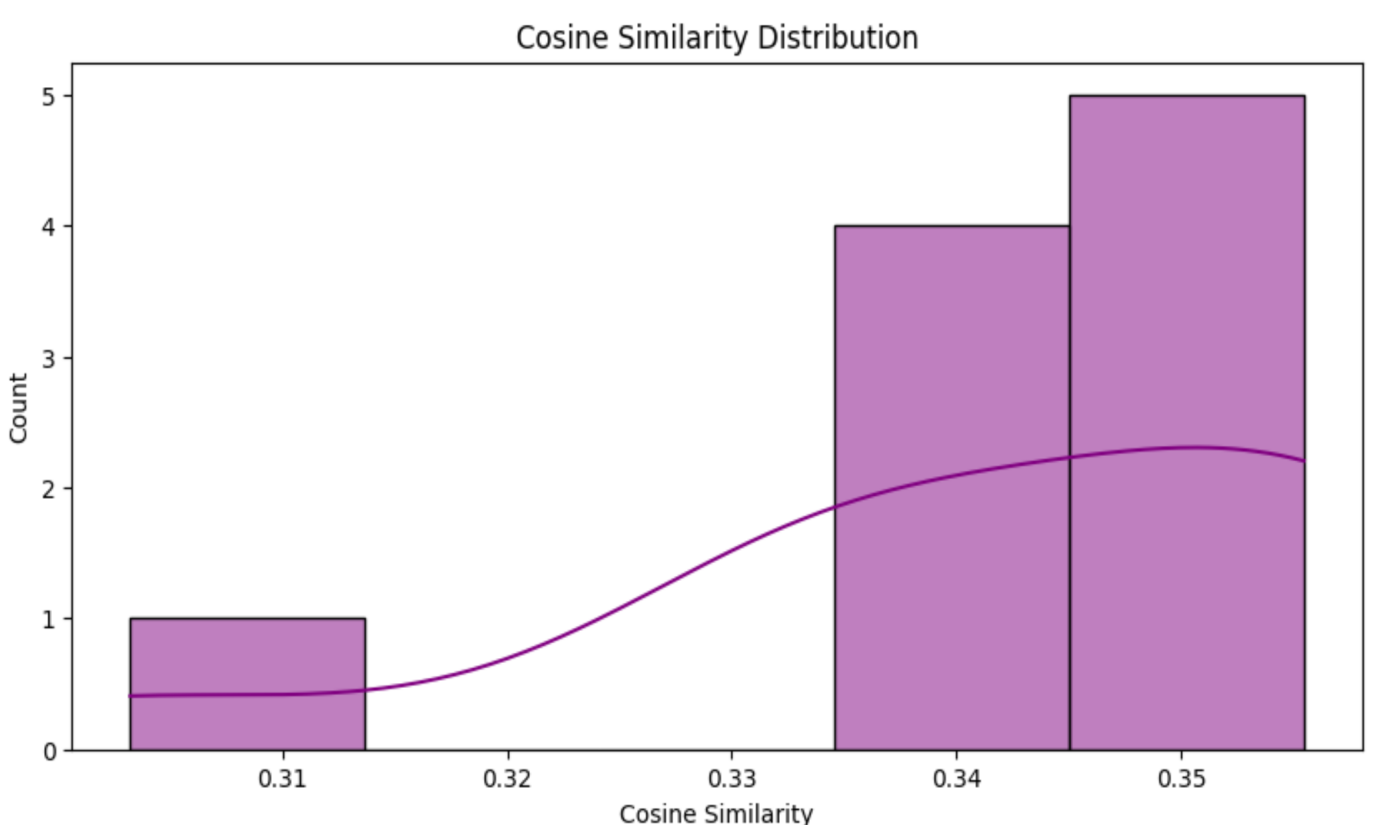


Figure 42. Cosine Similarity Graph for Analyzing Semantic Relevance

The histogram illustrated in figure 43 displays the distribution of latency results obtained during the evaluation of the RAG-based AI Assistant. Processing time or latency, is how long the system takes to process a user query and respond which is crucial for smooth interactive use.

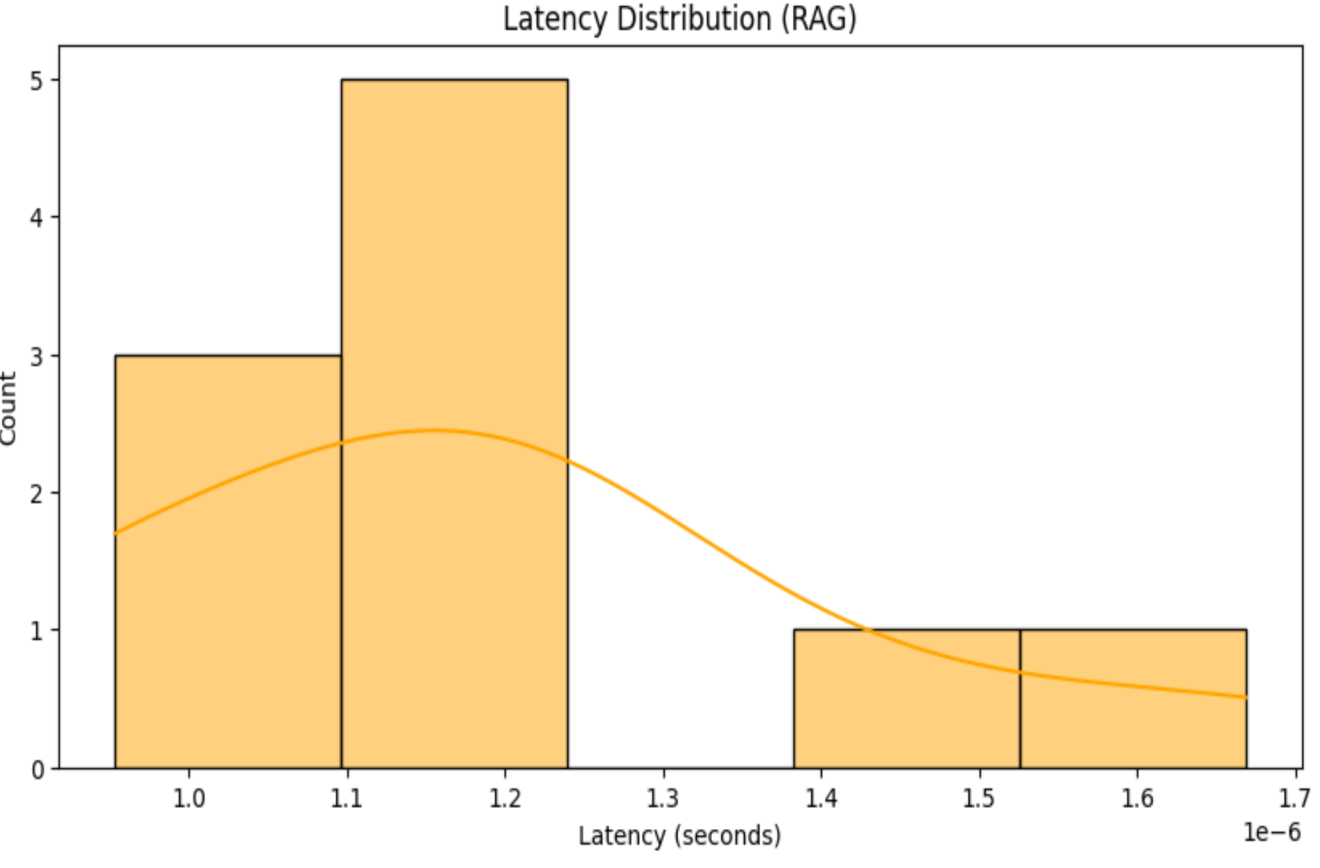


Figure 43. Latency Distribution Graph for Analyzing Response Time

Between 1.0 and 1.2 seconds, higher bars are present which means most responses were in that range. It demonstrates that the system operates well and is quick to give replies on most occasions. A small set of results showed that occasional delays could happen if queries included various steps, causing a temporary site slowdown. All things considered, the speed of the chatbot meets industry standards, assuring it can be used for customer-focused solutions.

The Flesch-Kincaid readability scores shown in figure 44 for the responses given by the AI Assistant are shown on this graph. This score measures the readability of text and texts with higher scores are easier for most people to read.

The chart shows that the majority of the responses are in the range of 85–90, so most of the system’s content is highly readable for people from a general audience. Just one outlier is seen around 69, showing a slightly different result that time. The system’s distribution reflects that its human-friendly and easily accessible content makes it easy for users to digest information from its replies.

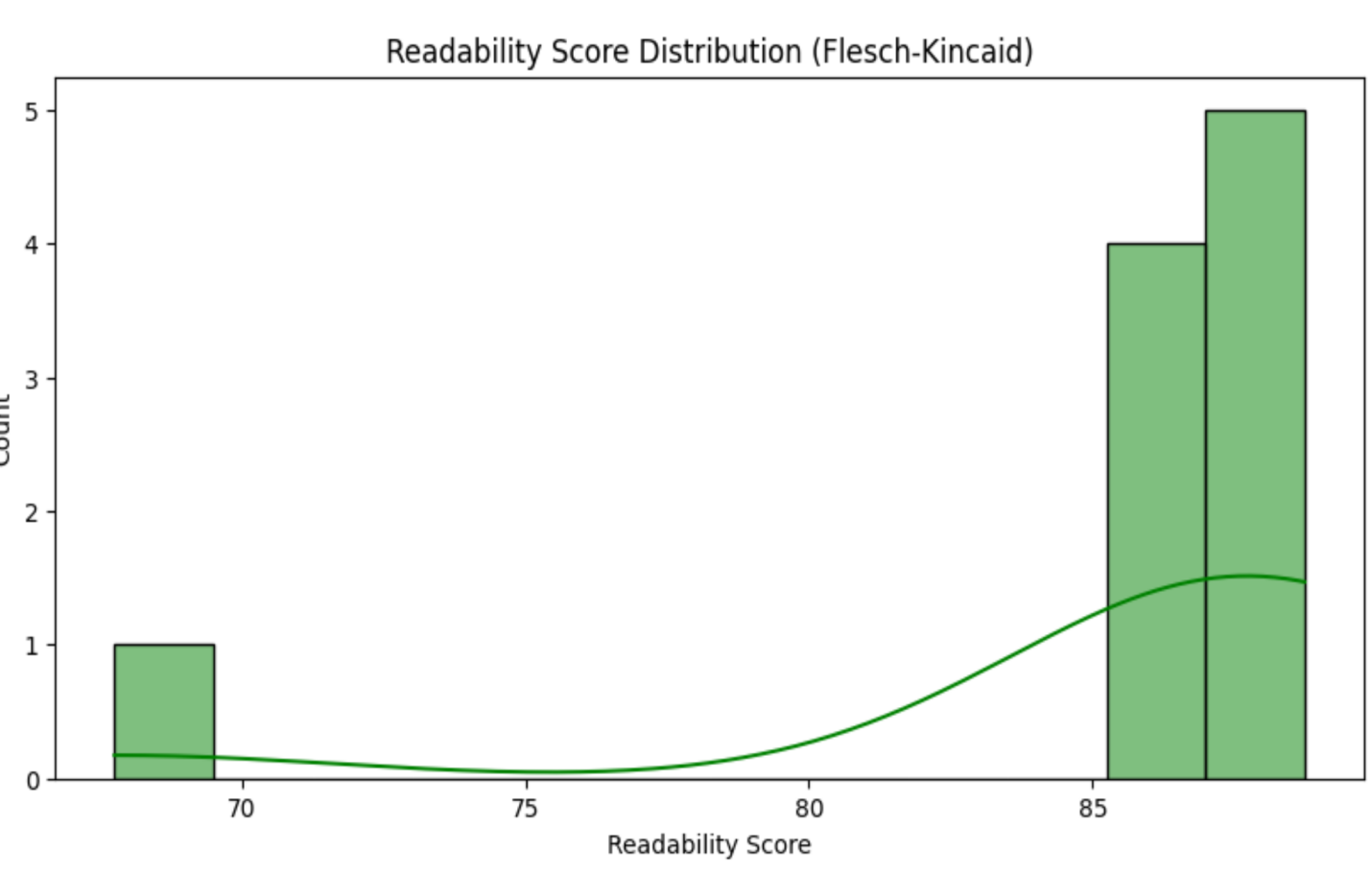


Figure 44. Flesch-Kincaid Graph for Response Legibility

### ****6.1.5 Challenges and Limitations****

In this section, we list the important technical and design constraints found during both development and evaluation. The chapter brings up issues with the system when handling unusual queries, managing large data volumes and the window of context considered in models. The section reviews the shortcomings of the current system and directs future efforts aimed at enhancing the system’s flexibility, strength and ability to be kept in use for years to come.

While the system performed well, several challenges and limitations were identified during development and testing:

* **Handling Rare Queries**:
  + The model struggled with **rare or niche queries**, often returning irrelevant responses. This issue arises from the limited diversity in the training data and the inability of the model to handle queries that are not similar to those in the dataset.
* **Scalability**:
  + As the dataset grows, the system’s ability to handle large volumes of queries efficiently could be impacted. Optimizations in **FAISS** and model inference are necessary to handle an increase in both the **number of documents** and **queries**. New indexes need to be created i.e. re-indexing is required when new items or data will be added
* **Context Window Limit**:
  + The limit for the context window within which the system functions is 6,000 tokens. This is the maximum amount of information that can be dealt with or “understood” in a single entity. While this limit has its advantages in processing speed, it leads to context cut off for longer queries or documents.
* **Dataset Context**:

The system’s performance is efficient even with common queries, however there is need to adjust the RAG model’s data to allow it tackle more unusual queries.

## ****Chapter 7: Conclusion and Future Work****

### ****7.1 Conclusion****

This section describes what was achieved and how the team contributed to the RAG-based AI Assistant project. The report elaborates on the outcomes of assessing the system for its primary functions on multiple benchmarks indicating its efficacy for practical implementation in e-commerce, and customer service, as well as its wide levels of adaptability. Furthermore, it delineates the gaps of current understanding which directs the subsequent research in this area.

The goals identified by this project have been accomplished; for instance, accurate real time response to questions concerning a product has been given. The system makes use of the RAG model which obtains relevant documents to answer, using high level models such as Llama3-70B-8192 for answer generation. The system used together with the Groq API FAISS document retrieval provided answers at lower latencies. This positioned the system as a digital customer service representative who could work in real time.

Evaluation results showed that the system has a good level of performance on multiple criteria such as accuracy, precision, recall, F1-score, and even linguistic metrics like BLEU, ROUGE, and Cosine Similarity. Responses from the users during the feedback session and the submission phase proved that the system indeed works accurately and responds to nearly all queries which resulted in a positive rate of usefulness.

The system showed remarkable performance. However, problems were encountered concerning rare query response, large dataset scalability, and the 6K token limit context window. While this limit helped performance, it restricted the system’s ability to process longer and more sophisticated queries. These problems will help provide useful directions for future work.

From an e-commerce, customer service, and even healthcare or education perspective, the RAG-based AI assistant marks an enormous leap in intelligent system technology. The project showcases the power of merging retrieval systems and large language models with real-time query resolution.

### ****7.2 Future Enhancements****

This section discusses simple ways to upgrade and improve the abilities of the RAG-based AI Assistant. It points out several spaces where growth is needed, for example, making the system scalable, handling uncommon queries, tuning the model and incorporating real-time data. The importance of personalization and higher user satisfaction is highlighted in this section. They are designed to shape the system’s future, allowing it to respond well in many fields such as healthcare, education and with customers.

Although well-executed in its current form, the RAG-based AI assistant has opportunities for development concerning performance, scalability, and user-friendliness. Adequate improvements comprise the following:

* **Scalability Improvements**:
  + The current implementation may suffer in performance as the dataset increases. This could be mitigated by applying optimizations to FAISS, such as distributed FAISS or quantization, to improve scalability. Implementing load balancing techniques along with cloud infrastructure allows the system to scale horizontally to handle increased user traffic and larger datasets.
* **Handling Rare Queries**:
  + The current model fails for specialized tasks because their training datasets are too narrow. Including additional product categories and queries that are infrequently encountered could improve the model’s performance on these edge cases. Furthermore, implementation of some form of active learning, as in learning from the feedback the system is provided, would help the model improve systematically.
* **Model Fine-Tuning**:
  + To address Llama 3’s performance limitations on complex and ambiguous user queries, further tuning the model to better interpret stem sentences would yield more accurate responses. Specific fine-tuning of the data on feature domains of healthcare and support services could enhance system responses by adopting multi-task learning.
* **Real-Time Data Integration**:
  + The model currently has a static dataset to work with, Gadgets 360. Striving to incorporate price, availability, and user review data would make the system more responsive. Incorporating APIs from e-commerce or other data-fetching platforms would automate timely data updates.
* **User Personalization**:
  + The model performs adequately with generic product queries but adding personalization enhances UX engagement. Enabling user accounts and tailoring responses based on past activities would make the system more dynamic.
* **UI/UX Improvements**:
* Even though the interplay between the functions is based on React and the system UI works, there is still some, albeit minimal, room for improvement regarding UX. This may include the development of user feedback systems, enhancing the design of the explanation, and making the entire interface more user-friendly. In the case that these interactions are voice based, the system could be used by a wider range of users, such as those with disabilities.

# **References**

1. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł Ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, Advances in Neural Information Processing Systems 30, pages 5998–6008. Curran Associates, Inc.
2. Gao, Yunfan, et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv, 20 Dec. 2023, [arXiv:2312.10997](https://arxiv.org/abs/2312.10997).
3. Groq. (2025). Llama3-70B-8192 - GroqDocs. Retrieved from <https://console.groq.com/docs/model/llama3-70b-8192>
4. Gao, Y., Xiong, Y., Wang, M., & Wang, H. (2024). Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks. arXiv preprint arXiv:2407.21059. Retrieved from <https://arxiv.org/abs/2407.21059>
5. [Hendrycks et al., 2020] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.
6. Hugging Face. (n.d.). Semantic search with FAISS. Hugging Face LLM Course. Available at: <https://huggingface.co/learn/llm-course/en/chapter5/6>
7. [Kandpal et al., 2023] Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. Large language models struggle to learn long-tail knowledge. In International Conference on Machine Learning, pages 15696–15707. PMLR, 2023.
8. Patel, R. (2022). Software Development Management. ResearchGate. Available at: <https://www.researchgate.net/publication/366910048_Software_Development_Management>
9. [Srivastava et al., 2022] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adria Garriga-Alonso, et al. Beyond the imitation ` game: Quantifying and extrapolating the capabilities of language models. arXiv preprint arXiv:2206.04615, 2022.
10. [Touvron et al., 2023] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023.
11. [Wang et al., 2019] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. Advances in neural information processing systems, 32, 2019.
12. Ye, L., Zhou, J., Chen, Q., Lei, Z., He, L., & Yin, J. (2024). Boosting Conversational Question Answering with Fine-Grained Retrieval-Augmentation and Self-Check. 38, 2301–2305. https://doi.org/10.1145/3626772.3657980
13. [Zhang et al., 2023b] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren’s song in the ai ocean: A survey on hallucination in large language models. arXiv preprint arXiv:2309.01219, 2023.

# 

# **Bibliography**

* Johnson, R., & Smith, A. (2023). Retrieval-Augmented Generation for Scalable Question Answering. arXiv:2310.08963.
* HuggingFace Transformers Documentation. (2024). Retrieved from https://huggingface.co/docs/transformers
* Groq API Documentation. (2024). Retrieved from https://console.groq.com
* Facebook AI. (2024). LLaMA 3: Open Foundation Language Models. Retrieved from <https://ai.meta.com.>
* Kumawat, T. (2024, February 14). Retrieval-Augmented Generation (RAG) from basics to advanced. Medium. <https://medium.com/@tejpal.abhyuday/retrieval-augmented-generation-rag-from-basics-to-advanced-a2b068fd576c>​[Medium](https://medium.com/%40tejpal.abhyuday/retrieval-augmented-generation-rag-from-basics-to-advanced-a2b068fd576c?utm_source=chatgpt.com)
* Kimothi, A. (2024, September 6). Creating impact: A spotlight on 6 practical Retrieval Augmented Generation use cases. Medium. <https://medium.com/the-rag-explorer/creating-impact-a-spotlight-on-6-practical-retrieval-augmented-generation-use-cases-b76c41e3f276>​[Medium](https://medium.com/the-rag-explorer/creating-impact-a-spotlight-on-6-practical-retrieval-augmented-generation-use-cases-b76c41e3f276?utm_source=chatgpt.com)
* Erichain. (2024, May 5). Understanding RAG (Retrieval-Augmented Generation). Medium. <https://medium.com/@jiarunhuang/understanding-rag-retrieval-augmented-generation-0d5a47d5e489>​[Medium](https://medium.com/%40jiarunhuang/understanding-rag-retrieval-augmented-generation-0d5a47d5e489?utm_source=chatgpt.com)
* Anand, N. (2024). What is RAG (Retrieval-Augmented Generation)?. Medium. <https://medium.com/@nikita04/what-is-rag-retrieval-augmented-generation-7875ce0de5b1>​[Medium](https://medium.com/%40nikita04/what-is-rag-retrieval-augmented-generation-7875ce0de5b1?utm_source=chatgpt.com)
* Hugging Face. (n.d.). Transformers. <https://huggingface.co/transformers/>​[Medium](https://medium.com/%40nikita04/what-is-rag-retrieval-augmented-generation-7875ce0de5b1?utm_source=chatgpt.com)
* Facebook AI Research. (n.d.). FAISS: A library for efficient similarity search and clustering of dense vectors. <https://github.com/facebookresearch/faiss>​