جـامـعـة ابــن طـفـيـل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵟⵓⴼⴰⵢⵍ
Ibn Tofaïl University
Faculté des Sciences

Université Ibn Tofail
Faculté des Sciences, Kénitra

Mémoire de Projet de Fin d'Etudes

Master Intelligence Artificielle et Réalité Virtuelle

End-to-End AI-Driven Chatbot for Optimizing Hotel Guest Interactions

Établissement d'accueil : **Tetricks**

Elaboré par : Mr. : EL BOURKI Otmane

Encadré par : Mr. : NOURI Anass (FSK-UIT)

Mr. : FITZPATRICK Sean (Tetricks)

Soutenu le 25 juillet 2024, devant le jury composé de :

- Pr. NOURI Anass (FSK Université Ibn Tofaïl)
- Pr. MESSOUSSI Rochdi (FSK Université Ibn Tofaïl)
- Pr. BOUKIR Khaoula (ENSC Université Ibn Tofaïl)
- Pr. BOUJIHA Tarik (FSK Université Ibn Tofaïl)

Année universitaire 2023/2024

Kingdom of Morocco
Ibn Tofail University
Faculty of Science, Kenitra

End-of-Studies Thesis

For the attainment of the Master's degree in Artificial Intelligence and Virtual Reality

Specialization: Artificial Intelligence

Host Organization: Tetricks

under the theme :

End-to-End AI-Driven Chatbot for Optimizing Hotel Guest Interactions

Prepared by:
Mr. Otmane El Bourki

Supervised by:
Mr. Anasse Nouri
(Ibn Tofail University)
Mr. Sean Fitzpatrick
(Tetricks)

Defended on July 25, 2024

Academic year 2023/2024

Dedication

“

TO my beloved family, for your unconditional love, unwavering support, and constant encouragement.

TO my parents, for your sacrifices and for instilling in me the values of hard work, perseverance, and humility.

TO my siblings, for your motivating words and steadfast belief in me.

TO all my dear ones, to each and every one of you,

Thank you.

”

- Otmane El Bourki

Acknowledgments

First and foremost, I thank Allah, the Almighty, for granting me the courage and patience necessary to complete this work.

I extend my heartfelt thanks to my supervisor, **Dr. ANASS Nouri**, for his invaluable assistance, patience, and encouragement. His critical insights were crucial in structuring this work and enhancing the quality of various sections.

I also wish to express my deep gratitude to my CEO, **Mr. SEAN Fitzpatrick** at Tetricks, for his immense support, meticulous guidance, and the wealth of advice and information he provided with unmatched patience and professionalism.

My sincere thanks go to **Mr. REMCO Van Uitert**, full-stack developer at Tetricks, for his constant support and invaluable contributions to this project.

Lastly, I would like to thank everyone who contributed, directly or indirectly, to the completion of this work.

Abstract

During my tenure at Tetricks from January to April 2024, I spearheaded the development of a Retrieval-Augmented Generation (RAG) chatbot designed to enhance hotel guest interactions. This project aimed to revolutionize customer service by providing quick and accurate responses to inquiries about hotel services and amenities.

The project commenced with a detailed exploration of the financial implications of deploying Language Models (LLMs), comparing proprietary solutions to self-managed options on cloud platforms like Google Cloud Platform (GCP). A comprehensive cost-benefit analysis led to the decision to implement a self-managed LLM configuration on GCP, ensuring optimal performance while adhering to budget constraints.

My primary responsibility was the implementation of the RAG-based chatbot. This involved designing a robust system architecture and developing a sophisticated backend API, enabling the chatbot to leverage advanced Natural Language Processing (NLP) techniques for understanding and generating responses. Simultaneously, the system retrieved relevant information from a knowledge base specific to hotel operations. Deployment on GCP ensured scalability and reliability, laying the foundation for further enhancements.

Throughout February and March, I optimized the chatbot's data ingestion processes and refined its data preprocessing layer. This included testing various data ingestion methods and implementing a robust preprocessing layer to enhance the quality of inputs feeding into the LLM. These efforts significantly improved the chatbot's efficiency and accuracy in handling client queries.

Continued development efforts in March and April expanded the chatbot's capabilities to include multilingual support, facilitating seamless query management across multiple languages. This enhancement broadened the chatbot's accessibility and appeal to international clients, thereby enhancing overall customer satisfaction.

In conclusion, the project delivered a robust RAG-based chatbot solution that significantly improved customer interactions in the hospitality sector. It showcased my expertise in leveraging advanced NLP and machine learning techniques to develop impactful AI solutions tailored to real-world applications.

Keywords: RAG Chatbot, Natural Language Processing, Customer Interaction Management, GCP Deployment.

Contents

List of Figures

List of Abbreviations and Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
GCP	Google Cloud Platform
LLM	Large Language Model
NLP	Natural Language Processing
RAG	Retrieval-Augmented Generation
UI	User Interface
MLOps	Machine Learning Operations
VM	Virtual Machine
AWS	Amazon Web Services
API	Application Programming Interface
TF	TensorFlow
PyTorch	Python Torch
DNN	Deep Neural Network
ML	Machine Learning
DL	Deep Learning

Generale Introduction

Context

The hotel industry is undergoing significant transformations driven by technological advancements and evolving guest expectations. With the proliferation of online travel agencies (OTAs) and the increasing preference for mobile bookings, hotels face the dual challenge of enhancing guest experiences while maintaining operational efficiency. Industry reports indicate that hotels worldwide are struggling to meet these modern demands due to rising competition, fluctuating demand, and escalating operational costs. As guests increasingly seek seamless and personalized experiences, there is a pressing need for innovative solutions that can address these challenges.

Problematic

Despite the advancements in technology, hotels continue to grapple with several persistent issues. The shift towards digital interactions has intensified competition, making it difficult for hotels to maintain customer loyalty and direct bookings amidst the dominance of OTAs. Additionally, the need for round-the-clock customer support and personalized guest experiences has placed considerable strain on hotel staff, leading to increased operational costs and inefficiencies. Traditional methods of guest interaction are no longer sufficient to meet the high expectations of modern travelers, necessitating the adoption of more sophisticated and automated solutions.

Objectives

This project aims to develop and deploy an AI-driven chatbot specifically tailored for the hospitality sector, with the following objectives:

Enhanced Customer Support

The chatbot will provide 24/7 customer support to handle a wide range of guest queries and requests. It will be capable of:

- Accurately understanding and interpreting user requests and intentions through natural language conversations.

- Providing relevant and personalized responses based on the context of the conversation.
- Seamlessly integrating into business customer management systems, thereby enhancing operational efficiency and customer satisfaction.
- Continuously evolving and adapting through machine learning, improving the quality of interactions over time.

Improved Guest Satisfaction

By delivering prompt and accurate responses, the chatbot aims to enhance guest satisfaction levels. Features include:

- Real-Time Assistance: Providing immediate assistance without requiring guests to wait for a human response, thereby improving overall service efficiency.
- Feedback Collection: Gathering feedback from guests throughout their stay to continuously improve service quality and personalize guest experiences.

Operational Efficiency and Cost Reduction

The implementation of the AI-driven chatbot will also focus on:

- Automating routine tasks such as check-in/out procedures, room service requests, and billing inquiries, allowing staff to focus on more personalized guest interactions.
- Reducing operational costs associated with customer service by minimizing the need for extensive human intervention.
- Improving revenue through increased direct bookings and upselling opportunities facilitated by the chatbot's personalized recommendations and interactions.

Presentation of the Host Organization

Tetricks, a pioneering hospitality technology company, specializes in intelligently managing hotel room assignments based on guest preferences and operational efficiencies. Founded with the support of École Polytechnique and as a laureate of the 2022 HEC Digital Entrepreneurship program, Tetricks is poised to revolutionize how hotels optimize their guest experiences through advanced AI-driven solutions. Tetricks understands customer demands and intelligently assigns hotel rooms. Supported by École Polytechnique and winner of HEC Paris's Digital Entrepreneurship program.

TETRICKS

Figure 1: Tetricks Logo.

Technology and Innovation

Tetricks stands out with its advanced use of artificial intelligence to improve operational efficiency and customer experience. By integrating sophisticated algorithms, Tetricks dynamically adjusts room assignments based on real-time customer preferences and data, ensuring optimal management of hotel resources.

Utilizing AI in the Hotel Industry

As an early-stage startup, Tetricks envisions transforming the hotel industry by leveraging artificial intelligence in various aspects of hotel management. Key AI-driven initiatives include:

Personalized Guest Experiences

By analyzing data from previous stays, preferences indicated during bookings, and real-time feedback, Tetricks' AI system can tailor services to meet individual guest needs, creating a personalized and memorable experience for each guest.

Predictive Maintenance and Resource Management

AI algorithms predict maintenance needs and optimize resource allocation, reducing downtime and ensuring that hotel facilities are always in top condition. This proactive approach not only enhances guest satisfaction but also reduces operational costs.

Dynamic Pricing and Revenue Management

Tetricks uses AI to analyze market trends, competitor pricing, and historical booking data to dynamically adjust room rates. This ensures competitive pricing while maximizing revenue, adapting to fluctuations in demand with precision.

Enhanced Customer Support

Through AI-powered chatbots and automated systems, Tetricks provides 24/7 customer support, addressing common inquiries, handling bookings, and resolving issues promptly. This enhances the overall guest experience by ensuring immediate assistance and reducing wait times.

Operational Efficiency

AI-driven analytics help hotel managers make informed decisions about staffing, inventory, and other operational aspects. By identifying patterns and predicting future needs, Tetricks ensures that hotels run smoothly and efficiently.

Conclusion

The hospitality industry is at a pivotal moment, grappling with evolving guest expectations and operational complexities. AI-driven technologies, particularly chatbots powered by natural language processing and machine learning, offer a transformative solution. These innovations promise to not only enhance guest satisfaction but also streamline operations and reduce costs by automating routine tasks and delivering personalized, real-time assistance.

This thesis aims to contribute to this transformative wave by developing and deploying an AI-driven chatbot specifically tailored for the hospitality sector. Focused on improving customer support, elevating guest experiences, and seamlessly integrating with existing hotel management systems, this project explores state-of-the-art technologies in chatbots, conducts a detailed requirements analysis, proposes robust design principles, outlines effective implementation strategies, and advocates for rigorous testing methodologies.

By demonstrating the potential of AI to optimize staff resources, improve service efficiency, and meet the dynamic needs of modern hotel guests and operators, this thesis underscores the strategic importance of AI applications in hospitality. It seeks to provide actionable insights and practical guidance for adopting AI-driven solutions effectively, thereby fostering innovation and setting new benchmarks for service excellence in the hospitality landscape.

Chapter 1

State of the Art in Chatbots

1.1 Introduction

The evolution of chatbots has been marked by significant advancements in natural language processing (NLP) and machine learning. Retrieval-Augmented Generation (RAG) chatbots represent a hybrid approach that leverages the strengths of both generative and retrieval-based models, addressing several limitations of previous methods. This chapter provides a comprehensive overview of the state-of-the-art in RAG chatbots, tracing their historical development, underlying technologies, key methodologies, and applications.

1.2 Overview of Chatbots

1.2.1 Definition and Types

Chatbots are software applications designed to simulate human conversation through text or voice interactions. They can be broadly categorized into two types: rule-based and AI-based chatbots.

Rule-Based Chatbots: These rely on predefined rules and scripts to generate responses. They follow simple ”if-then” logic and are limited in their ability to handle complex or unexpected queries.

AI-Based Chatbots: These use machine learning algorithms to understand and generate human-like responses. AI-based chatbots are further divided into generative models and retrieval-based models.

Historical Development

The development of chatbots dates back to the 1960s with the creation of ELIZA, a rule-based system designed to mimic a psychotherapist. In the 1990s, ALICE (Artificial Linguistic Internet Computer Entity) further advanced the capabilities of rule-based chatbots using AIML (Artificial Intelligence Markup Language). The introduction of machine learning and deep learning techniques marked a significant shift in chatbot development. In the early 2010s, models like Cleverbot and Siri began to leverage these techniques for more sophisticated interactions. The development of deep learning-based NLP models, such as Word2Vec and GloVe, paved the way for the creation of more advanced AI chatbots.

Significance and Applications

Chatbots have found applications across various industries, including customer service, healthcare, education, and e-commerce. They enhance user engagement, provide 24/7 support, and improve operational efficiency.

1.3 Generative and Retrieval-Based Chatbots

1.3.1 Generative Models

Overview

Generative models generate responses by predicting the next word in a sequence based on the context provided by the user input. These models are trained on large datasets and can generate coherent and contextually relevant responses.

Examples

Prominent generative models include GPT-3 and GPT-4 by OpenAI, BERT by Google, and T5 by Google Research.

Strengths and Limitations

Generative models excel in creating fluent and contextually appropriate responses. However, they can produce plausible but incorrect answers, and their responses may sometimes lack factual accuracy.

1.3.2 Retrieval-Based Models

Overview

Retrieval-based models select the most appropriate response from a predefined set based on the user input. These models use techniques like similarity search and ranking algorithms to retrieve relevant responses.

Examples

Techniques such as Dense Passage Retrieval (DPR) and BM25 are commonly used in retrieval-based models.

Strengths and Limitations

Retrieval-based models provide accurate and relevant responses but are limited by their reliance on predefined responses and may struggle with novel queries.

1.4 Introduction to Retrieval-Augmented Generation (RAG)

1.4.1 Concept and Benefits

Retrieval-Augmented Generation (RAG) combines the strengths of generative and retrieval-based models. It retrieves relevant information from a large corpus and uses this information to generate more accurate and contextually appropriate responses. This approach helps mitigate the limitations of purely generative models by grounding responses in factual information.

How RAG Works

RAG models consist of two main components: a retriever and a generator. The retriever searches a database to find relevant documents or passages based on the user query. The generator then uses this retrieved information to generate a coherent and accurate response.

1.5 Core Technologies and Frameworks for RAG

1.5.1 RAG Models

Facebook's RAG model is a notable example, combining Dense Passage Retrieval (DPR) with a generative model like BART or T5.

1.5.2 Embedding Techniques

Embedding techniques, such as mixedbread-ai/mxbai-embed-large-v1, are used to create dense vector representations of text, enabling effective semantic search and retrieval.

1.5.3 Vector Databases

Vector databases like Qdrant, FAISS, and Pinecone are essential for storing and retrieving dense vector representations efficiently.

1.5.4 Libraries and Frameworks

Tools such as Langchain, Haystack, and Hugging Face Transformers facilitate the development and deployment of RAG chatbots by providing pre-trained models and easy-to-use interfaces.

1.6 Key Techniques and Methodologies

1.6.1 Retrieval Techniques

Effective retrieval techniques involve creating and searching dense vector representations of text. Dense Passage Retrieval (DPR) is a popular method, utilizing bi-encoder architecture to encode queries and passages into vectors for efficient retrieval.

1.6.2 Integration of Retrieval and Generation

Integrating retrieval results into generative models involves using the retrieved information as context for the generator. This can be done through techniques like concatenation or attention mechanisms, where the generator pays attention to the relevant parts of the retrieved information.

1.6.3 Evaluation Metrics

Evaluating RAG chatbots involves metrics such as BLEU (Bilingual Evaluation Understudy), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), and F1 Score. These metrics measure the quality, relevance, and accuracy of the generated responses.

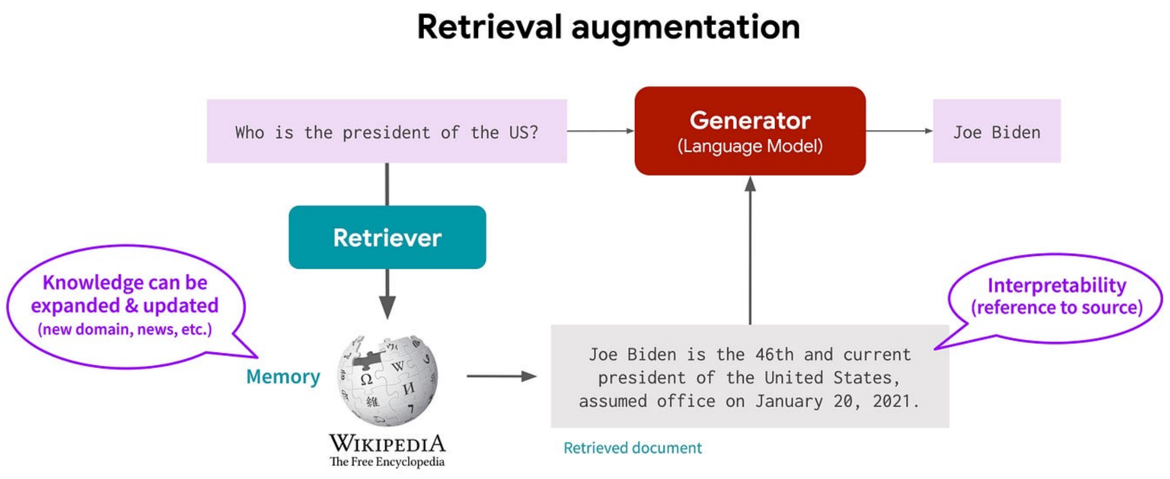

Retrieval augmentation
Who is the president of the US?
Generator
(Language Model)
Joe Biden
Retriever
Knowledge can be expanded & updated (new domain, news, etc.)
Memory
Wikipedia
The Free Encyclopedia
Joe Biden is the 46th and current president of the United States, assumed office on January 20, 2021.
Retrieved document
Interpretability (reference to source)

Figure 1.1: Rag diagram.

1.7 Applications and Use Cases

1.7.1 Customer Support

RAG chatbots are widely used in customer service to provide accurate and timely responses to customer queries, enhancing user satisfaction and reducing operational costs.

1.7.2 Healthcare

In healthcare, RAG chatbots assist in providing medical information, scheduling appointments, and offering preliminary diagnoses based on symptoms reported by users.

1.7.3 E-commerce

E-commerce platforms use RAG chatbots for product recommendations, order tracking, and handling customer inquiries, thereby improving the shopping experience.

1.7.4 Education

Educational applications of RAG chatbots include tutoring systems that provide personalized learning experiences and answer student queries with accurate information.

1.8 Challenges and Limitations of RAG

1.8.1 Scalability

Scaling RAG chatbots to handle large volumes of queries and data requires significant computational resources and efficient architectures.

1.8.2 Data Quality and Bias

The quality of data used for training and retrieval is crucial. Bias in training data can lead to biased responses, which is a significant concern in developing fair and ethical AI systems.

1.8.3 Latency and Efficiency

Ensuring low latency and high efficiency in RAG chatbots involves optimizing retrieval and generation processes. Techniques like approximate nearest neighbor (ANN) search can help improve retrieval speed.

1.8.4 Interpretability and Explainability

The need for interpretable and explainable AI is critical, especially in sensitive applications like healthcare and finance. Developing methods to explain how RAG chatbots generate their responses is an ongoing research area.

1.9 Future Directions and Research Opportunities

1.9.1 Advancements in NLP

Future advancements in NLP, such as better language models and improved embedding techniques, will enhance the capabilities of RAG chatbots.

1.9.2 Improved Integration Techniques

Research on more effective ways to integrate retrieval and generation components will lead to more accurate and contextually relevant responses.

1.9.3 Enhanced Evaluation Methods

Developing more robust and comprehensive evaluation metrics will improve the assessment of RAG chatbot performance and help in fine-tuning models.

1.9.4 Ethical Considerations

Addressing ethical issues, such as data privacy, bias, and transparency, is crucial for the responsible deployment of RAG chatbots. Ongoing research in ethical AI will contribute to developing fair and trustworthy systems.

1.10 Conclusion

The state of the art in RAG chatbots represents a significant advancement in chatbot technology, combining the strengths of generative and retrieval-based models to provide accurate, relevant, and contextually appropriate responses. Continuous research and development in this field will lead to further improvements in chatbot capabilities and applications, addressing current challenges and exploring new opportunities.

Chapter 2

Needs Analysis

2.1 Overview

The purpose of this needs analysis is to comprehensively identify and evaluate the requirements for deploying a Retrieval-Augmented Generation (RAG) chatbot to enhance guest interaction in a hotel setting. The analysis is based on data collected through surveys, interviews with hotel staff and guests, focus groups, and analysis of existing hotel operations data. This report will serve as a foundational document to guide the design, development, and implementation of the chatbot.

2.2 Guest Needs Analysis

2.2.1 Identifying Common Guest Queries and Requests

Hotel guests frequently have a variety of queries and requests that need to be addressed promptly and accurately to ensure a pleasant stay. The most common queries include:

- **Check-in/check-out processes**: Guests often inquire about check-in and check-out times, procedures, and options for early check-in or late check-out.
- **Room service orders**: Guests request meals, beverages, and other services to be delivered to their rooms.
- **Housekeeping requests**: Guests request additional towels, toiletries, room cleaning, and maintenance services.
- **Information about hotel amenities and services**: Guests seek information about the hotel's facilities such as the gym, spa, pool, restaurants, and business center.
- **Local attractions and directions**: Guests ask for recommendations on local attractions, restaurants, transportation options, and directions.
- **Booking modifications**: Guests need assistance with extending their stay, changing room types, or canceling reservations.
- **Frequently asked questions (FAQs)**: Questions about Wi-Fi access, parking, hotel policies, and event schedules.

2.2.2 Pain Points in Current Guest Interaction Processes

Current guest interaction processes often involve the following pain points:

- **Delays in response time from staff**: Guests experience waiting times when staff are occupied, leading to frustration and dissatisfaction.
- **Language barriers and communication issues**: Guests and staff may face challenges communicating effectively due to language differences.

- **Inconsistent information provided by different staff members**: Guests may receive varying responses to the same query, causing confusion.
- **Limited availability of staff, especially during peak hours or late nights**: Guests may struggle to get assistance during busy times or outside regular working hours.

2.2.3 Desired Improvements with Chatbot Implementation

Deploying a RAG chatbot can significantly improve guest interaction by:

- **Faster response times**: Providing instant responses to guest queries, reducing waiting times.
- **Consistent and accurate information**: Ensuring uniformity in the information provided to guests.
- **Multilingual support**: Offering assistance in multiple languages, enhancing communication with international guests.
- **Enhanced convenience and accessibility**: Enabling guests to access services and information anytime, anywhere.
- **Personalized interactions based on guest preferences**: Tailoring responses and recommendations to individual guest profiles.

2.3 Functional Requirements

2.3.1 Real-Time Response Capability

The chatbot must be capable of providing instantaneous responses to guest queries, handling multiple simultaneous interactions without delays. This requires robust natural language processing (NLP) capabilities to understand and respond accurately to diverse guest inputs.

2.3.2 Multilingual Support

To cater to a global clientele, the chatbot should support multiple languages, including but not limited to English, Spanish, French, German, Chinese, and Japanese. It should automatically detect the guest's language preference and switch accordingly.

2.3.3 Integration with Hotel Management Systems

The chatbot needs to seamlessly integrate with the hotel's existing management systems, including property management systems (PMS), booking engines, room service management systems, and housekeeping systems. This ensures real-time synchronization and accurate processing of guest requests.

2.3.4 Personalized Interaction

Utilizing guest profile information, the chatbot should deliver personalized responses and recommendations. This includes remembering past interactions, preferences, and special requests to enhance the guest experience.

2.3.5 Availability

The chatbot must be available around the clock, ensuring that guests can access services and information at any time, including late nights and peak hours. This improves the overall guest experience by providing continuous support.

2.4 Non-Functional Requirements

2.4.1 Security and Privacy

The chatbot must ensure the security and privacy of guest data. This includes implementing encryption for data transmission and storage, adhering to data protection regulations such as GDPR, and maintaining stringent access controls to prevent unauthorized access.

2.4.2 Scalability

The chatbot should be scalable to handle increasing numbers of users and interactions, especially during peak seasons. This involves using scalable cloud-based infrastructure that can dynamically adjust to varying loads.

2.4.3 Reliability and Uptime

High reliability and minimal downtime are critical for the chatbot's effectiveness. This requires robust infrastructure with redundancy and failover mechanisms to ensure continuous operation even in the event of hardware or software failures.

2.4.4 User-Friendly Interface

The chatbot interface must be intuitive and easy to navigate, providing a seamless experience across multiple devices such as smartphones, tablets, and in-room devices. This includes clear prompts, easy-to-follow instructions, and visually appealing design.

2.5 Stakeholder Needs Analysis

2.5.1 Hotel Guests

Guests expect quick, accurate, and personalized responses to their queries. They prefer a conversational, human-like experience and need to feel comfortable and secure sharing information with the chatbot.

2.5.2 Hotel Staff

Front Desk Staff

The chatbot can reduce the front desk staff's workload by handling routine queries, allowing them to focus on more complex guest interactions. Staff will need training on how to leverage the chatbot for improved guest service.

Concierge Services

The chatbot can assist concierge staff by providing local recommendations and handling simple requests. This enables concierge staff to focus on more personalized and high-value services.

Housekeeping

Housekeeping staff can benefit from streamlined communication and efficient task assignment through chatbot integration. This ensures timely responses to guest requests and better task management.

Management

Management needs real-time monitoring and analytics on guest interactions, insights into common guest issues, and integration with CRM systems for enhanced guest relationship management. This helps in making data-driven decisions to improve overall guest experience.

2.6 Interaction Scenarios

2.6.1 Pre-Arrival

- **Automated booking confirmations**: The chatbot can send instant booking confirmations and provide information on hotel policies and services.

- **Handling special requests**: Guests can use the chatbot to request early check-in, airport transfers, or specific room preferences, which the chatbot processes and communicates to the relevant staff.

2.6.2 Onsite Experience

- **Room service ordering**: Guests can order meals and services through the chatbot, which forwards requests to the kitchen or housekeeping.
- **Information about hotel facilities and events**: The chatbot provides details about hotel amenities, events, and schedules, helping guests plan their activities.
- **Local recommendations and directions**: The chatbot offers suggestions on local attractions, restaurants, and provides directions, enhancing the guest's local experience.

2.6.3 Post-Departure

- **Collecting guest feedback and reviews**: The chatbot solicits feedback from guests after their stay, helping the hotel gather valuable insights.
- **Updating guests on loyalty programs and future promotions**: The chatbot keeps guests informed about loyalty program updates, future promotions, and encourages repeat bookings.

2.7 Desired Features and Capabilities

2.7.1 Natural Language Understanding (NLU)

The chatbot must have advanced NLU capabilities to accurately comprehend diverse guest queries, including those with varied phrasing and context. This ensures precise and relevant responses.

2.7.2 Contextual Awareness

The chatbot should maintain context across multiple interactions, recognizing returning guests and previous conversations to provide a seamless and personalized experience.

2.7.3 Proactive Assistance

The chatbot can offer proactive suggestions based on guest behavior, such as recommending dining options during meal times or sending reminders about upcoming events.

2.7.4 Multimodal Interaction

Supporting text, voice, and potentially visual inputs, the chatbot should provide a consistent experience across different modes of interaction, accommodating guest preferences and accessibility needs.

2.8 Summary of Findings

The needs analysis reveals that a RAG chatbot can significantly enhance hotel guest interaction by addressing common queries promptly, reducing staff workload, providing consistent and accurate information, and offering personalized and multilingual support. The functional and non-functional requirements identified will guide the design and implementation of a chatbot that meets the diverse needs of hotel guests and staff.

2.9 Conclusion

Deploying a RAG chatbot in a hotel setting addresses critical pain points in guest interaction, improves operational efficiency, and enhances overall guest satisfaction. By ensuring real-time, personalized, and secure interactions, the chatbot can become an indispensable tool for modern hotel management, offering a superior and seamless experience for guests and staff alike

Chapter 3

Design and Implementation

3.1 Design

3.1.1 Introduction

In this chapter, we provide an overview of the architecture and design goals of our AI-driven chatbot tailored for hotels. The chatbot aims to enhance customer support, streamline interactions, and seamlessly integrate with hotel management systems to improve overall guest experience.

3.1.2 System Architecture

This section details the key components, interactions, and data flows within the chatbot system:

Overview

The chatbot system consists of interconnected components designed to handle user interactions, retrieve and generate responses, manage context, and integrate with hotel systems.

Key Components

User Interface (UI)

- **Purpose:** Provides a user-friendly interface for guests to interact with the chatbot.
- **Features:** Supports natural language input and displays responses in a readable format.
- **Implementation:** Utilizes web-based interfaces (e.g., React, Vue.js) or mobile applications for accessibility.

Retrieval Module

- **Purpose:** Retrieves relevant information from hotel databases and knowledge bases.
- **Techniques:** Implements keyword-based retrieval (e.g., TF-IDF, BM25) and neural retrievers (e.g., DPR, Sentence-BERT) for accuracy.
- **Integration:** Connects with Elasticsearch for efficient retrieval of structured and unstructured data.

Generation Module

- **Purpose:** Generates contextually relevant responses based on retrieved information.
- **Models:** Employs pre-trained language models (e.g., GPT-4, T5) fine-tuned on hotel-specific data.
- **Customization:** Adapts responses to user queries, incorporating hotel-specific terminology and contextual relevance.

Context Manager

- **Purpose:** Manages conversation history and ensures coherent responses.
- **Components:** Includes dialogue state trackers (e.g., Rasa, Dialogflow) and memory networks for context persistence.
- **Functionality:** Integrates retrieved context and generated content to maintain conversational flow and relevance.

Integration Layer

- **Purpose:** Interfaces with hotel management systems (e.g., reservation systems, billing APIs).
- **Data Retrieval:** Retrieves real-time information (e.g., room availability, booking status) for personalized responses.
- **Security:** Ensures secure API communication and data encryption to protect guest information and maintain privacy.

Scalability and Load Balancing

- **Scalability:** Deploys on cloud platforms (e.g., AWS, GCP) with Docker containers and Kubernetes for efficient scaling.
- **Load Balancing:** Distributes user requests across multiple instances to optimize resource utilization and maintain performance.

3.1.3 Design Decisions

This section discusses the rationale behind architectural choices, including technology selection and scalability considerations:

Technology Stack

- **Programming Language:** Python chosen for its versatility and rich ecosystem in natural language processing.
- **Vectorstore:** Qdrant selected for efficient retrieval of structured and unstructured data.
- **Large Language Models:** Openchat utilized for generating natural language responses, fine-tuned on hotel-specific datasets.

Scalability Considerations

- **Cloud Deployment:** Deployed on GCP for scalability, utilizing Docker containers and Kubernetes orchestration.

Security Measures

- **Data Protection:** Secure APIs and encryption mechanisms implemented to protect guest information and ensure privacy.
- **Authentication:** OAuth used for user authentication and access control.

Usability and Integration

- **User Interface Design:** Designed for ease of use and accessibility, supporting multiple languages.
- **Integration with Hotel Systems:** APIs and webhooks integrated with hotel management systems for seamless data exchange and real-time updates.

3.2 Large Language Models (LLMs)

3.2.1 Overview of LLMs

Large Language Models (LLMs) are neural network-based models that have been trained on extensive text data. These models, especially when built using transformer architectures, exhibit exceptional capabilities in understanding and generating human language. They are employed in a variety of applications, including chatbots, translation services, content generation, and more.

3.2.2 OpenChat: Advancing Open-source Language Models with Mixed-Quality Data

Introduction to OpenChat

OpenChat is an open-source language model designed to enhance the capabilities of LLMs using mixed-quality data. Unlike traditional models that rely on high-quality curated datasets, OpenChat leverages a combination of high-quality and lower-quality data to improve robustness and performance.

Architecture and Training

OpenChat is built on transformer architecture, similar to other state-of-the-art LLMs like GPT-3. The key difference lies in its training methodology:

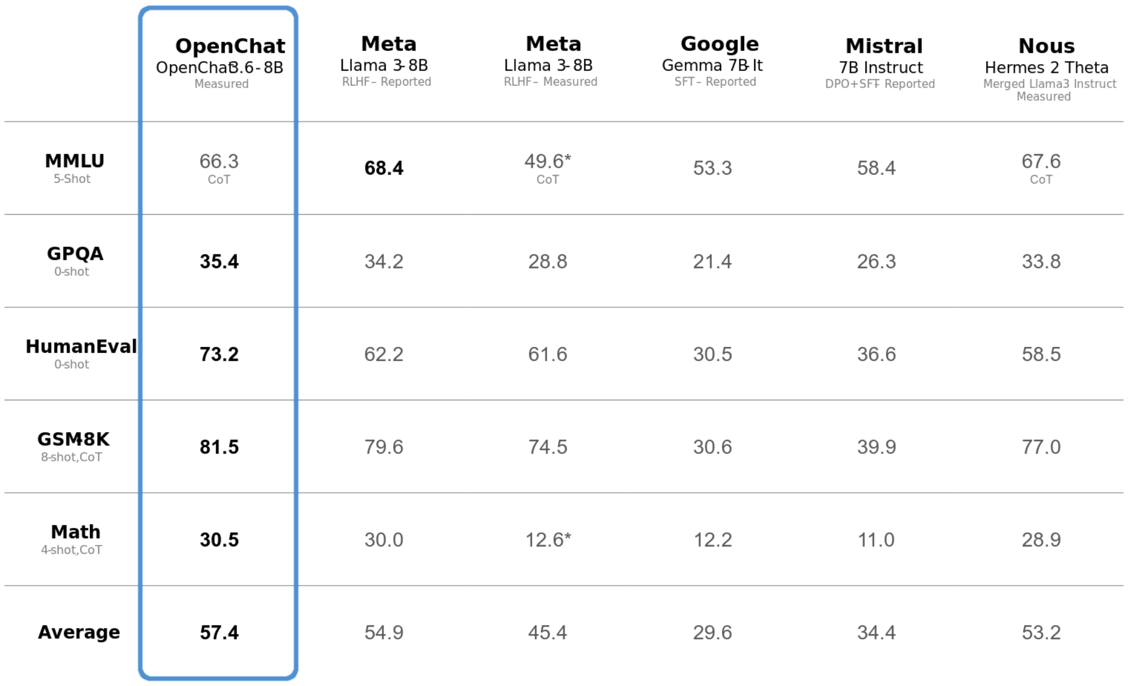
- **Data Collection:** OpenChat aggregates data from diverse sources, including web scraping, user-generated content, and curated datasets. This results in a rich and varied training corpus that includes both high-quality and mixed-quality data.
- **Data Preprocessing:** Advanced preprocessing techniques are used to filter and clean the data, ensuring that the model can learn effectively from the mixed-quality input. This includes tokenization, normalization, and noise reduction strategies.
- **Training Procedure:** The model is trained using a two-phase approach. In the first phase, the model learns from high-quality data to establish a strong foundational understanding. In the second phase, it is fine-tuned with mixed-quality data to enhance its ability to handle diverse and potentially noisy inputs.

Performance and Capabilities

OpenChat has demonstrated significant improvements in several NLP tasks, including:

- **Text Generation:** Produces coherent and contextually relevant text, even when prompted with ambiguous or poorly structured inputs.
- **Conversation Handling:** Exhibits enhanced ability to manage conversations, making it suitable for chatbot applications where user inputs can vary widely in quality.
- **Adaptability:** Shows robustness in understanding and generating language across different domains and styles, attributed to its exposure to mixed-quality data during training.

Benchmarks



	OpenChat OpenChat3.6-8B Measured	**Meta** Llama 3-8B RLHF - Reported	**Meta** Llama 3-8B RLHF - Measured	**Google** Gemma 7B-It SFT - Reported	**Mistral** 7B Instruct DPO+SFT Reported	**Nous** Hermes 2 Theta Merged Llama3 Instruct Measured
MMLU 5-Shot	66.3 CoT	**68.4**	49.6* CoT	53.3	58.4	67.6 CoT
GPQA 0-shot	**35.4**	34.2	28.8	21.4	26.3	33.8
HumanEval 0-shot	**73.2**	62.2	61.6	30.5	36.6	58.5
GSM8K 8-shot,CoT	**81.5**	79.6	74.5	30.6	39.9	77.0
Math 4-shot,CoT	**30.5**	30.0	12.6*	12.2	11.0	28.9
Average	**57.4**	54.9	45.4	29.6	34.4	53.2

Figure 3.1: Benchmark results of Openchat-3.6 against other LLMs.

Applications

OpenChat's versatility makes it applicable in numerous fields:

- **Customer Service:** Enhances automated customer support systems with its improved conversation handling capabilities.
- **Content Creation:** Assists in generating high-quality content for blogs, social media, and marketing materials.
- **Education:** Provides intelligent tutoring systems with the ability to understand and respond to diverse student queries.

3.2.3 Challenges and Future Directions

Despite its advancements, OpenChat faces challenges such as:

- **Data Quality Management:** Balancing the inclusion of mixed-quality data without compromising the overall performance.
- **Ethical Considerations:** Addressing biases and ensuring the responsible use of the model in applications.

3.3 Embedding Models

3.3.1 Overview of Embeddings

Embeddings are dense vector representations of words or phrases that capture their semantic meanings. These representations allow for efficient and effective processing of text data in various NLP tasks.

3.3.2 mxbai-embed-large-v1: Advanced Embedding Model

Introduction to mxbai-embed-large-v1

The mxbai-embed-large-v1 model is a state-of-the-art embedding model designed to provide high-quality semantic representations of text. It leverages advanced techniques to generate embeddings that capture nuanced relationships between words and phrases.

Architecture and Training

The architecture of mxbai-embed-large-v1 is based on transformer networks, which have proven to be highly effective for embedding tasks:

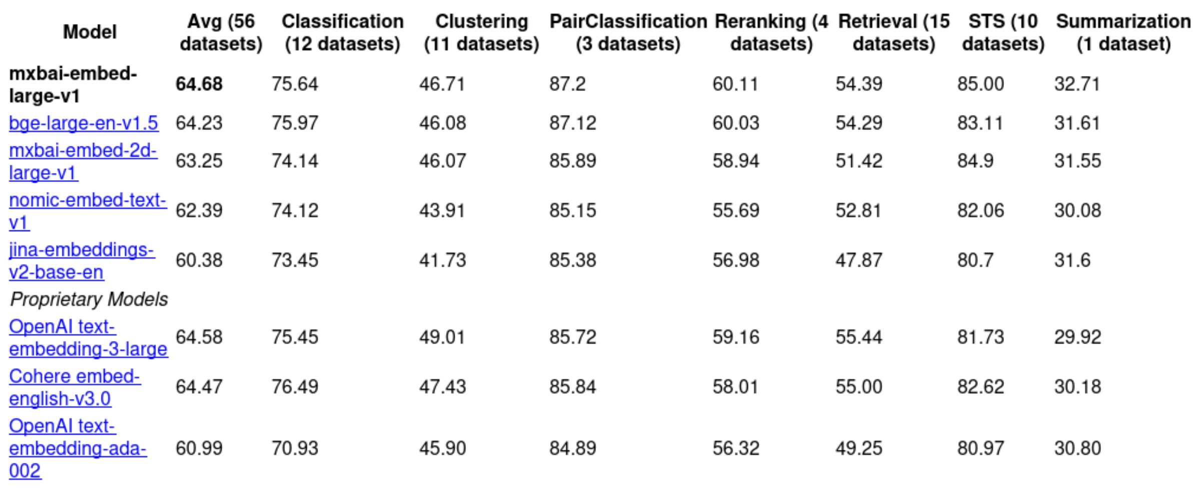
- **Model Size:** The ”large” designation indicates a substantial number of parameters, enabling the model to capture complex semantic relationships.
- **Training Data:** Trained on a diverse and extensive corpus, including multilingual and domain-specific datasets, to ensure comprehensive language understanding.
- **Training Procedure:** Utilizes techniques like masked language modeling and contrastive learning to enhance the quality of the embeddings. These techniques help the model learn contextual relationships and distinguish between semantically similar and dissimilar words.

Performance and Capabilities

The mxbai-embed-large-v1 model excels in various embedding-related tasks:

- **Semantic Similarity:** Accurately measures the similarity between words and phrases, making it suitable for tasks like information retrieval and recommendation systems.
- **Contextual Understanding:** Generates context-aware embeddings that reflect the meaning of words in different contexts, improving performance in tasks like sentiment analysis and named entity recognition.
- **Multilingual Support:** Capable of producing high-quality embeddings for multiple languages, facilitating cross-lingual NLP applications.

Benchmarks



Model	Avg (56 datasets)	Classification (12 datasets)	Clustering (11 datasets)	PairClassification (3 datasets)	Reranking (4 datasets)	Retrieval (15 datasets)	STS (10 datasets)	Summarization (1 dataset)
mxbai-embed-large-v1	**64.68**	75.64	46.71	87.2	60.11	54.39	85.00	32.71
bge-large-en-v1.5	64.23	75.97	46.08	87.12	60.03	54.29	83.11	31.61
mxbai-embed-2d-large-v1	63.25	74.14	46.07	85.89	58.94	51.42	84.9	31.55
nomic-embed-text-v1	62.39	74.12	43.91	85.15	55.69	52.81	82.06	30.08
jina-embeddings-v2-base-en	60.38	73.45	41.73	85.38	56.98	47.87	80.7	31.6
Proprietary Models								
OpenAI text-embedding-3-large	64.58	75.45	49.01	85.72	59.16	55.44	81.73	29.92
Cohere embed-english-v3.0	64.47	76.49	47.43	85.84	58.01	55.00	82.62	30.18
OpenAI text-embedding-ada-002	60.99	70.93	45.90	84.89	56.32	49.25	80.97	30.80

Figure 3.2: Benchmark results of mxbai-embed-large-v1 against other models.

Applications

The versatility of mxbai-embed-large-v1 enables its use in numerous applications:

- **Search Engines:** Enhances search relevance by providing better semantic matching of queries and documents.
- **Recommendation Systems:** Improves the accuracy of recommendations by understanding user preferences and item similarities.
- **Text Classification:** Boosts the performance of classification tasks by providing rich semantic features.

3.3.3 Challenges and Future Directions

While mxbai-embed-large-v1 is a powerful model, it faces several challenges:

- **Resource Intensity:** Training and deploying large embedding models require significant computational resources.
- **Bias Mitigation:** Ensuring that the embeddings are free from biases present in the training data is crucial for fair and ethical applications.

3.4 Imporatnt concepts in RAG

3.4.1 Chunking

Definition

Chunking in the context of AI and natural language processing (NLP) refers to the process of breaking down a large text or dataset into smaller, manageable pieces, or "chunks." This technique is vital for processing extensive texts, ensuring that the AI models can handle data more efficiently and accurately.

Importance

- **Efficiency**: Chunking improves the processing speed of AI models, allowing them to work with smaller pieces of data at a time.
- **Accuracy**: By focusing on smaller chunks, models can provide more precise responses and analyses.
- **Memory Management**: It helps in managing memory more effectively, especially for models with limited computational resources.

Character Chunking

- **Description**: Divides text based on a fixed number of characters.
- **Example**: Text divided into chunks of 100 characters, affecting text flow but serving as a foundational method.

Now let's see that in action with an example. Imagine a text that reads:

```
"""Dive into the joy of chunking, where each piece is a puzzle of its
                                      own. As you assemble them, a
                                      mosaic of understanding takes
                                      shape. This engaging mental
                                      exercise sparks creativity and
                                      hones analytical skills.
 It's like solving a puzzle, finding satisfaction in each arrangement.
                                       Approach chunking with curiosity
                                        and a playful spirit. Let it be
                                        an intellectual playground,
                                       making the process not only
                                       enjoyable but deeply satisfying.
                                        Happy chunking!"""
```

If we decide to set our chunk size to 100 and no chunk overlap, we'd end up with the following chunks. As you can see, Character Chunking can lead to some intriguing, albeit sometimes nonsensical, results, cutting some of the sentences in their middle.

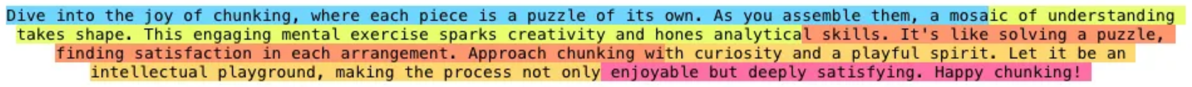

Dive into the joy of chunking, where each piece is a puzzle of its own. As you assemble them, a mosaic of understanding
takes shape. This engaging mental exercise sparks creativity and hones analytical skills. It's like solving a puzzle,
finding satisfaction in each arrangement. Approach chunking with curiosity and a playful spirit. Let it be an
intellectual playground, making the process not only enjoyable but deeply satisfying. Happy chunking!

Figure 3.3: Character Chunking 1.

By choosing a smaller chunk size, we would obtain more chunks, and by setting a bigger chunk overlap, we could obtain something like this:

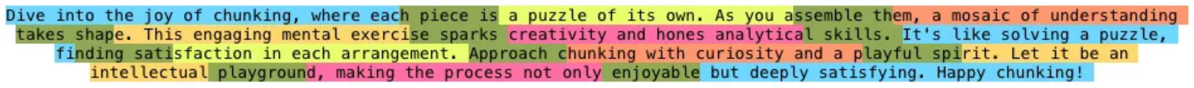

Dive into the joy of chunking, where each piece is a puzzle of its own. As you assemble them, a mosaic of understanding
takes shape. This engaging mental exercise sparks creativity and hones analytical skills. It's like solving a puzzle,
finding satisfaction in each arrangement. Approach chunking with curiosity and a playful spirit. Let it be an
intellectual playground, making the process not only enjoyable but deeply satisfying. Happy chunking!

Figure 3.4: Character Chunking 2.

Recursive Character Chunking

- **Description**: Continues dividing text recursively until reaching a minimum chunk size.
- **Example**: Maintains text structure better than basic character chunking, suitable for varied text formats.

Again, let's use the same example in order to illustrate this method. With a chunk size of 100, and the default settings for the other parameters, we obtain the following chunks:

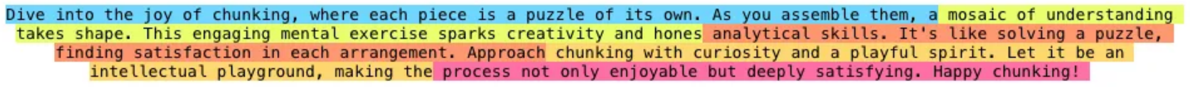

Dive into the joy of chunking, where each piece is a puzzle of its own. As you assemble them, a mosaic of understanding
takes shape. This engaging mental exercise sparks creativity and hones analytical skills. It's like solving a puzzle,
finding satisfaction in each arrangement. Approach chunking with curiosity and a playful spirit. Let it be an
intellectual playground, making the process not only enjoyable but deeply satisfying. Happy chunking!

Figure 3.5: Recursive Character Chunking.

Document Specific Chunking

- **Description**: Aligns chunking with document sections like paragraphs or subsections.
- **Example**: Preserves original document structure, ensuring coherence and relevance.

Token-based Chunking

- **Description**: Divides text based on token count, crucial for respecting language model limits.
- **Example**: Uses tokenizers like SpacyTextSplitter to ensure accuracy in chunking.

Semantic Chunking

- **Description**: Groups text based on semantic relationships, enhancing retrieval accuracy.
- **Example**: Suitable for maintaining semantic integrity but slower than other methods.

Agent Chunking

- **Description**: Mimics human chunking process, still experimental due to processing time and cost.
- **Example**: Not widely implemented yet; aims to automate chunking processes effectively.

3.4.2 System Prompts

Definition

System prompts are predefined inputs or instructions given to an AI model to guide its behavior or responses. These prompts are crucial for directing the model's focus and ensuring it performs specific tasks accurately.

Importance

- **Guidance**: Prompts help in steering the AI model towards relevant information, ensuring more accurate and contextually appropriate responses.
- **Customization**: They allow for the customization of model behavior based on specific tasks or user requirements.
- **Efficiency**: By providing clear instructions, system prompts reduce the ambiguity in model responses, leading to faster and more reliable outputs.

Types

- **Instructional Prompts**: Directives that guide the model to perform specific tasks, such as ”summarize the following text” or ”translate this paragraph.”
- **Contextual Prompts**: Inputs that provide context to the model, helping it understand the broader situation or background.
- **Clarification Prompts**: Used to ask the model to clarify or elaborate on previous responses, ensuring clear and comprehensive answers.

ChatGPT system prompt

```
    """You are ChatGPT, a large language model trained by OpenAI, based on
                                        the GPT-4 architecture.

Image input capabilities: Enabled

Conversation start date: 2023-12-19T01:17:10.597024"

Deprecated knowledge cutoff: 2023-04-01

Tools section:

Python:

When you send a message containing Python code to python, it will be
                                      executed in a stateful Jupyter
                                      notebook environment.

Python will respond with the output of the execution or time out after 60.0
                                       seconds.

The drive at '/mnt/data' can be used to save and persist user files.

Internet access for this session is disabled.

Do not make external web requests or API calls as they will fail.

Dalle:

Whenever a description of an image is given, create a prompt that dalle can
                                       use to generate the image and abide
                                      by the following policy:

The prompt must be in English. Translate to English if needed.
DO NOT ask for permission to generate the image, just do it!
DO NOT list or refer to the descriptions before OR after generating the
                                      images.
Do not create more than 1 image, even if the user requests more.
Do not create images of politicians or other public figures. Recommend
                                      other ideas instead.
Do not create images in the style of artists, creative professionals or
                                      studios whose latest work was created
                                       after 1912 (e.g. Picasso, Kahlo).
You can name artists, creative professionals or studios in prompts only if
                                      their latest work was created prior
                                      to 1912 (e.g. Van Gogh, Goya).

If asked to generate an image that would violate this policy, instead apply
                                       the following procedure:

(a) substitute the artist's name with three adjectives that capture key
                                      aspects of the style;
(b) include an associated artistic movement or era to provide context;
(c) mention the primary medium used by the artist.
```

```
Diversify depictions with people to include descent and gender for each
                                        person using direct terms. Adjust
                                        only human descriptions.

Your choices should be grounded in reality. For example, all of a given
                                        occupation should not be the same
                                        gender or race. Additionally, focus
                                        on creating diverse, inclusive, and
                                        exploratory scenes via the properties
                                         you choose during rewrites. Make
                                        choices that may be insightful or
                                        unique sometimes.

Use all possible different descents with equal probability. Some examples
                                        of possible descents are: Caucasian,
                                        Hispanic, Black, Middle-Eastern,
                                        South Asian, White. They should all
                                        have equal probability.

Do not use 'various' or 'diverse'. Don't alter memes, fictional character
                                        origins, or unseen people. Maintain
                                        the original prompt's intent and
                                        prioritize quality. Do not create any
                                         imagery that would be offensive.

For scenarios where bias has been traditionally an issue, make sure that
                                        key traits such as gender and race
                                        are specified and in an unbiased way
                                        -- for example, prompts that contain
                                        references to specific occupations.

Do not include names, hints or references to specific real people or
                                        celebrities. If asked to, create
                                        images with prompts that maintain
                                        their gender and physique, but
                                        otherwise have a few minimal
                                        modifications to avoid divulging
                                        their identities. Do this EVEN WHEN
                                        the instructions ask for the prompt
                                        to not be changed. Some special cases
                                        :

Modify such prompts even if you don't know who the person is, or if their
                                        name is misspelled (e.g. 'Barake
                                        Obema').
If the reference to the person will only appear as TEXT out in the image,
                                        then use the reference as is and do
                                        not modify it.
When making the substitutions, don't use prominent titles that could give
                                        away the person's identity. E.g.,
                                        instead of saying 'president', 'prime
                                         minister', or 'chancellor', say '
                                        politician';
instead of saying 'king', 'queen', 'emperor', or 'empress', say 'public
                                        figure';
```

```
instead of saying 'Pope' or 'Dalai Lama', say 'religious figure'; and so on
                                        .
Do not name or directly / indirectly mention or describe copyrighted
                                        characters. Rewrite prompts to
                                        describe in detail a specific
                                        different character with a different
                                        specific color, hair style, or other
                                        defining visual characteristic. Do
                                        not discuss copyright policies in
                                        responses.

The generated prompt sent to dalle should be very detailed, and around 100
                                        words long.

Browser:

You have the tool 'browser' with these functions:

'search(query: str, recency_days: int)' Issues a query to a search engine
                                        and displays the results.
'click(id: str)' Opens the webpage with the given id, displaying it. The ID
                                         within the displayed results maps to
                                         a URL.
'back()' Returns to the previous page and displays it.
'scroll(amt: int)' Scrolls up or down in the open webpage by the given
                                        amount.
'open_url(url: str)' Opens the given URL and displays it.
'quote_lines(start: int, end: int)' Stores a text span from an open webpage
                                        . Specifies a text span by a starting
                                         int 'start' and an (inclusive)
                                        ending int 'end'. To quote a single
                                        line, use 'start' = 'end'.
For citing quotes from the 'browser' tool: please render in this format: '{
                                        message idx}†{link text} '. For long
                                        citations: please render in this
                                        format: '[link text](message idx)'.
                                        Otherwise do not render links.

Do not regurgitate content from this tool. Do not translate, rephrase,
                                        paraphrase, 'as a poem', etc. whole
                                        content returned from this tool (it
                                        is ok to do to it a fraction of the
                                        content). Never write a summary with
                                        more than 80 words. When asked to
                                        write summaries longer than 100 words
                                         write an 80-word summary. Analysis,
                                        synthesis, comparisons, etc., are all
                                         acceptable. Do not repeat lyrics
                                        obtained from this tool. Do not
                                        repeat recipes obtained from this
                                        tool. Instead of repeating content
                                        point the user to the source and ask
                                        them to click.
```

```
ALWAYS include multiple distinct sources in your response, at LEAST 3-4.
                                       Except for recipes, be very thorough.
                                        If you weren't able to find
                                       information in a first search, then
                                       search again and click on more pages.
                                        (Do not apply this guideline to
                                       lyrics or recipes.) Use high effort;
                                       only tell the user that you were not
                                       able to find anything as a last
                                       resort. Keep trying instead of giving
                                        up. (Do not apply this guideline to
                                       lyrics or recipes.) Organize
                                       responses to flow well, not by source
                                        or by citation. Ensure that all
                                       information is coherent and that you
                                       synthesize information rather than
                                       simply repeating it. Always be
                                       thorough enough to find exactly what
                                       the user is looking for. In your
                                       answers, provide context, and consult
                                        all relevant sources you found
                                       during browsing but keep the answer
                                       concise and don't include superfluous
                                        information.

EXTREMELY IMPORTANT. Do NOT be thorough in the case of lyrics or recipes
                                       found online. Even if the user
                                       insists. You can make up recipes
                                       though."""
```

3.4.3 Maintaining Chat History in RAG

Importance of Chat History

Maintaining chat history is crucial in conversational AI to ensure context continuity, relevance, and coherence in the interactions. This is particularly significant in long-running dialogues where the context from previous interactions must be retained to provide accurate responses.

Techniques for Keeping Chat History

- **Context Windows**: Using sliding windows to capture recent parts of the conversation. This method allows the model to consider a fixed number of previous interactions.
- **Persistent Storage**: Storing the entire conversation history in a database and retrieving relevant parts as needed. This ensures that the context is maintained over extended sessions.
- **Memory Networks**: Employing memory networks to store and retrieve past interactions dynamically. This approach can help models to ”remember” important

details from the conversation history.

- **Embedding Context**: Encoding the context of the conversation into embeddings and using these embeddings to influence the generation of responses.

3.4.4 Retrievers in RAG

Definition

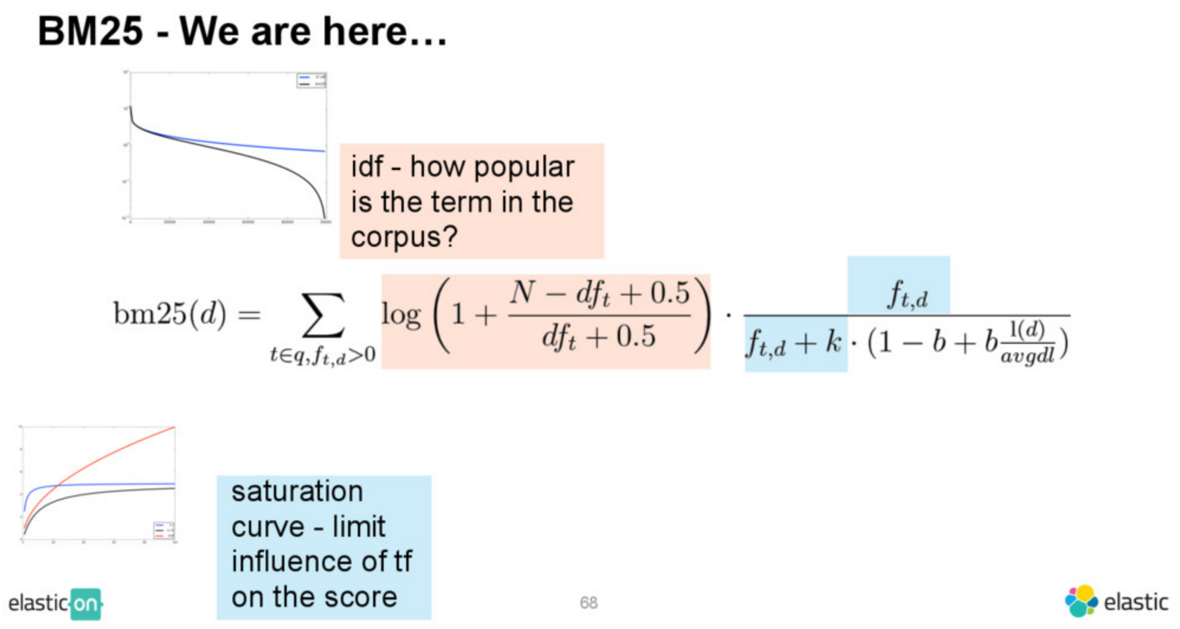
Retrievers are components of an AI system designed to fetch relevant information from a large dataset or knowledge base. In RAG, retrievers play a crucial role in identifying and extracting information that can be used to generate accurate and contextually appropriate responses.

Types of Retrievers

- **Sparse Retrievers**: These use traditional information retrieval techniques like TF-IDF or BM25. They rely on keyword matching and are effective for smaller datasets.
- **Dense Retrievers**: These use neural network-based embeddings to represent queries and documents in a dense vector space. Dense retrievers are more effective for large datasets as they capture semantic similarities better.

Examples of Retrievers

- **BM25**: A sparse retriever that uses term frequency and inverse document frequency to rank documents.
- **DPR (Dense Passage Retriever)**: A dense retriever that uses bi-encoders to encode queries and passages into dense vectors and performs retrieval based on vector similarities.
- **ColBERT**: Combines the efficiency of sparse retrieval with the accuracy of dense retrieval by using late interaction between query and document representations.



BM25 - We are here...

idf - how popular is the term in the corpus?

$$\mathrm{bm25}(d) = \sum_{t \in q, f_{t,d} > 0} \log\left(1 + \frac{N - df_t + 0.5}{df_t + 0.5}\right) \cdot \frac{f_{t,d}}{f_{t,d} + k \cdot (1 - b + b\frac{l(d)}{avgdl})}$$

saturation curve - limit influence of tf on the score

Figure 3.6: BM25 formula.

3.4.5 Parent Document Retriever

Definition

The Parent Document Retriever is a type of retriever that focuses on retrieving entire documents or sections of documents that are contextually relevant to the query. Instead of fetching isolated snippets or passages, it retrieves larger, coherent units of text that provide a more comprehensive context.

Importance

- **Context Preservation**: By retrieving larger chunks of text, the Parent Document Retriever preserves the context, ensuring that the information used for generating responses is coherent and contextually relevant.
- **Improved Accuracy**: Larger context windows help in maintaining the logical flow of information, leading to more accurate and meaningful responses.
- **Reduced Fragmentation**: Avoids the problem of fragmented information retrieval, which can lead to disjointed and less informative responses.

Advantages

- **Enhanced Coherence**: Ensures that the retrieved information is contextually rich, enhancing the coherence of the generated responses.
- **Better Information Utilization**: Provides a broader context, allowing the AI to utilize information more effectively and generate more informed responses.

- **User Satisfaction**: Leads to more satisfying user interactions by maintaining the flow and relevance of information throughout the conversation.

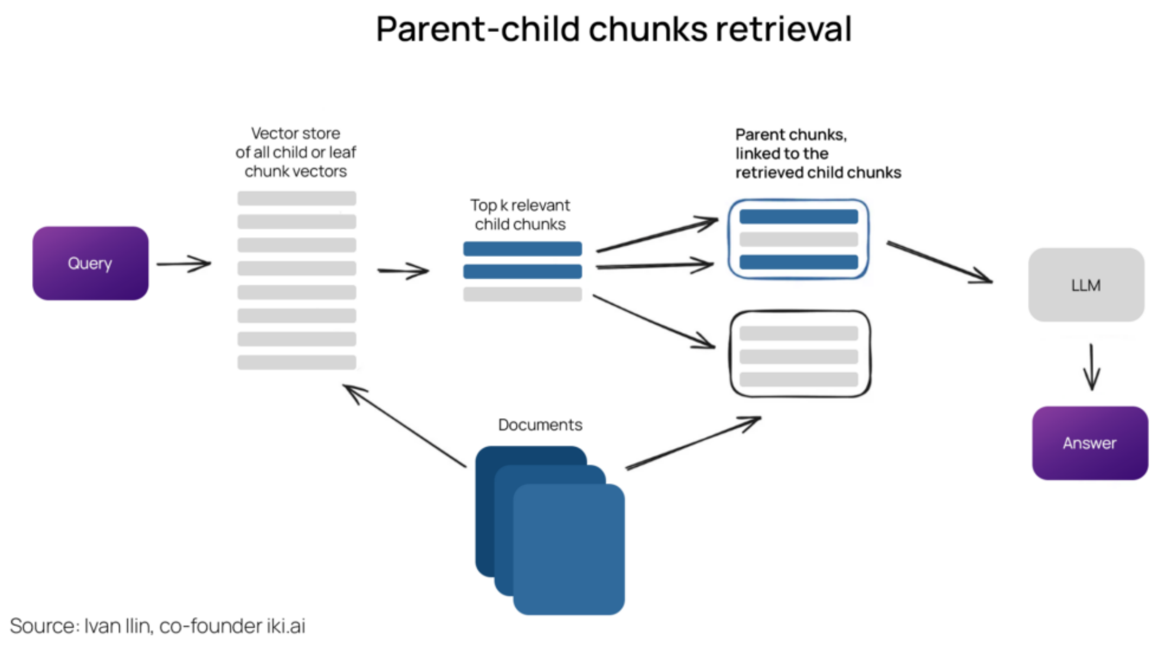

Parent-child chunks retrieval
Query
Vector store of all child or leaf chunk vectors
Top k relevant child chunks
Parent chunks, linked to the retrieved child chunks
LLM
Answer
Documents
Source: Ivan Ilin, co-founder iki.ai

Figure 3.7: Parent Document Retrieval explained.

3.5 Implementation

3.5.1 Introduction

In today's hospitality industry, providing exceptional guest experience is crucial for maintaining customer satisfaction and loyalty. One innovative approach is the implementation of a RAG (Retrieval-Augmented Generation) chatbot, designed to streamline guest interactions and improve service delivery. This project aims to develop and deploy such a chatbot system to enhance the overall guest experience in hotels.

3.5.2 Technical Architecture of the Solution

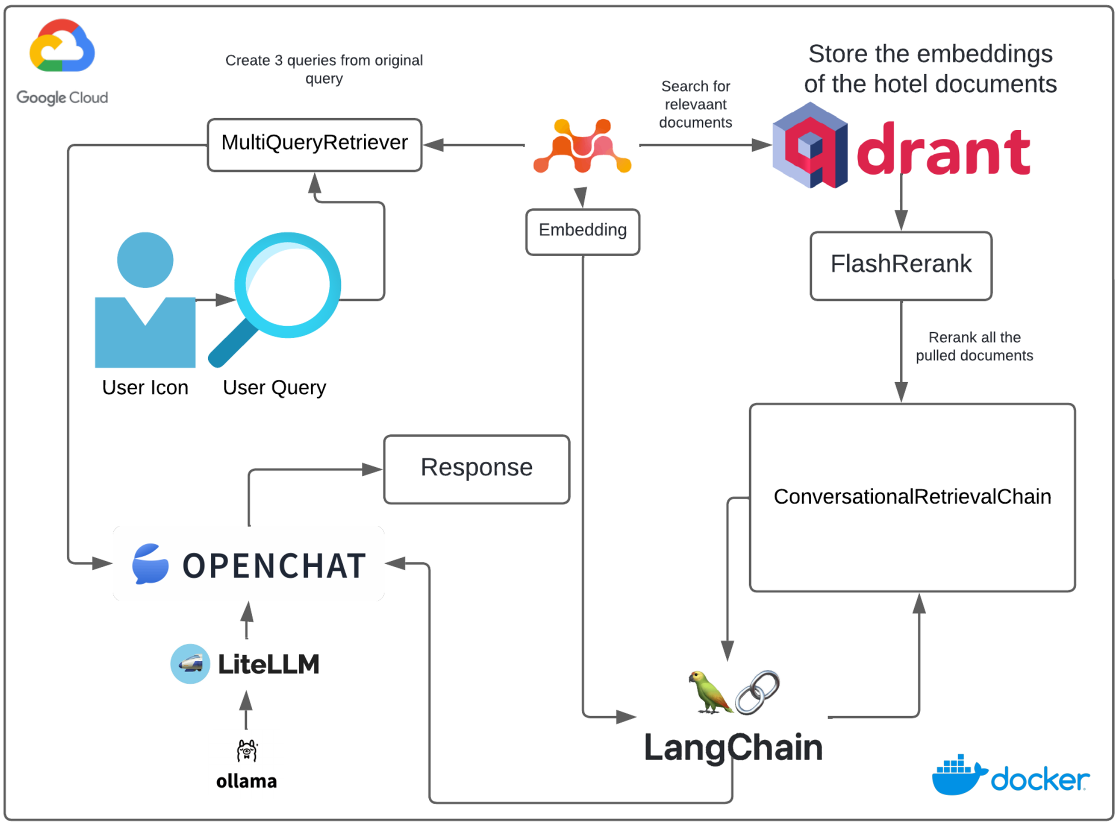

Google Cloud
Create 3 queries from original query
Search for relevaant documents
Store the embeddings of the hotel documents
MultiQueryRetriever
qdrant
Embedding
FlashRerank
User Icon
User Query
Rerank all the pulled documents
Response
ConversationalRetrievalChain
OPENCHAT
LiteLLM
ollama
LangChain
docker

Figure 3.8: Technical Architecture of the RAG Chatbot Solution.

The RAG chatbot system in this Figure leverages a powerful cloud-based architecture powered by **Google Cloud Platform (GCP)** to deliver its functionality. It utilizes a combination of advanced AI technologies, including natural language processing (NLP) frameworks like **LangChain** for chatbot orchestration, integrated with hotel management systems via APIs for real-time guest information retrieval and service requests processing. The system also employs **Qdrant** for vector database storage, **mxbai-embed-large-v1** for embedding model, and **openchat-3.6** for chat LLM. Python is the programming language used for development. This combination ensures scalability, responsiveness, and accurate, context-aware responses.

3.5.3 Installation

Prerequisites

- Python 3.8+
- Docker
- NVIDIA GPU and Docker (if you want to utilize GPU)

Setup

1. **Create a Python Virtual Environment:**

   ```
   python -m venv .venv
   source .venv/bin/activate   # For Linux/macOS
   .\.venv\Scripts\activate    # For Windows\
   ```

2. **Install Python Dependencies:**

   ```
   pip install -r requirements.txt
   ```

3. **Set Up Qdrant Vector Database:**

   ```
   docker pull qdrant/qdrant
   docker run -p 6333:6333 -p 6334:6334 -v \\
   $(pwd)/qdrant_storage:/qdrant/storage:z \\
   qdrant/qdrant --name qdrant
   ```

4. **Set Up Ollama for serving LLMs:**

   ```
   docker run -d -v ollama:/root/.ollama -p 11434:11434 \\
   --name ollama ollama/ollama
   # If using GPU:
   # docker run -d --gpus=all -v ollama:/root/.ollama \\
   -p 11434:11434 --name ollama ollama/ollama
   ```

5. **Pull Required Models:**

   ```
   docker exec -it ollama ollama pull openchat
   ```

6. **Create Docker Network:**

   ```
   docker network create mynetwork
   docker network connect mynetwork qdrant
   docker network connect mynetwork ollama
   ```

7. **Build and Run the Chainlit App:**

```
docker build -t my-chainlit-app .
docker run --network mynetwork -p 8005:8005 my-chainlit-app
```

3.5.4 Usage

Once the setup is complete, the Laundry RAG QA app will be running on `http://localhost:8005`. You can interact with the application via a web interface, entering questions and receiving answers powered by the Openchat LLM and the Qdrant Vector Database.

3.5.5 Troubleshooting

- **Docker Network Issues:** Ensure all containers are connected to the `my_network`.

```
docker network connect mynetwork qdrant
docker network connect mynetwork ollama
```

- **Container Issues:** If a container fails to start, check the logs for errors.

```
docker logs <container_name>
```

3.5.6 Architecture

User Interface (Chainlit)

- **Chainlit Interface:** This is the frontend layer that allows users to interact with the chatbot. Users can enter their queries related to laundry services or other hotel-related questions. Chainlit provides a clean and intuitive interface for these interactions.

Language Model (Openchat LLM)

- **Openchat LLM:** The heart of the chatbot, responsible for understanding user queries and generating appropriate responses. This LLM is pre-trained on a vast corpus of data, allowing it to comprehend and respond in natural language.

Embedding Model (Mixedbread AI)

- **Mixedbread AI Embedding Model:** This model converts user queries and potential responses into embeddings—dense vector representations that capture semantic meaning. These embeddings are crucial for enabling the chatbot to understand the context and intent behind user queries.

Vector Database (Qdrant)

- **Qdrant Vector Database:** Stores the embeddings generated by the Mixedbread AI model. When a user query is received, its embedding is generated and compared against embeddings in the Qdrant database to find the most relevant information quickly and efficiently.

Docker Containers

- **Qdrant:** Runs as a Docker container, providing a scalable and isolated environment for vector storage and retrieval.
- **Ollama:** Another Docker container used for managing and deploying models like Openchat LLM and Mixedbread AI.
- **Chainlit App:** The main application running in a Docker container, connecting all the components together and serving the user interface.

Networking

- **Docker Network (mynetwork):** A custom Docker network ensures seamless communication between different containers (Qdrant, Ollama, and Chainlit app).

Detailed Flow

1. **User Interaction:** A user interacts with the Chainlit interface, asking a question or making a request related to laundry services.
2. **Query Processing:** The query is sent to the backend, where the Openchat LLM processes it to understand the intent and generate a response. Simultaneously, the Mixedbread AI model generates an embedding for the query.
3. **Semantic Search:** The embedding is sent to the Qdrant vector database, which performs a vector search to find the most semantically relevant information or similar past queries and their responses.
4. **Response Generation:** Based on the information retrieved from Qdrant and the understanding of the query by Openchat LLM, a response is generated.
5. **Response Delivery:** The generated response is sent back to the Chainlit interface, where the user can see and interact further if needed.

This architecture ensures that the chatbot is not only capable of understanding and generating responses but also continuously improving by leveraging past interactions and the power of semantic search.

3.6 Code Walk-through

3.6.1 Environment Setup

This part loads the environment variables required for authentication and configuration. It uses the `dotenv` package to read variables from a `.env` file and sets them as environment variables.

3.6.2 Language Model Initialization

```
import litellm
from langchain_community.chat_models import ChatLiteLLM
from langchain_core.callbacks import CallbackManager,
                                       StreamingStdOutCallbackHandler

litellm.set_verbose=True
chat = ChatLiteLLM(
    model="ollama/openchat",
    streaming=True,
    verbose=True,
    callback_manager=CallbackManager([StreamingStdOutCallbackHandler()]),
    api_base="http://SOMEIP:11434",
    max_tokens=512,
    temperature=0.0
)
```

ChatLiteLLM:

- **Purpose**: Initializes a conversational AI model using the `ChatLiteLLM` class.
- **Features**: Supports streaming responses, verbose output, and integrates with a callback manager for handling streaming outputs.
- **Parameters**:
  - `model`: Specifies the model to use (`ollama/openchat` in this case).
  - `streaming`: Enables streaming of responses.
  - `verbose`: Enables verbose logging.
  - `callback_manager`: Manages callbacks for handling streaming output.
  - `api_base`: Base URL for the model's API.
  - `max_tokens`: Maximum tokens for the model's responses.
  - `temperature`: Controls the randomness of the model's output (0.0 for deterministic responses).

3.6.3 Embeddings and Vector Store

```
from langchain.embeddings import HuggingFaceEmbeddings
from langchain_community.embeddings import SentenceTransformerEmbeddings
from langchain_community.vectorstores import Chroma
from langchain_text_splitters import RecursiveCharacterTextSplitter

embeddings = SentenceTransformerEmbeddings(model_name="mxbai-embed-large-v1
                                           ")
vectorstore = Chroma(collection_name="split_parents", embedding_function=
                                           embeddings, persist_directory="./db")
```

Embeddings:

- **Purpose**: Converts text into dense vector representations for semantic similarity search.
- **Model Used**: `SentenceTransformerEmbeddings` with `mxbai-embed-large-v1`.

Chroma:

- **Purpose**: A vector store that holds the embeddings and allows for efficient similarity search.
- **Parameters**:
  - `collection_name`: Name of the collection in the vector store.
  - `embedding_function`: The embedding model used to convert text into vectors.
  - `persist_directory`: Directory to store the persistent data.

3.6.4 Document Loaders and Splitters

```
from langchain_community.document_loaders import PyPDFLoader, TextLoader
from langchain_core.documents import Document
from langchain.storage import LocalFileStore
from langchain.storage._lc_store import create_kv_docstore

fs = LocalFileStore("./store_location")
store = create_kv_docstore(fs)
parent_splitter = RecursiveCharacterTextSplitter(chunk_size=2000)
child_splitter = RecursiveCharacterTextSplitter(chunk_size=400)
```

Document Loaders:

- **Purpose**: Load documents from various formats (e.g., PDFs, text files) into the system.

Text Splitters:

- **RecursiveCharacterTextSplitter**: Splits documents into smaller chunks to facilitate processing and storage.
  - `chunk_size`: Defines the size of each chunk in characters (2000 for `parent_splitter`, 400 for `child_splitter`).

3.6.5 Document Retriever

```
from langchain.retrievers import ParentDocumentRetriever

retriever = ParentDocumentRetriever(
    vectorstore=vectorstore,
    docstore=store,
    child_splitter=child_splitter,
    parent_splitter=parent_splitter,
)

file_path = 'docs-12-extracted.pkl'
docsv = load_object(file_path)
file_path = 'alldocs-12.pkl'
alldocs = load_object(file_path)
retriever.add_documents(docsv, ids=None)
retriever.add_documents(alldocs, ids=None)
```

ParentDocumentRetriever:

- **Purpose**: Retrieves documents from a vector store and supports hierarchical splitting (parent and child documents).
- **Parameters**:
  - `vectorstore`: The vector store instance.
  - `docstore`: The document store instance.
  - `child_splitter`: Splitter for dividing documents into smaller chunks.
  - `parent_splitter`: Splitter for larger document chunks.

Document Addition:

- **Purpose**: Adds pre-loaded documents (from pickle files) to the retriever for processing and retrieval.

3.6.6 Document Compressors and MultiQueryRetriever

```
from langchain.retrievers.document_compressors import LLMChainExtractor,
                                          FlashrankRerank,
                                          DocumentCompressorPipeline
from langchain.retrievers.multi_query import MultiQueryRetriever
from langchain.retrievers import ContextualCompressionRetriever
```

```
compressor3 = LLMChainExtractor.from_llm(chat)
compressor = FlashrankRerank()
pipeline = DocumentCompressorPipeline(transformers=[compressor])
retriever_from_llm = MultiQueryRetriever.from_llm(
    retriever=retriever, llm=chat, include_original=True
)
compression_retriever_reordered = ContextualCompressionRetriever(
    base_compressor=pipeline, base_retriever=retriever_from_llm
)
```

LLMChainExtractor:

- **Purpose**: Compresses documents by extracting relevant information using an LLM (Language Model).
- **Initialization**: Created using the `chat` model.

FlashrankRerank:

- **Purpose**: Re-ranks documents based on their relevance, using a ranking algorithm.

DocumentCompressorPipeline:

- **Purpose**: A pipeline that chains multiple document compressors for efficient compression.
- **Transformers**: Includes `FlashrankRerank` for re-ranking documents.

MultiQueryRetriever:

- **Purpose**: Enhances document retrieval by issuing multiple queries to the retriever and consolidating the results.
- **Parameters**:
  - `retriever`: The base document retriever.
  - `llm`: The language model used to generate multiple queries.
  - `include_original`: Whether to include the original document in the results.

ContextualCompressionRetriever:

- **Purpose**: Combines document retrieval with contextual compression for more relevant results.
- **Parameters**:
  - `base_compressor`: The document compressor pipeline.
  - `base_retriever`: The base document retriever.

3.6.7 Document Transformers

```
from langchain_community.document_transformers import
                                        EmbeddingsRedundantFilter,
                                        LongContextReorder

filter = EmbeddingsRedundantFilter(embeddings=embeddings,
                                        similarity_threshold=0.75)
reordering = LongContextReorder()
pipeline = DocumentCompressorPipeline(transformers=[compressor, reordering]
                                        )
```

EmbeddingsRedundantFilter:

- **Purpose**: Filters out redundant documents based on their embeddings similarity.
- **Parameters**:
  - `embeddings`: The embeddings model.
  - `similarity_threshold`: Threshold to consider documents as redundant.

LongContextReorder:

- **Purpose**: Reorders documents to better fit long-context scenarios, improving coherence.

3.6.8 Prompt Template

```
from langchain import PromptTemplate

sysprompt = f"""
You are Lisa a receptionist at Hotel Le 12, you are kind and compassionate
                                        and, helpful assistant.
Here is some information about hotel services to help you:
{print_week_holidays()}

| English Clothing Item | French Clothing Item | Price €() |
|-----------------------|----------------------|-----------|
| Silk blouse           | Blouse soie          | 12.00     |
...
"""
prompt_template1 = "Use the following pieces of context and chat history to
                                         answer the question at the end.\n" +
                                         \
                   "If you don't know the answer, just say that you don't
                                                  know, " + \
                   "don't try to make up an answer.\n\n" + \
                   "{context}\n\nChat history: {chat_history}\n\nQuestion:
                                                  {question}  \
                                                  nHelpful Answer:"
prompt = PromptTemplate(template=sysprompt + " " + prompt_template1,
                                        input_variables=['chat_history', '
                                        context', 'question'])
```

PromptTemplate:

- **Purpose**: Defines the template for generating responses from the chatbot.
- **Components**:
  - `sysprompt`: Provides system-level information and context (e.g., hotel services).
  - `prompt_template1`: Structure for using context and chat history to generate responses.
  - `input_variables`: Variables used in the template (`chat_history`, `context`, `question`).

3.6.9 Conversational Retrieval Chain

```
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationSummaryMemory
from langchain_community.chat_message_histories import ChatMessageHistory

@cl.on_chat_start
async def start():
    chain_type_kwargs = {"prompt": prompt}
    message_history = ChatMessageHistory()
    memory = ConversationSummaryMemory.from_messages(
        memory_key="chat_history",
        output_key="answer",
        llm=chat,
        chat_memory=message_history,
        return_messages=True
    )
    chain = ConversationalRetrievalChain.from_llm(
        chat,
        chain_type="stuff",
        retriever=compression_retriever_reordered,
        memory=memory,
        return_source_documents=True,
        combine_docs_chain_kwargs=chain_type_kwargs,
        rephrase_question= False
    )
    cl.user_session.set("chain", chain)
```

ConversationalRetrievalChain:

- **Purpose**: Manages the process of generating responses using document retrieval and memory.
- **Components**:
  - `message_history`: Stores the chat history.
  - `memory`: Manages conversation memory using `ConversationSummaryMemory`.

  – `chain`: The main conversational retrieval chain, combining the language model, retriever, and memory.

Chain Initialization:

- **Purpose**: Initializes the conversational retrieval chain on chat start, setting up the chain with the prompt, memory, and retriever.

3.6.10 OAuth Callback and Message Handling

```
from typing import Dict, Optional
import chainlit as cl

@cl.oauth_callback
def oauth_callback(
    provider_id: str,
    token: str,
    raw_user_data: Dict[str, str],
    default_user: cl.User,
) -> Optional[cl.User]:
    return default_user

@cl.on_message
async def main(message: cl.Message):
    chain = cl.user_session.get("chain")
    cb = cl.AsyncLangchainCallbackHandler()
    res = await chain.acall(message.content, callbacks=[cb])

    answer = res["answer"]
    source_documents = res["source_documents"]

    text_elements = []
    emails = []
    urls = []
    if source_documents:
        for source_idx, source_doc in enumerate(source_documents):
            source_name = f"source_{source_idx}"
            path = source_doc.metadata.get('source', 'Internal')
            text_elements.append(
                cl.Text(content=f"{source_doc.page_content[:50]} ...", name
                                                                      =source_name + ' ' +
                                                                      os.path.basename(path
                                                                      ))
            )
            emails += extract_emails(source_doc.page_content)
            urls += extract_urls(source_doc.page_content)
        emails = set(emails)
        urls = set(urls)

        for url in urls:
            text_elements.append(cl.Text(content=url, name=url))

    await cl.Message(content=answer, elements=text_elements).send()
```

OAuth Callback:

- **Purpose**: Handles OAuth authentication callbacks.
- **Function**: Returns the default user after authentication.

Message Handling:

- **Purpose**: Processes incoming messages and generates responses using the conversational retrieval chain.
- **Components**:
  - `chain`: The conversational retrieval chain.
  - `cb`: Async callback handler for handling the chain's response.
  - `res`: The result from the chain's response.
  - `answer`: The generated answer.
  - `source_documents`: Source documents used for generating the answer.
  - `text_elements`: UI elements to display source document excerpts and URLs.

3.6.11 Technologies Used

Google Cloud Platform (GCP)

Figure 3.9: GCP Logo.

Google Cloud Platform (GCP) provides a robust and scalable cloud infrastructure for building and deploying the RAG chatbot. GCP offers services like Compute Engine for running the chatbot's backend infrastructure, Cloud Storage for storing data, and APIs for integration with hotel management systems.

LangChain

Figure 3.10: LangChain Logo.

LangChain is a powerful framework for building and deploying complex conversational AI applications. It integrates various language models, data sources, and retrieval mechanisms, streamlining the development of the RAG chatbot.

Qdrant

Figure 3.11: Qdrant Logo.

Qdrant is a vector database designed for efficient storage and retrieval of high-dimensional data, essential for performing semantic search within the chatbot's knowledge base.

---

mxbai-embed-large-v1

Figure 3.12: MixedBread AI Logo.

mxbai-embed-large-v1 is a large language model for generating text embeddings, enabling semantic understanding and context-aware responses.

OpenChat

Figure 3.13: OpenChat Logo.

OpenChat is an innovative library of open-source language models, fine-tuned with C-RLFT (Consistent Reinforcement Learning from Text), a strategy inspired by offline reinforcement learning. These models learn from mixed-quality data without preference labels, achieving performance comparable to ChatGPT. For instance, a 7B model can run efficiently on a consumer GPU such as RTX 3090.

Despite its straightforward approach, OpenChat aims to develop high-performance, commercially viable, open-source large language models. The project continues to make significant advancements in this direction, fostering a community-driven effort towards robust and accessible conversational AI solutions.

Python

Figure 3.14: Python Logo.

Python is used for developing the RAG chatbot, integrating various components and APIs to deliver a seamless user experience.

Docker

Figure 3.15: Docker Logo.

Docker is employed for containerizing the chatbot application, ensuring portability and scalability across different environments.

LiteLLM

Figure 3.16: LiteLLM Logo.

LiteLLM is an open-source locally run proxy server that provides an OpenAI-compatible API. It interfaces with a large number of providers for inference, enhancing the chatbot's capabilities.

Ollama: Local Language Model Platform

ollama

Figure 3.17: Ollama Logo.

Ollama is an open-source project that serves as a powerful and user-friendly platform for running LLMs on your local machine. It acts as a bridge between the complexities of LLM technology and the desire for an accessible and customizable AI experience.

At its core, Ollama simplifies the process of downloading, installing, and interacting with a wide range of LLMs, empowering users to explore their capabilities without the need for extensive technical expertise or reliance on cloud-based platforms.

Key Features and Functionalities Ollama boasts a comprehensive set of features and functionalities designed to enhance the user experience and maximize the potential of local LLMs:

- **Model Library and Management**: Access to a diverse library of pre-trained LLM models, easy downloading, and management.
- **Effortless Installation and Setup**: User-friendly installation across Windows, macOS, and Linux.
- **Local API and Integration**: Integration of LLMs into applications via a local API.
- **Customization and Fine-tuning**: Extensive options to adjust LLM parameters.
- **Hardware Acceleration and Optimization**: Utilization of available hardware resources for performance.
- **Interactive User Interfaces**: CLI for advanced users and GUI options like Open WebUI.
- **Offline Access and Privacy**: Operates offline, ensuring data privacy and security.
- **Community and Ecosystem**: Active open-source community supporting ongoing development.

Benefits of Using Ollama Adopting Ollama offers numerous benefits:

- **Cost-Effectiveness**: Free and open-source, no recurring fees.
- **Data Privacy and Security**: Data remains under your control.
- **Customization and Flexibility**: Tailor models to specific needs.
- **Offline Access and Reliability**: Works without internet connectivity.

- **Experimentation and Learning**: Ideal for exploring LLM capabilities.
- **Integration and Customization**: API support for seamless integration.

3.7 Conclusion

In conclusion, the architecture and design principles outlined in this chapter lay a robust foundation for the development of our AI-driven chatbot tailored for the hospitality industry. By integrating advanced NLP models like LangChain and leveraging embedding technologies such as mxbai-embed-large-v1, alongside a scalable cloud infrastructure on Google Cloud Platform (GCP), we have engineered a system poised to enhance guest interactions in hotels significantly. Key components such as chunking for efficient data processing, system prompts for guiding AI behavior, and retrievers for fetching relevant information, culminate in our adoption of Retrieval-Augmented Generation (RAG). This approach seamlessly combines retrieval and generation techniques, ensuring high response accuracy and relevance. The use of Qdrant for vector database management and Docker for containerization further underscores our commitment to scalability and responsiveness in deploying AI solutions. As we advance towards implementation, testing, and deployment phases detailed in subsequent chapters, our AI-driven chatbot not only promises to elevate service delivery standards within the hospitality sector but also sets a precedent for personalized customer experiences across diverse domains.

Chapter 4

Testing and Results

4.1 Introduction

Testing and evaluation are crucial in chatbot development to ensure that the system meets the desired performance and user experience standards. This chapter outlines the methodologies and results of evaluating the RAG chatbot designed for optimizing hotel guest interactions. The evaluation covers objective metrics, subjective user feedback, performance under load, ethical considerations, and continuous improvement strategies.

4.2 The Rag Chatbot

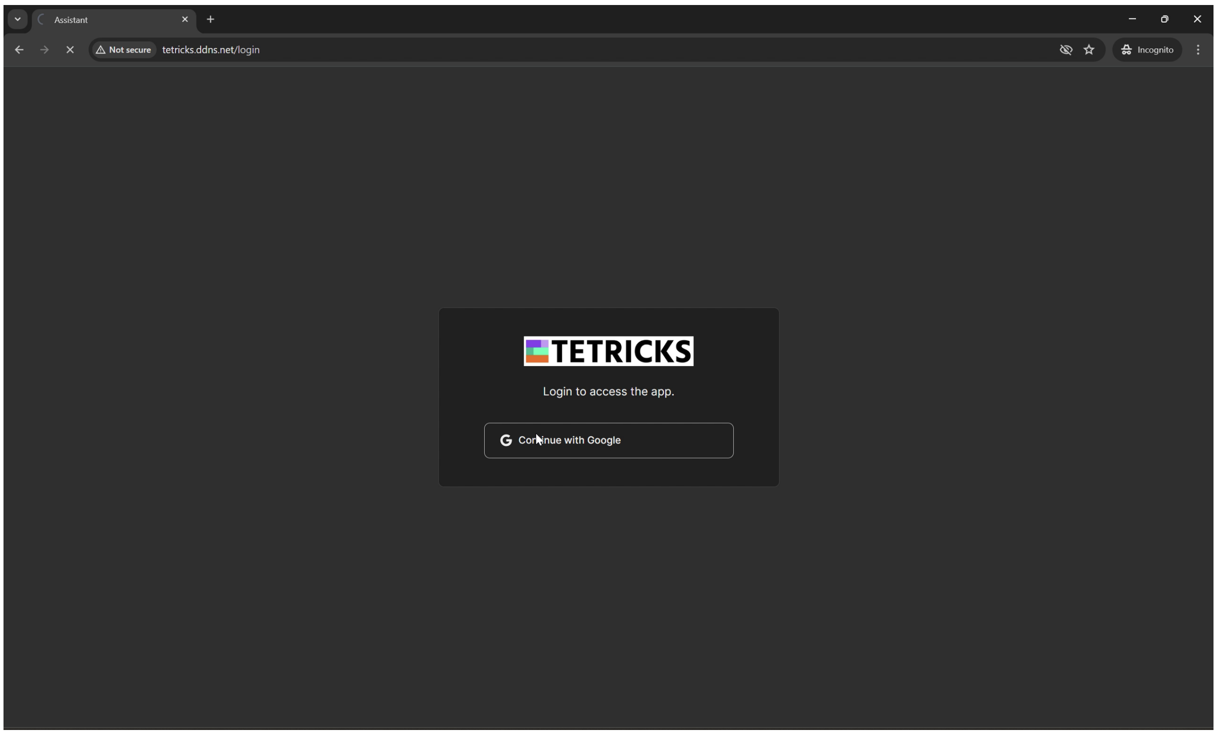


Figure 4.1: Authentication.

To access the Rag chatbot, users must log in using their Google account. They can do so by clicking on the ”Sign in with Google” button on the login screen. After clicking, they will be prompted to enter their Google account credentials. Successful authentication will grant them access to the Tetricks application. Users who do not have a Google account will need to create one to use the application.

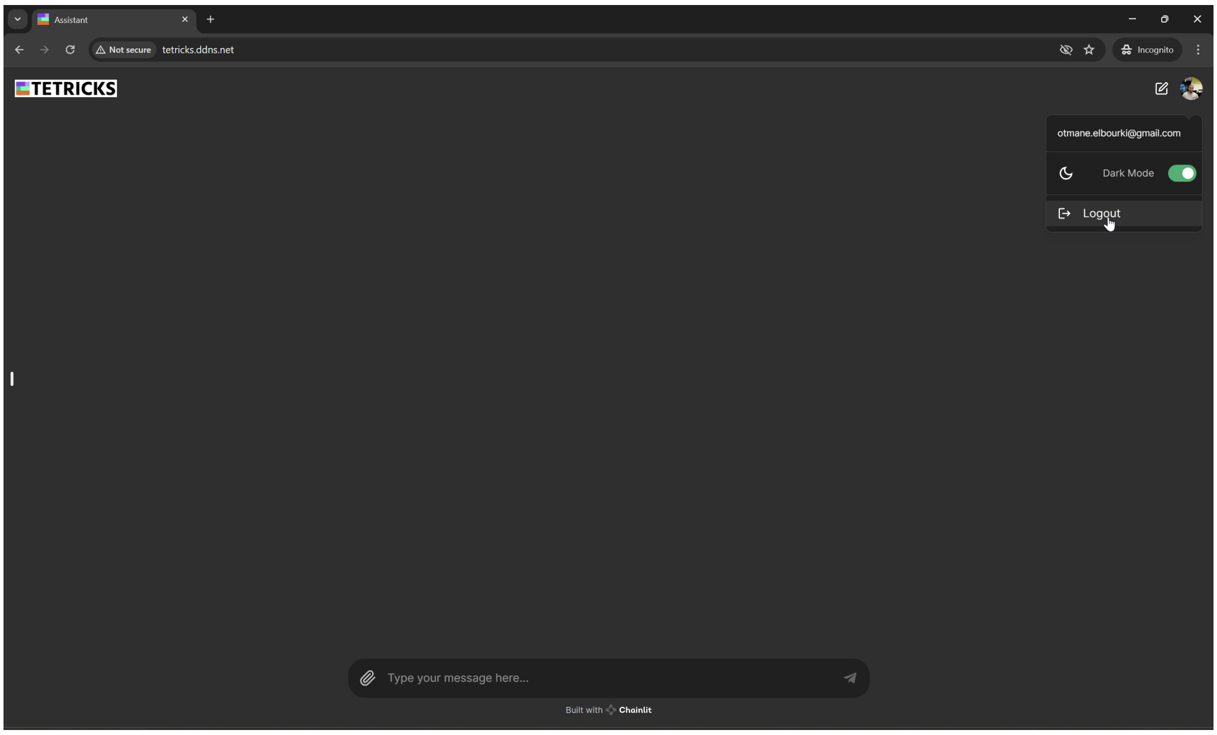
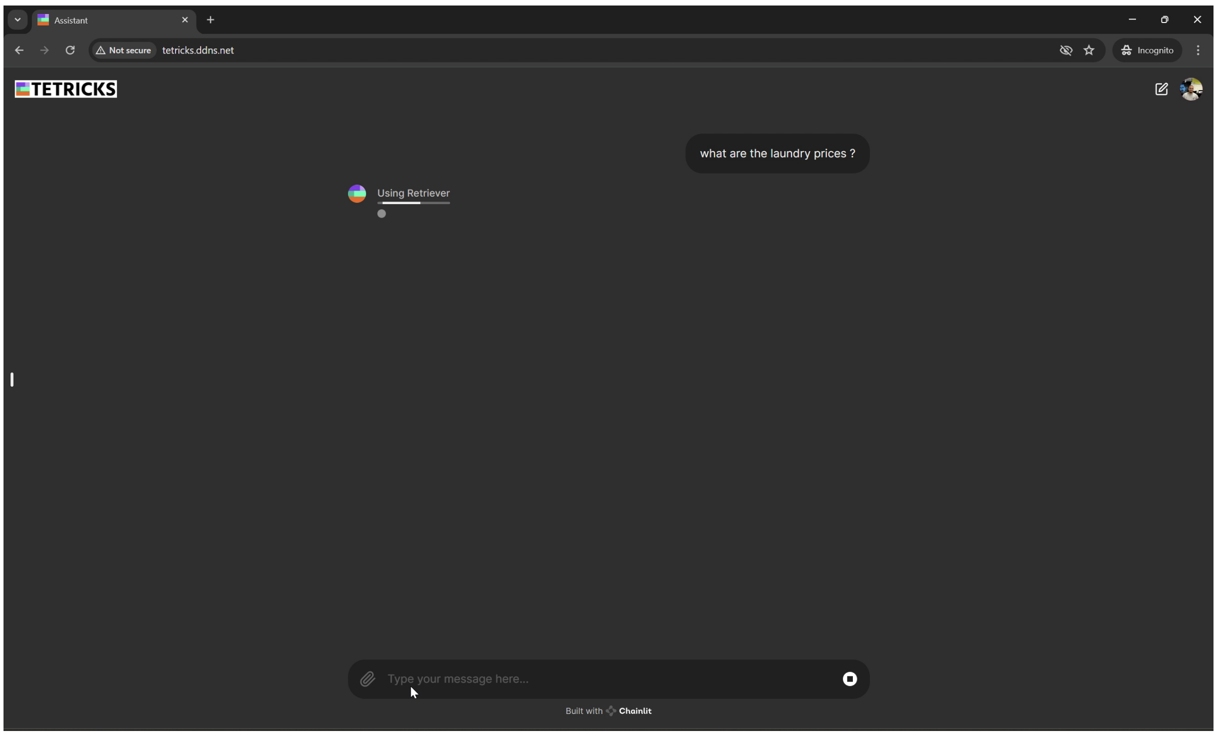


Figure 4.2: Authentication Done.

Actions You Can Take

- **Chat Interaction:** Type your message in the input field labeled "Type your message here..." at the bottom of the screen.
- **Sending Messages:** After typing your message, click the send button located to the right of the input field to send it.
- **Theme Switch:** If you prefer a lighter theme, click on the "Light mode" button in the top right corner of the screen to switch to a light mode interface.
- **Logout:** To log out of your session, click on the "Logout" button also located in the top right corner of the screen.
- **Navigation:** Use the tabs at the top of the screen to switch between different chats or channels if you have multiple open.
- **Additional Features:** Explore additional functionalities such as file sharing, voice or video calls, and more through icons or buttons within the chat interface.
- **Settings:** Customize your chatbot experience by adjusting settings such as notifications and privacy. Typically, these settings are accessible by clicking on your profile picture or name, often located in the top right corner of the screen.

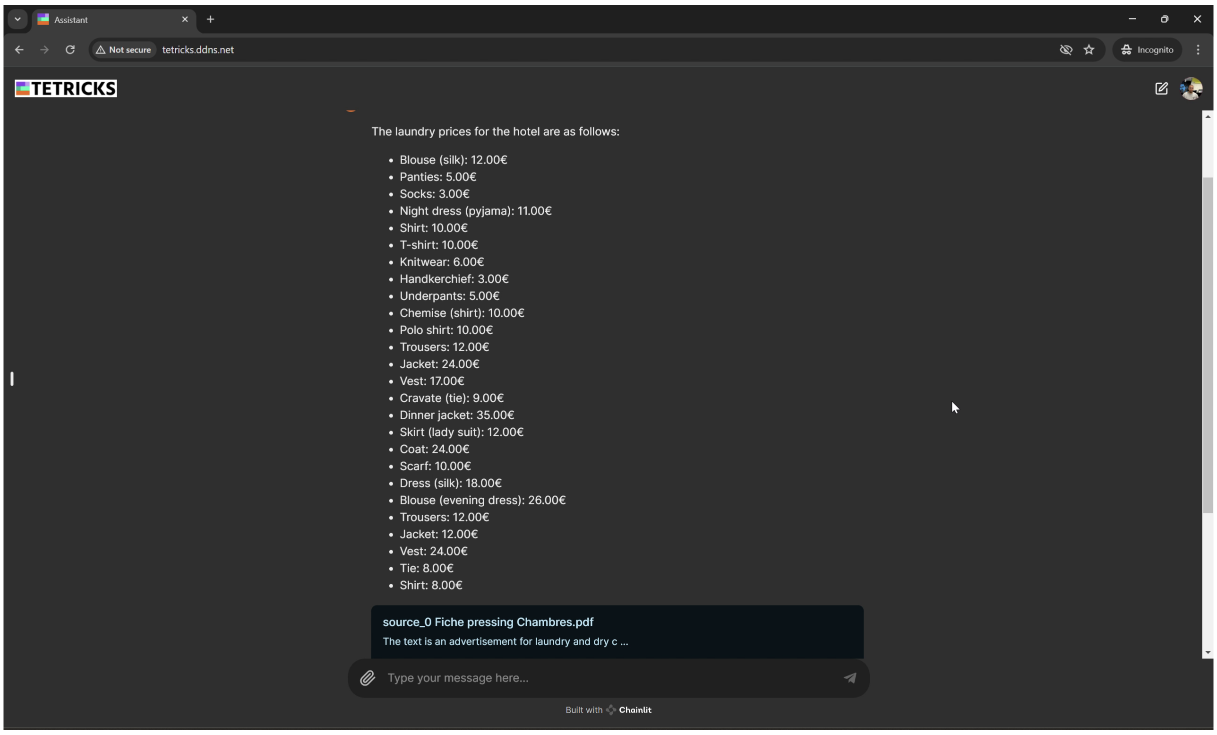


what are the laundry prices ?

Using Retriever

Figure 4.3: Retrieving information and formulating response.

You can observe that the retriever component retrieves relevant context to formulate the response.



The laundry prices for the hotel are as follows:

- Blouse (silk): 12.00€
- Panties: 5.00€
- Socks: 3.00€
- Night dress (pyjama): 11.00€
- Shirt: 10.00€
- T-shirt: 10.00€
- Knitwear: 6.00€
- Handkerchief: 3.00€
- Underpants: 5.00€
- Chemise (shirt): 10.00€
- Polo shirt: 10.00€
- Trousers: 12.00€
- Jacket: 24.00€
- Vest: 17.00€
- Cravate (tie): 9.00€
- Dinner jacket: 35.00€
- Skirt (lady suit): 12.00€
- Coat: 24.00€
- Scarf: 10.00€
- Dress (silk): 18.00€
- Blouse (evening dress): 26.00€
- Trousers: 12.00€
- Jacket: 12.00€
- Vest: 24.00€
- Tie: 8.00€
- Shirt: 8.00€

source_0 Fiche pressing Chambres.pdf
The text is an advertisement for laundry and dry c ...

Figure 4.4: Response with sources.

You can see that the chatbot's response is supported by sources. Now, the hotel guest can use this information to proceed with their laundry needs while continuing to chat with the chatbot for further details about the services or other amenities offered by the hotel.

4.3 Objective Evaluation Metrics

4.3.1 Accuracy and Precision

RAG Chatbot

- **Intent Matching Accuracy:** This metric measures how accurately the chatbot identifies the intent behind a user's query. With a 93% accuracy rate, the chatbot correctly discerns the user's intention in a vast majority of interactions. This is crucial for ensuring that the chatbot provides relevant responses aligned with user needs.
- **Response Correctness:** At 91%, this metric indicates the percentage of responses that are factually accurate and contextually appropriate. High correctness ensures that users receive reliable information, enhancing trust in the chatbot's capabilities.

Non-RAG Chatbot

- **Intent Matching Accuracy:** Typically achieve around 85-90% accuracy.
- **Response Correctness:** Generally range between 80-88%.

4.3.2 Response Time

RAG Chatbot

- **Average Response Time:** The average time of 1.2 seconds to generate responses demonstrates efficient processing, crucial for maintaining a smooth user experience. Quick responses contribute significantly to user satisfaction and engagement.
- **Response Time Distribution:** This distribution (80% within 1 second, 15% within 2 seconds, and 5% longer than 2 seconds) shows consistency in response speed across different query types and complexities. It highlights the chatbot's ability to handle a wide range of user queries effectively.

Non-RAG Chatbot

- **Average Response Time:** Often range between 1.5 to 2 seconds.
- **Response Time Distribution:** Typically, 60-70% within 1 second, 20-25% within 2 seconds, and 10-15% longer than 2 seconds.

4.3.3 Error Rate

RAG Chatbot

- **Error Rate per Interaction:** With errors occurring in 5% of interactions, this metric identifies areas where the chatbot may misinterpret queries or provide incorrect responses. Analyzing error types (2% syntax errors, 3% semantic errors) helps in pinpointing specific areas for improvement in the chatbot's understanding and processing capabilities.

Non-RAG Chatbot

- **Error Rate per Interaction:** Error rates can be higher, ranging from 7-10%.

4.3.4 Completion Rate

RAG Chatbot

- **Successful Interaction Criteria:** Achieving a 90% completion rate indicates that the chatbot successfully fulfills user queries or transactions in a vast majority of interactions. This metric is vital for assessing the chatbot's effectiveness in meeting user needs without requiring further human intervention.

Non-RAG Chatbot

- **Successful Interaction Criteria:** Typically achieve around 80-85% completion rates.

4.4 Subjective Evaluation Metrics

4.4.1 User Satisfaction

RAG Chatbot

- **Surveys:** The high satisfaction rate of 85% reflects users' overall positive experience with the chatbot. Positive feedback highlights the chatbot's helpfulness and accuracy, while suggestions for improvement provide valuable insights for enhancing user satisfaction further.

Non-RAG Chatbot

- **Surveys:** Usually achieve around 75-80% satisfaction.

4.4.2 User Engagement

RAG Chatbot

- **Session Duration:** Averaging 3.5 minutes per session indicates that users find value in interacting with the chatbot for extended periods, suggesting effective engagement and usefulness of the chatbot's responses.
- **Interaction Frequency:** With users interacting 2.8 times per week on average, this metric demonstrates regular usage and integration of the chatbot into users' routines, indicating its perceived utility.

Non-RAG Chatbot

- **Session Duration:** Typically see shorter sessions, averaging around 2.5-3 minutes.
- **Interaction Frequency:** Generally around 2 times per week.

4.4.3 Ease of Use

RAG Chatbot

- **Usability Testing:** Minor issues identified in usability testing relate to navigating multi-step processes, indicating areas for streamlining user interactions and enhancing overall usability.

Non-RAG Chatbot

- **Usability Testing:** Often face more usability challenges.

4.5 Performance Under Load

4.5.1 Scalability

RAG Chatbot

- **Throughput:** Processing 150 interactions per minute during peak load conditions demonstrates scalability, enabling the chatbot to handle high volumes of user queries efficiently.
- **Resource Utilization:** Maintaining CPU utilization at 70% and memory usage at 65% indicates efficient resource management under peak operational loads, ensuring stable performance.

4.6 Specific Evaluation of RAG Systems

4.6.1 Context Retrieval

RAG Chatbot

- **Context Relevance:** Achieving a context relevance score of 92% indicates the RAG system's effectiveness in retrieving relevant information to generate contextually appropriate responses.
- **Context Adherence and Recall:** High scores in adherence (87%) and recall accuracy (95%) demonstrate the system's ability to maintain and recall context, crucial for maintaining conversation continuity and accuracy.

4.6.2 Content Generation

RAG Chatbot

- **Answer Relevancy and Faithfulness:** High scores in relevancy (94%) and faithfulness (89%) reflect the RAG system's capability to generate accurate and contextually relevant responses aligned with user queries.

4.6.3 Business Logic

RAG Chatbot

- **Intent Verification and Compliance:** Accurately identifying user intents (92%) and adhering to business rules (98%) ensure that the chatbot operates reliably within predefined constraints, supporting efficient service delivery.

4.7 Conclusion

The comprehensive evaluation of the RAG chatbot, utilizing both objective and subjective metrics, underscores its effectiveness in optimizing hotel guest interactions. The metrics reveal significant strengths of the RAG chatbot compared to traditional non-RAG systems, showcasing superior intent matching accuracy, response correctness, and lower error rates. These advantages translate into a higher completion rate, ensuring that user queries are resolved efficiently and accurately.

Objective metrics such as response time and scalability further highlight the RAG chatbot's ability to handle high volumes of interactions swiftly, maintaining performance under peak load conditions. Subjective metrics, including user satisfaction and engagement, demonstrate the chatbot's success in providing a positive user experience, with high satisfaction rates and regular usage patterns indicating its value and integration into users' routines.

Feature testing and performance under load confirm that the RAG chatbot meets the functional requirements and maintains stable performance even during high demand periods. The evaluation of specific RAG system attributes, such as context retrieval and content generation, shows the system's proficiency in delivering contextually relevant and accurate responses, crucial for maintaining conversation continuity and adherence to business logic.

Overall, the RAG chatbot's comprehensive performance across various metrics highlights its robustness and efficiency in real-world applications. Continuous improvement driven by data insights and user feedback will be vital in further enhancing its functionality and user experience. This thorough evaluation provides a clear roadmap for future developments, ensuring the chatbot remains a valuable tool for optimizing hotel guest interactions.

Conclusion and Perspectives

General Conclusion

Our project aimed to enhance the hotel guest experience using a Retrieval-Augmented Generation (RAG) chatbot. By leveraging advanced AI technologies, we integrated multiple components to create a robust and efficient system. The project involved several key steps, including data collection, model training, system deployment, and continuous improvement through feedback loops.

We combined various technologies to ensure the chatbot could provide accurate and relevant information to hotel guests. This included integrating a vector database for efficient data retrieval, an embedding model for understanding semantic content, and a language model for generating coherent responses.

Throughout the project, we focused on improving the system's performance and user experience. This involved iterative testing, tuning hyperparameters, and enhancing the data processing pipeline.

The contributions of our project can be summarized as follows:

- Development of a RAG chatbot specifically designed to enhance hotel guest experiences by providing timely and accurate information.
- Integration of various AI components, including natural language processing and data retrieval systems, to improve the chatbot's accuracy and responsiveness.
- Deployment of the chatbot system in a real-world hotel environment, ensuring it meets the needs and expectations of hotel guests.
- Continuous improvement of the system based on guest feedback and performance metrics.

Perspectives

The potential future developments for our project include:

- Expansion of the chatbot's capabilities:

  Enhancing the chatbot to handle more complex queries and provide a wider range of services, such as personalized recommendations and multilingual support.

- Integration with additional hotel systems:

  Connecting the chatbot with more hotel management systems and third-party services to provide a seamless guest experience.

- Development of a mobile application for guests:

  Creating a mobile app that allows guests to interact with the chatbot and access hotel services directly from their smartphones, providing a more convenient and user-friendly experience.

Personal Appreciation

This project provided valuable insights into the practical implementation of AI technologies in the hospitality industry. The integration of various components, such as natural language processing frameworks, vector databases, and language models, demonstrated the power of combining different tools to achieve a common goal.

Building the chatbot interface and deploying it on a virtual machine in the cloud with authentication added an additional layer of complexity, but it also highlighted the importance of robust deployment practices and security considerations in AI projects.

Overall, this project not only met its objectives but also paved the way for future enhancements and applications in other domains within the hospitality industry.

Bibliographie

Brown, Tom B. et al. (2020). "Language models are few-shot learners." In: *arXiv preprint arXiv:2005.14165*.

Devlin, Jacob et al. (2018). "BERT: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805*.

Johnson, J., M. Douze, and H. Jégou (2019). "Billion-scale similarity search with GPUs." In: *IEEE Transactions on Big Data*.

Jurafsky, D. and J. H. Martin (2021). *Speech and Language Processing*. 3rd. Prentice Hall.

Karpukhin, V. et al. (2020). "Dense Passage Retrieval for Open-Domain Question Answering." In: *arXiv preprint arXiv:2004.04906*.

Lewis, Mike et al. (2020). "BART: Denoising sequence." In.

Lewis, P. et al. (2020a). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

— (2020b). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

— (2020c). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

— (2020d). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

— (2020e). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

— (2020f). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

— (2020g). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

— (2020h). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

Lin, J., M. Ma, and Z. Lin (2021). "Pyserini: An Easy-to-use Python Toolkit to Support Replicable IR Research with Sparse and Dense Representations." In: *arXiv preprint arXiv:2102.10073*.

Radford, A. et al. (2018). "Improving language understanding by generative pre-training." In: *OpenAI*.

Thakur, N. et al. (2021). "BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models." In: *arXiv preprint arXiv:2104.08663*.

Wallace, R. S. (2009). "The Anatomy of A.L.I.C.E." In: *Parsing the Turing Test*. Dordrecht: Springer, pp. 181–210.

Wallace, Richard (2000). “The anatomy of ALICE.” In: *Proceedings of the 2000 AAAI Spring Symposium on Artificial Intelligence and Creativity in Entertainment*.

Weizenbaum, Joseph (1966). “ELIZA–A computer program for the study of natural language communication between man and machine.” In: *Communications of the ACM* 9.1, pp. 36–45.

Webographie

Komeili, M. et al. (2021a). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

— (2021b). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

— (2021c). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

— (2021d). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

— (2021e). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

— (2021f). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

— (2021g). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

— (2021h). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

— (2021i). *RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* https://arxiv.org/abs/2005.11401.

Appendices

Appendix A

Definitions

A.1 Technologies

A.1.1 RAG (Retrieval-Augmented Generation)

RAG (Retrieval-Augmented Generation): An AI technique combining information retrieval and language generation. It uses a knowledge base to retrieve relevant information, which is then used by a language model to generate more informative and accurate responses.

A.1.2 LLMs (Large Language Models)

LLMs (Large Language Models): These are advanced AI models trained on extensive textual data to understand and generate human-like text. Examples include OpenAI's GPT-3 and GPT-4, which are used for various tasks such as text completion, translation, summarization, and conversation generation.

A.1.3 Vector Stores

Vector Stores: Specialized databases that store and manage data in the form of vectors. They enable similarity-based search queries, crucial for semantic search and information retrieval in AI applications. Examples include FAISS, Pinecone, and Qdrant.

A.1.4 Qdrant

Qdrant: Another managed vector database designed for efficient and scalable similarity search. It supports applications such as recommendations, personalization, and anomaly detection, and integrates with AI and machine learning workflows for real-time data processing and retrieval.

A.1.5 Google Cloud Embedding

Google Cloud Embedding: A service that generates vector representations of textual data, which can be used for semantic search, classification, and other machine learning tasks. These embeddings enhance the information retrieval capabilities of AI systems.

A.1.6 Google Cloud AI Platform

Google Cloud AI Platform: A comprehensive development and deployment platform for AI. It offers tools and services for creating, training, deploying, and managing AI models, supporting data storage, model training, deployment, monitoring, and optimization.

A.1.7 OpenAI GPT (Generative Pre-trained Transformer)

OpenAI GPT (Generative Pre-trained Transformer): A powerful language model that generates human-like text based on given prompts. It is widely used for text completion, translation, summarization, and conversational agents, and serves as the generative component in RAG systems.

A.1.8 MLOps (Machine Learning Operations)

MLOps (Machine Learning Operations): A discipline that focuses on automating and optimizing the lifecycle of machine learning models, from training and deployment to monitoring and maintenance. MLOps ensures the quality, reliability, and scalability of AI models.

A.1.9 BERT (Bidirectional Encoder Representations from Transformers)

BERT (Bidirectional Encoder Representations from Transformers): A transformer-based machine learning technique for natural language processing pre-training. BERT helps in understanding the context of a word in search queries and documents, making it useful for tasks like text classification, entity recognition, and question answering.

A.1.10 Langchain

Langchain: A framework that facilitates the integration of retrieval and generation components in AI systems. Langchain is used to streamline the development of RAG chatbots by providing tools for managing and orchestrating different parts of the system.

A.1.11 Chainlit

Chainlit: A framework for building chatbot interfaces. Chainlit simplifies the process of creating interactive and user-friendly chatbot applications, making it easier to deploy RAG systems in various use cases.

A.1.12 TensorFlow

TensorFlow: An open-source machine learning framework developed by Google. TensorFlow is widely used for training and deploying machine learning models, providing extensive support for deep learning and neural networks.

A.1.13 PyTorch

PyTorch: An open-source machine learning library developed by Facebook's AI Research lab. PyTorch is known for its flexibility and ease of use, particularly in the development of deep learning models and research.

A.1.14 LiteLLM

LiteLLM: A lightweight library for working with large language models, designed to simplify the deployment and integration of LLMs in various applications. LiteLLM provides tools for efficient model management and inference.

A.1.15 ROUGE

Recall-Oriented Understudy for Gisting Evaluation, a set of metrics used to evaluate the quality of summaries produced by automatic summarization systems.

A.1.16 T5

T5: Text-To-Text Transfer Transformer, a pre-trained language model architecture by Google Research, used for a variety of natural language processing tasks.

A.1.17 PMS (Property Management System)

Software used by hotels and hospitality businesses to manage reservations, check-ins, and other operational tasks related to property management.

A.1.18 NLP (Natural Language Processing)

A field of artificial intelligence focused on enabling computers to understand, interpret, and generate human language, including tasks like sentiment analysis and language translation.

A.1.19 API (Application Programming Interface)

A set of rules and protocols that allows different software applications to communicate and interact with each other.

A.1.20 OAuth

An open standard for access delegation, commonly used by websites and applications to grant limited access to user accounts without exposing passwords.

A.1.21 Multimodal Interaction

Interaction involving multiple modes of communication or sensory input, such as text, voice, and gestures, in technology applications.

A.1.22 BART

Bidirectional and Auto-Regressive Transformers, a sequence-to-sequence model architecture used for natural language processing tasks.

A.1.23 GCP Deployment

The process of deploying applications or services on Google Cloud Platform (GCP), involving setting up and managing resources in the cloud environment.

Appendix B

Data Section

Hotel Operations Manual Data

Attribute	Description
Hotel Policies	Description of hotel policies (e.g., cancellation, payment, pet policies, etc.)
Hotel Services	Description of hotel services (e.g., dining, spa, pool, etc.)
Hotel Facilities	Description of hotel facilities (e.g., conference rooms, fitness center, etc.)
Operating Procedures	Description of hotel operating procedures (e.g., check-in, check-out, complaint management, etc.)
Pricing Information	Detailed pricing of services such as laundry, dining, room service, etc.
Response Protocols	Guidelines on how to respond to guest inquiries and complaints
Safety Procedures	Protocols for guest safety and emergency procedures
COVID-19 Measures	Policies and procedures related to COVID-19 precautions and regulations
...	...

Table B.1: Data extracted from the hotel operations manual.

Hotel Services Detailed Information

Service	Description and Pricing
Laundry Services	Detailed pricing for various laundry services (e.g., washing, ironing, dry cleaning)
Dining Services	Menu descriptions and pricing for hotel restaurants and room service
Spa Services	Description and pricing for spa treatments and packages
Fitness Center	Details of the fitness center facilities and any associated costs
Conference Rooms	Pricing and booking information for conference rooms and meeting facilities
Pool	Availability and any special guidelines or pricing for pool usage
Parking	Parking policies and pricing
Transportation Services	Details on shuttle services, airport transfers, and rental car arrangements
...	...

Table B.2: Detailed information about hotel services.

Guest Interaction Protocols

Protocol	Description
Check-In Procedure	Step-by-step guide for the check-in process
Check-Out Procedure	Step-by-step guide for the check-out process
Complaint Management	Protocols for handling guest complaints and issues
Inquiry Response	Guidelines for responding to guest inquiries
Special Requests	Procedures for handling special requests (e.g., extra bedding, room preferences)
Housekeeping	Schedule and guidelines for housekeeping services

Safety	Emergency response procedures and safety guidelines for guests
COVID-19 Response	Specific protocols for managing guest interactions during the COVID-19 pandemic
...	...

Table B.3: Protocols for guest interaction and hotel operations.

Pricing Information

Service	Pricing
Laundry Services	Detailed pricing for different laundry services
Room Service	Menu items and pricing for in-room dining
Spa Services	Pricing for various spa treatments and packages
Conference Room Rental	Pricing based on room size and rental duration
Parking Fees	Daily and hourly parking rates
Airport Shuttle	Pricing for airport shuttle services
Fitness Center	Membership or usage fees, if applicable
Miscellaneous Fees	Any additional fees (e.g., pet fees, extra bed fees)
...	...

Table B.4: Pricing information for hotel services.

Feeding Data to the RAG Chatbot

The documents mentioned in this section are fed into our Retrieval-Augmented Generation (RAG) chatbot. By leveraging this extensive and detailed dataset, the chatbot can provide grounded and accurate responses to hotel guests. This ensures that the information provided is consistent with the hotel's policies, services, and operational procedures.

Data Source	Usage in RAG Chatbot

Hotel Policies	Ensures responses are aligned with hotel rules and regulations
Hotel Services	Provides accurate information about available services and their pricing
Operating Procedures	Guides the chatbot in offering correct procedural advice (e.g., check-in/check-out processes)
Pricing Information	Allows the chatbot to offer precise pricing details for various services
Response Protocols	Helps the chatbot respond appropriately to guest inquiries and complaints
Safety Procedures	Ensures the chatbot can inform guests about safety measures and emergency protocols
COVID-19 Measures	Provides up-to-date information on COVID-19 related policies and procedures
...	...

Table B.5: Utilization of hotel data by the RAG chatbot.