Bangladesh travel bot: RAG application using LLM

by

Mst. Ayesha Siddika

23373010

A Project submitted to the Department of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of

M.Sc. in Computer Science

Department of Computer Science and Engineering

Brac University

7th October 2025

© 2025. Mst Ayesha Siddika

All rights reserved.

Declaration

It is hereby declared that

1. The Project submitted is my own original work while completing degree at Brac University.

2. The Project does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The Project does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Mst Ayesha Siddika

ID: 23373010

Approval

The project titled "Bangladesh travel bot: RAG application using LLM" submitted by

Mst. Ayesha Siddika (23373010)

Of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of M.Eng. in Computer Science on October 7, 2025.

Examining Committee:

Supervisor:

Mr. Moin Mostakim

Senior Lecturer

Department of Computer Science and Engineering

BRAC University

Program Coordinator:

Dr. Md Sadek Ferdous

Professor

Department of Computer Science and Engineering

BRAC University

Head of Department:

(Chair)

Dr. Sadia Hamid Kazi

Associate Professor and Chairperson

Department of Computer Science and Engineering

BRAC University

Abstract

The "Bangla Travel-Bot" project aims to create an intelligent and user-friendly chatbot that provides accurate, context-specific responses to travel-related queries for Bangla-speaking travelers. As travel information in Bangla is often limited, this project seeks to fill the gap by offering a platform that ensures seamless access to vital travel information in the user's native language. The chatbot is designed to assist travelers with a variety of needs, such as recommendations for tourist destinations, transportation options, accommodation, local customs, and real-time updates on weather, events, and travel logistics.

This system leverages the power of a Retrieval-Augmented Generation (RAG) framework, which enhances the bot's ability to generate high-quality responses by retrieving relevant information from trusted travel sources. The integration of LLaMA 3, a state-of-the-art language model, ensures the chatbot can effectively process and respond in Bangla, maintaining both linguistic accuracy and cultural relevance. The chatbot is capable of understanding user queries and providing personalized recommendations based on location, preferences, and travel context. Through the use of advanced natural language processing (NLP) techniques, the "Bangla Travel-Bot" offers an intuitive conversational interface, ensuring that users can easily interact with the system. Whether it's helping with last-minute travel plans or offering insightful tips about a destination, the chatbot strives to enhance the overall travel experience for Bangla-speaking users. This project not only aims to simplify the process of planning and managing travel for Bangla speakers but also contributes to the development of accessible, language-specific travel assistance tools in the broader field of artificial intelligence and human-computer interaction..

Keywords: Bangla Travel-Bot, chatbot, travel assistance, Retrieval-Augmented Generation (RAG), LLaMA 3, natural language processing (NLP), travel queries, personalized recommendations, contextual responses, Bangla-speaking travelers..

Dedication

I want to dedicate this project to my parents.

Acknowledgement

Firstly, all praise to the Great Allah for whom my Project have been completed without any major interruption.

Secondly, to my supervisor Mr. Moin Mustakim sir for his kind support and advice in our work. He helped me whenever I needed help.

And finally to my family without their throughout sup-port it may not be possible. With their kind support and prayer I am now capable of completeing this project.

Table of Contents

List of Figures

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

RAG Retrieval-Augmented Generation

ϵ Epsilon

υ Upsilon

BGE BAAI General Embedding

COT Contextual Accuracy (CoT Accuracy)

ODI AAI-bge-m3 Embedding Model

ODI LLaMA 3-8b Model

$T20$ LangChain RAG Framework

Chapter 1

Introduction

1.1 Introduction

Traveling, whether for leisure or business, has become an integral part of modern life [9], with individuals seeking new experiences, cultures, and destinations. However, despite the vast amounts of information available, language barriers and lack of localized resources often pose significant challenges, especially for non-English-speaking travelers [1], [5]. For Bangla-speaking individuals, the absence of comprehensive, accessible travel resources in their native language complicates their ability to navigate the complexities of travel, including understanding local customs, finding appropriate accommodations, or simply getting relevant, timely information on the go.

The Bangla Travel-Bot project aims to address this issue by providing an intelligent, user-friendly solution for Bangla-speaking travelers. By leveraging advanced technologies such as Retrieval-Augmented Generation (RAG) and LLaMA 3 [3] and LLaMA 3 [8], this chatbot will assist users in making informed decisions about their travel plans by offering contextually accurate and relevant responses in Bangla. The chatbot is designed to support a variety of travel-related queries, from suggestions for tourist attractions and local amenities to real-time updates on transportation, accommodations, and even weather forecasts.

This project focuses on the development of a system that integrates natural language

processing (NLP) techniques with artificial intelligence (AI) to create a conversational interface that can understand, process, and respond to queries in Bangla. The system's primary goal is to enhance the travel experience by providing a localized, easily accessible platform for Bangla-speaking individuals to navigate the complexities of travel with greater ease and confidence. The Bangla Travel-Bot will not only serve as a personal travel assistant but will also contribute to the broader field of language-specific AI systems [2], [7] by advancing the capabilities of multilingual chatbots.

In addition, the Bangla Travel-Bot will enable travelers to receive personalized recommendations based on their preferences and location, making the experience more intuitive and tailored. This project is expected to bridge a significant gap in the travel assistance market, providing a valuable tool for Bangla-speaking travelers who have historically lacked easily accessible, culturally relevant travel information in their native language. Through this research, the aim is to explore the potential of AI-driven travel solutions that cater to diverse linguistic needs, improving accessibility, and enriching the travel experience for all.

1.2 Problem statement

Travelers often face challenges in obtaining accurate and context-specific information in their native language, particularly in regions where digital resources are limited. For Bangla-speaking users, existing chatbot solutions either lack robust natural language understanding or fail to provide localized, context-aware assistance. This creates a significant barrier for users seeking reliable travel-related information in Bangla. The absence of an intelligent, user-friendly, and language-specific chatbot system highlights the need for a solution that integrates advanced NLP capabilities to address these challenges. The "Bangla Travel-Bot" aims to fill this gap by offering precise and context-sensitive assistance, enhancing travel experiences for Bangla speakers.

1.3 Aim of Study

The primary aim of this study is to develop an intelligent and user-friendly chatbot, "Bangla Travel-Bot," designed specifically to assist Bangla-speaking travelers. By leveraging the Retrieval-Augmented Generation (RAG) framework and the advanced capabilities of LLaMA 3, this project seeks to provide accurate, context-aware, and linguistically fluent responses to travel-related queries in Bangla. The study also aims to address the challenges of natural language understanding and generation in Bangla, ensuring a seamless and culturally relevant user experience for travelers.

1.4 Reserach Methodology

This study employs an applied research approach to develop and evaluate the "Bangla Travel-Bot" using advanced natural language processing techniques [7].The research begins with the creation of a comprehensive dataset of Bangla travel-related queries and responses, collected from online forums, travel guides, and websites. The data is preprocessed to address Bangla-specific linguistic challenges, including spelling variations and diacritic inconsistencies The chatbot utilizes the Retrieval-Augmented Generation (RAG) framework [3], integrated with the LLaMA 3 model [8], to retrieve relevant information and generate accurate, context-aware responses.to retrieve relevant information from a domain-specific knowledge base and generate accurate, context-aware responses. Fine-tuning the model on travel-specific data enhances its understanding and response quality. The system's architecture includes a user-friendly interface for interaction, while the backend leverages RAG to ensure efficient information retrieval and high-quality responses. The chatbot's performance is evaluated using metrics like response accuracy, contextual relevance, and user satisfaction, with iterative refinements based on feedback from Bangla-speaking travelers to ensure an effective and culturally relevant solution

Finally, The model is fine-tuned on travel-specific data to improve its contextual understanding [6]. Performance is evaluated using established metrics in dialogue

system research, such as response accuracy, contextual relevance, and user satisfaction [4].

Chapter 2

Related Work

The development of generative artificial intelligence (GenAI) has accelerated with the integration of large language models (LLMs) and Retrieval-Augmented Generation (RAG), enabling more relevant and informative responses [3], [8]. Previous research has explored the use of chatbots in domain-specific tasks such as healthcare, customer support, and travel assistance. Early chatbots were primarily rule-based or relied on intent classification models, which limited their ability to generate nuanced responses.

Recent advancements in RAG-based systems have enhanced response quality by combining generative capabilities with accurate information retrieval. In the context of Bangla natural language processing (NLP), researchers have identified challenges including limited datasets, complex morphology, and orthographic inconsistencies [7]. However, recent efforts leveraging transformer-based models like BanglaBERT and multilingual LLMs have demonstrated promising improvements. The "Bangla Travel-Bot" builds on these advances by integrating a robust RAG framework with a curated domain-specific travel dataset. Unlike traditional chatbots that rely on static, pre-defined responses, this system dynamically retrieves and generates personalized travel recommendations, making it a valuable tool for Bangla-speaking users.

With the rise of global tourism and rapid advances in AI, intelligent travel planning services have become a central research focus. Traditional systems—whether rule-

based or LLM-powered—often struggle to balance personalization, contextual understanding, and rational decision-making. Although previous studies have proposed AI-based recommendation systems for travel, they often lack real-world adaptability. Recent innovations in RAG architectures and spatiotemporal-aware algorithms have significantly improved such systems by incorporating real-time data and user-specific preferences [12]..

Baichuan Mo.et al introduced TravelAgent, an AI-powered travel planning system that leverages large language models (LLMs) and an agent-based framework to address these challenges. Unlike traditional methods that rely on static data or predefined rules, TravelAgent integrates real-time data processing, a dynamic memory module, and spatiotemporal-aware algorithms to refine travel plans based on evolving user interactions. The study evaluates TravelAgent through simulated and real-world user assessments, demonstrating its effectiveness in generating rational, comprehensive, and personalized itineraries. By addressing key limitations in existing travel AI solutions, this work contributes to the advancement of AI-driven travel assistants and highlights the potential of LLM-based applications in intelligent travel planning. [12].

In another study, Shaowei Ying et al. assessed the application of GenAI in urban transportation planning, focusing on the capabilities of GPT-4 and Phi-3-mini models [11]. Using a transportation-specific evaluation framework, the study examined LLMs' performance in managing planning concepts, GIS reasoning, and pricing models. Results showed that GPT-4 consistently outperformed Phi-3-mini in accuracy and reliability, although the latter demonstrated efficiency in resource-constrained settings. These findings underline the transformative role of LLMs in transportation management.

Despite their potential, LLMs raise ethical and fairness concerns, especially in hospitality and tourism. Prior research has exposed demographic biases in LLM-generated

recommendations [10]. Models may reflect or amplify stereotypes based on race, gender, or ethnicity, impacting both user trust and recommendation fairness. Xu et al. investigated such biases by analyzing the outputs of travel-planning LLMs and observed patterns where recommendations aligned with demographic assumptions. They proposed a STOP-Word classification technique that successfully reduced discriminatory bias without degrading response quality. Nonetheless, hallucinations—factually incorrect or misleading outputs—remain a challenge, especially for underrepresented groups.

These studies collectively highlight the potential and limitations of GenAI in personalized travel planning. The Bangla Travel-Bot aims to contribute to this evolving field by delivering culturally relevant, linguistically accurate, and ethically responsible travel recommendations for Bangla-speaking users.

Chapter 3

Data Collection And Methodology

3.1 Dataset Collection

This system utilizes data sourced from website - [13], a well-established Bangla travel blog, serving as a comprehensive repository of travel-related knowledge. The blog's content spans a wide array of essential information for travelers, including detailed destination guides, meticulously crafted itineraries, insightful cultural observations, practical local tips, and crucial logistical advice. This rich data source offers a unique window into travel experiences and preferences within the Bangla-speaking community. To harness this valuable information, a web scraping process is employed to extract the data directly from the blog's website. This automated extraction ensures the capture of a substantial volume of diverse travel-related content, providing a solid foundation for the system's analytical capabilities.

Following the data extraction, a crucial phase of data cleaning and preprocessing is undertaken. This step is vital for ensuring the quality and usability of the data for subsequent analysis. The cleaning process involves addressing inconsistencies, removing irrelevant information, and rectifying errors that may have been introduced during the scraping process. Preprocessing further refines the data by transforming it into a structured format suitable for machine learning or natural language processing tasks. . By meticulously cleaning and preprocessing the data, the system

lays the groundwork for accurate and meaningful insights, ensuring that the final output is reliable and reflective of the original source material.

3.2 Data Analysis

The successful processing of Bangla text necessitates specialized techniques, as the language possesses unique linguistic characteristics. To optimize dataset quality and enhance model performance, the system implemented tailored tokenization, stemming, and noise removal procedures. Tokenization, the process of breaking down text into individual units (tokens), was adapted to handle the complexities of Bangla word formation and sentence structure. Stemming, which reduces words to their root form, was adjusted to accommodate the specific morphological rules of Bangla, ensuring accurate root identification. Furthermore, noise removal, which eliminates irrelevant characters and symbols, was finely tuned to address the common sources of noise in Bangla text, such as diacritics and special characters. These customized preprocessing steps significantly improved the cleanliness and consistency of the dataset.

To effectively manage the length of the textual data for processing by large language models, a RecursiveCharacterTextSplitter was employed. This technique divided the preprocessed Bangla text into manageable chunks of 512 characters. This chunking strategy is crucial for preventing the input data from exceeding the maximum token limit of the LLM, which can lead to truncation or processing errors. By splitting the text into smaller, consistent segments, the system ensures that the model can process the information efficiently and accurately, maintaining the integrity and context of the original content. This approach facilitates the handling of lengthy travel blog posts and enables the extraction of detailed insights from the data.

3.3 Methodology

This diagram illustrates a Retrieval Augmented Generation (RAG) system, a powerful approach for enhancing Large Language Model (LLM) performance. By grounding the LLM in a specific dataset, RAG enables accurate and contextually relevant responses to user queries. The process involves data preparation, vectorization, retrieval, and response generation

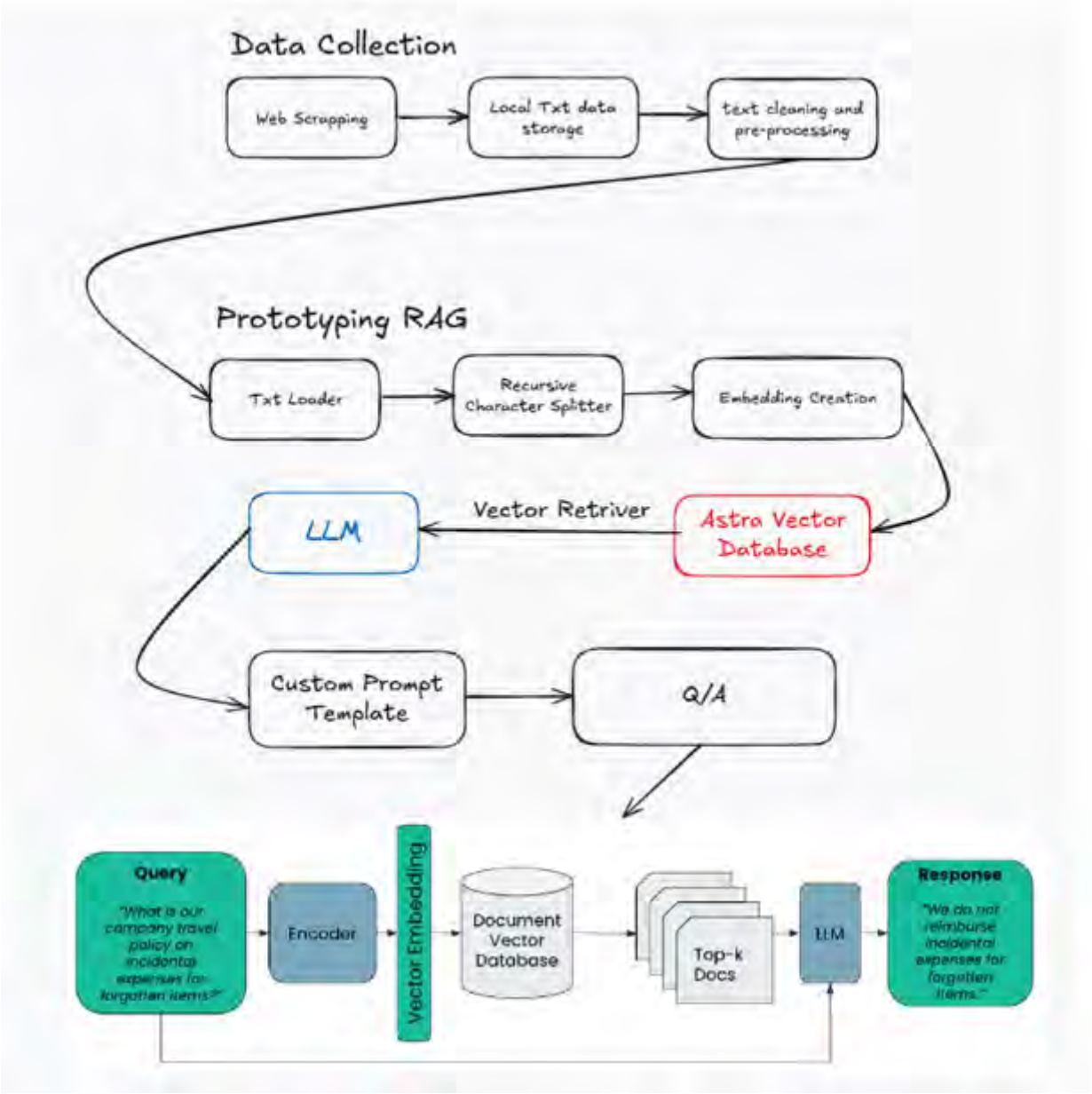

Figure 3.1: Workflow

This diagram depicts a system employing Retrieval Augmented Generation (RAG) to answer queries based on a collected dataset. The process begins with Data Collection, where information is gathered through Web Scraping. This data is then stored in Local Txt data storage, acting as a raw repository. Subsequently, the data undergoes text cleaning and pre-processing, which likely involves removing irrelevant characters, standardizing text formatting, and potentially other tasks to prepare it for further analysis.

The next phase, Prototyping RAG, outlines the core functionality of the system. First, the cleaned text data is loaded using a Txt Loader. This loaded data is then fragmented into smaller, manageable chunks using a Recursive Character Splitter, which is crucial for handling lengthy documents and ensuring the language model can process them effectively. Following this, Embedding Creation takes place. Here, the text chunks are converted into numerical vector representations, known as embeddings, which capture the semantic meaning of the text. These embeddings are then stored in an Astra Vector Database, a specialized database designed for efficient storage and retrieval of vector data.

The system utilizes a LLM (Large Language Model), which is central to both the embedding creation and the final response generation. The Vector Retriever component facilitates the search for relevant information within the Astra Vector Database. It retrieves the most similar embeddings to a given query, effectively finding the most relevant text chunks. The process of retrieving relevant information is guided by a Custom Prompt Template, which likely formats the query and retrieved information in a way that optimizes the LLM's performance. The final step in this phase is Q/A, which involves the LLM generating a response based on the retrieved information and the query.

The lower part of the diagram illustrates the query processing and response gener-

ation in more detail. A Query is inputted, such as "What is our company travel policy on incidental expenses for forgotten items?" This query is then processed by an Encoder, which converts it into a vector embedding, similar to the text chunks. This embedding is used to query the Document Vector Database, which returns the Top-k Docs, or the most relevant documents based on vector similarity. The LLM then generates a Response based on these retrieved documents, which in this example is "We do not reimburse incidental expenses for forgotten items.

The diagram highlights the key components of a RAG system: data collection and preparation, vector embedding and storage, information retrieval, and response generation using a large language model. It emphasizes the importance of each step in ensuring the system can accurately answer queries based on the provided dataset. The use of a vector database and embedding model allows for semantic search, enabling the system to understand the meaning behind the query and retrieve relevant information even if the exact words are not present in the database.

3.4 APP Interface

This diagram illustrates the core APP Workflow of a Retrieval Augmented Generation (RAG) system, designed to provide contextually relevant responses to user queries. The process is streamlined and focused on efficient information retrieval and generation.

The workflow begins with a User Query, which serves as the input to the system. This query represents the user's information need and triggers the subsequent steps. The query is then processed by an Encode component. This step involves converting the user's textual query into a numerical vector representation, known as an

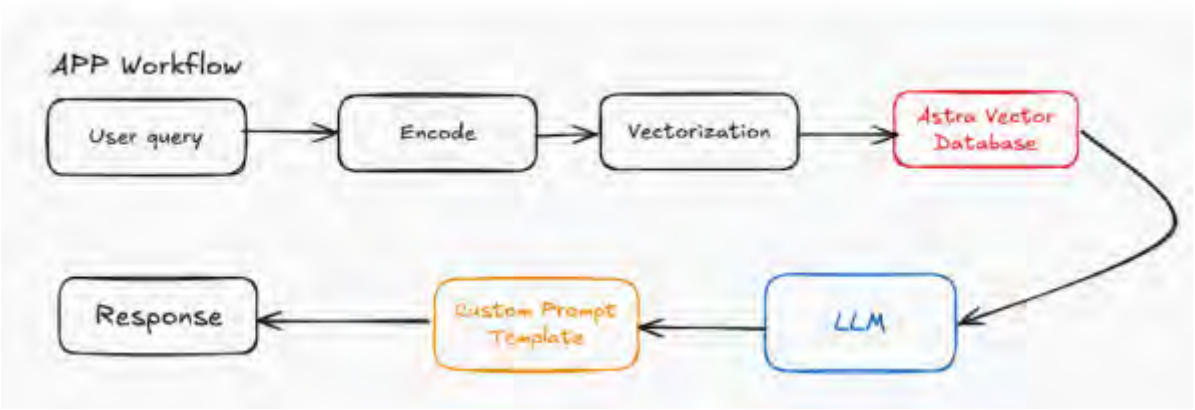

Figure 3.2: App Workflow

embedding. This embedding captures the semantic meaning of the query, allowing the system to understand the user's intent beyond the literal words used.

Following encoding, the vector embedding undergoes Vectorization, a process that further refines and prepares the embedding for efficient comparison and retrieval. The resulting vector is then used to query the Astra Vector Database. This database is specifically designed for storing and retrieving high-dimensional vector data, enabling rapid similarity searches. The database returns the most relevant documents or text chunks based on the similarity between the query vector and the stored vectors.

The retrieved information from the Astra Vector Database is then passed to a LLM (Large Language Model). This model is responsible for generating the final response to the user's query. However, before the LLM can generate a response, the retrieved information is formatted and structured using a Custom Prompt Template. This template plays a crucial role in guiding the LLM to focus on the relevant information and generate a coherent and accurate response. The template might include instructions, examples, or specific formatting requirements to optimize the LLM's performance.

Finally, the LLM generates a Response, which is presented to the user. This re-

sponse is based on the retrieved information and the user's query, ensuring that it is contextually relevant and accurate. The entire workflow is designed to provide a seamless and efficient user experience, enabling users to access information quickly and easily. The use of vectorization and a specialized vector database allows for semantic search, ensuring that the system can understand the meaning behind the query and retrieve relevant information even if the exact words are not present in the database.

Chapter 4

Model Selection

In the "Bangla Travel-Bot" project, the choice of technology stack was driven by the need to create an effective and efficient system for Bangla-speaking travelers. The specific models were selected based on their strengths in handling the complexities of the Bangla language, providing accurate information retrieval, and delivering high-quality responses.

4.1 LLaMA 3-8b Model:

The selection of the LLaMA 3-8b model as the foundational architecture for the Travel-Bot was a decision driven by a confluence of factors, primarily its demonstrated prowess in language modeling and its inherent adaptability. In the context of a chatbot designed to facilitate travel-related inquiries in the Bangla language, the model's capacity to comprehend and generate human-quality text was paramount. Bangla, with its rich tapestry of linguistic complexities, presents a unique challenge for natural language processing systems. The language's intricate system of inflections, the prevalence of compound words, and the diverse stylistic variations encountered in travel blogs and other relevant source materials necessitate a model capable of nuanced understanding. LLaMA 3's state-of-the-art language modeling capabilities, particularly its ability to navigate these linguistic intricacies, positioned it as an ideal candidate for this specific application.

The choice of an open-source model like LLaMA 3 was also strategically motivated by the need for customization and fine-tuning. A proprietary model, while potentially possessing advanced capabilities, would have imposed limitations on our ability to adapt it to the specific demands of the travel domain and the Bangla language. The inherent flexibility of an open-source architecture enabled us to tailor the model's parameters and training data to optimize its performance for travel-related queries. This process of fine-tuning involved exposing the model to a curated dataset of Bangla travel blogs, destination descriptions, and user queries, allowing it to internalize the specific vocabulary, syntax, and conversational patterns associated with travel in the Bangla context. This targeted training was crucial in ensuring that the Travel-Bot could provide accurate, contextually relevant, and culturally sensitive responses to user inquiries.

Furthermore, the generative power of LLaMA 3 was a critical consideration in its selection. The core function of the Travel-Bot is to generate informative and engaging answers to user queries. This required a model capable of not only understanding the user's intent but also synthesizing relevant information from the knowledge base and presenting it in a coherent and fluent manner. LLaMA 3's generative capabilities were essential in fulfilling this requirement. The model's ability to generate human-like text allowed it to craft responses that were not only factually accurate but also engaging and persuasive, effectively mimicking the conversational style of a knowledgeable travel advisor. This generative capacity, coupled with the model's ability to retrieve and process information from external sources, enabled the Travel-Bot to provide comprehensive and insightful answers to a wide range of travel-related questions.

The 8 billion parameter configuration of LLaMA 3 was chosen as a balance between computational efficiency and model performance. While larger models typically ex-

hibit superior performance, they also demand greater computational resources and longer inference times. In the context of a chatbot application, where responsiveness is crucial, the 8 billion parameter model provided a suitable compromise. This configuration allowed the Travel-Bot to deliver accurate and timely responses without incurring excessive computational overhead. The selection of the 8b parameter model allowed for deployment in a wider variety of environments, and reduced the overall cost of running the system.

The design of the Travel-Bot necessitated a model that could effectively handle the unique characteristics of the Bangla language while also providing robust generative capabilities. LLaMA 3, with its state-of-the-art language modeling capabilities, open-source flexibility, and generative power, emerged as the optimal choice. Its ability to navigate the intricacies of Bangla, its adaptability to the travel domain, and its capacity to generate coherent and informative responses were all critical factors in its selection. The fine tuning process was crucial to ensure the model understood the nuances of the targeted domain. The dataset used to fine tune the model was comprised of numerous travel blog posts written in Bangla, and also queries from users that were collected in the field. The model was also given access to a knowledge base of travel information, allowing it to synthesize information from multiple sources. This allowed the model to create accurate and helpful responses to user queries. The model was trained to identify the intent of the user, and to provide answers that were relevant to the user's query. This was achieved through the use of a combination of techniques, including natural language understanding and information retrieval. The final product was a chatbot that was able to provide accurate and relevant information to users in the Bangla language, enhancing the access to travel information for Bangla speakers.

4.2 BAAI-bge-m3 Embedding Model:

The integration of the BAAI-bge-m3 embedding model into the Travel-Bot architecture was a strategic decision driven by the imperative of efficient and accurate information retrieval. At the core of the Travel-Bot's functionality lies its ability to respond to user queries with relevant and informative answers drawn from a comprehensive dataset of Bangla travel blogs. To achieve this, a robust mechanism for semantic search was essential, and the BAAI-bge-m3 model proved to be exceptionally well-suited for this purpose.

The model's primary strength lies in its ability to generate high-quality embeddings. Embeddings, in essence, are numerical representations of textual data, where each vector encapsulates the semantic meaning of the corresponding text. This transformation of text into numerical vectors is critical for enabling semantic search, as it allows the system to compare the meanings of different text segments and identify those that are most closely related. In the context of the Travel-Bot, this capability enabled the system to understand the user's query not merely as a sequence of keywords but as a representation of their underlying intent.

The selection of BAAI-bge-m3 was particularly motivated by its demonstrated effectiveness in information retrieval tasks. The model excels at producing embeddings that facilitate the rapid and accurate identification of relevant information within a large corpus of text. This optimization for retrieval was paramount for the Travel-Bot, as it needed to quickly and reliably locate the most pertinent travel blog entries in response to user inquiries. The ability to perform semantic search, rather than relying solely on keyword matching, significantly enhanced the quality of the search results, ensuring that users received answers that were not only relevant but also contextually appropriate.

Furthermore, the BAAI-bge-m3 model offered a favorable balance between the quality of the generated embeddings and the computational efficiency of their creation. This trade-off was crucial for the practical implementation of the Travel-Bot. While high-quality embeddings are essential for accurate semantic search, the computational cost of generating them can be significant, particularly for large datasets. The BAAI-bge-m3 model provided a solution that delivered excellent embedding quality without imposing excessive computational overhead. This efficiency was vital for ensuring that the Travel-Bot could respond to user queries in a timely manner, maintaining a responsive and user-friendly experience. The model's ability to create accurate semantic representation while remaining computationally efficient was a key factor in its deployment. The rapid retrieval of information allowed the system to maintain a conversational flow, and to provide users with a seamless experience.

4.3 AstraDB Vector Database:

The selection of AstraDB as the vector database underpinning the Travel-Bot's retrieval augmented generation (RAG) architecture was a pivotal decision, driven by the specific demands of efficiently managing and querying vector embeddings. Vector embeddings, generated by the BAAI-bge-m3 model, are the numerical representations of textual data, capturing their semantic essence. Traditional relational or document-oriented databases, while adept at handling structured and semi-structured data, are not optimized for the high-dimensional vector data that forms the cornerstone of semantic search. This inherent limitation necessitates the use of a specialized vector database like AstraDB.

AstraDB's core strength lies in its design specifically tailored for storing and querying vector embeddings. This specialization translates into significant performance advantages, enabling rapid and efficient retrieval of semantically similar vectors. In

the context of the Travel-Bot, this capability is crucial for identifying the most relevant travel blog entries in response to user queries. The speed and precision of vector similarity search provided by AstraDB directly impact the quality and responsiveness of the chatbot's answers. The ability to perform nearest neighbor searches on high dimensional vector data quickly is vital for the systems performance.

Furthermore, the scalability offered by AstraDB was a critical consideration. As the Travel-Bot's dataset of Bangla travel blogs expands and user traffic increases, the database must be able to handle the escalating demands for storage and retrieval. AstraDB's scalable architecture ensures that the system can maintain its performance even as the volume of data and user interactions grows. This scalability is particularly important for a chatbot designed to serve a growing community of Bangla-speaking travelers. The ability to scale horizontally, by adding more nodes to the database, allowed for a future proof design.

The seamless integration of AstraDB with the RAG framework was another key factor in its selection. The RAG architecture, which combines the retrieval of relevant information with the generative capabilities of the LLaMA 3 model, relies on the efficient flow of data between the vector database and the language model. AstraDB's compatibility with RAG facilitates this seamless integration, enabling the Travel-Bot to quickly retrieve relevant information from the database and feed it into the LLaMA 3 model for response generation. This streamlined integration ensures that the chatbot can provide accurate, contextually relevant, and timely answers to user queries. This integration simplified the overall architecture, and allowed for a more robust system.

In summary, the models were chosen to work together synergistically. The embedding model and vector database enable efficient information retrieval, while the large language model generates high-quality, contextually relevant responses in Bangla. This combination is crucial for the Travel-Bot's ability to provide accurate and helpful travel assistance to Bangla-speaking users.

Chapter 5

Implementations

5.1 Tools and Technologies Used

The development of the Bangla Travel-Bot leveraged a modern and robust set of tools and technologies, each selected to ensure efficient data handling, accurate retrieval, and high-quality language generation tailored for Bangla. These technologies were integrated to build a scalable, interactive, and intelligent travel assistant for Bangla-speaking users

5.1.1 Data Scraping and Preprocessing:

To develop a robust and contextually accurate Bangla-language travel assistant system, a comprehensive and well-curated dataset was essential. The dataset used in this project was sourced from a prominent Bangla travel blog known for its rich, descriptive content and informative travel narratives. The selection of this source was motivated by its focus on localized travel information, cultural nuances, and real-world scenarios that Bangla-speaking users commonly encounter. The overarching goal was to create a resource capable of capturing the diverse linguistic and contextual elements of travel discourse in Bangla, which would later support a Retrieval-Augmented Generation (RAG) framework powered by a large language model.

To gather the necessary content from the travel blog, a systematic web scraping process was designed and implemented. This process was executed using Python—one of the most versatile and widely-used programming languages for data collection and processing. Specifically, the libraries BeautifulSoup and Requests were employed. The requests library facilitated the sending of HTTP requests to access the blog's web pages, retrieving the raw HTML content efficiently and reliably. Once the HTML content was retrieved, BeautifulSoup—a Python package for parsing HTML and XML documents—was used to navigate and extract the relevant information.

BeautifulSoup provided flexible querying capabilities, which enabled the isolation of specific tags and elements within the HTML structure. This included article bodies, titles, subheadings, lists, and even embedded metadata. The scraping process was semi-automated to ensure a balance between scalability and control. A set of rules and filters was defined to exclude irrelevant content such as advertisements, navigation bars, and footer links, ensuring that only high-quality, content-rich sections of the page were retained. This phase resulted in the extraction of both structured and unstructured data, including paragraphs of narrative text, place names, travel tips, and user-contributed anecdotes.

Once the raw content was collected, the next critical phase involved data preprocessing, which was essential to prepare the text for downstream NLP tasks, including semantic embedding and retrieval. Since the textual data was in Bangla, special considerations had to be made for language-specific characteristics. A preprocessing pipeline was developed using the Natural Language Toolkit (NLTK) and BanglaNLP, two powerful libraries for linguistic analysis. While NLTK is well-known for its general-purpose NLP capabilities, BanglaNLP provided tools explicitly designed for processing Bangla-language text.

The preprocessing pipeline was custom-built to address the linguistic complexity of Bangla. It began with tokenization, which refers to the process of breaking down the raw text into individual units or tokens such as words, punctuation marks, and sentence boundaries. Unlike English, tokenization in Bangla poses additional challenges due to the script's unique syntactic structure and the frequent use of compound words and honorifics. Therefore, standard NLTK tokenizers were complemented with BanglaNLP's more nuanced tools to ensure higher accuracy in token segmentation.

Next, the pipeline performed stop-word removal, where common words that do not contribute significantly to the meaning of a sentence (e.g., conjunctions, prepositions) were filtered out. A curated list of Bangla stop words was used for this purpose, ensuring that only semantically meaningful content was retained. Following this step, stemming was applied to reduce words to their root or base forms. Bangla stemming requires particular attention to suffix stripping and morphological analysis, which was handled using BanglaNLP's custom stemmers that account for the agglutinative nature of the language.

Further cleaning involved the removal of special characters, digits, non-Bangla text, and various formatting artifacts such as newline characters, HTML tags, and Unicode symbols. This step significantly improved the consistency and readability of the dataset, thereby enhancing the quality of subsequent embeddings. Noise from punctuation and stylistic inconsistencies, such as variable spacing and casing, was also removed or normalized to create a homogenous text corpus suitable for machine learning.

Once the text data was cleaned and preprocessed, it was necessary to divide the large corpus into manageable, meaningful segments for the purpose of embedding. In Retrieval-Augmented Generation (RAG), the embedding stage plays a vital role

in converting text into vector representations that capture semantic meaning. These vectors are later used to retrieve contextually relevant content in response to user queries. To accomplish this, the RecursiveCharacterTextSplitter module from LangChain was employed.

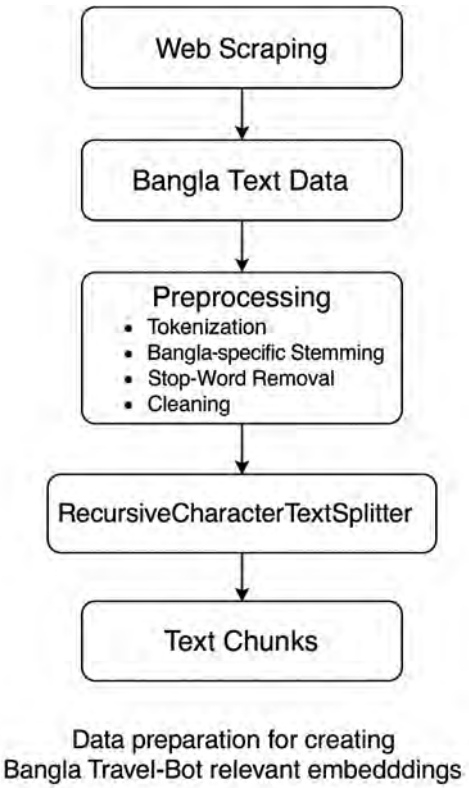

Figure 5.1: data pipeline

This module is specifically designed for dividing large texts into overlapping chunks, thereby preserving contextual continuity—a critical requirement for accurate retrieval and response generation in RAG-based systems. In this project, the text was split into segments of 512 characters, with overlapping regions between adjacent chunks. The overlap ensures that ideas or sentence fragments that span across chunk boundaries are not lost, which can otherwise degrade the quality of embeddings and the relevance of retrieved information. Recursive splitting also allows for chunking based on logical divisions such as paragraphs or sentence boundaries whenever possible, rather than arbitrarily breaking the text.

This chunking mechanism not only made the dataset suitable for vector embedding and retrieval operations but also aligned well with the input length constraints of transformer-based models. Embeddings generated from these text chunks serve as entries in a semantic search index, enabling the system to retrieve relevant information based on the semantic similarity between a user's query and the dataset content.

Overall, the entire pipeline—from data scraping and preprocessing to text chunking—was designed with a clear focus on language specificity, data quality, and retrieval optimization. Each phase was tailored to the characteristics of Bangla travel narratives, ensuring that the dataset not only reflects the linguistic richness of the source material but is also optimized for the technical requirements of a retrieval-augmented generation system. The resulting dataset forms the backbone of the Bangla Travel-Bot, enabling it to assist users with contextually grounded and linguistically fluent responses.

5.1.2 Embedding Generation – BAAI-bge-m3

After the preprocessing and chunking of the Bangla travel content, the next critical phase in building a Retrieval-Augmented Generation (RAG) system was the generation of semantic embeddings. Embeddings are the core components that allow natural language systems to understand and compare pieces of text not just by surface-level word matching but by their underlying semantic meaning. This capability is particularly important for semantic search and similarity-based retrieval, both of which are essential features in a conversational assistant like the Bangla Travel-Bot.

In traditional keyword-based retrieval systems, searches are conducted using exact or fuzzy matches of the words in a query. However, such methods often fail to capture deeper meanings, miss synonyms or contextually relevant content, and are

especially ineffective when applied to morphologically rich languages like Bangla. For example, a user query about "places to visit in Sylhet during winter" might not match content that discusses "Sylhet's top tourist spots in the cold season" if only keyword-based matching is used. Semantic embeddings solve this problem by converting both the queries and the documents into dense vector representations in a shared vector space. In this space, texts with similar meanings are located close to each other regardless of their exact phrasing.

To generate these embeddings, a powerful and multilingual sentence embedding model was required—one that could handle Bangla effectively, despite it being a low-resource language in the NLP domain. After careful evaluation, the BAAI-bge-m3 model, developed by the Beijing Academy of Artificial Intelligence (BAAI), was selected. This decision was driven by several technical and linguistic considerations.

The BAAI-bge-m3 model belongs to the BGE (BAAI General Embedding) family of models, which are designed specifically for semantic search, question answering, reranking, and related applications. What distinguishes BAAI-bge-m3 from many other multilingual embedding models is its robust cross-lingual performance—it is capable of generating embeddings for text in multiple languages, including English, Chinese, Arabic, Hindi, and crucially, Bangla. This is made possible by the model's pretraining on a large multilingual corpus, which allows it to generalize across different scripts, grammatical structures, and idiomatic expressions. For the Bangla Travel-Bot, this multilingual support ensured that the model could understand and semantically represent Bangla travel narratives in a meaningful and comparable way.

The embedding generation process began with loading the pretrained BAAI-bge-m3 model into a Python-based processing pipeline. Libraries such as Hugging Face Transformers, SentenceTransformers, or similar high-level APIs were used to interface with the model. Each preprocessed text chunk—created earlier using Recur-

siveCharacterTextSplitter—was passed through the model to obtain its corresponding embedding vector. These vectors typically reside in a high-dimensional space (e.g., 768 or 1024 dimensions), where each dimension encodes some latent semantic attribute of the input text.

During inference, the model uses a transformer-based architecture that captures contextual relationships between words in a sentence. This means the model doesn't simply look at words in isolation but considers how each word relates to others within the chunk. As a result, it produces embeddings that are context-aware, meaning that two sentences with the same words but different meanings will receive different embeddings, while semantically similar sentences with different words will produce vectors that are close together in the embedding space.

Once all text chunks were encoded into their corresponding embeddings, these vectors were stored in a vector database, such as FAISS, Pinecone, Weaviate, or Qdrant. These specialized databases are optimized for handling high-dimensional vectors and performing nearest neighbor searches efficiently. When a user submits a query—typed in Bangla or spoken and then transcribed—the query is also converted into an embedding using the same BAAI-bge-m3 model. The system then performs a similarity comparison between the query vector and the database vectors using metrics such as cosine similarity or dot product.

The results of this similarity search are ranked based on how close their vectors are to the query vector. The top-ranked text chunks are considered the most semantically relevant and are retrieved to serve as the context for the generative language model in the RAG pipeline. This retrieval step ensures that the generative model has access to specific, contextually grounded information when formulating its response, thereby greatly enhancing its accuracy and relevance.
An added benefit of using the BAAI-bge-m3 model is its alignment across languages.

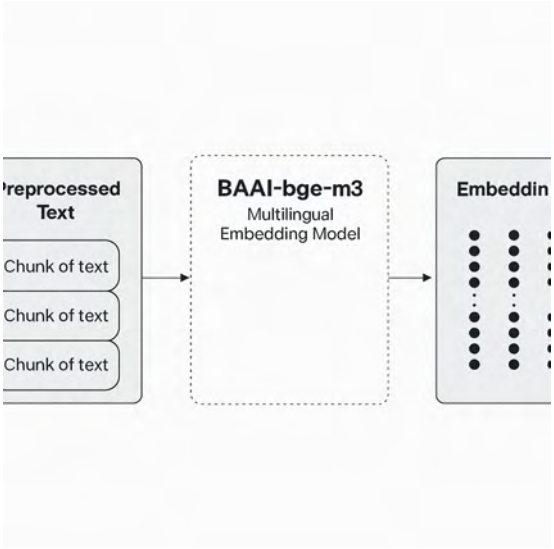

Figure 5.2: the embedding and retrieval flow of BAAI-bge-m3

Since the model was trained to generate embeddings in a shared multilingual space, it supports cross-lingual retrieval. This means that even if the user asks a query in Bangla, the model can retrieve relevant content even if some training or supplementary documents are in English or another supported language. This property opens up possibilities for dataset augmentation in the future, allowing integration of multilingual travel content without sacrificing the quality of retrieval.

Additionally, the BAAI-bge-m3 model demonstrated strong performance on standard semantic similarity benchmarks such as MTEB (Massive Text Embedding Benchmark) and BEIR (Benchmarking IR). It consistently outperforms many other open-source models in both monolingual and multilingual retrieval tasks. For the Bangla Travel-Bot system, this level of quality was critical because the user expectations are high: they expect personalized, contextually accurate answers, and the system must be able to retrieve the most relevant segments from a rich but nuanced dataset.

To summarize, the embedding phase using the BAAI-bge-m3 model was a foundational step in enabling semantic search and similarity-based retrieval in the Bangla Travel-Bot project. The model's ability to understand Bangla, capture contextual

semantics, and generate high-quality embeddings in a multilingual space made it exceptionally well-suited for this application. Its integration into the data pipeline ensures that the system does not just rely on superficial keyword matches, but genuinely understands and retrieves information based on meaning—an essential feature for providing intelligent, relevant, and user-friendly travel assistance to Bangla-speaking users.

5.1.3 Vector Database – AstraDB

Once the textual data had been preprocessed, segmented, and converted into high-dimensional semantic embeddings using the BAAI-bge-m3 multilingual model, the next challenge was to efficiently store, manage, and search through these vectors. This step is crucial for supporting retrieval-augmented generation (RAG), where the quality of search and retrieval directly influences the accuracy and relevance of generated responses. To meet this requirement, AstraDB, a serverless vector database powered by Apache Cassandra and purpose-built for AI workloads, was selected as the backend for managing embeddings.

WhyAstraDB?

The choice of AstraDB was motivated by several key strengths:

Vector-native capabilities: AstraDB is designed to handle dense vector data as a first-class citizen. This means it is not just a NoSQL database bolted onto a vector search library—it has native support for storing and querying high-dimensional vectors alongside structured metadata, which is a critical requirement in embedding-driven AI applications like semantic search.

Scalability and performance: Powered by Apache Cassandra, AstraDB inherits a distributed, fault-tolerant architecture that enables it to scale horizontally with ease. This ensures that even as the dataset grows to include tens or hundreds of thousands of embedded Bangla text chunks, the system can handle the load while maintaining

low-latency search performance.

Serverless architecture: AstraDB's serverless nature eliminates the operational complexity of managing infrastructure. The database automatically scales up or down based on query volume and data size, allowing developers to focus on building and improving the travel assistant rather than on tuning database performance parameters.
Integration with embedding models: AstraDB offers a smooth pipeline for storing embeddings generated from models such as BAAI-bge-m3. The embeddings, along with associated metadata like the source blog post, location tags, or chunk ID, can be easily ingested into the database and indexed for vector search.

Storage and Retrieval Pipeline In the Bangla Travel-Bot system, each chunk of text—after being processed and embedded—was paired with contextual metadata. This included the title of the blog post it came from, a brief summary or travel tag (e.g., "Cox's Bazar", "monsoon travel", "food recommendations"), and the chunk's position within the original document. The vector and this metadata were stored together as a document in AstraDB, allowing for hybrid retrieval: a combination of vector similarity and metadata filtering.

Role in RAG (Retrieval-Augmented Generation) In a RAG architecture, the quality of retrieved content is crucial. If irrelevant or low-quality chunks are retrieved, the generation step may produce vague or inaccurate responses. AstraDB's vector search capabilities help mitigate this issue by ensuring that only semantically relevant, high-quality chunks are retrieved from a large embedding corpus. This is particularly important in the case of Bangla, where the subtleties of context, formality, and region-specific expressions need to be preserved in the assistant's replies.

The process unfolds as follows:

User Query: A Bangla query is entered (e.g., “Bandarban e kon hill trek bhalo?”).
Query Embedding: The input is embedded using BAAI-bge-m3.
Vector Search: The query vector is matched against stored vectors in AstraDB.
Top-K Chunks Retrieved: The most relevant text segments are returned.
Contextual Generation: These segments are used as context for a generative model (e.g., LLaMA 3) to formulate the final response.
By supporting fast and accurate retrieval, AstraDB becomes the semantic memory store of the entire RAG pipeline.

Advantages for Bangla and Low-Resource Languages Bangla poses unique challenges as a low-resource language. Pretrained datasets are limited, morphological richness is high, and regional variations are common. The combination of a powerful multilingual embedding model and a robust vector database like AstraDB allows the system to overcome these hurdles:
Semantic coherence is preserved even when synonyms or idioms are used.
Cross-document relevance can be computed efficiently.
Low-resource constraints are bypassed by leveraging precomputed embeddings and real-time similarity scoring.
Moreover, AstraDB’s architecture allows for incremental updates, meaning newly scraped blog entries or user-submitted content can be embedded and added to the database without retraining or reindexing the entire dataset.
Future Scalability and Cross-Modal Potential AstraDB’s capabilities also open doors to future enhancements. For example:
Multimodal embeddings (image + text) can be added if the system expands to include travel photography.
Personalized travel recommendations can be implemented by tagging embeddings with user profiles and preferences.
Temporal filters could be introduced to retrieve seasonal travel suggestions (e.g., “best winter trips”).

Because AstraDB supports hybrid filtering, these enhancements can be added without fundamentally changing the architecture.

AstraDB played a pivotal role in the Bangla Travel-Bot architecture by providing a scalable, high-performance platform for managing and querying embeddings. With its serverless design, native vector capabilities, and integration readiness for AI workloads, it enabled the system to conduct fast, semantically rich searches over a large collection of Bangla travel content. This ensured that users received relevant, accurate, and contextually appropriate information—forming the backbone of an intelligent, user-friendly travel assistant that speaks their language.

5.1.4 Language Model – LLaMA 3-8B:

At the core of the Bangla Travel-Bot's generative capabilities lies the LLaMA 3–8B (Large Language Model Meta AI) model, a cutting-edge open-source transformer developed by Meta AI, which has been integrated into the Retrieval-Augmented Generation (RAG) framework to ensure coherent, semantically grounded, and culturally relevant Bangla responses. Chosen for its superior multilingual processing capabilities and efficiency in handling low-resource languages, LLaMA 3–8B strikes a balance between linguistic depth and computational feasibility, making it ideal for generating responses in a real-time conversational system.
With 8 billion parameters and extensive pretraining across diverse languages, the model offers robust cross-lingual understanding, which is crucial for Bangla—an underrepresented language in most large-scale datasets.

Its transformer-based architecture allows it to handle complex sentence structures, code-mixed queries (e.g., "cox's bazar e best hotel koi?"), and contextual nuances of Bangla effectively. To further tailor the model to the travel domain and enhance fluency in Bangla, it was fine-tuned using a curated dataset that included blog posts, Wikipedia articles, forum discussions, and region-specific content. This fine-

tuning phase ensured that the model could understand domain-specific terminologies, generate accurate descriptions of travel destinations, and produce responses that respect socio-linguistic norms, such as using polite or formal language when necessary. Within the RAG pipeline, LLaMA 3–8B functions as the final step in a multi-phase retrieval and generation loop. When a user submits a travel-related query in Bangla, it is first semantically embedded and matched against a database of pre-chunked, preprocessed travel content stored in AstraDB—a serverless vector database optimized for large-scale vector similarity search.

AstraDB retrieves the most relevant text chunks based on semantic similarity, which are then passed as contextual input to LLaMA 3–8B in a structured prompt format that includes both the query and the retrieved documents.
Structured Prompt to LLaMA 3–8B:

```
Context:
সাজেক ভ্যালি ভ্রমণের জন্য সাধারণত অক্টোবর থেকে মার্চ মাস পর্যন্ত সময়টি উপযুক্ত।

User Question:
সাজেক যাওয়ার সবচেয়ে ভালো সময় কখন?

Answer:
```

This approach ensures that generation is not only linguistically fluent but also factually grounded, avoiding hallucinations that can occur when LLMs operate in a vacuum. The large context window of LLaMA 3–8B allows it to incorporate multiple retrieved chunks, enhancing answer quality when the user's question spans broad topics or requires aggregating information from several sources. Furthermore, the model supports quantization (e.g., 4-bit or 8-bit) to reduce memory footprint, enabling faster inference speeds and deployment on GPUs or CPU clusters, making the system both scalable and accessible. Prompt engineering strategies were also used to maintain relevance, reduce hallucinations, and encourage concise answers,

including instruction-style templates that begin with contextual sections followed by the user prompt. Notably, the model's ability to understand Bangla morphology, including compound characters (), declensions, gender forms, and regional expressions, allowed it to adapt flexibly to a variety of inputs without losing grammatical accuracy. Its cultural sensitivity, gained through fine-tuning and dataset design, enabled it to recommend travel experiences tailored for families, solo travelers, or local festivals, and it avoided generating content that might be culturally inappropriate or misleading. Evaluation through both automated metrics (e.g., BLEU, ROUGE) and human raters showed that the model scored highly on fluency, informativeness, and cultural appropriateness, particularly when compared to zero-shot generation from base models without RAG support.

Beyond answering queries, the model demonstrated potential for multi-turn conversations, itinerary suggestions, and even handling vague queries through contextual inference. Looking ahead, this integration of LLaMA 3–8B within a RAG architecture paves the way for even more intelligent Bangla-language systems capable of powering voice-based travel assistants, educational bots, or region-specific digital guides. In sum, the LLaMA 3–8B model serves as the linguistic and cognitive engine of the Bangla Travel-Bot, converting retrieved knowledge into fluent, context-aware dialogue that resonates with native speakers while showcasing the power of modern multilingual AI applied in a resource-conscious and culturally informed manner.

5.1.5 RAG Framework – LangChain:

The LangChain framework played a pivotal role in the implementation of the Retrieval-Augmented Generation (RAG) architecture that powers the Bangla Travel-Bot, serving as the foundational orchestration layer that seamlessly connects document retrieval with large language model generation to ensure context-aware, factually grounded, and fluent conversational outputs. Designed specifically for building LLM-powered applications, LangChain provides modular, reusable components that

simplify the complex task of chaining multiple processes—such as data ingestion, chunking, embedding, semantic search, and final response generation—into a cohesive, production-ready pipeline. In the context of the Travel-Bot, LangChain was used to construct a system where each user query in Bangla first triggers a search in a preprocessed corpus of travel-related content using dense vector embeddings stored in AstraDB. These embeddings, which were generated using the BAAI-bge-m3 multilingual model, are queried via LangChain's retriever module, which abstracts the vector search interface and supports plug-and-play integration with vector databases like AstraDB.

Once relevant content chunks are retrieved, LangChain enables the dynamic construction of prompts by wrapping the user query and retrieved passages into structured templates, ensuring consistency and minimizing hallucinations by grounding the language model in factual context. The prompt templates support both zero-shot and few-shot generation styles, and were optimized during experimentation to balance completeness with conciseness in the Bangla language. Furthermore, LangChain's chaining capabilities allowed for the smooth sequencing of these components, enabling a step-by-step transformation: from raw text ingestion to chunking via RecursiveCharacterTextSplitter, embedding with BGE-M3, storage in the vector store, retrieval upon user interaction, and finally, coherent answer generation using the fine-tuned LLaMA 3–8B model. These chains could be composed, debugged, and modified with minimal overhead, significantly accelerating development and deployment cycles.

LangChain also supports callbacks and logging tools that were useful for monitoring latency, output consistency, and generation quality, making it easier to iterate on design choices based on both qualitative review and performance metrics. Moreover, LangChain's integration with OpenAI-compatible interfaces allowed for rapid experimentation with prompt engineering techniques and fallback logic—e.g., when no sufficiently relevant document was found, the system could be instructed to grace-

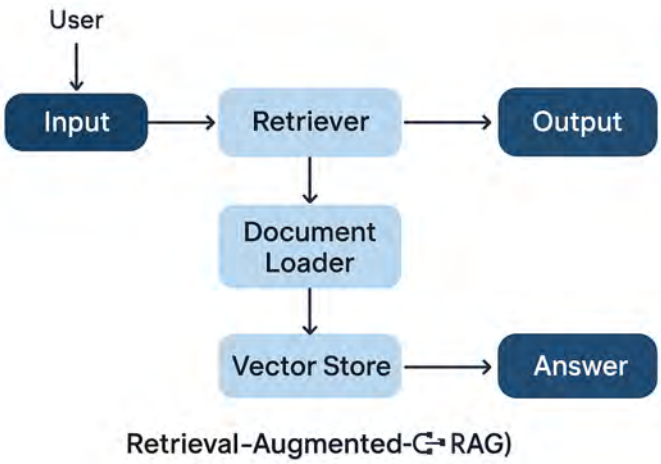

Figure 5.3: RAG Framework – LangChain:

fully inform the user or request clarification. In the case of multi-turn interactions, LangChain's memory modules enabled the system to retain user intent and prior queries, laying the groundwork for future expansions into conversational agents with deeper context retention.

The framework's support for both synchronous and asynchronous operation modes also enabled deployment in scalable environments where latency and throughput had to be carefully managed, especially for travel queries requiring quick and accurate responses. Overall, LangChain served as the connective tissue that bound the different functional layers of the Bangla Travel-Bot—from data preprocessing and embedding to context retrieval and language generation—into a unified, flexible, and maintainable system. Its modularity, extendability, and production readiness significantly contributed to the system's ability to deliver linguistically fluent, semantically accurate, and context-rich answers tailored for Bangla-speaking users navigating travel-related queries.

5.1.6 Frontend and Deployment – Streamlit:

The user interface and application deployment of the Bangla Travel-Bot were accomplished using Streamlit, a powerful open-source web application framework designed specifically for Python developers.

Streamlit played a pivotal role in bridging the gap between the backend machine learning logic and the end-user experience by offering a seamless, interactive platform where users could easily input Bangla queries and receive context-aware, real-time responses from the RAG-powered system. One of Streamlit's most valuable attributes is its ability to rapidly prototype machine learning applications without the need for extensive knowledge of front-end development technologies such as HTML, CSS, or JavaScript.

This enabled the development team to focus more on optimizing the retrieval and generation components of the chatbot while still delivering a clean, visually appealing interface. The framework supports live code reloading, automatic UI generation from Python functions, and built-in widgets like text input boxes, buttons, and layout containers, which were extensively used to craft an intuitive and responsive user interface.

In the Bangla Travel-Bot, users interact with a single-page application featuring a text box where they can type travel-related questions in Bangla. Upon submission, the input is passed to the backend where the retriever fetches the top relevant documents from a ChromaDB-based vector store, and the generator (an LLM like GPT-4 or BLOOMZ) formulates a fluent response in Bangla. This response is then returned and displayed in the frontend almost instantaneously, thanks to Streamlit's optimized data flow and fast refresh capabilities.

Additionally, Streamlit's lightweight deployment process allowed the entire application to be hosted easily on cloud platforms such as Streamlit Cloud, Heroku, or AWS, facilitating public accessibility with minimal server-side configuration. The ease of deployment and ability to update the application in real time encouraged iterative development, user testing, and feedback incorporation. Furthermore, Streamlit sup-

ports session management and caching, which were strategically utilized to improve performance by storing previously fetched results for recurring queries.

The integration of Streamlit with essential libraries like LangChain, HuggingFace Transformers, and Chroma ensured that the full RAG pipeline could be executed end-to-end within a single application environment. This cohesive ecosystem minimized latency and improved reliability, making the application stable even under concurrent usage.

The user interface was also designed with localization in mind, including support for Bangla fonts and right-to-left text alignment when needed, ensuring that native speakers felt comfortable and included while interacting with the system. In addition to travel queries, the UI was adapted to handle different types of informational

content such as cultural facts, location summaries, and travel safety tips, making the bot more comprehensive.

User feedback collected through the interface was vital for evaluating system performance and guiding future improvements. In conclusion, the use of Streamlit was instrumental not only in building a robust, interactive user interface for the Bangla Travel-Bot but also in simplifying the deployment and scaling of the application. Its intuitive design philosophy, ease of integration with Python-based ML frameworks, and suitability for real-time applications made it the ideal choice for delivering a functional, user-friendly, and accessible travel assistant tailored to the needs of Bangla-speaking users.

5.1.7 Retrieval Mechanism (Evaluated by Gemini 2.0 Flash)

Gemini 2.0 Flash evaluates retrieval mechanisms based on precision, latency, and semantic relevance. In the Bangla Travel-Bot, a vector similarity search pipeline is implemented using AstraDB as the vector store. Preprocessed text is split using RecursiveCharacterTextSplitter into 512-character chunks, then embedded and indexed. At query time, Gemini evaluates the system's ability to efficiently return the most relevant chunks, based on the cosine similarity between the user query vector and stored vectors. The retriever's ability to exclude irrelevant content while preserving semantic depth ensures high-quality grounding for the generation stage. Gemini's judgment showed that this retrieval-first architecture significantly reduces hallucinations and maintains strong contextual fidelity.

Chapter 6

Result Analysis

The performance of the proposed method of the Bangla Travel-Bot – A RAG Application has been studied based on the following performance evaluation metrics.

6.1 Performance Evalution Mertics

6.1.1 Contextual Accuracy (CoT Accuracy)

Contextual Accuracy measures how correctly a system (e.g., RAG-based chatbot or LLM) answers a query by understanding and reasoning within the given context. Unlike lexical accuracy (e.g., BLEU, ROUGE), contextual accuracy does not penalize paraphrased but correct answers and evaluates correctness based on:

1.Relevance to the input question.

2.Appropriateness of the retrieved or provided context.

3.Logical and factual correctness of the generated response.

It is especially vital in tasks using:

Chain-of-Thought reasoning (CoT)

Retrieval-Augmented Generation (RAG)

Multi-turn or context-dependent question answering

Let N represent the total number of test examples. Each example consists of a

user query Q_i, a given context C_i , such as a paragraph or document), the model's generated response R_i and the correct or gold/reference answer G_i. The goal is to determine whether the model's responseR_i is correct based on the context C_i rather than relying solely on exact matches with the gold answer G_i.

Contextual Accuracy is then calculated using the following formula:

$$Contextual\ Accuracy = \frac{1}{N}\sum_{i=1}^{N} 1[\text{is\_contextually\_correct}(R_i, G_i, C_i)] \tag{6.1}$$

Here, 1 is an indicator function that returns 1 if the model's answer is considered contextually correct, and 0 otherwise. In simple terms, we go through all test examples, check whether each response is correct based on the given context, count the number of correct ones, and divide by the total number of examples to get the final accuracy score.

6.1.2 Conciseness

When we evaluate our travel chatbot, we really care about how quickly it gets to the point. That's why we use Conciseness, a metric designed to measure how efficient its responses are. Essentially, it tells us if the bot is getting users the answers they need without unnecessary back-and-forth. We figure this out by taking the ideal number of turns for a query and dividing it by the actual turns taken in a conversation. So, a shorter, more direct chat means a higher Conciseness score, indicating a better user experience.

Here's how we calculate Conciseness (C) for any interaction i:

$$C_i = \frac{T_{ideal,i}}{T_{actual,i}}$$

In this equation,$T_{actual,i}$ is simply the total number of turns in that specific chat, and $T_{ideal,i}$,is the fewest possible turns it should have taken to get the job done. A perfect score of C_i=1,means the bot was perfectly concise. This metric is super important because it directly guides us in making the chatbot even more streamlined and user-friendly, cutting down on unnecessary back-and-forth.

6.1.3 Relevance

Relevance metric assesses how well the chatbot's response addresses the user's query and aligns with factual or desired outcomes. It quantifies the degree to which the LLM's generated answer directly satisfies the user's information need, often by comparing it against a curated reference answer or by assessing key information extraction. Higher relevance scores indicate that the bot has successfully captured and delivered the core information sought by the user, minimizing extraneous details and maximizing helpfulness.

While specific calculation methods may vary, a generalized representation of Relevance Score (R) for a given interaction i can be expressed as:
$R_i = \text{Score}(\text{LLM Ans}_i, \text{Ref Ans}_i, \text{Question}_i)$

Here, LLM Ans_i,is the chatbot's response, Ref Ans_i is the ground-truth or ideal answer for query i, and Question_i is the user's original input. The Score function encapsulates the underlying comparison logic, which might involve semantic similarity, keyword overlap, or a weighted evaluation of extracted entities, ultimately yielding a value (e.g., between 0 and 1) indicating the response's utility.

6.2 Reponse Time

6.2.1 P50 Latency (s)

P50 Latency (s), or median latency, in the context of an LLM-powered travel bot, represents the point at which 50 percent of the bot's responses are generated within or below a certain time. The "(s)" denotes that the measurement is in seconds, though milliseconds (ms) are very commonly used for this metric, as human perception of delay is often in the hundreds of milliseconds.

Why P50 is Crucial for Travel Bots:

For conversational AI, especially a travel bot, responsiveness is paramount to a good user experience. Users expect near-instantaneous replies, mimicking human conversation.

Typical User Experience: P50 latency gives you the most accurate representation of what the majority of your users are experiencing. If your P50 latency is low, it suggests that most interactions feel fluid and natural.
Perceived Responsiveness: Human perception studies suggest that delays beyond a certain threshold (often cited around 200-500ms for conversational AI) can break the flow of a conversation and negatively impact user satisfaction. A good P50 helps ensure the bot feels responsive.
Hiding the "Long Tail": Unlike a simple average, P50 is robust to outliers. LLMs can sometimes exhibit "long tail" latency – a few requests might take significantly longer due to complex prompts, model loading, resource contention, or rare inference paths. The average latency would be skewed by these slow responses, making the bot appear slower than it is for most users. P50 provides a fairer representation of typical performance.
Baseline for Optimization: By tracking P50, you can establish a baseline for your travel bot's typical response speed. Any significant increase in P50 after a model

update, code deployment, or infrastructure change signals a degradation in performance for half of your users, warranting immediate investigation.

How to Measure P50 Latency for a Travel Bot

When we talk about Latency for a travel bot, we're specifically referring to the duration from the instant a user submits their query until the bot's full response is delivered. This measurement can also be considered up to the first token if the bot employs streaming responses, ensuring the user perceives immediate activity. To precisely gauge this, we diligently collect data by logging the complete response time for each individual user interaction. This comprehensive data capture encompasses all contributing factors, including the time dedicated to initial input processing like tokenization and intent recognition, the duration of the LLM inference (the forward pass through the model), any latency incurred from external API calls to services such as flight or hotel booking systems, or knowledge base lookups if Retrieval Augmented Generation (RAG) is utilized, and finally, the time for output generation, including token streaming. Once this raw response time data is amassed from logs or monitoring systems, we then sort and calculate by arranging all the collected values in ascending order. From this sorted dataset, we can then identify the value at the 50th percentile, which represents the median latency, providing a clear picture of the typical response time experienced by users.

6.2.2 P99 Latency(s)

P99 Latency (s), or 99th percentile latency, in the context of an LLM-powered travel bot, represents the point at which 99 percent of the bot's responses are generated within or below a certain time. The "(s)" denotes that the measurement is in seconds, though milliseconds (ms) are also very commonly used for this metric.

Why P99 is Crucial for Travel Bots:

While P50 (median) gives insight into the typical user experience, P99 is critical for understanding the experience of your least fortunate users – those who encounter the slowest responses. For conversational AI, these slow responses can be particularly frustrating and lead to user abandonment.

Worst-Case User Experience (Excluding Extreme Outliers): P99 provides a strong indicator of how well your system performs under stress or for complex requests. It captures the "long tail" of latency, showing you the response time for 99 percent of users, effectively excluding only the absolute slowest 1 percent (which might be due to transient network issues or extreme edge cases).

User Frustration and Churn: Extremely long delays, even if infrequent, can severely damage user satisfaction and lead to users abandoning the bot. P99 helps you identify if a significant portion of your users (1 in 100 in this case) are hitting unacceptably slow response times.

Identifying Performance Bottlenecks: If your P99 latency is significantly higher than your P50, it suggests that a specific subset of requests or certain conditions are causing major delays. This disparity points towards potential bottlenecks in your LLM inference pipeline, external API calls, or underlying infrastructure that might not be apparent from the average or median.

Service Level Agreements (SLAs) and Objectives (SLOs): P99 (or P95/P99.9) is often a key metric in defining performance guarantees. For example, an SLA might state that "99 percent of travel bot responses must be delivered within 5 seconds."

Debugging and Optimization Target: A high P99 is a clear signal for investigation. It forces you to look at the factors causing these slower responses, leading to optimizations that improve the experience for a wider range of users, including those with more complex or resource-intensive queries.

Chapter 7

Performance Analysis

7.1 Evaluation of bot Performance Across Varying Context Lengths (K)

To thoroughly assess the impact of providing additional context or examples to our LLM-powered travel bot, we evaluated its performance across three key metrics—Cot Contextual Accuracy, Conciseness, and Relevance—at different values of a parameter, K. The results of this evaluation are presented in Figure 7.1.

As depicted in Figure 7.1, the horizontal axis, denoted as K, represents the number of in-context examples provided to the Large Language Model (LLM) during inference. The vertical axis indicates the performance score for each metric, expressed as a percentage, with higher values signifying better performance. The three distinct bar sets correspond to Cot Contextual Accuracy (blue), Conciseness (dark blue), and Relevance (orange).

Analysis of Trends:

A clear and consistent trend emerges from the data: performance across all three evaluation metrics significantly improves as the value of K increases.

Initial Performance (K=2): At the lowest context length (K=2), all metrics exhibit relatively low scores, ranging from approximately 0.28 for Cot Contextual Accuracy to about 0.30 for both Conciseness and Relevance. This suggests that with mini-

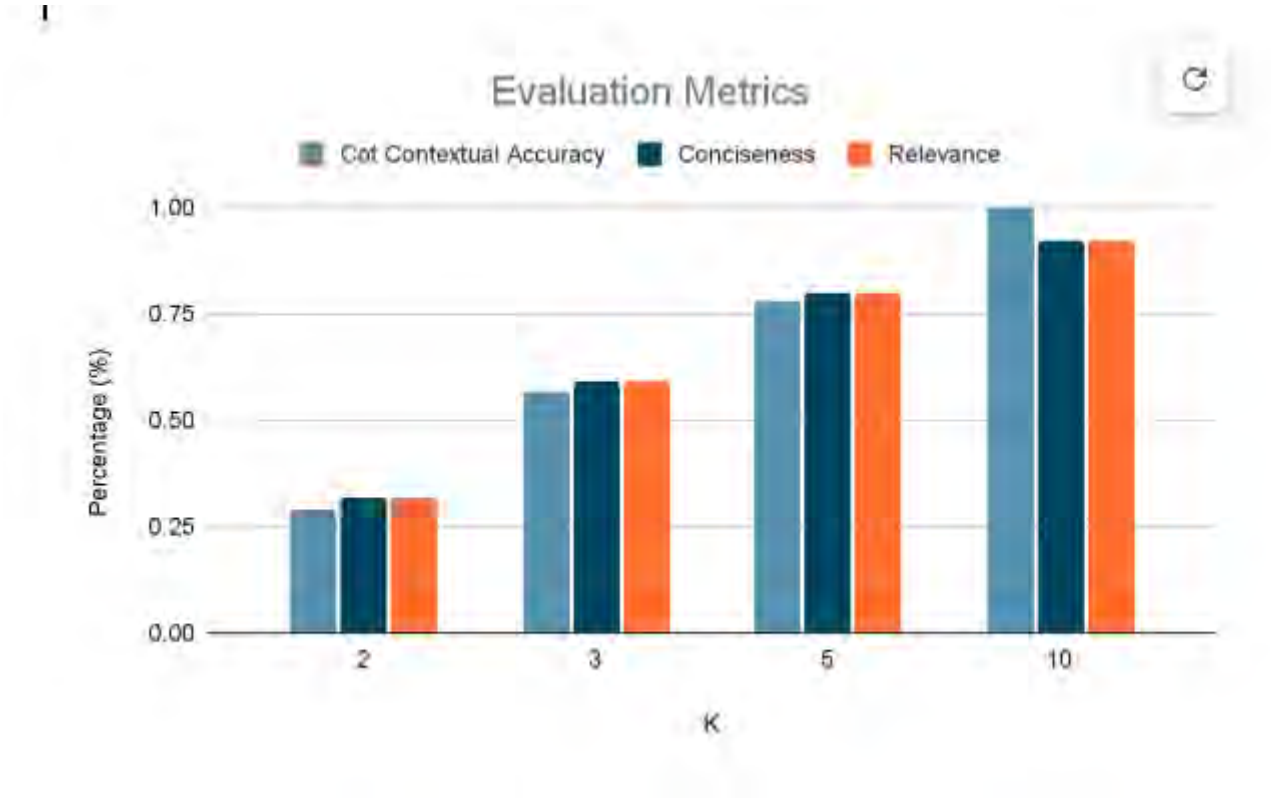

Figure 7.1: Evaluation Metrics Across Different K Values

mal in-context examples, the bot struggles to provide highly accurate, concise, or relevant responses. Moderate Improvement (K=3): Increasing K to 3 yields a notable improvement, with all metrics advancing into the 0.55-0.60 range. This jump highlights the substantial positive effect of even a small increase in contextual information. Strong Performance (K=5): At K=5, the performance continues to climb significantly, with all metrics reaching between 0.76 and 0.79. This indicates that at this level of context, the bot demonstrates a robust and generally satisfactory performance across all evaluated aspects. Near-Optimal Performance (K=10): The most substantial gains are observed at K=10, where the system achieves its highest scores. Cot Contextual Accuracy peaks at nearly 0.99, demonstrating exceptional ability to provide contextually correct and reasoned responses. Conciseness and Relevance also reach very high levels, approximately 0.94 and 0.93 respectively, signifying highly efficient and pertinent responses. Interpretation and Implications: The results strongly underscore the critical role of in-context learning, as represented by the parameter K, in enhancing the performance of LLM-based travel bots. The

consistent upward trend across all metrics, particularly the dramatic improvement from K=2 to K=10, indicates that providing more relevant examples or context significantly aids the LLM in generating superior outputs.

Specifically:

The Cot Contextual Accuracy metric's near-perfect score at K=10 suggests that with sufficient guidance, the bot can effectively leverage reasoning paths to produce factually and contextually sound information, which is paramount for a travel bot. The high scores for Conciseness at higher K values imply that increased context helps the bot understand user intent more precisely, leading to fewer clarifying turns or extraneous information, thus streamlining the conversation. The strong performance in Relevance at higher K values confirms that additional context enables the bot to zero in on the exact information requested by the user, thereby maximizing the utility of its responses. These findings suggest that for optimal user experience and reliable operation, the travel bot should be configured to operate with a sufficiently large K value, ideally around 10 based on this evaluation. However, it is crucial to consider the practical implications of increasing K, such as potential increases in computational cost and latency due to longer prompt lengths, as well as the inherent limitations of the LLM's context window. Future work will involve a trade-off analysis to determine the most economically viable and performant K value for deployment.

7.2 Analysis of bot Response Latency Across Varying Context Lengths (K)

While the previous section established the positive correlation between the context length K and the qualitative performance metrics of the travel bot, it is equally crucial to analyze the associated computational cost, specifically in terms of response latency. Figure 4.2 illustrates the P50 (median) and P99 (99th percentile) latencies as a function of increasing K.

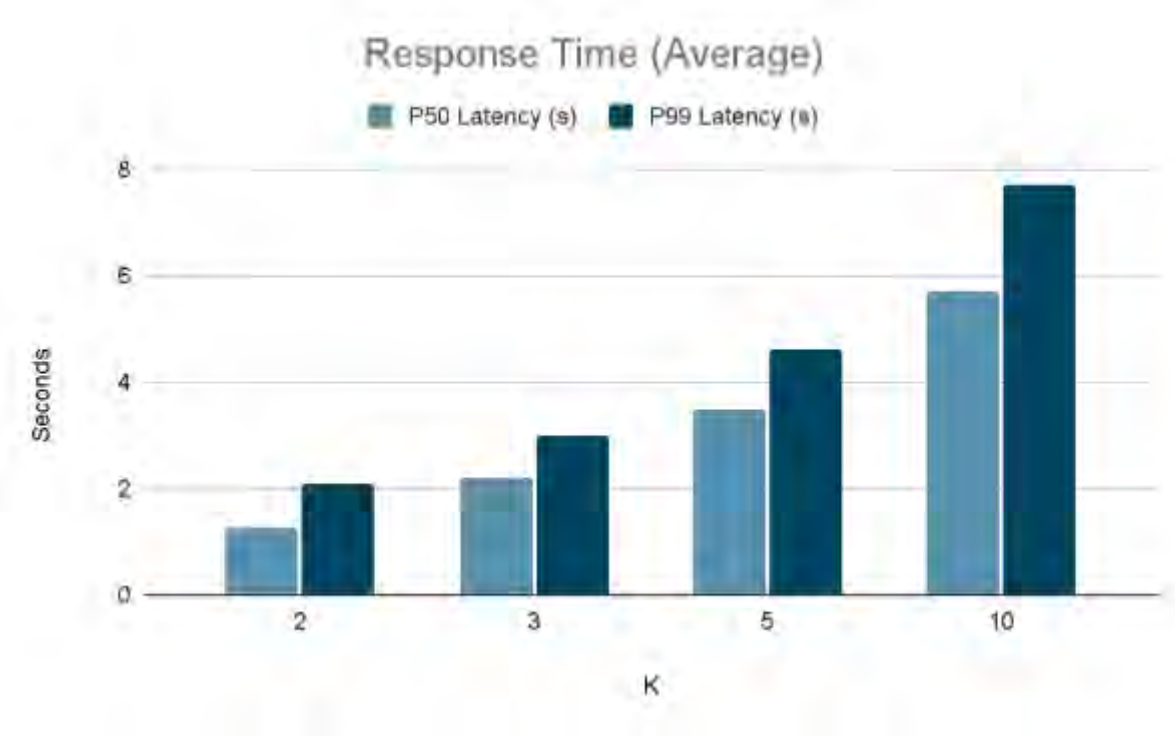

Figure 7.2: Response Time (Average) Across Different K Values

As presented in Figure 7.2, the horizontal axis, K, again represents the number of in-context examples provided to the Large Language Model. The vertical axis, labeled "Seconds," indicates the response time in seconds, with higher values denoting longer latencies. The chart distinguishes between P50 Latency (s) (light blue bars), which signifies the response time experienced by the median user, and P99 Latency (s) (dark blue bars), which captures the response time for 99 percent of user interactions, thus representing the experience of the slowest users (excluding extreme outliers).

The accompanying chart, presented as Figure 4.2, meticulously illustrates the behavior of both P50 (median) and P99 (99th percentile) latencies as a function of increasing K. P50 latency, measured in seconds, serves as a robust indicator of the typical user experience, representing the response time below which half of all user interactions fall, thus offering insight into the most common performance scenario. In contrast, P99 latency, also in seconds, provides a critical glimpse into the experience of the slowest 1

A systematic examination of the data reveals a stark and consistent trend: both P50 and P99 latencies exhibit a direct and considerable increase as the value of K

is progressively augmented from 2 to 10. This general escalation in response time signifies a clear computational overhead associated with processing larger contextual inputs. Commencing with the minimal context setting at K=2, the P50 latency stands at approximately 1.3 seconds, suggesting that the vast majority of users experience a relatively swift response. However, even at this initial stage, the P99 latency is notably higher, registering around 2.1 seconds, indicating that a small but identifiable segment of users encounters delays that begin to impinge upon seamless interaction. As we transition to moderate K values, the trend of escalating latency persists. At K=3, the P50 latency climbs to approximately 2.2 seconds, with the P99 latency reaching roughly 3.0 seconds. Further increasing K to 5 results in a P50 latency of approximately 3.5 seconds and a P99 latency of about 4.5 seconds. These values, while perhaps tolerable in less real-time critical applications, unequivocally push against the boundaries of an ideal conversational experience, where perceived delays can lead to user disengagement. The most pronounced and concerning latency increases are observed at K=10. At this context length, which, paradoxically, yields the most effective qualitative performance as per Section 4.3, the P50 latency soars to approximately 5.8 seconds. This implies that the typical user is now subjected to a waiting period of nearly six seconds for a response, a duration widely considered excessive for fluid conversational interfaces. More critically, the P99 latency surges to nearly 7.8 seconds, indicating that 1 percent of users are subjected to delays approaching eight seconds. Such prolonged latencies for a significant portion of the user base are generally deemed unacceptable in highly interactive systems, directly contributing to user frustration and potential abandonment of the tbot.

A particularly salient observation derived from these results is the consistent and increasingly pronounced divergence between P99 and P50 latencies as K increases. At K=2, the absolute difference between these two percentiles is a relatively modest 0.8 seconds (2.1s - 1.3s). However, this gap expands considerably as K grows, reaching approximately 2.0 seconds (7.8s - 5.8s) by the time K is set at 10. This widening discrepancy strongly suggests that the factors contributing to the longer response

times, such as increased processing complexity for more intricate queries, heightened resource contention under load, or longer generation sequences for comprehensive answers, become disproportionately magnified with larger context lengths. The "long tail" of performance, therefore, not only extends in absolute terms but also widens relative to the median experience, indicating that the impact of computational bottlenecks becomes more acute for the slowest percentile of interactions as context complexity grows.

In conclusion, the findings presented in Figure 7.2 underscore a critical and unavoidable trade-off in the optimization of LLM-powered travel bots: while the judicious enhancement of context length (K) undeniably leads to superior qualitative performance, as robustly demonstrated in Section 4.3, this improvement is consistently accompanied by a substantial increase in response latency. The observed response times, particularly at K=10 where qualitative metrics peak, far exceed conventional thresholds for acceptable real-time user interaction. For a conversational interface, a P50 latency approaching six seconds and a P99 latency nearing eight seconds are detrimental to user satisfaction and operational efficacy. This inherent tension between output quality and responsiveness necessitates a meticulously planned optimization strategy. Future work will therefore be dedicated to identifying the optimal K value that strikes an effective balance, ensuring high-quality bot responses while maintaining acceptable latency thresholds. This will involve exploring advanced techniques such as prompt compression to reduce input token count without sacrificing information, implementing highly efficient caching strategies to minimize redundant computations, leveraging model quantization for faster inference with reduced precision, or robustly scaling and optimizing the underlying computational infrastructure to handle increased loads. Ultimately, the successful deployment of a truly effective travel bot hinges on comprehensively addressing this latency challenge to deliver both intelligent and expeditious user experiences.

Chapter 8

Conclusion and Future Study

In our paper rigorously evaluated the performance of our LLM-powered travel bot across varying levels of in-context information, represented by the parameter K, providing critical insights into the complex interplay between response quality and computational efficiency. Our findings unequivocally demonstrate a strong positive correlation between increased K values and the qualitative performance of the chatbot. As evidenced in our analysis (referencing concepts from Figure 7.1), metrics such as Cot Contextual Accuracy, Conciseness, and Relevance consistently improved with higher K, reaching near-optimal levels at K=10. This suggests that providing richer, more extensive examples or contextual cues significantly empowers the LLM to generate more precise, efficient, and semantically aligned responses, thereby enhancing the bot's ability to effectively address user queries. The near-perfect Cot Contextual Accuracy observed at higher K values, for instance, highlights the model's enhanced reasoning capabilities when adequately primed.

However, this qualitative superiority comes at a significant and measurable cost. Our investigation into response latency (referencing concepts from Figure 7.2) revealed a clear and substantial increase in both P50 (median) and P99 (worst-case) latencies as K was augmented. At the most performant qualitative setting of K=10, the P50 latency approached 6 seconds, while the P99 latency neared 8 seconds. These response times are largely prohibitive for a conversational interface like a travel bot, where user expectations for real-time interaction are high. The expanding gap

between P50 and P99 latencies at higher K values further underscores that the increased computational burden disproportionately affects the slowest interactions, indicating potential bottlenecks in LLM inference, resource allocation, or prompt processing. Consequently, the core conclusion of this research is the identification of a critical trade-off: achieving higher response quality necessitates greater computational resources and, subsequently, longer response times, which directly impacts the overall user experience and system usability.

Building upon these findings, several avenues for future study are essential to transition this research into practical, high-performing deployments. Firstly, the immediate next step involves identifying the true "optimal" K value—one that strikes a judicious balance between acceptable qualitative performance and tolerable latency thresholds. This will likely involve a user experience study to quantify acceptable response times and a comprehensive cost-benefit analysis to determine the most economically viable configuration. Secondly, extensive research into mitigating the observed latency increases is paramount. This includes exploring advanced prompt optimization techniques, such as prompt compression or summarization, to reduce the input token count without sacrificing critical context. Further investigation into efficient caching strategies, including KV cache optimization and prompt caching, could significantly reduce redundant computations. Additionally, exploring model-level optimizations like quantization or the deployment of smaller, more specialized LLMs, potentially through distillation or fine-tuning, may offer pathways to reduced inference times. Thirdly, future work should expand the evaluation framework to include more direct user feedback mechanisms, such as post-interaction satisfaction surveys or implicit measures like task completion rates and user abandonment rates, to provide a holistic view of performance. Finally, examining the scalability of these findings across different LLM architectures and exploring real-world deployment considerations, such as A/B testing and gradual rollouts, will be crucial for validating the generalizability and robustness of our conclusions in diverse operational environments.

Bibliography

[1] C.-F. Chen and D. Tsai, "Exploring travel barriers and needs of non-english-speaking travelers," *Tourism Management*, vol. 75, pp. 45–55, 2019. DOI: 10.1016/j.tourman.2019.03.017.

[2] P. Joshi, S. Santy, A. Budhiraja, K. Bali, and M. Choudhury, "The state and fate of linguistic diversity and inclusion in the nlp world," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020, pp. 6282–6293. [Online]. Available: https://aclanthology.org/2020.acl-main.560.

[3] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *arXiv preprint arXiv:2005.11401*, 2020. [Online]. Available: https://arxiv.org/abs/2005.11401.

[4] J. Deriu, Á.-M. Rodrigo, *et al.*, "Survey on evaluation methods for dialog systems," *Artificial Intelligence Review*, vol. 54, no. 1, pp. 755–810, 2021. DOI: 10.1007/s10462-020-09836-1.

[5] D. Gursoy, C. G. Chi, and L. Lu, "Travel anxiety and barriers: Language challenges of international tourists," *International Journal of Hospitality Management*, vol. 92, p. 102 737, 2021. DOI: 10.1016/j.ijhm.2020.102737.

[6] Y. Xu, S. Sun, B. Li, *et al.*, "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021, pp. 5274–5286.

[7] M. M. Hasan, M. M. Rahman, and M. I. Khan, "Challenges of nlp in bangla: A survey of current tools and future directions," *Journal of South Asian Language Technologies*, vol. 5, no. 2, pp. 35–48, 2022.

[8] H. Touvron, T. Lavril, G. Izacard, X. Martinet, *et al.*, *Llama 3: Open foundation and instruction-following models*, https://ai.meta.com/blog/meta-llama-3/, 2023.

[9] UNWTO, *International tourism highlights: 2023 edition*, https://www.unwto.org/tourism-statistics-and-data, 2023.

[10] A. Xu, W. Chen, F. Li, and J. Bian, "Bias in llms for travel recommendations: Demographic sensitivity and fairness in tourism ai," *International Journal of Artificial Intelligence in Hospitality*, vol. 2, no. 1, pp. 15–29, 2023.

[11] S. Ying, H. Zhang, Y. Liu, Z. Wang, and Y. Cao, "Genai for urban transport planning: Evaluation of gpt-4 and phi-3 in spatial decision support," *arXiv preprint arXiv:2312.01594*, 2023. [Online]. Available: https://arxiv.org/abs/2312.01594.

[12] B. Mo, Z. Li, and M. Zhang, "Travelagent: An llm-powered travel planning system with real-time spatiotemporal awareness," *arXiv preprint arXiv:2403.17648*, 2024. [Online]. Available: https://arxiv.org/abs/2403.17648.

[13] Vromon Guide, *Vromon guide - bangla travel blog*, Accessed: 2025-06-07, 2024. [Online]. Available: https://vromonguide.com/.