

Building an Intelligent QA/Chatbot for Transportation with LangChain and Open Source LLMs

Team 11

Ethan Do, Aneesh Sunkarapalli, Aarush Patil, Neil Akalwadi, Rami
Ghaleb

CS4624 Multimedia, Hypertext, and Information Access
Virginia Tech, Blacksburg, VA

Instructor and Client: Dr. Mohamed Farag (mmagdy@vt.edu)

March 2025

Contents

1	Abstract	4
2	Introduction	5
2.1	Problem	5
2.2	Motivation	5
2.3	General Approach	6
3	Requirements	7
3.1	Scope	7
3.2	Features	8
3.3	Deliverables	8
3.4	Expectations	9
4	Design	10
4.1	Frontend Layer	10
4.2	Backend Layer	10
4.3	Database Layer	11
4.4	Data Flow Architecture	11
5	Implementation	14
5.1	Data Processing and Retrieval	14
5.2	LLM Integration	15
5.3	Backend API Development	15
5.4	Frontend Development and UI	16
5.5	User Authentication	16
5.6	Maintaining Conversational Contexts	16
6	Testing and Evaluation	18
6.1	RAG Testing	18

6.2	Document Chunk Retrieval Validation	18
6.3	LLM Performance Evaluation	19
6.4	Evaluation Results	19
7	User Manual	20
7.1	Login Process	20
7.2	UI Overview	20
7.3	Working with Collections (Chat and Chat History)	21
7.3.1	Starting a New Chat Collection	22
7.3.2	Accessing and Managing Existing Collections	22
7.4	Internal Chat Interface	23
8	Developer Manual	26
8.1	Development Environment Setup	26
8.1.1	Prerequisites	26
8.1.2	Backend (Flask and LangChain)	27
8.1.3	Frontend (React)	27
8.1.4	Database Setup (SQLite and ChromaDB)	27
8.2	Project Structure	27
8.2.1	Frontend Directory Structure	27
8.2.2	Backend Directory Structure	28
8.3	Run system	28
8.4	Run locally	29
9	Lessons Learned and Future Work	30
9.1	Lessons Learned	30
9.2	Timeline/Schedule	31
9.3	Problems Encountered	31
9.4	Solutions to the Encountered Problems	32
9.5	Future Work	32
10	Timeline and Milestones (Plans for the Remainder of the Semester)	34
10.1	Milestone 1: Requirements and Initial Thoughts	34
10.2	Milestone 2: Data Preparation and Retrieval Infrastructure	35
10.3	Milestone 3: LLM Integration and Pipeline Development	35
10.4	Milestone 4: User Interface Development	35
10.5	Milestone 5: Deployment, Testing, and Final Evaluation	36
11	Acknowledgments	37

Chapter 1

Abstract

The project involved developing a question-answering (QA) chatbot to assist transportation engineers who use complex traffic simulation software on a frequent basis. The chatbot helps engineers quickly interpret information from simulation manuals by asking questions. It utilizes LangChain for pipeline integration, ChromaDB for data storage and retrieval, and open-source Large Language Models (LLMs) for generating responses relevant to the prompt and context of the chatlog. It also exercises Retrieval-Augmented Generation (RAG) to help users obtain answers by retrieving related information from the domain-specific documentation (traffic simulation manuals in this case). Users interact with all of these technologies through a web application interface that possesses functionalities like conversation collections made possible by maintaining context and history, as well as user authentication for accessing and managing past collections of interactions.

This report documents our team's development process and highlights our workflow and approach to the problem; addressing progress in initial data processing tasks, including efficient document segmentation and chunking. It also outlines our experience with incorporating open source LLMs. We define clear milestones and deliverables for our development and tackle the challenges of maintained conversation contexts, potential improvements to our web-based user interface, and the implementation of an authentication system for users to login.

Chapter 2

Introduction

2.1 Problem

Traffic engineers encounter difficulties when trying to use traffic simulation software due to their use of complex and extensive manuals. They're accustomed to having to use manuals that are long and technical, making the process of quickly locating specific information time-consuming and unnecessarily challenging. These engineers consistently deal with unique scenarios or have questions that can't be directly addressed by standard manuals. This limits their ability to quickly resolve issues they might encounter when trying to apply their skills in the realm of simulation software.

2.2 Motivation

Recognizing these challenges, our team is determined to improve the efficiency and user experience for traffic engineers by creating an accessible/easy to use interactive solution. We hope to provide a platform that can offer context-aware assistance geared towards the user's questions. Through seamless follow-up questions and stored conversation contexts, we believe that traffic engineers will have a tool that effectively helps them based on their situation. The ultimate goal is to improve the workflow of traffic engineers by simultaneously improving their productivity when dealing with simulation tools.

2.3 General Approach

We are adopting an iterative development process for building our specialized QA chatbot application to properly address the outlined issues and follow through with what we intend to build. Our approach consists of:

1. Integrating LangChain's retrieval system with ChromaDB to chunk, vectorize, and retrieve relevant information directly from the traffic simulation manuals will ensure accurate responses to user queries.
2. Evaluating different open-source LLMs (primarily Hugging Face resources). By testing we can try to determine what model we deem as the most capable of delivering relevant responses that properly address prompts given by the user.
3. Using Flask, we will construct a backend that can handle API requests, manage user interactions, and promote the flow of data between the frontend, ChromaDB, and the LLM used.
4. Designing and implementing a user interface built with React. The frontend will allow traffic engineers to easily ask questions by interacting with the chatbot, as well as viewing past interactions via chat logs.
5. Enhancing user experience and data security by implementing a reliable user authentication system; this allows users to store conversation histories and revisit prior interactions attached to their profile.

Chapter 3

Requirements

3.1 Scope

The scope of this project is defined by the design, implementation, testing, and deployment of the intelligent QA/chatbot system we are trying to create for transportation engineers. The primary technical specifications of the project include:

- A frontend built using React, styled with Tailwind CSS or Material UI for optimal user experience across all devices.
- A backend that utilizes Flask to provide robust API endpoints, manage secure user sessions, handle queries, and communicate seamlessly with both frontend and database layers.
- A SQLite database with SQLite3 to securely store user authentication details and chat history.
- The integration of LangChain and ChromaDB to chunk, vectorize, and retrieve relevant information from the uploaded simulation manuals.
- The selection and integration of open-source LLMs (from Hugging Face's library, for example)
- Containerization using Docker to deploy the application to a web server with appropriate power to consistently exercise the LLM-powered chatbot

3.2 Features

Our QA/chatbot system provides various features that define the experience of our primary users, the traffic engineers, such as:

- LangChain and ChromaDB integrated with a chosen, active LLM from a set of options to deliver contextually accurate responses based on the traffic simulation manuals.
- ChromaDB provides an efficient way to store, chunk, and vectorize the text to make accurate retrieval of information from the traffic simulation manuals possible and reduce response times to user queries.
- Users can access structured conversation histories based on prior sets of interactions they've had with the chatbot, making it easier to come back to responses and knowledge that they might need to utilize later on.
- The project architecture takes into account possible integration of additional data sources in the future for more nuanced responses.

3.3 Deliverables

The final form of the project will be in the form of a set of deliverables to the client, consisting of:

- A fully developed and responsive web application deployed to the internet.
- Secure authentication infrastructure enabling user profiles and associated chat histories.
- Comprehensive user and developer manuals detailing the application's functionality, technical implementation, and future improvements.
- Well-documented source code along with the scripts needed for data preprocessing and model testing.

3.4 Expectations

The project team and the client both expect to see some improvements in traffic engineers' productivity as a result of the use of our application. The system is expected to provide personalized assistance and greatly reduce the need to manually search through the lengthy manuals. It will enhance the user experience for the traffic engineers by providing a way for them to have a sort of "hands-on" experience, in terms of interactions with the application, and access conversations with easily retrievable historical context of the chat. The client expects the delivered solution to be effective and well-documented so it can serve as a functioning platform for its place within the world of intelligent transportation systems.

Chapter 4

Design

4.1 Frontend Layer

The frontend layer's task is to provide a responsive user interface that allows our userbase (primarily traffic engineers) to interact with the chatbot application easily. This layer is built primarily using React to promote ease of use, due to its component-based structure, and quick interaction. The styling is handled using Tailwind CSS to maintain an aesthetic design when the application is being viewed on the web. The primary functionalities are:

- User-friendly input fields for questions/queries.
- An interactive chatbot interface for simple user communication.
- A display of conversation histories with associated conversation contexts.
- An account management interface to handle personalized user experiences and our utilized authentication processes.

4.2 Backend Layer

The backend layer's task is to manage the business logic of our approach, data associated with users, and serve as the middle-party to handle the frontend interface, database, and machine learning components. Since it's built on the Flask framework, the backend provides RESTful APIs to handle incoming queries, authenticate users easily, and promotes the passing of data across different layers of the project. The primary functionalities are:

-
- Managing user authentication and handling data for each session/instance.
 - Processing requests from the frontend and communicating them back to the LLM layers and data retrieval.
 - Function as a bridge between the ChromaDB retrieval system, LangChain pipeline, and frontend interactions.

4.3 Database Layer

The database layer consists of two main components that both handle the aspects of data storage and retrieval that our project requires:

- **SQLite Database:** Securely manages user data, from authentication information to user profiles and associated conversation histories (collections). SQLite offers scalability and efficient querying for different-structured data, making it ideal for whatever our application might require. With the SQLite 3 incorporation, it remains local to the user's machine rather than in the cloud.
- **ChromaDB (Vector Store):** Acts as the vector database to store pre-processed and vectorized chunks from the traffic simulation manuals. The database efficiently supports similarity searches for contextually accurate retrieval of relevant manual segments when queried through LangChain's retrieval pipeline.

4.4 Data Flow Architecture

Our data flow architecture is how data travels across the system between the frontend, backend, databases, and the LangChain pipeline:

1. The user would submit a query through the React frontend interface (in the form of a question to the chatbot) to be sent to the Flask backend API.
2. Flask authenticates this request and sends the query to the LangChain retrieval pipeline.
3. LangChain interacts with ChromaDB to retrieve vectorized document chunks that are found as relevant to what the user is asking the bot.

-
4. The retrieved document chunks and the query from the user are passed to the LLM then integrated through LangChain to generate a response relevant to the context of the question/query.
 5. The LLM-generated response is sent to the Flask backend and the user's conversation history in the SQLite database is updated as a result (so that the context of the conversation with the chatbot can be preserved).
 6. The backend then sends the full response to the frontend to be displayed to the user, answering their initial question to the chatbot.

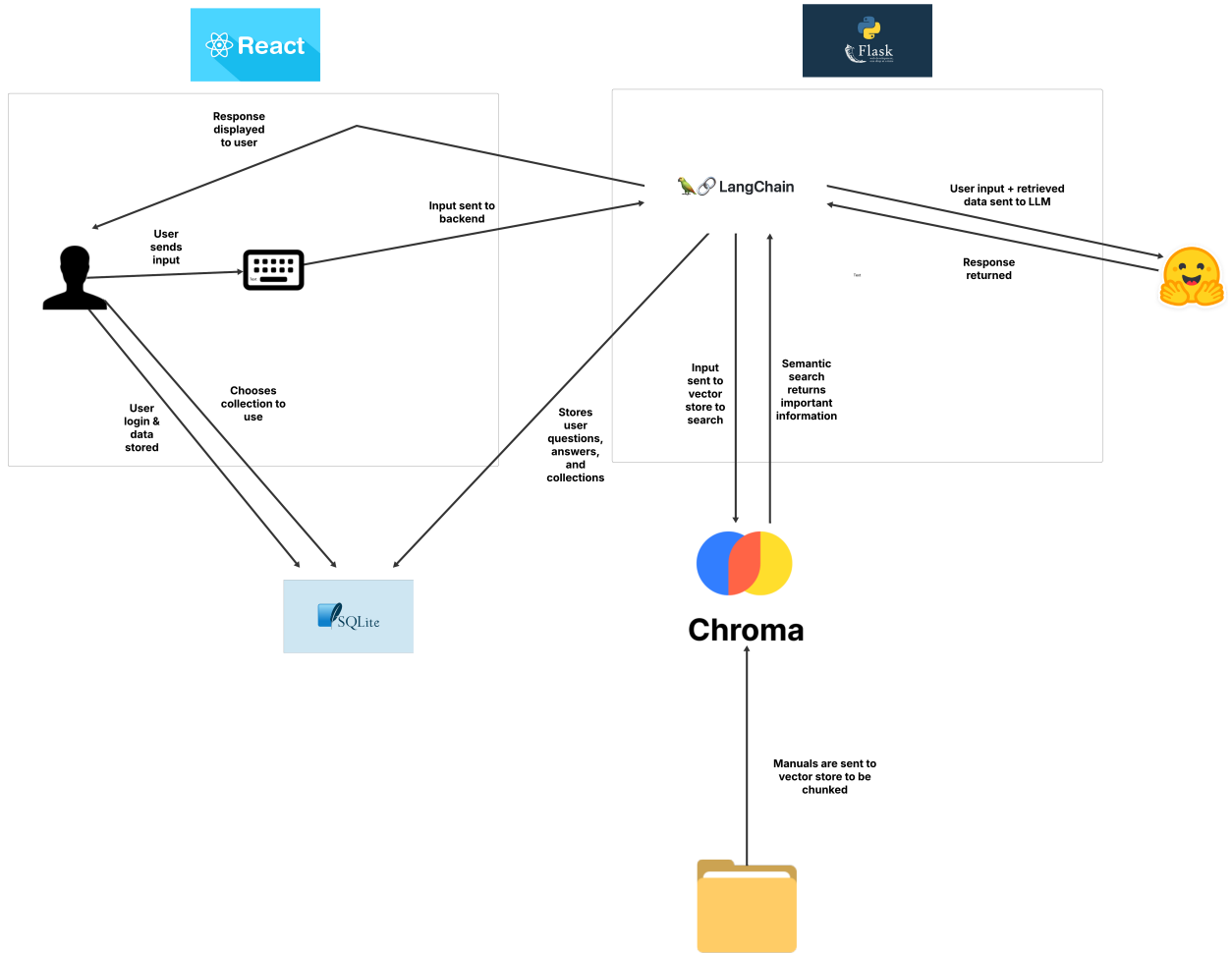


Figure 4.1: System Architecture Diagram

Chapter 5

Implementation

Our implementation strategy is divided into six major parts: Data Processing and Retrieval, LLM Integration, Backend API Development, Frontend Development and UI, User Authentication, and Maintaining Conversational Contexts. Each segment presents the main technical work done, the frameworks and tools utilized, and how they were incorporated into the overall system design.

5.1 Data Processing and Retrieval

To build a well-performing RAG pipeline, we used a formal data processing workflow with LangChain and ChromaDB. The process was: capturing and parsing traffic simulation manuals using LangChain's document loaders and divided them into context-aware pieces (chapter-division and byte-limited) using custom text splitters.

- Generated embeddings using open-source models like sentence-transformers from Hugging Face.
- Pushed the vectorized pieces into ChromaDB to facilitate fast, similarity-based retrieval when users query.

All implementation behind data chunking, embedding, and storing is handled within LLM/LLM.py. It governs the center of the retrieval cycle that supplies the chatbot pipeline.

5.2 LLM Integration

We evaluated several open-source LLMs from Hugging Face, with a focus on light, quantized models to balance performance with efficiency. Testing was performed via Google Colab since it provides easy access to GPU acceleration. With models benchmarked for accuracy, latency, and relevance to transport-related queries, we selected the best of them and added it to our LangChain pipeline:

- Loaded the model using Hugging Face’s Transformers library or llama-cpp for quantized models.
- Created scripts to directly integrate ChromaDB retrieval results into the model pipeline through LangChain.

5.3 Backend API Development

The backend API is built using Flask, serving as the middleman between the frontend, the LangChain pipeline, and the databases. It assumes responsibility for communication, logic flow, and user-specified interactions. Its fundamental functionalities include:

- RESTful API endpoints handle user queries, run through the LangChain pipeline, and return the chatbot responses to the React frontend.
- Standard authentication integration with safe storage of user sessions in the SQLite database.
- SQLite utilized for storage of user metadata and chat logs against authenticated accounts.
- Thorough error handling and logging implemented for facilitation of reliability and easier debugging at runtime.

Everything from backend functionality exists in the backend/ directory, primarily within main.py, auth.py, and LangChain and ChromaDB integration-specific modules.

5.4 Frontend Development and UI

The frontend is built with React, providing a responsive and clean interface for the interaction with the chatbot. Tailwind CSS is used to style, keeping the looks crisp and consistent. Some of the key frontend features are:

- Modular React functional components designed for maintainability and scalability for pages like chat, login, and collections.
- State management with React hooks and context to control user sessions, API requests, and dynamic UI updates to chat history and UI state.
- Integration of authentication system for secure login.
- Multi-session chat collections, seamless query sending, chat history persistence, and minimal chat UI optimized for simplicity.

All the frontend code lives in the frontend/src directory, organized into components/, pages/, and auth/ for scalable code.

5.5 User Authentication

For user authentication and management, we employed standard user authentication for easy frontend-backend access control:

- The backend is implemented in Flask with token authentication for the users, and SQLite connection to store the metadata of the users (e.g., email, password, session history).
- The frontend uses React and utilizes our authentication system to initiate authentication flows and obtain access tokens, which are securely sent to the backend for validation.

5.6 Maintaining Conversational Contexts

To allow follow-up questions and maintain context for different collections/conversations, we stored chat histories in the SQLite database.

-
- We utilized this to maintain the conversation between user queries and chatbot answers to ensure the conversation can be used as a reference and even continued.
 - Each user's chat history is stored in SQLite, divided into simulation scenarios (collections), so they can come back and continue old sessions.

Chapter 6

Testing and Evaluation

Comprehensive testing and evaluation are an essential component of our project because it's how we can verify that the developed QA chatbot meets the specified requirements and delivers a satisfactory experience. This chapter highlights how we went about testing our application and its core parts.

6.1 RAG Testing

Retrieval-Augmented Generation is the main part of our chatbot's capability to provide accurate and relevant responses to the context. To test our RAG pipeline, we:

- Manually prepared a set of queries targeting known information within our simulation manual dataset. Each generated response was evaluated to look at accuracy in relation the context of the question and the information present within the manuals.
- Repeated queries multiple times to verify that the retrieval process was consistently examining the correct chunks of the document for context and that the responses were consistent across multiple interactions with the chatbot.

6.2 Document Chunk Retrieval Validation

Efficient retrieval of information chunks stored within ChromaDB is another integral part of the chatbot's performance. To test our vector store, we:

-
- Retrieved document chunks that we manually reviewed and compared with the expected sections of the original documentation to measure relevance and completeness
 - Adjusted chunk sizes and embedding models to determine the best configurations that would balance context without harming the accuracy of retrieved information
 - Tested repeatability by submitting the same queries across multiple sessions to ensure that retrieval results were accurate on the basis of the text and not highly randomized/varied

6.3 LLM Performance Evaluation

To guarantee the reliability and responsiveness of our selected Hugging Face model (TinyLlama), we:

- Created a comprehensive set of queries that we tested against the LLM using Google Colab environments. The generated responses were then looked at for accuracy and clarity by comparing them against what responses we expected to be produced.
- Conducted trials of multiple possible models (that we were considering using as our main) under identical conditions, comparing the quality of their results/responses and leading to the final model selection.

6.4 Evaluation Results

These initial test results showed us that:

- ChromaDB retrieval testing showed consistent retrieval in correct document chunks across different test queries which shows a strength in the RAG performance.
- Hugging Face LLM tests indicate solid accuracy and contextual relevance in the generated responses which are satisfactory considering the desired functionality of the project.
- Usability and ease-of-use tests affirmed that the frontend interface, developed with React, delivers a user-friendly experience.

Chapter 7

User Manual

This manual provides the user with step-by-step guidance on how to use the transportation chatbot, from logging in to interacting with the chatbot and managing your chat history.

7.1 Login Process

Follow these steps to log into the system:

1. Go to the application's homepage.
2. Click the **Register Here** button located near the bottom of the login box for first time account creation
3. Enter email and a password of choice to create account.
4. Account and associated chat collections can be accessed with login.

7.2 UI Overview

After logging in, users will see the primary interface, which includes the following components:

- **Main External Panel:** The main landing page/external area showcases the different existing collections that can be entered by the user upon clicking the specific chatlog they would like to see

-
- **Collections Panel:** The left sidebar side displays chat collections that each have their own context.
 - **Main Internal Panel:** The main internal area provides access to the chatbot interaction window.

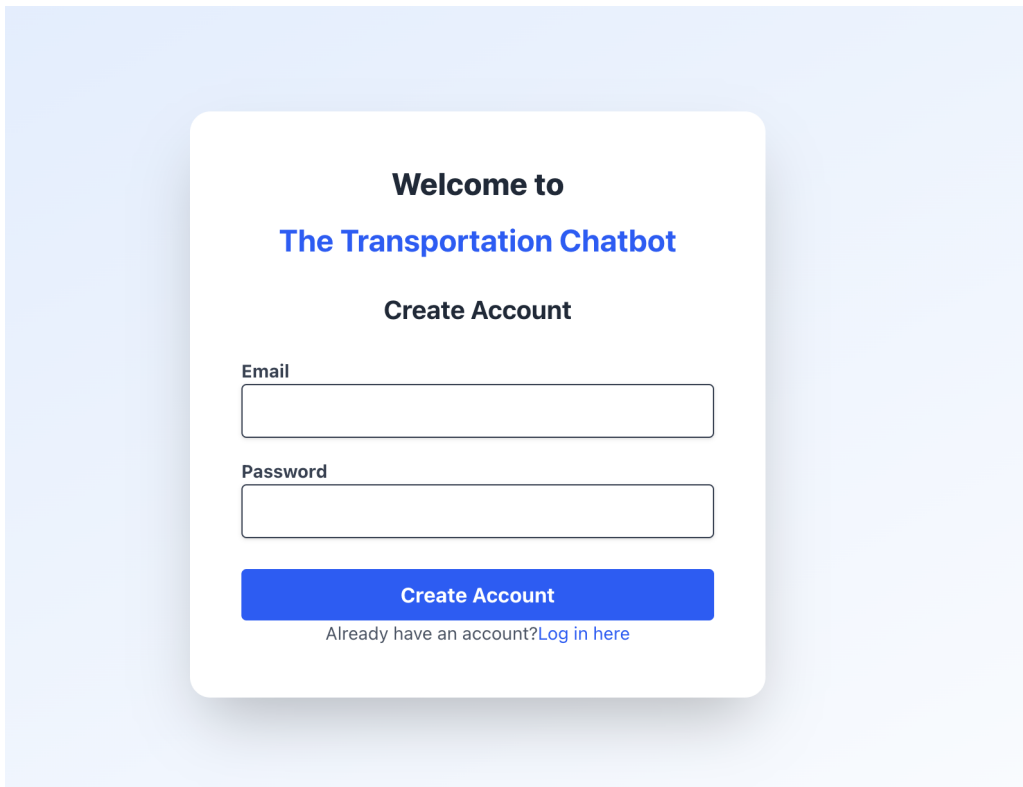


Figure 7.1: Account Interface (Create Account)

7.3 Working with Collections (Chat and Chat History)

The chatbot system organizes your interactions into Collections, each corresponding to a specific context inherently set by the user's queries.

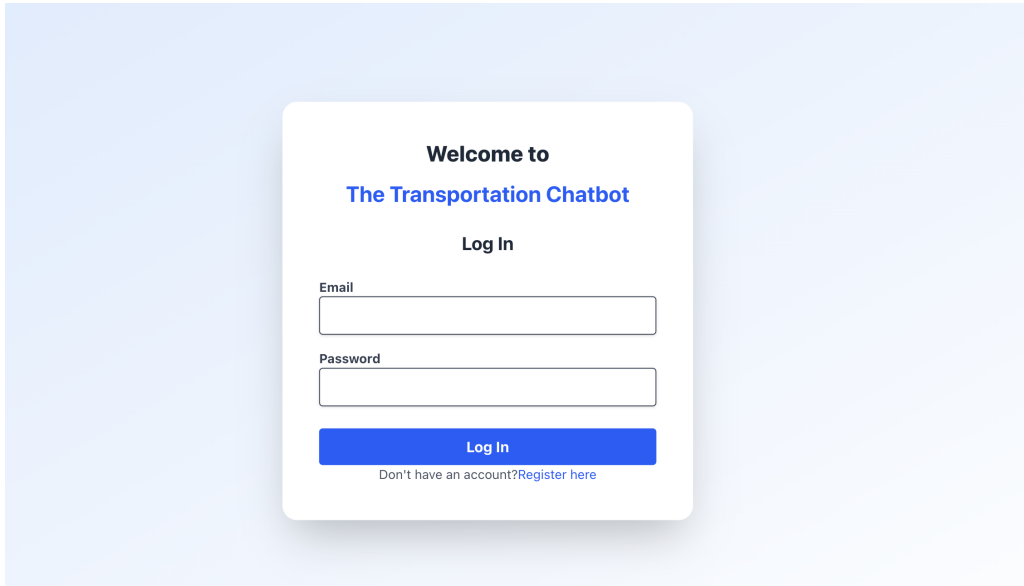


Figure 7.2: Account Interface (Login)

7.3.1 Starting a New Chat Collection

1. In the Collections Panel, click the New Collection button.
2. Enter a descriptive name for your collection.
3. Confirm by clicking the New Collection button. The new collection now appears in your panel.

7.3.2 Accessing and Managing Existing Collections

To access previous chat conversations:

1. Select the desired collection from the Collections Panel.
2. The chat history for the selected collection appears in the main area of the panel.
3. To delete a specific collection, click the trashcan icon on the left of the tap for the specific chat collection.

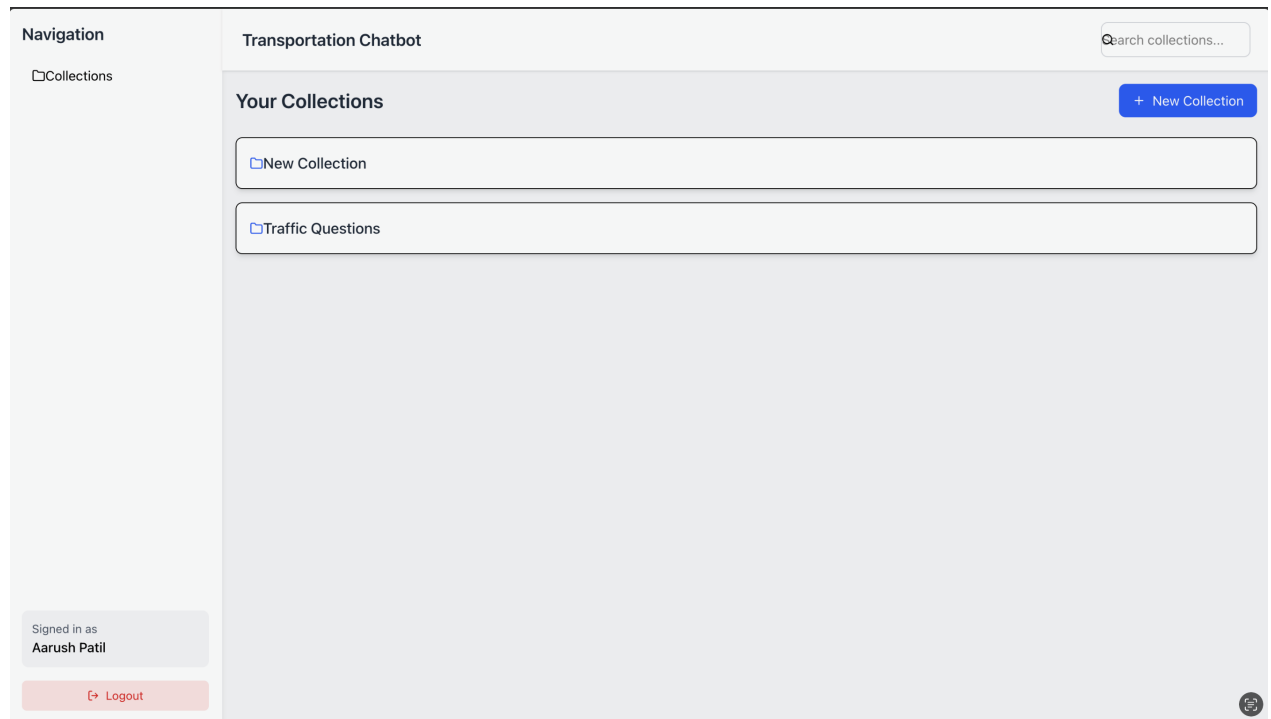


Figure 7.3: Landing Page

7.4 Internal Chat Interface

The internal chat interface is where users directly interact with the chatbot:

1. After selecting or creating a collection, navigate to the chat interface in the main area panel.
2. Type your query into the provided text input box at the bottom of the chat window.
3. Press the Send button or hit the Enter key to submit your question.
4. View the response displayed under your query.
5. Enter follow-up questions or additional queries if desired, since the chatbot maintains conversational context.

Collections





-  what is traffic 
-  what is an O-D numbe... 
- [+ New Collection](#)

Figure 7.4: Dialog for Creating a New Collection

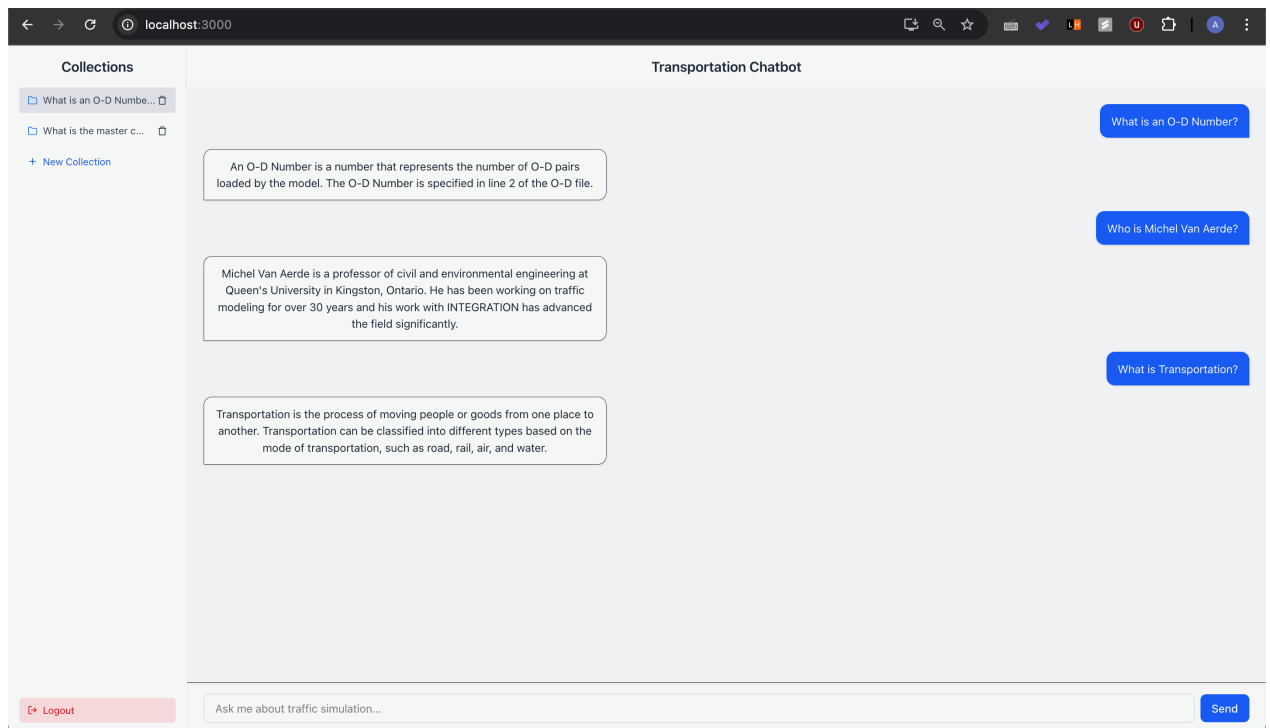


Figure 7.5: UI Overview

Chapter 8

Developer Manual

Source Code: <https://github.com/aarushpatil/Capstone>

This developer manual provides detailed instructions into setting up and maintaining the transportation chatbot system.

8.1 Development Environment Setup

To properly set up the development environment, ensure that the following software and dependencies are installed:

8.1.1 Prerequisites

- VS Code (or any IDE of choice): <https://code.visualstudio.com/download>
- Python (version 3.11 or newer): <https://www.python.org/downloads/>
- Node.js and npm (latest LTS version recommended. We used node js v24.0.0 and npm v11.3.0): <https://nodejs.org/en/download>
- SQLite: <https://www.sqlite.org/download.html>
- Docker (for containerization and deployment): <https://docs.docker.com/engine/install/>
- Google Colab (optional, recommended for LLM experimentation)
- **Clone the repo using the following command:**

```
git clone https://github.com/aarushpatil/Capstone.git
```

8.1.2 Backend (Flask and LangChain)

1. Launch docker desktop application (from install)
2. Enter the root directory and make a docker image to run later

```
cd Capstone
docker build -t chatbot ./
```

8.1.3 Frontend (React)

- (a) Navigate to the frontend directory and install npm dependencies:

```
cd frontend
npm install
```

8.1.4 Database Setup (SQLite and ChromaDB)

- i. SQLite does not require setup as it is already included within the docker image.
- ii. ChromaDB initializes automatically via scripts in the backend environment. Data ingestion and embeddings are handled by scripts located at `scripts/data_processing.py`.

8.2 Project Structure

The project is structured into clear, modular directories and files for ease of navigation, maintenance, and scalability:

8.2.1 Frontend Directory Structure

- `public/` - Holds static assets
- `src/` - Main source folder containing:

-
- **components/** - Reusable React components
 - **pages/** - Main views of the application
 - **styles/** - Styling files
 - **utils/** - Utility functions, such as API calls
 - **App.js** - The application's main entry point

8.2.2 Backend Directory Structure

- **backend/** - Backend source folder containing:
 - **main.py** - Main Flask application
 - **LLM/** - Handles language model operations
 - * **LLM.py** - Implements the RAG pipeline
 - * **chunking.py** - Splits manuals into chunks
 - **chroma_db/** - Stores cached ChromaDB with manuals
 - **requirements.txt** - Lists Python dependencies
 - **Dockerfile** - Defines the Docker setup for the backend and frontend

8.3 Run system

We containerize our backend system using Docker:

- (b) Deploy containers locally and start server: In the current terminal (let's call it terminal 1), we will navigate to the Capstone directory, then run the backend with the following commands:

```
cd ..
docker run -p 5050:5050 -v $(pwd):/app -it chatbot /bin/bash
cd backend
python main.py
```

Open a new terminal (terminal 2), where we will navigate to the Capstone directory and run frontend with the following commands:

```
cd Capstone
docker run -p 3000:3000 -v $(pwd):/app -it chatbot /bin/bash
cd frontend
```

```
npm start
```

- (c) Open <http://localhost:3000> on your browser of choice to interact with the application

8.4 Run locally

If you prefer to run the application locally without Docker, follow the steps below (after cloning the repository):

Backend port 5050:

```
cd Capstone/backend
pip install -r requirements.txt
python main.py
```

Then, open a second terminal:

Frontend port 3000:

```
cd Capstone/frontend
npm install
npm start
```

Then open your browser and navigate to:

<http://localhost:3000>

Chapter 9

Lessons Learned and Future Work

9.1 Lessons Learned

Throughout the development of our project, our team consistently faced obstacles and had to exercise problem-solving techniques to make it past the issues we encountered. Reflecting on our experiences, we found that:

- Our team initially underestimated how complex it would be to process and chunk the data of the manuals and the manuals themselves. Thankfully, spending time to make sure that our document preprocessing was successful led to better retrieval of related/relevant information from the manuals.
- Determining a useful LLM to use in the context of our goals for the project was something that turned out to be more time-consuming than we thought. We then learned how important it was to test across multiple models to determine how model performance would affect the chatbot's quality of responses, as well as the time it would take to answer.
- Focusing heavily on our UI design during the frontend development process made it so that we could easily identify the strengths and weaknesses in our core application and logic. This provided us insight on why the frontend's functionality is very vital even when compared to the backend and database layers which seemingly do most of the

heavy lifting when it comes to making sure our processes are running effectively for the application's use.

9.2 Timeline/Schedule

Developments	General Date	Current Status
Determining Initial Requirements	March 2025	Completed
Tech Stack Research and Setup	March 2025	Completed
Initial Google Colab Model Deployment	March 2025	Completed
Data Chunking Implementation	March 2025	Completed
ChromaDB Integration	March 2025	Completed
Retriever Accuracy Testing	March 2025	In Progress
LLM Evaluation and Selection	March 2025	In Progress
LangChain Pipeline Integration	March 2025	In Progress
Prompt and Response Quality Development	April 2025	Upcoming
Frontend UI/UX Development	April 2025	Upcoming
Full System Testing	May 2025	Upcoming
Final Deployment and Project Delivery	May 2025	Upcoming

Table 9.1: Detailed Project Schedule

9.3 Problems Encountered

As previously mentioned, our team faced various challenges from the technical to communicative facets of the project like:

- Difficulty in effectively chunking the manuals to make sure that relevant data was being retrieved accurately and consistently in order to provide the user with solid responses to their questions/queries.
- Determining the best open-source LLM to use from the options that we had available to us that made the most sense in the context of what our project does (functioning as a QA chatbot).
- Determining how to get the ChromaDB retrieval systems and the LangChain pipeline working together properly.

-
- Making sure that frontend and backend components were effectively linked together and weren't causing bugs to occur in standard use of the application.
 - The responsiveness of the user interface when casually using the application; seeing how smooth it was to navigate from page to page.
 - Communicating when a member was working on a certain component of the project

9.4 Solutions to the Encountered Problems

We addressed these problems by:

- Testing different chunk sizes multiple times to make sure the manuals were chunked properly and the data retrieval was reliable. This made it so that the built chatbot was verified to consistently provide responses to users that satisfied their needs.
- Setting up a clear and organized testing method to compare the different open-source LLMs and determine the best one for our chatbot specifically.
- Delving into documentation and scripts to make sure that the ChromaDB retrieval system was integrated with the LangChain pipeline and functioned as expected.
- Defining simple API guidelines for the Flask backend so that the frontend and backend could communicate properly without causing bugs during a standard use of the chatbot and application as a whole.
- Regularly testing out the application's interface to ensure it wasn't made less responsive due to changes in development.
- Focusing during our team meetings and assigning defined tasks to promote that our members could communicate clearly about what specifically they were working on at that point in time.

9.5 Future Work

We believe that future teams that are tasked with this project should try to improve upon our iteration by:

-
- Training the LLM with additional traffic simulation items like manuals or datasets to expand the chatbot's capabilities with the increased knowledge it possesses as a result of learning from more material.
 - Conducting some tuning to optimize the LLMs, available as options in the application, in an attempt to improve the quality of the responses without sacrificing the time taken to produce and display the response to a user query.
 - Improving overall aesthetics and layout wherever possible for the user interface, making the experience of interacting with the application as a whole much smoother.
 - Fleshing out the Admin functionality, providing more abilities in the application besides alternating the active LLM being used for a collection.
 - Making improvements to the scalability of the system to allow the chatbot to handle any possible increase in use, from a larger number of users to more substantial queries.

Chapter 10

Timeline and Milestones (Plans for the Remainder of the Semester)

This chapter covers the timeline and milestones that we created for a successful completion of the transportation chatbot project.

10.1 Milestone 1: Requirements and Initial Thoughts

Planned completion: Early March 2025 (Completed)

- Meet with the client and gather a comprehensive list of project requirements that clearly define the user expectations.
- Conduct research on available NLP tools, RAG frameworks and potential open-source LLMs.
- Create and present an initial project presentation that defines the goals and expected outcomes.
- Set up a test by deploying a model in a Google Colab environment to verify the workflow plan.

10.2 Milestone 2: Data Preparation and Retrieval Infrastructure

Planned completion: Late March 2025 (Completed)

- Collect, pre-process, and chunk the documentation data from the traffic simulation software. Ensure that the information is structured for efficient and effective retrieval.
- Evaluate and select the best embedding model for vectorizing document chunks.
- Utilize ChromaDB as an easy-to-use vector store for storing embeddings and enabling efficient information retrieval.
- Conduct initial tests to verify the speed and accuracy of the integrated ChromaDB retriever with realistic test queries.

10.3 Milestone 3: LLM Integration and Pipeline Development

Planned completion: Mid-April 2025 (In Progress)

- Perform comparative testing among various Hugging Face LLMs and select the best performing model.
- Integrate the LLM into the LangChain pipeline and ensure that ChromaDB correctly retrieves results and model inputs.
- Design, implement, and refine the prompt templates to optimize for domain-specific accuracy.
- Develop a complete function pipeline to effectively generate responses from user queries through retrieval-augmented generation (RAG).

10.4 Milestone 4: User Interface Development

Planned completion: Late April 2025 (Upcoming)

-
- Use SQLite to develop a secure backend service to store and retrieve conversation histories.
 - Design and build a user-friendly frontend interface with React. Include intuitive chat sessions and chat history collections.
 - Conduct user experience testing to refine frontend interactions, improve navigation, and optimize the overall usability of the chatbot application.

10.5 Milestone 5: Deployment, Testing, and Final Evaluation

Planned completion: Early May 2025 (Upcoming)

- Containerize the frontend and backend using Docker, and prepare for deployment on a dedicated web server.
- Conduct comprehensive system testing, including backend API tests, frontend usability tests, conversational response accuracy tests, and stress tests.
- Collect and analyze the testing data to evaluate overall system performance and identify areas for optimization and improvement.
- Deploy the final, fully-tested system to production while ensuring a stable operation for users.
- Prepare the final documentation and presentation materials for project handoff and demonstration.

Chapter 11

Acknowledgments

We'd like to thank our instructor and client Dr. Mohamed Farag for his insight and support throughout development of the project, from the initial stages to the final product. In his role as our client, Dr. Farag directed us with detailed instructions to ensure that our final deliverable would incorporate all of the desired specificities of the application so that it would meet his standards. His interest in intelligent transportation systems and machine learning carried over into our client meetings to foster an engaging environment. We appreciate his time and effort in guiding us, which thoroughly contributed to our experience in building the project and the project itself.

Guest Lecturer Yusuf Elnady also provided us with a great starting point in the Google Colab notebook he showcased. The QA system he developed, along with the successful implementation of the ML model, gave us code that we could utilize and adopt to fit our project's specific context and needs regarding traffic simulation.

Chapter 12

References

- LangChain Documentation <https://python.langchain.com>, accessed March 2025.
- ChromaDB GitHub Repository <https://github.com/chroma-core/chroma>, accessed March 2025.
- Hugging Face Models Documentation <https://huggingface.co/docs>, accessed March 2025.
- TheBloke (2024) CapybaraHermes-2.5-Mistral-7B-GGUF Model. Hugging Face Repository <https://huggingface.co/TheBloke/CapybaraHermes-2.5-Mistral-7B-GGUF>, accessed March 2025.