

Intelligent QA/Chatbot for Transportation

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

Group: 11

CS4624

Mohammed Farag

Virginia Tech, Blacksburg VA 24061

4/1/2025

Presentation Outline

3 - Problem

4 - Motivation

5 - Project Description

6 - Requirements/Deliverables/Features

7 - Approach/Plan

8-9 - System Architecture Diagram

10 - Technologies and Frameworks

11-12 - Backend

13-17 - Frontend

18 - Database

19 - Docker Containerization

20 - User Manual

21 - Developer Manual

22 - Challenges We Faced

23 - Improvement/Next Steps

24 - Timelines and Milestones

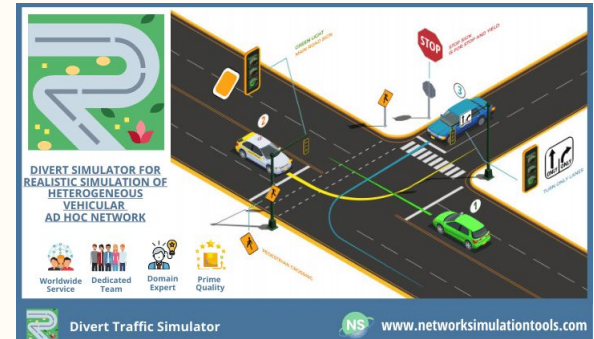
25 - Roles

26 - Client

27 - References

Problem

- Traffic engineers struggle with having to read through long and confusing manuals to learn how to effectively run traffic simulations.
- Looking for a specific detail in traffic simulations from a paper manual will be highly time consuming and inefficient.
- Engineers may have personal difficulties that cannot be easily solved through a standard manual.
- Following a question, there may be additional questions that arise related to the previous queries.



Motivation

- Our team wants to make a change and improve efficiency for traffic engineers.
- We want to allow a personalized experience for traffic engineers to get help catered to their needs.
- We'd like to enhance workflow by being able to have follow up questions.
- Create an accessible and convenient experience for traffic engineers.



Project Description

- We are building a web application-based QA (question-answering) chatbot that assists traffic engineers in understanding how to run traffic simulations
- Utilizes Large language models (LLMs) and Retrieval-augmented generation (RAG) loaded with traffic simulation manual data to generate responses within the context of the user-requested question by extracting related information from the data and providing assistance on running a simulation
- The homepage of the web app contains collections, essentially chat logs, collections are stored, allowing users to look at their chat history and revisit previous conversations with maintained context (such as a specific network)
- Users require login authentication in order to access their profile on the web app

Requirements

- Build a chatbot that assists traffic engineers
- Create a pipeline for a full chatbot flow, including prompts, retrievers, and LLM models
- Implement retrieval-augmented generation using LangChain and an open-source LLM

Deliverables

- A full-stack web application for users to run the QA/Charbot, complete with the following features:
 - Responsive web UI
 - Hosted application on a dedicated web server
 - User authentication to store conversation history

Features

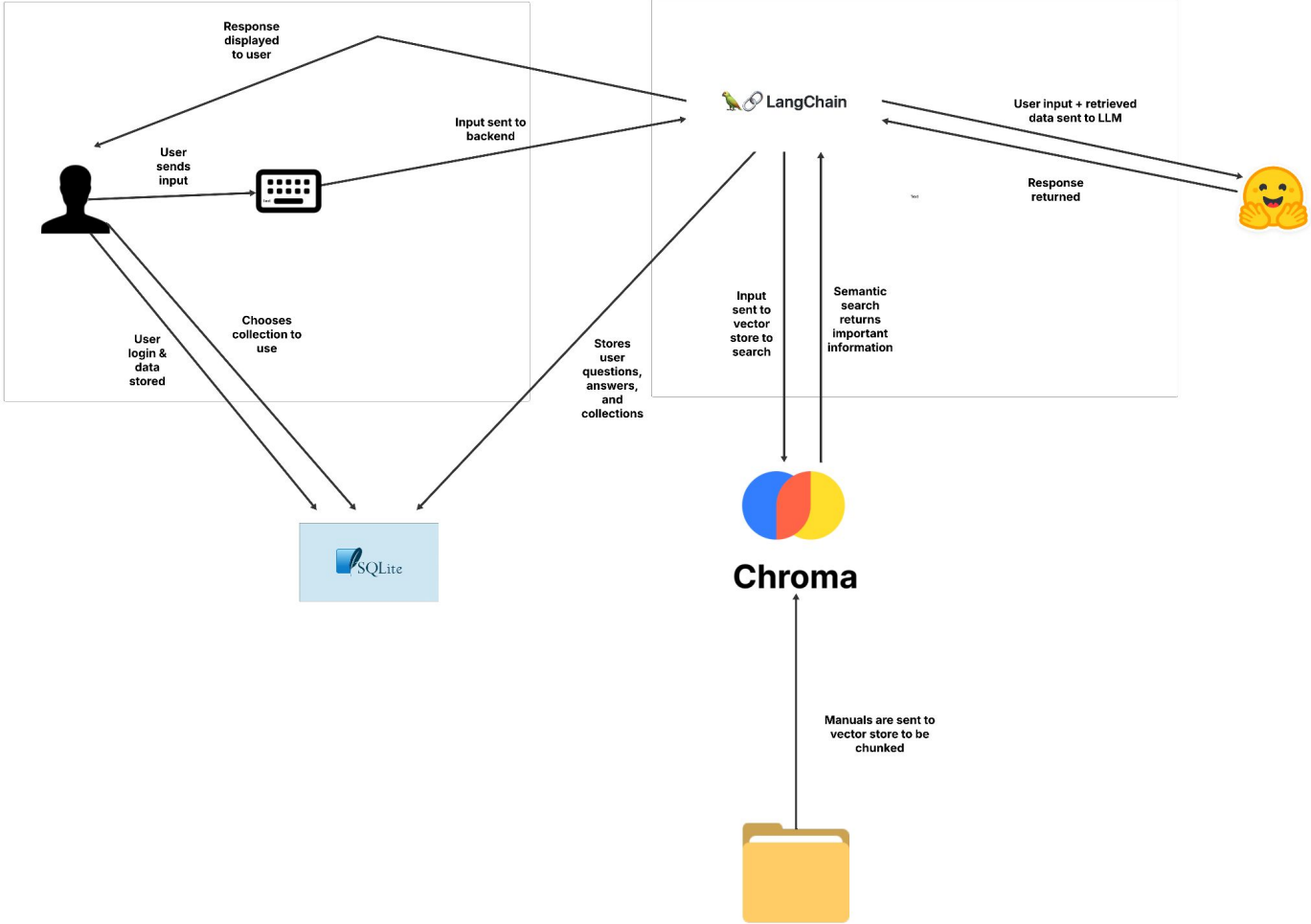
- Use ChromaDB to chunk, vectorize, and store data efficiently for the LLM
- Implement memory modules to allow follow ups/conversation

Approach/Plan

To ensure smooth progress, we will utilize an iterative process while implementing our project.

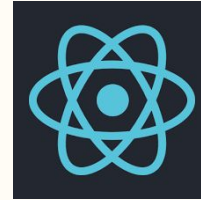
- Integrate LangChain with ChromaDB to retrieve data from the manuals
- Determine the most efficient LLM for our use case
- Build our backend to handle API requests for prompts
- Design a user-friendly UI for users to input questions
- Include authentication to save user information
- Add a memory module to store conversations and allow continuous conversations

Diagram for system architecture/design



Technologies and Frameworks

- **Frontend:** React, Tailwind CSS
- **Backend:**
 - **Flask:** For API and managing communication between frontend, database and LLM
 - **ChromaDB:** Store the manuals into chunks, and retrieve relevant chunks efficiently.
 - **SQLite:** Store user credentials and past conversation context.
 - **LangChain:** Integrates ChromaDB's relevant manual chunks, past conversation context from MongoDB with the LLM.
 - **Hugging Face Models:** Find a good model from hugging face to answer user questions.
- **Development & Testing Tools:**
 - **Google Colab:** Test different LLMs
 - **Postman:** Testing backend APIs
 - **Docker:** Containerize the application for easier deployment



LangChain + Chroma



Hugging Face



Flask



docker

Backend(LLM)

API Endpoint to query a response

```
@app.route("/api/chat", methods=["POST"])
def chat():
    if "user" not in session:
        return jsonify({"status": "error", "response": "Not authorized"}), 401
    try:
        data = request.get_json()
        print("Received request:", data, flush=True)
        user_message = data.get("message", "").strip()

        # Call the helper from llm.py to get the response from the LLM.
        llm_response = get_llm_response(user_message)

        return jsonify({"response": llm_response, "status": "success"})
    except Exception as e:
        print("Error in /api/chat:", str(e), flush=True)
        return jsonify({"response": f"Error processing your request: {str(e)}", "status": "error"}), 500
```

Take query and information from ChromaDB and get response from LLM

```
def get_llm_response(query: str, context = "") -> str:
    """
    Takes a user query string and returns the LLM's best answer
    using the already-initialized qa_chain.
    """
    from langchain.prompts import PromptTemplate
    prompt_template = """
    You are to act like a traffic simulation assistant.
    You will be given a question, previous chat history with the user, and information from a traffic simulation map.
    You need to analyze this information from the manual and answer the question asked.

    If the information provided isn't enough to answer the question asked then respond with "I don't know"

    Answer concisely.

    Context:
    {context}

    Question: {question}

    Answer: """"
    custom_prompt = PromptTemplate(
        input_variables=(context, query),
        template=prompt_template,
    )
```

Send user query into ChromaDB vector storage to get relevant information

```
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    return_source_documents=True,
    chain_type_kwargs={"prompt": custom_prompt}
)
try:
    result = qa_chain.invoke({"query": query})

    print("Chroma DB Retrieved Documents: -----")
    for doc in result["source_documents"]:
        print("Page Content:")
        print(doc.page_content)
        print("Metadata:")
        print(doc.metadata)
        print("-" * 20)
    print("Chroma DB Retrieved Documents End: -----")

    return result["result"]
except Exception as e:
    print("Error in get_llm_response:", str(e))
    return f"Error: {str(e)}"
```

Backend(Chapter Chunking)

Chunking The Manuals By Chapter

```
def getFirstManual():
    pdf_file = "./INTEGRATION_Manual_1.pdf"
    page_numbers = [1, 2, 4, 5, 8, 12, 30, 51, 59, 61]
    chapter_strings = extract_chapters_by_page_numbers(pdf_file, page_numbers)
    return chapter_strings

def getSecondManual():
    pdf_file = "./INTEGRATION_Manual_2.pdf"
    page_numbers = [1, 2, 4, 5, 8, 12, 30, 51, 59, 61]
    chapter_strings = extract_chapters_by_page_numbers(pdf_file, page_numbers)
    return chapter_strings

def getManualChunks():
    return getFirstManual() + getSecondManual()
```

Further Chunking Within Chapters

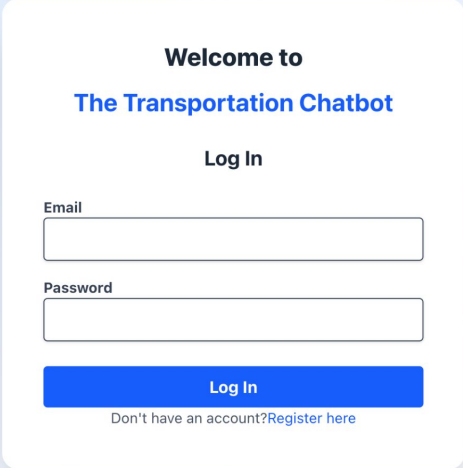
```
def getTextSplitted():
    from langchain.docstore.document import Document
    from langchain.text_splitter import RecursiveCharacterTextSplitter

    manual_chapters = getManualChunks()
    splitter = RecursiveCharacterTextSplitter(
        separators=['\n\n', '\n'],
        chunk_size=500,
        chunk_overlap=30
    )

    final_chunks = []
    for chapter in manual_chapters:
        doc = Document(page_content=chapter)
        chunks = splitter.split_documents([doc])
        final_chunks.extend(chunks)

    return final_chunks
```

Frontend



Welcome to
The Transportation Chatbot

Log In

Email

Password

Log In

[Don't have an account? Register here](#)

Frontend

Transportation Chatbot Search collections... [Logout](#)

Your Collections

Create New Collection

Collection name [Create](#) [Cancel](#)

traffic simulations 🗑️

simulation guide 🗑️

modeling approach 🗑️

Frontend

Create New
Collection

Existing
Collections

Transportation Chatbot

Search Collections

Search collections... [Logout](#)

Your Collections

Create New Collection

Collection name

traffic simulations

simulation guide

modeling approach

Logout

Frontend

The screenshot displays a web browser window at localhost:3000. The page is titled "Transportation Chatbot". On the left, a sidebar labeled "Collections" contains two items: "What is an O-D Numbe..." and "What is the master c...", along with a "+ New Collection" link. The main chat area features three messages in white boxes with rounded corners:

- Message 1: "An O-D Number is a number that represents the number of O-D pairs loaded by the model. The O-D Number is specified in line 2 of the O-D file."
- Message 2: "Michel Van Aerde is a professor of civil and environmental engineering at Queen's University in Kingston, Ontario. He has been working on traffic modeling for over 30 years and his work with INTEGRATION has advanced the field significantly."
- Message 3: "Transportation is the process of moving people or goods from one place to another. Transportation can be classified into different types based on the mode of transportation, such as road, rail, air, and water."

On the right side of the chat area, there are three blue buttons with white text: "What is an O-D Number?", "Who is Michel Van Aerde?", and "What is Transportation?". At the bottom left, there is a red "Logout" button. At the bottom right, there is a text input field with the placeholder "Ask me about traffic simulation..." and a blue "Send" button.

Frontend

User collections

localhost:3000

Collections

- What is an O-D Number...
- What is the master c...
- + New Collection

Transportation Chatbot

What is an O-D Number?

Who is Michel Van Aerde?

What is Transportation?

Logout

Ask me about traffic simulation... Send

Conversation with chatbot

Logout functionality

Users can submit questions

Database

Users Table

```
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS users (
        id TEXT PRIMARY KEY,
        email TEXT UNIQUE,
        password TEXT
    )
    """
)
```

Collections Table

```
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS collections (
        collection_id TEXT PRIMARY KEY,
        user_id TEXT,
        name TEXT,
        FOREIGN KEY(user_id) REFERENCES users(id)
    )
    """
)
```

Chat History Table

```
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS chat_history (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        collection_id TEXT,
        role TEXT,
        content TEXT,
        timestamp TEXT,
        FOREIGN KEY(collection_id) REFERENCES collections(collection_id)
    )
    """
)
```

Docker Containerization

Single docker file for both the backend and frontend

```
RUN apt-get update && apt-get install -y --no-install-recommends \  
    clang \  
    cmake \  
    git \  
    ninja-build \  
    vim \  
    build-essential \  
    curl \  
    sqlite3 \  
    libsqlite3-dev \  
&& rm -rf /var/lib/apt/lists/*  
  
# Copy over the Python requirements.  
COPY requirements.txt .  
  
# Install Python packages from requirements.txt.  
RUN pip install --no-cache-dir -r requirements.txt
```

User Manual

- Navigate to the URL of the project
- Create an account using email and password
- Upon account creation the user will automatically be logged in
- Create a new collection by clicking the “new collection” button on the top right
- Enter a collection by clicking on the corresponding box
- Ask the chatbot any questions regarding traffic simulations!

Developer Manual

- Clone the repository at <https://github.com/aarushpatil/Capstone>
- Build a docker image for the project (docker build -t chatbot ./)
- Run both the backend and frontend servers

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

- `cd frontend`
- `npm install`
- `npm run start`

Backend: `docker run -p 5050:5050 -v $(pwd):/app -it chatbot /bin/bash`

- `cd backend`
- `Python main.py`

Challenges We Faced

Chunking Data:

- Effectively chunking data by chapter to ensure accurate and in-depth responses
- Formatting information such as tables and charts

RAG Pipeline:

- Difficulty creating our ChromaDB and LangChain pipeline
- After running on Google Colab, there were issues in being able to run the LLMs locally due to dependency issues.
- Optimizing speed and accuracy of our model responses

Full-Stack Integration:

- Connecting our backend to our frontend via API endpoints

Improvements/Next Steps

Improve Response Accuracy:

- Evaluate various LLMs (Llama, Mistral, Zephyr, etc.) to determine the most optimal choice
- Fine-tune prompt template with more in depth context for better results
- Chunk data by chapters rather than fixed-sized

Speed Up Query Time:

- Utilize quantized models to reduce latency
- Optimize ChromaDB for quicker indexing

Timeline and milestones

- **Milestone 1:**

- Gather project requirements, research NLP tools and RAG frameworks, and create an initial presentation
- Set up a basic LLM test in Google Colab to validate the planned workflow

- **Milestone 2:**

- Process and chunk data for efficient retrieval, choose an embedding model, and implement ChromaDB as a vector store
- Run initial tests to evaluate the retriever's performance

- **Milestone 3:**

- Select the best-performing Hugging Face LLM through comparative testing
- Integrate the model into a LangChain pipeline with ChromaDB retrieval
- Build the full RAG pipeline to generate responses from user queries

- **Milestone 4:**

- Develop a SQLite backend to store and retrieve user chat history
- Create a React-based frontend with chat and session management
- Conduct user testing to improve interface usability and navigation

- **Milestone 5:**

- Containerize the full stack with Docker and prepare for deployment
- Run system-wide tests for backend, frontend, and overall chatbot performance
- Conduct user testing to improve interface usability and navigation
- Analyze test results and deploy final build
- Prepare final documentation and demo materials for project delivery

Roles, who is doing what



Aneesh:
Implementation, Frontend



Ethan:
LangChain/LLM/Database



Neil:
Implementation, Backend/API



Rami:
Implementation, Frontend



Aarush:
Implementation,
Backend/LLM

Client

Dr. Farag

- Ph.D. in computer science from Virginia Tech
- Research Associate for The Center for Sustainable Mobility
- Interests include intelligent transportation systems, connected/automated vehicles, C-V2X, machine learning, large-scale data analysis, large-scale system analysis and design, big data, and information retrieval



References

- <https://colab.research.google.com/drive/1Zs70RvMVBnn4VFUfS-Rst4dG0fojnEaT>
(Yusuf Elnady)
- Dr. Farag