Carnegie Mellon University

Capstone Final Presentation

Nexteer (AI Bots) Team

Faculty Adviser: Ryan Smalley

Introduction of the Team

Project Manager

Jacqueline Hsu

Database Management, NLP, Leadership

Praneet Yavagal

SDE, AI Modeling

Nachiketa Hebbar

AI Development, RAG, LLMs

Financial Manager

Rugved Somwanshi

SDE, LLMs, UI/UX AI Development

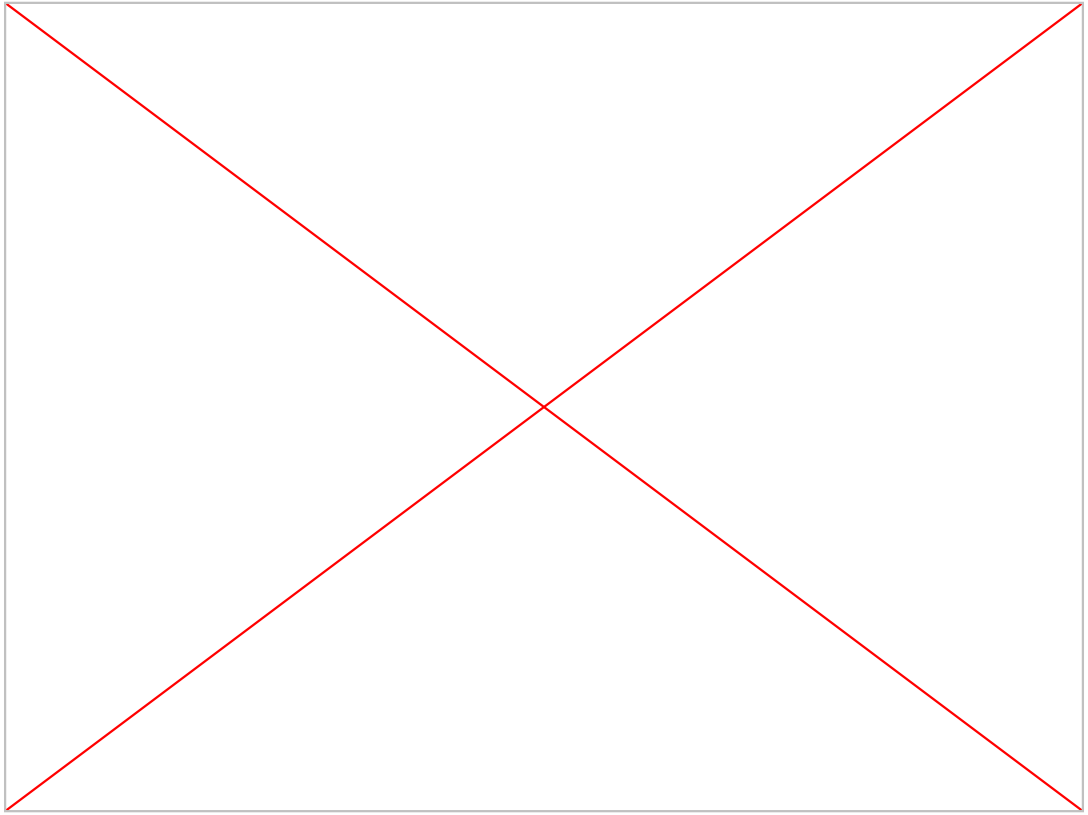
Khushali Daga

Deep Learning, NLP, RAG, LLMs

Louis Leng

Full-stack Development

Creative Video



Carnegie Mellon University

Agenda

01 Background and Overview of Project

02 Technical Deep Dive

03 Implementation

04 Documentation

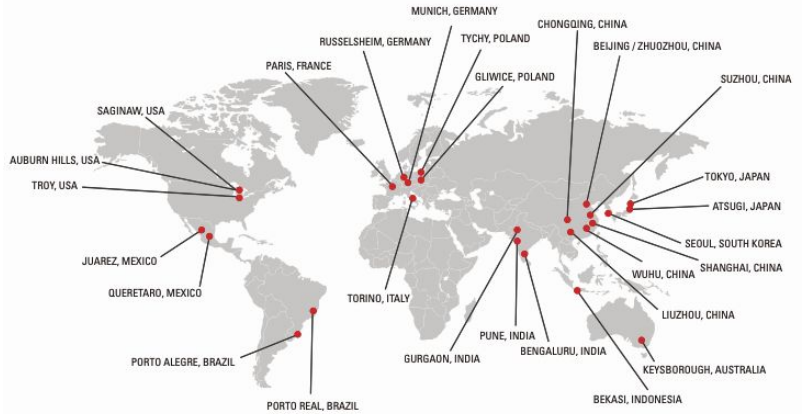
05 Competitor Analysis

06 Wrapping Up

Carnegie Mellon University

01 Background and Overview of Project

Nexteer Automotive



- Global leading motion control technology company that specializes in automotive steering and driveline systems
- Aims to accelerate mobility that is safe, green, and exciting through innovative motion control technologies

Carnegie Mellon University

Problem Statement

Nexteer's internal organization currently has multiple individual chatbots that each take care of a particular domain.

Pain points:

- Current chatbots lack the capability to accurately identify the domain of a question
- Query is not always redirected to the appropriate bot/expert
- Inefficiencies in process
- Decreased user satisfaction

Carnegie Mellon University

Objectives/Outcomes

- Develop a chatbot dispatcher that can streamline the process of directing queries to the most appropriate chatbot based on their skills and access
  - Metadata structure
  - Understanding the domain of user queries
  - Determine which agent is best suited to provide an answer
- Our solution will work alongside existing chatbots — not replacing them

Carnegie Mellon University

Deliverables

Proof of Concept

A redirection logic that Nexteer can implement for their own chatbot

Research that suggest our chosen methods are the most appropriate

Items

Code (model tests, redirection logic, chatbot dispatcher)

Documentation (reports, code comments, README.md)

Presentations slides (weekly meetings, midterm, final)

Recording (final presentation)

Carnegie Mellon University

Business Impact

“Saving money, time, and increasing efficiency”

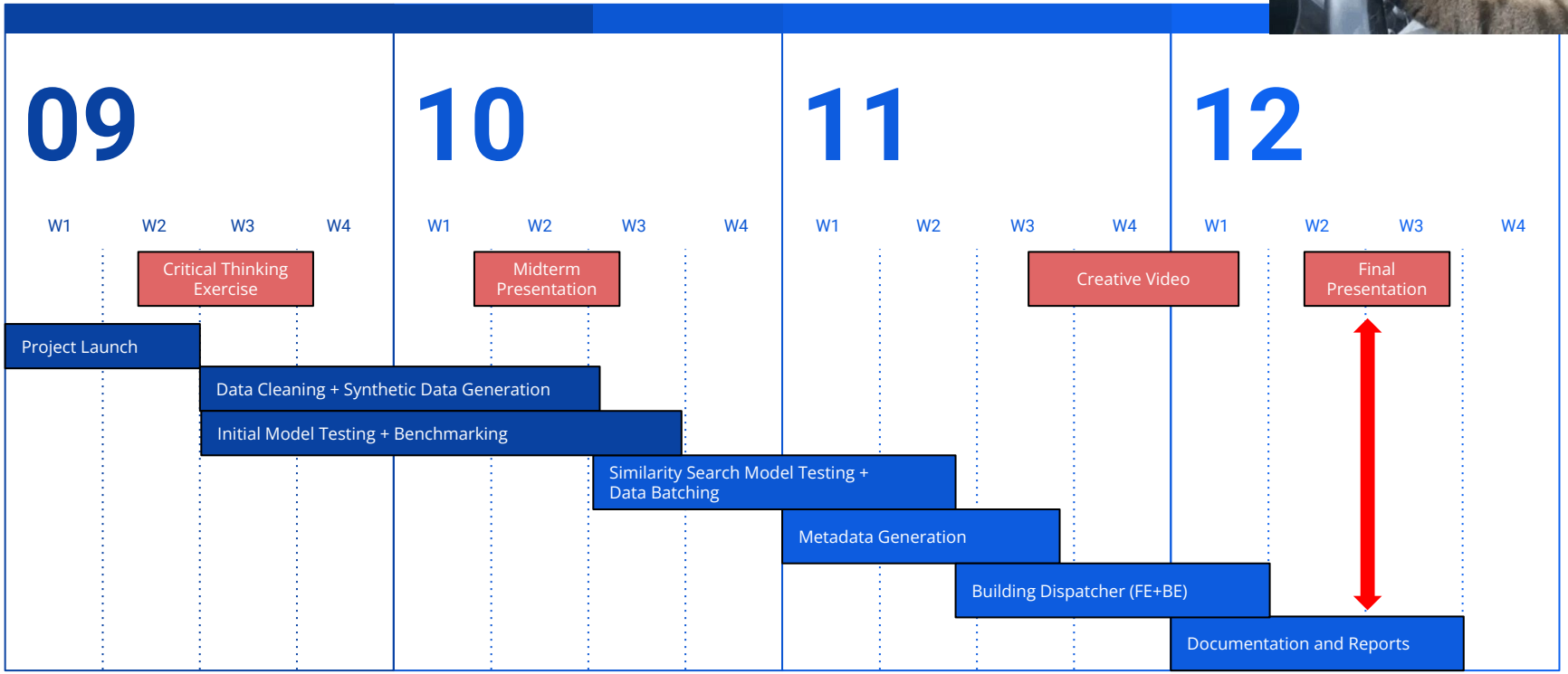
Research with different approaches —> **Saved time and money** for the company to hire people to do the work

Combining multiple functions into one bot —> **Shortened time** to knowledge/information

Experimented with different LLM models —> **Saved computational cost** depending on the scale of use (no. of users)

Carnegie Mellon University

Project Schedule



02
Technical Deep Dive

Academic Autopsy: Midterm Recap

- Initial Database Schema Design
- Synthetic Data Generation
- Model Benchmarking

Model	Accuracy	Inference Time(seconds)
Gemini	97.3%	3.08
GPT(4o-mini)	**96.7%**	**0.67**

Carnegie Mellon University

Technical Deep Dive

2.a Metadata Structure

Metadata Structure

Carnegie Mellon University

Example: Organizational Information Bot

Description

- Bot provides details about company hierarchy, roles, and operations.
- It can answer questions related to organizational structure, such as identifying department managers or understanding department functions.

Capabilities

- Retrieve organizational role and hierarchy information
- Identify departmental heads and management structure
- Access details on departmental functions and responsibilities
- Provide contacts for department-specific inquiries

Keywords

- Department hierarchy
- Organizational roles
- Department manager
- Management structure
- Organizational contacts
- Department functions
- Role responsibilities
- Leadership structure
- Employee roles
- Organizational chart
- Interdepartmental communication

Why Schema Update is Necessary

- Limited Insight into Bot Functionality
- Inefficient Bot Matching
- Scalability and Flexibility Challenges

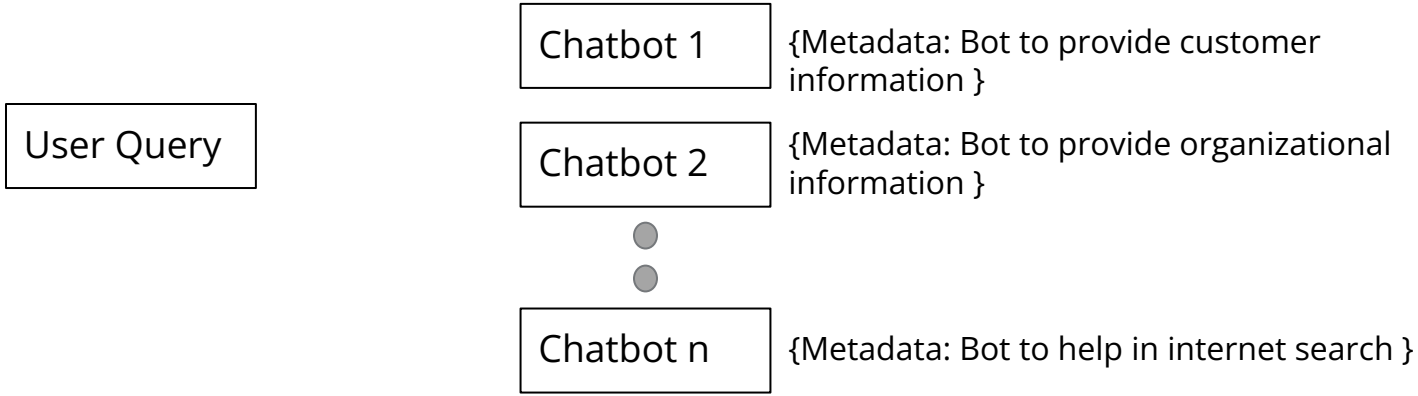
Carnegie Mellon University

2.b Metadata Generation

Auto Metadata Generation

Problem:

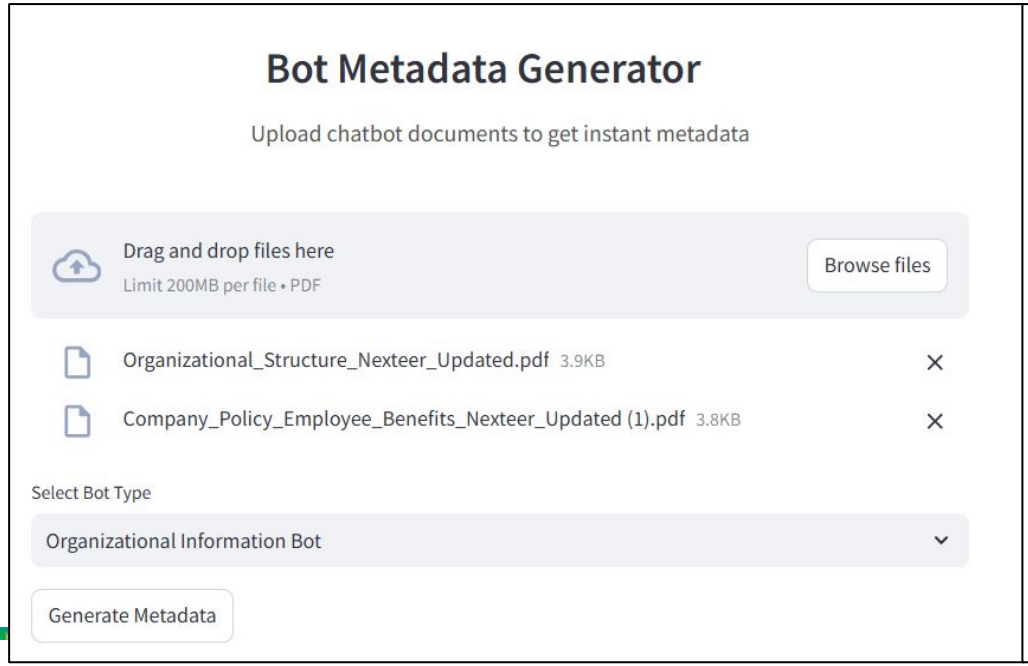
- Chatbots are selected for routing based on their metadata.
- As chatbots increase(>100), manually writing metadata for each chatbot is not feasible.

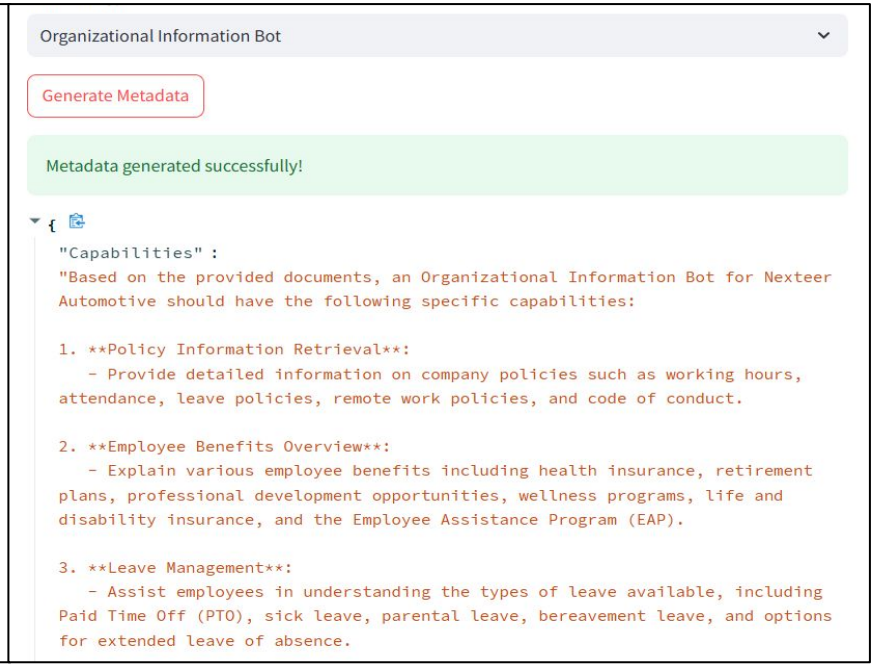


Carnegie Mellon University

Auto Metadata Generation

Solution: Built a web-based tool that allows uploading chatbot documents and generating instant metadata on demand.



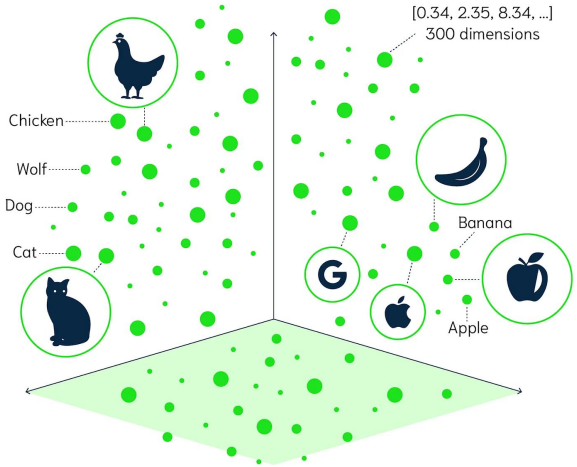


2.c Similarity Search

Similarity Search - Comparison & Motivation

Key Motivations:

1. Contextual Matching: Identifies semantically relevant results
2. Efficiency: Faster than full semantic search
3. Scalability: Handles large datasets effectively
4. User-Centric: Adapts to varied user queries



Example:

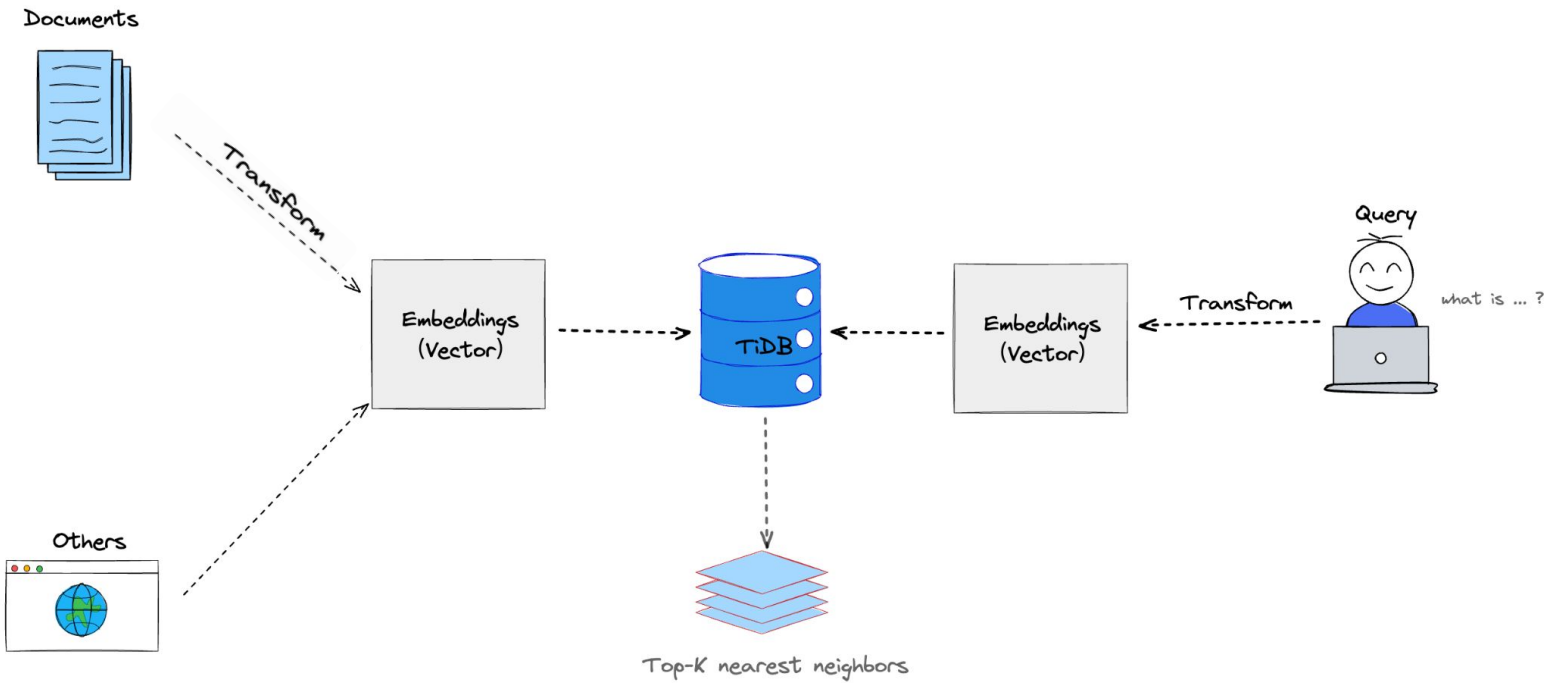
1. Query: "affordable laptops"
2. Lexical Search: Misses "budget computers"
3. Similarity Search: Captures semantic relationships, ensuring relevant matches

Carnegie Mellon University

Similarity Search - Comparison & Motivation

Method	Strengths	Limitations
Lexical Search	Fast, precise for exact matches	Misses context, synonyms
Semantic Search	Understands intent and context	Computationally expensive
Similarity Search	Balances accuracy and efficiency	Depends on embedding quality

Similarity Search - Implementation



Carnegie Mellon University

Similarity Search - Pros and Cons

PROS

1. Semantic Understanding
2. Versatility
3. Scalability
4. Personalization
5. Dimensionality Handling

CONS

1. Implementation Complexity
2. Embedding Quality Dependence
3. Interpretability Challenges
4. Sensitivity to Noise
5. Privacy Concerns

Similarity Search - Improvements

1. **Contextual Retrieval:** Incorporate context-aware embeddings

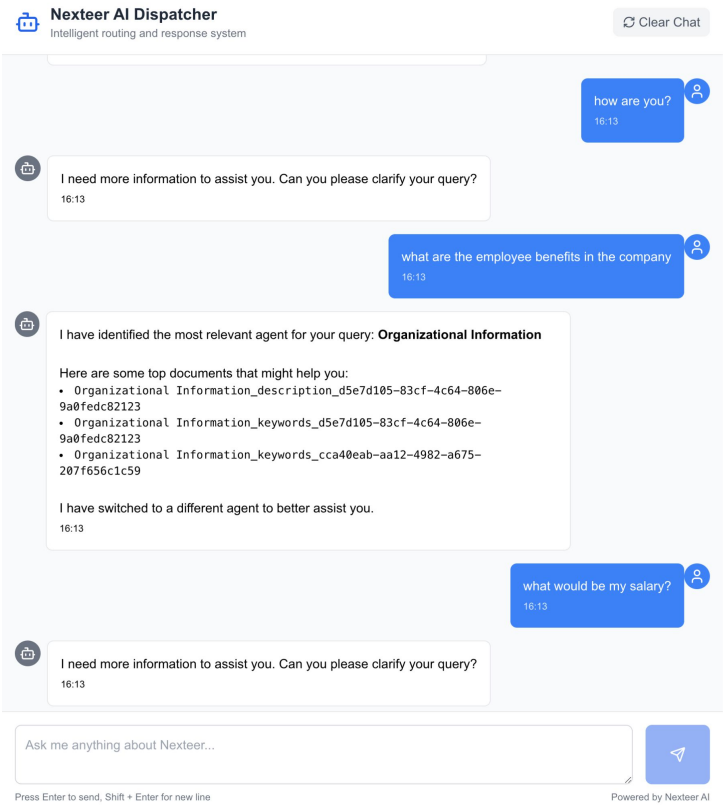
2. **Multi-Objective Search:** Consider multiple constraints (relevance, diversity)

3. **Hardware Acceleration:** Leverage GPUs/TPUs for faster computation

03
Implementation

Frontend

- Frontend Architecture
  - Modern chat interface built with Next.js
  - Real-time response handling
- Technical Implementation
  - TypeScript for type safety
  - Tailwind CSS for styling
- Concurrency Support



Carnegie Mellon University

Backend

- Built using **Python** with **FastAPI**
- Provides **REST API**
- Uses **LMStudio** for local LLM inference which supports OpenAI API standards
- Modular in nature
- Uses **SQLLite** and **ChromaDB** for storing metadata and vectors respectively.

Backend

```python
def check_if_current_agent_can_answer(self, query) -> bool:
    """
    Check if the current agent can answer the query.
    This will return True if the current agent can answer the query
    and False if the current agent cannot answer the query.
    """
    self.add_system_message(
        f"""Determine if the question needs a redirection to another agent or the current agent is capable of answering it.
            If the current agent is capable of answering it, then proceed with the current agent.
            Usually, internet_search is not the answer and try to use more of the specialized agents which we have
            The current agent is {str(self.current_agent)}. Here are the other agents for your context. {str(self.agents)} - DO NOT USE THESE NAMES IN THE RESPONSE.
            Usually, if the current agent is a specialized agent, then it is better to proceed with the current agent.
            So for example, if the current agent is 'agent1', then the response should be 'agent1' or '$OTHER_AGENT' to switch to another agent.
            ONLY ANSWER WITH THE CURRENT AGENT NAME OR $OTHER_AGENT. DO NOT ANSWER WITH ANY OTHER AGENT NAME. Be intelligent and think if the current agent can answer the question or not.
            BE VERY STRICT AND CAREFUL THINKING IF THE CURRENT AGENT CAN ANSWER THE QUESTION OR NOT. IT IS OKAY TO SWITCH IF IN DOUBT.
    """,
    )
    self.add_human_message(query)
    res = self.llm.invoke(self.conversation_history)
    json_res = res
```

FastAPI 0.1.0 OAS 3.1

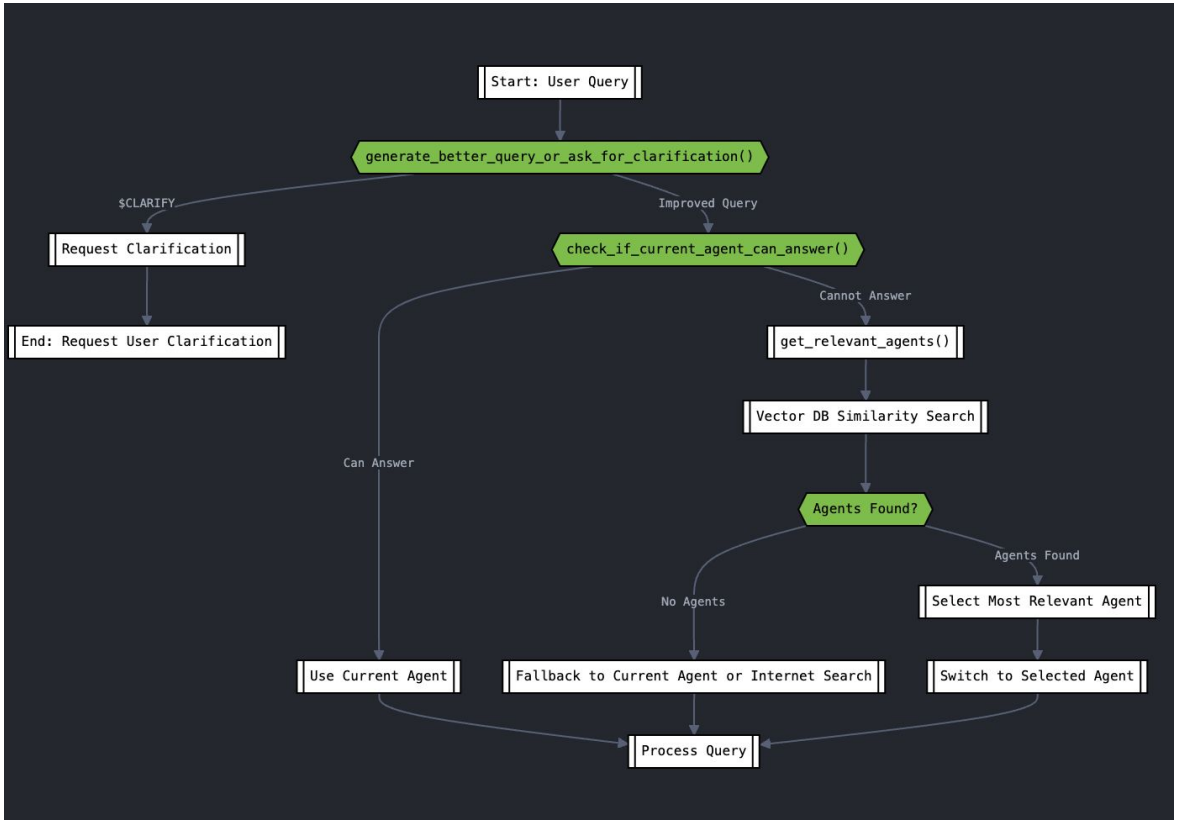
/openapi.json

default

GET	/	Read Root
POST	/metadata	Extract Metadata
GET	/agents	Get Similar Agents

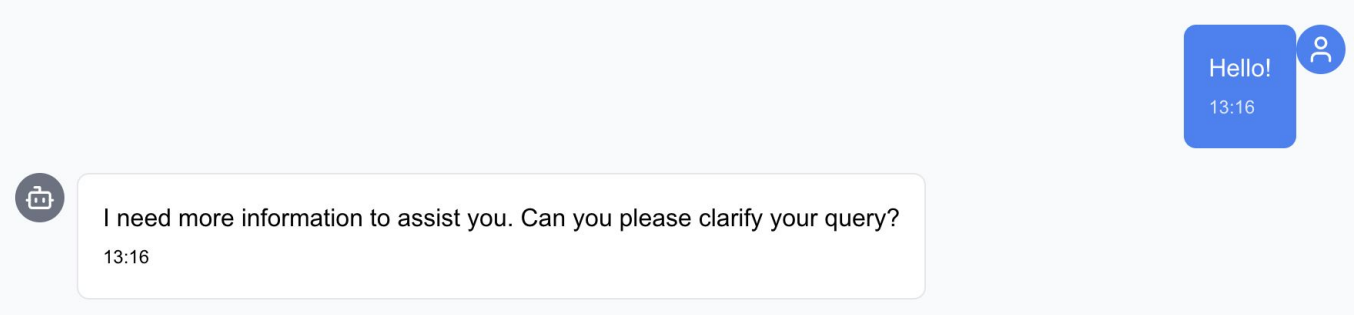
Carnegie Mellon University

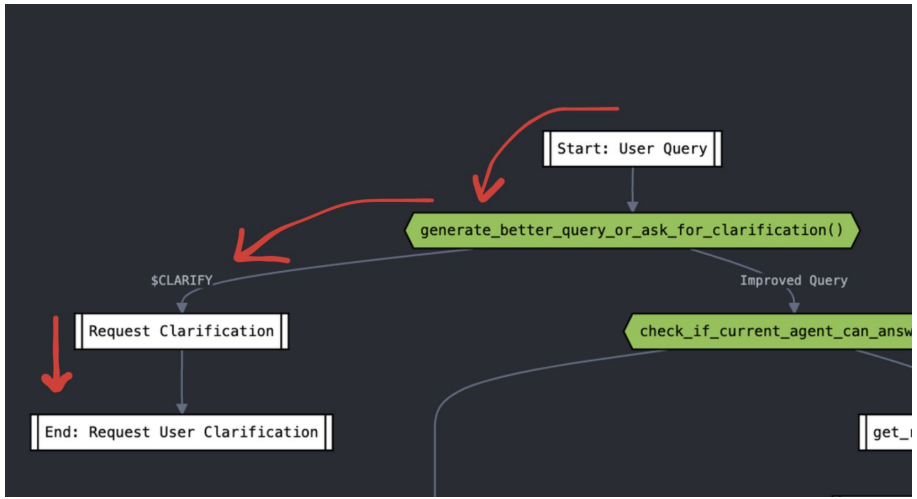
Backend Flow Diagram



Carnegie Mellon University

Case I: Generic/Gibberish Query





Carnegie Mellon University

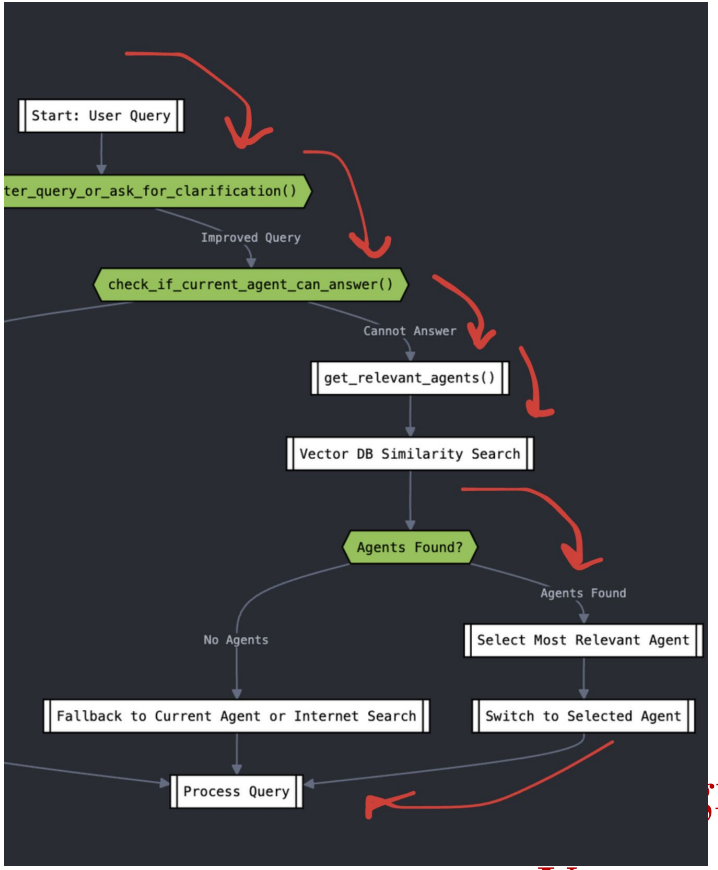
Case II: Switch to an appropriate agent

Tell me about the customer acquisition costs of nexteer
13:16

I have identified the most relevant agent for your query: **Customer Database Search**

Here are some top documents that might help you:
- Customer Database Search_description_66fb7937-c143-40e9-a709-d3c55b449918
- Customer Database Search_keywords_66fb7937-c143-40e9-a709-d3c55b449918
- Customer Database Search_capabilities_66fb7937-c143-40e9-a709-d3c55b449918

I have switched to a different agent to better assist you.
13:16



Comparison Between Models

Model	Check if the user query is good enough	Check if the current agent can answer well	Get the most relevant agent from similarity search	Speed	Costs (Dollars per 1M tokens)
Qwen Coder 2.5 7B	✅	✅	✅	⚡⚡⚡	0.3
Qwen 2.5 Instruct 14B	✅	✅	✅	⚡⚡	0.4
Llama 3.2 3B	✅	⚠️	✅	⚡⚡⚡⚡	0.06
Llama 3.2 1B	⚠️	❌	⚠️	⚡⚡⚡⚡	0.06
Open AI 4o mini	✅	✅	✅	⚡⚡	0.6

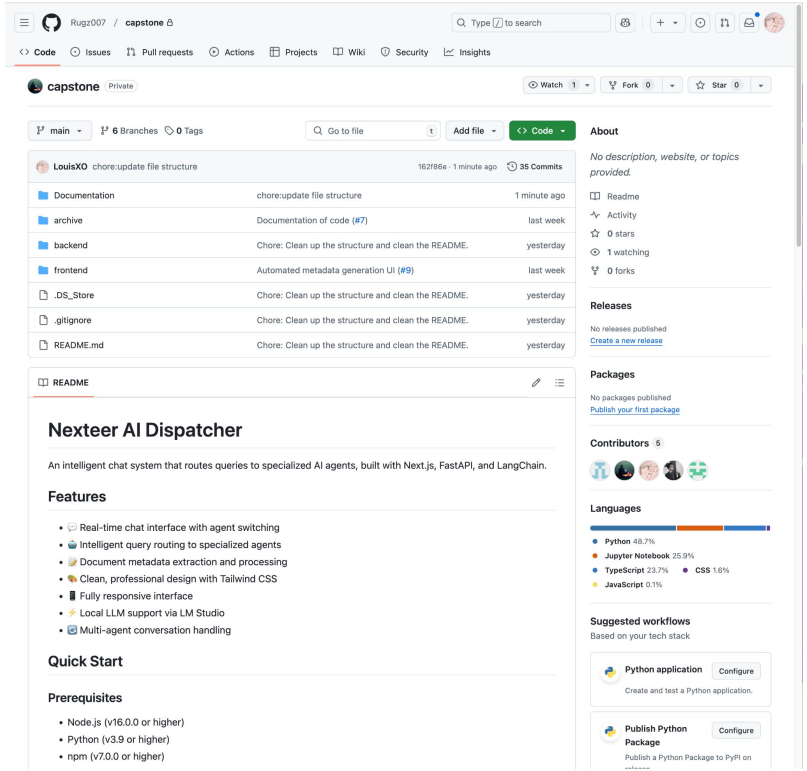
https://www.together.ai/pricing
https://artificialanalysis.ai/

Carnegie Mellon University

04
Documentation

GitHub Repo

- The Nexteer AI Dispatcher codebase is organized into three main components:
  - Frontend
  - Backend
  - Documentation



Carnegie Mellon University

Set Up

Frontend Setup

```
# Navigate to frontend directory
cd frontend

# Install dependencies
npm install

# Start development server
npm run dev
```

Backend Setup

```
# Navigate to backend directory
cd backend

# Create virtual environment
python -m venv venv

# Activate virtual environment
# On Windows:
venv\Scripts\activate
# On Unix or MacOS:
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Initialize vector store database
python -m database.init_db

# Start the server
uvicorn app:app --reload
```

Development Commands

Frontend

- `npm run dev` - Start development server
- `npm run build` - Build production bundle
- `npm run start` - Start production server
- `npm run lint` - Run ESLint

Backend

- `uvicorn main:app --reload` - Start development server
- `python -m pytest` - Run tests

Carnegie Mellon University

Metadata Generation Tool

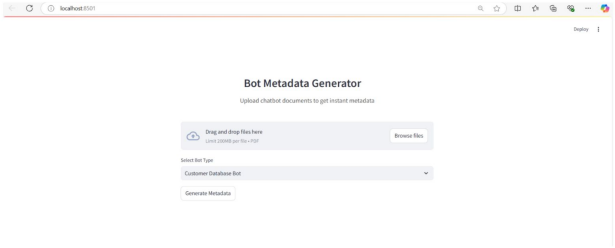
Auto Metadata Generation Web Tool Documentation

Overview

The Metadata Web App Tool is a user-friendly application that allows you to upload PDF documents containing chatbot-related information and automatically generates aggregated metadata for specific chatbot types. The generated metadata includes capabilities, descriptions, and specialization keywords, which can be downloaded in JSON format.

How to Use the Metadata Web App Tool

Step 1: Accessing the Tool

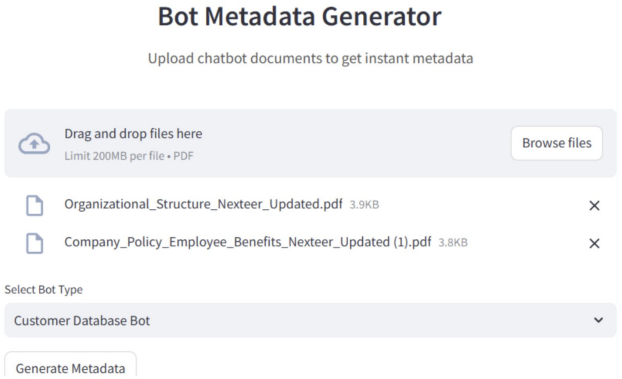
1. Open the web app in your browser by running the command “streamlit run python_webapp.py”
2. You will see the main interface with a clean and minimal design, featuring an upload area, bot type selector, and a "Generate Metadata" button.
3. 

Step 2: Uploading Documents

1. In the "Upload PDF files" section:
   - Drag and drop one or more PDF files into the upload box.
   - Alternatively, click the Browse files button to select files from your computer.
2. Ensure that the uploaded documents are in the PDF format and contain relevant content about the chatbot.

---

2. If you select Custom Bot, a text input field will appear for you to manually enter the bot's name.



Step 4: Generating Metadata

1. Click the Generate Metadata button to process the uploaded documents and generate metadata.
2. The app will extract text from the uploaded PDFs and call an AI model to generate metadata fields:
   - Capabilities: Lists specific functionalities the bot should have.
   - Description: Summarizes the bot's purpose and scope.
   - Specialization Keywords: Extracts keywords describing the bot's expertise and focus.

Carnegie Mellon University

05 Competitor Analysis

OpenAI Swarm Framework Overview

Structure:

- Multi-agent system with lightweight collaboration
- Agents are dynamically assigned tasks based on specialization
- Context is retained throughout agent transitions

Highlights:

- Efficient agent collaboration
- Seamless handoffs between agents for complex tasks

Carnegie Mellon University

IntelliBot Pro Framework

Components:

- RAG Bots
- API Communication
- Internet Search Bot

Key Features:

- Real-time document updates
- Automated metadata generation
- Optimized accuracy and efficiency

Carnegie Mellon University

OpenAI Swarm vs. IntelliBot Pro

Feature	Swarm	IntelliBot Pro
Purpose	General task resolution	Enterprise triage resolution
Agent Specialization	Lightweight, general-purpose	Domain-specific expertise
Fallback Mechanism	Absent	Internet Search
Real-Time Updates	Limited	Dynamic document updates

06
Wrapping Up

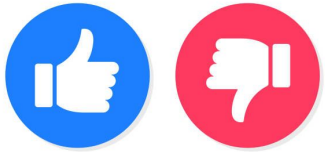
Lessons Learned

This project exposed us to the realities and challenges of operationalizing AI systems.

Challenge	Solution
Lack of clean data for testing	Generated synthetic data
Ambiguous user queries	Added clarifying question mechanisms
Handling scalability for more users	Adopted low-latency vector similarity search
Integration with legacy systems	Built modular APIs

Next Steps for Nexteer

1. **Integrate Feedback Loop for Monitoring**



   a. **Why:** Explicit user feedback to get alerted when dispatcher performance drops
   b. **How:** Implement scoring systems for users to rate chatbot response

2. **Add safeguards and guardrails for user safety**

   a. **Why:** LLMs have a tendency to hallucinate or produce biased responses in some cases
   b. **How:** Add user response filtering, and an additional specialized language model to detect potentially harmful or irrelevant responses

Carnegie Mellon University

Closing Thoughts

September - December, 4 months, 30+ meetings

Thank you, audience

Thank you, Nexteer

Thank you, Krzysztof

Thank you, Ryan

Thank you, Team!!!!

Carnegie Mellon University

Thank you for listening!

- Team Nexteer AI Bots

Q&A Time

Carnegie Mellon University