Jinyi Wang

DSCI 551 ChatDB 81

ChatDB: Movie Querying with ChatGPT and MongoDB

https://github.com/jinyi415/ChatGPT-demo.git

1. Introduction

With the rapid advancements in large language models (LLMs), natural language interfaces (NLIs) have become an increasingly viable solution for bridging the gap between human users and database systems. Traditional database interaction requires knowledge of formal query languages, posing a barrier for many users. To address this, ChatDB: Movie Querying with ChatGPT and MongoDB provides a user-friendly interface that allows users to query and manipulate movie-related data using plain English.

This project focuses on building a command-line-based NLI system that connects natural language inputs to MongoDB operations through the ChatGPT API. Users can perform a variety of tasks, such as retrieving top-rated movies, filtering by genre or release year, and even modifying data (e.g., inserting, updating, or deleting movie records). The underlying movie data is sourced dynamically from the TMDb (The Movie Database) API and stored in MongoDB collections.

By integrating ChatGPT for natural language understanding and MongoDB for flexible document storage, ChatDB demonstrates how conversational systems can simplify database access. The system supports complex queries, including aggregation and cross-collection joins using $lookup, and aims to provide robust and explainable query results. This project serves as a proof of concept for how AI-powered interfaces can enhance usability and accessibility in database-driven applications.

2. Planned Implementation (From Project Proposal)

The development of ChatDB was structured around three core components: data ingestion and storage, LLM-powered natural language query translation, and CLI-based user interaction. TMDb movie data was retrieved via API and stored in MongoDB, organized into collections such as movies, genres, and directors. This schema enables not only simple filtering queries but also $lookup operations that simulate joins across collections.

To interpret user input, the system integrates the ChatGPT API, which parses natural language queries and converts them into valid MongoDB operations. This includes basic find() queries with filters, sorting, and projection, as well as more advanced aggregations using $match, $group, and $sort. Data modification is also supported, allowing users to insert new records, update existing ones, or delete entries — all through conversational commands.

User interaction is conducted through a command-line interface (CLI), which facilitates real-time engagement with the database and displays results in a structured format. To improve efficiency and reduce repeated calls to external APIs (e.g., TMDb or ChatGPT), a local caching mechanism was implemented.

A breakdown of the key implementation components, their objectives, and current completion status is shown in the table below:

Pillar	Objective	Current Status
Data ingestion & indexing	Fetch movie data from TMDb, normalize fields (title, genre, year, director), and index in MongoDB for efficient querying	Delivered
ChatGPT-powered query translation	Use ChatGPT API to parse natural language and generate MongoDB queries for filtering, sorting, and joining	Delivered
CLI-based interaction	Build a command-line interface allowing users to query and modify movie records through natural language	Delivered
Data modification (CRUD)	Support insert, update, and delete operations using natural language commands	Delivered
Caching & optimization	Implement local cache to reduce repeated API calls and improve response time	Delivered

3. Architecture Design (Flow Diagram and its Description)

The system architecture of ChatDB follows a linear yet modular design, as illustrated in Figure 1. The process begins with the User Interface, implemented as a command-line interface (CLI). This interface serves as the entry point for all user interactions, allowing natural language inputs such as "Find top-rated comedies released after 2020" or "Delete all Spider-Man movies."

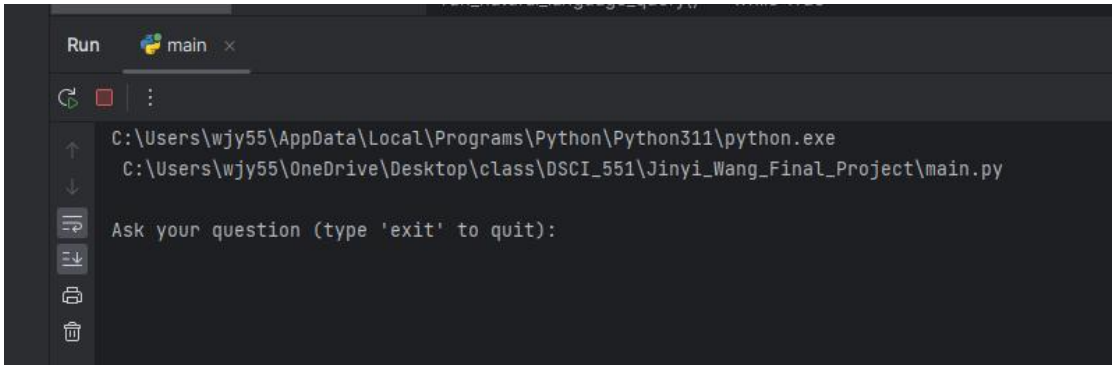

Figure 1. The process begins with the User Interface

Once a query is submitted, it is first processed through a System Prompt layer, where the user input is embedded into a predefined prompt template. This prompt is then sent to the ChatGPT API, which interprets the natural language and returns a structured MongoDB-style JSON query.

The interpreted query is passed into a Query Builder module. This module not only formats the query for MongoDB execution but also interacts with a Cache system. If a similar query has been processed recently, the system attempts to retrieve the result or the structured query from the cache to reduce latency and API costs. Otherwise, the newly generated query is stored in the cache for future reuse.

The final query is executed against the MongoDB database, which contains collections such as movies, genres, and directors. These collections support both simple filtering and complex operations like $lookup joins and aggregations.

If the requested data is not found in MongoDB, the system triggers the External Data Integration module, which queries the TMDb API. The retrieved data is formatted and inserted back into MongoDB, ensuring that subsequent queries for the same content can be resolved locally without repeating the external request.

This architecture ensures extensibility, efficiency, and robustness in handling both common and unseen movie-related queries.

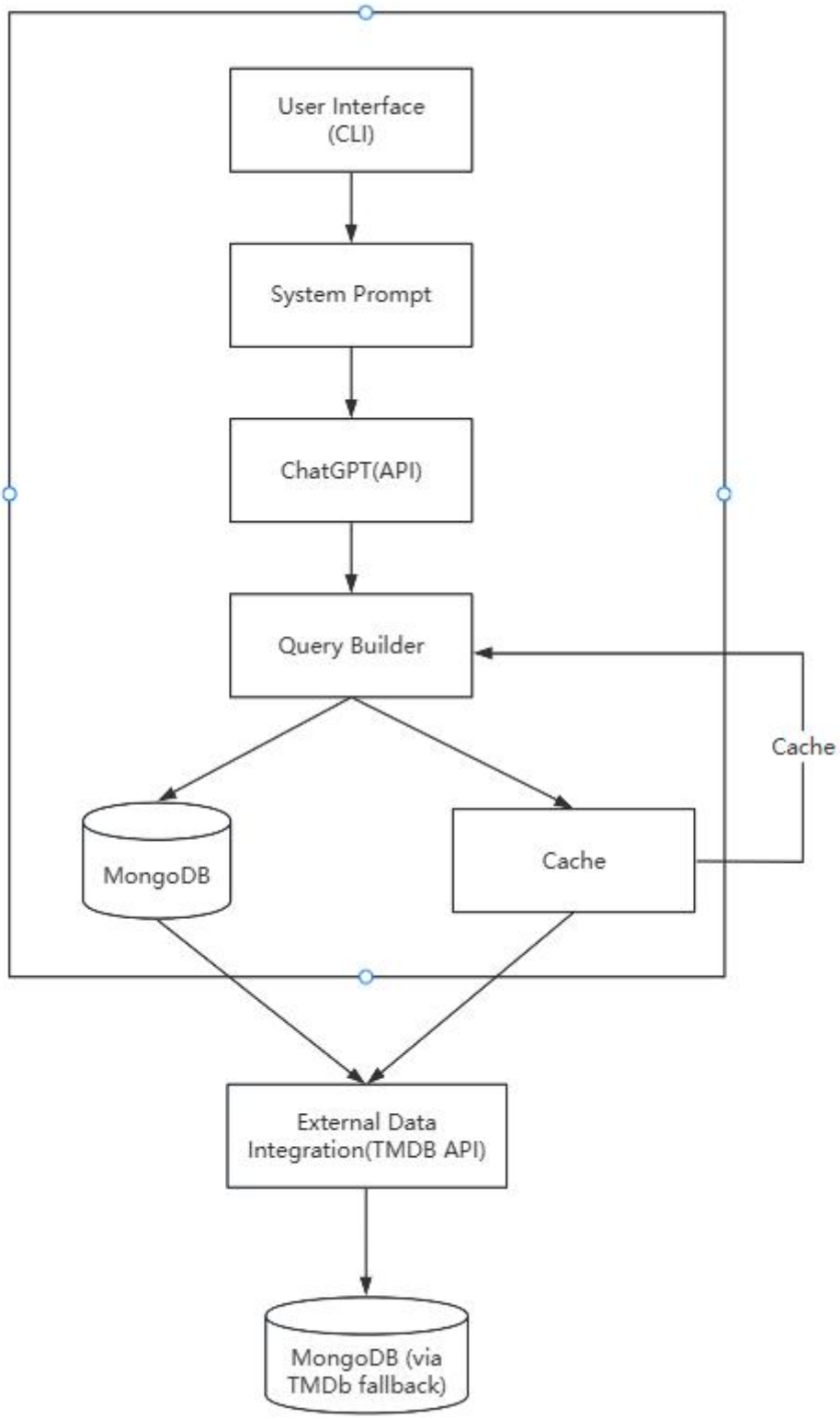

Figure 2. System Architecture of ChatDB: From Natural Language Input to MongoDB Execution and Fallback Handling

4. Implementation

4.1 Functionalities

ChatDB supports a wide range of natural language interactions with a MongoDB-based movie database, allowing users to explore, query, and modify data without writing any query language. The database consists of four core collections — user, movie, watchlist, and watched_history—which together support both content-level and user-level operations.

The first core functionality is schema exploration. When users input prompts such as “What collections are in the database?” or “Show me a few sample movie entries,” the system responds by listing available collections or displaying representative documents. This feature enables users to understand the structure and contents of the database without requiring any prior technical knowledge.

The second major functionality is query execution. The system processes user input by sending it to the ChatGPT API, which returns a structured JSON object representing a MongoDB query. This query is executed against the appropriate collection, and the results are displayed in a readable format. The system supports complex filtering, sorting, and even aggregation logic. For example, when a user requests “Find 3 best action movies after 2015 directed by Christopher Nolan,” the system extracts genre, year, rating, and director from the prompt, then builds a compound query accordingly. Cross-collection operations are also supported using MongoDB’s $lookup operator. In a query such as “Show me the watched history of user Jinyi,” the system joins the watched_history and movie collections to return a detailed, merged result that includes both metadata and personal activity.

The third core functionality is data modification. The system can insert, update, or delete records based entirely on natural language input. For instance, a command like “Add a new movie called Inception from 2010 directed by Christopher Nolan” will result in a new document being added to the movie collection. A command like “Update the genre of Inception to Science Fiction” triggers an update operation on the corresponding field. Similarly, a prompt such as “Delete all Spider-Man movies” removes matching documents from the database. Each of these operations is interpreted by ChatGPT, converted into MongoDB operations, executed with basic validation, and followed by confirmation messages to ensure transparency and correctness.

To improve performance and reduce API consumption, ChatDB incorporates a caching

mechanism. Both previous ChatGPT query results and TMDb movie metadata are stored locally. When similar or repeated requests are made, the system first checks its local cache before issuing external API calls. This optimization significantly reduces latency and improves cost efficiency.

4.2 Tech Stack

ChatDB is implemented entirely in Python. MongoDB serves as the backend database and is accessed through the PyMongo library. Natural language queries are interpreted using OpenAI's GPT-4 model via the ChatGPT API. To enhance the completeness of the movie dataset, the system integrates with the TMDb API. When a movie is not found in the local database, the system queries TMDb for the missing information, stores the result in MongoDB, and returns it to the user seamlessly.

Caching is handled using Python's in-memory dictionaries, with persistence achieved through the pickle module, allowing data to be saved and restored across sessions. Development and testing were carried out in Jupyter Notebook and PyCharm, while API testing was conducted using Postman. The user interface is implemented as a command-line loop, where users can continuously input natural language queries and receive structured, formatted responses. The system's modular design also makes it easy to extend functionality or adapt it for new use cases in the future.

4.3 Implementation Screenshots

The screenshots below demonstrate core interactions within ChatDB. One example shows the user entering the query "Find 3 best action movies after 2015." The system processes the request, constructs a MongoDB query, and returns a sorted list of relevant movie documents from the movie collection. Another example demonstrates a join query, where a user asks for their watched history. The system uses a $lookup operation between the watched_history and movie collections to return complete movie details, including titles, genres, and watch dates. A final example shows data modification: the user first inserts a new movie entry using natural language, then deletes all entries matching a given keyword. The system confirms both operations and reflects the changes in the database in real time.

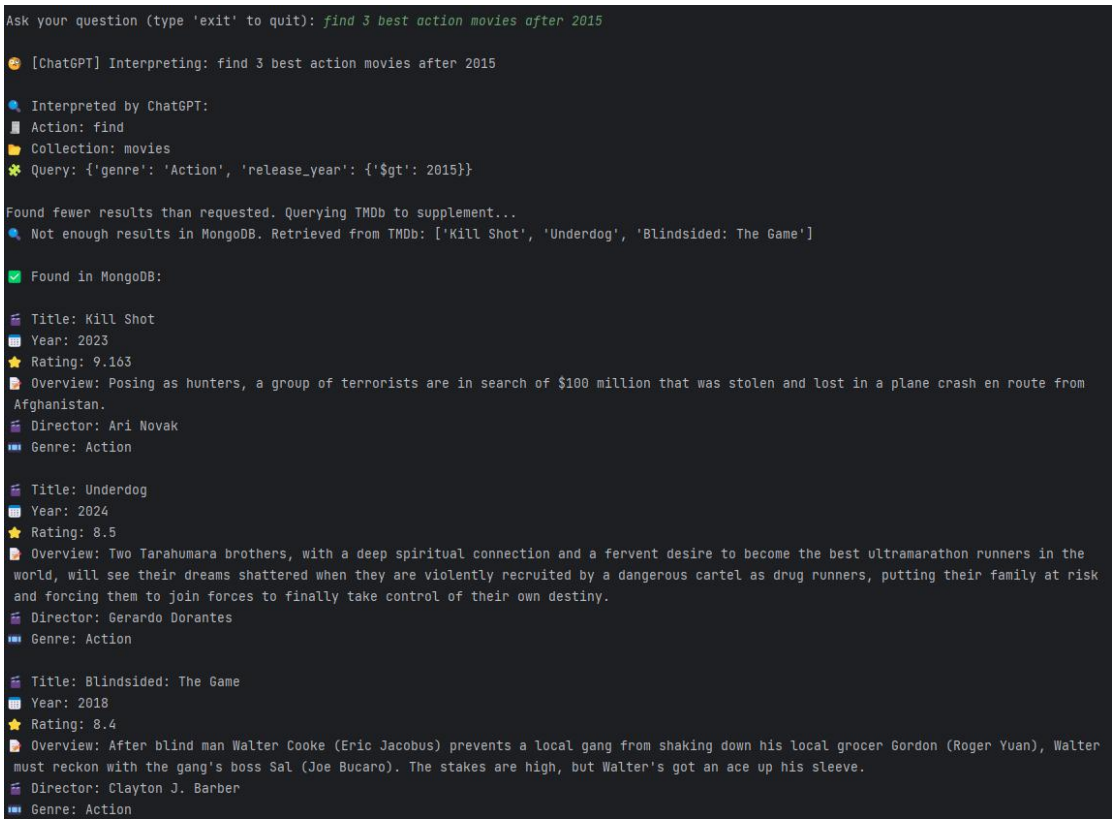

Figure 3. Querying Top-Rated Movies Based on Genre and Year

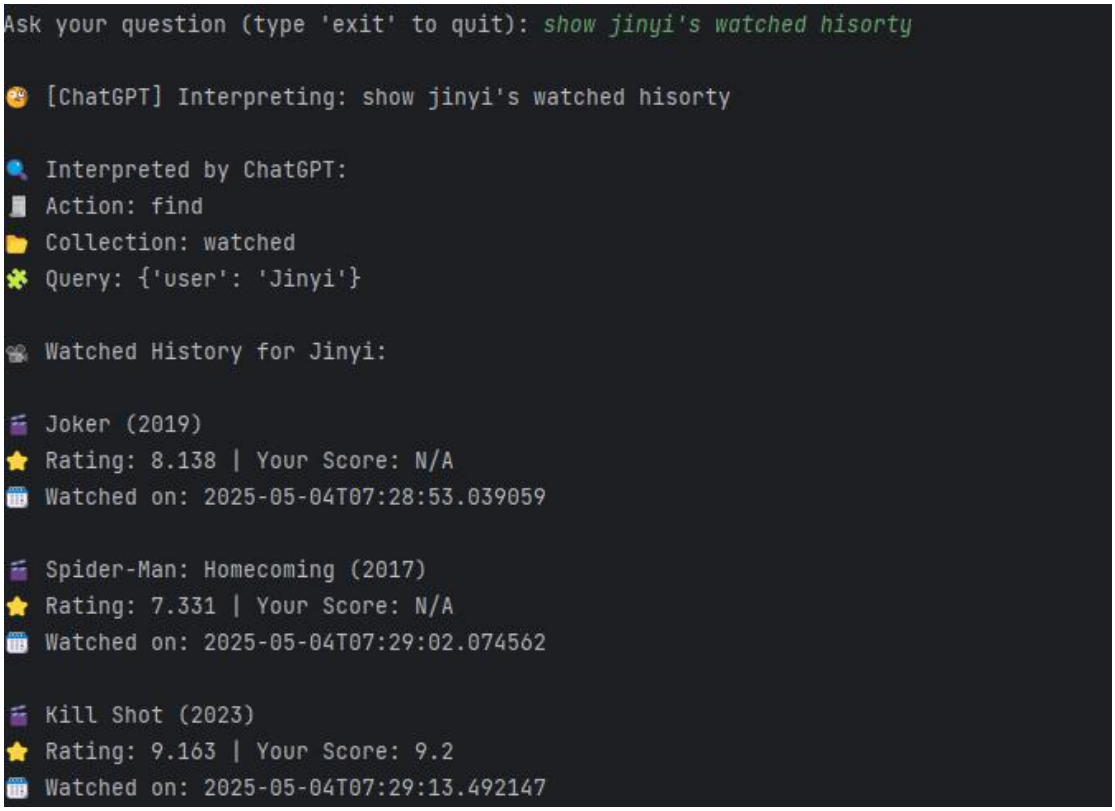

Figure 4. Retrieving User Watch History via $lookup Join

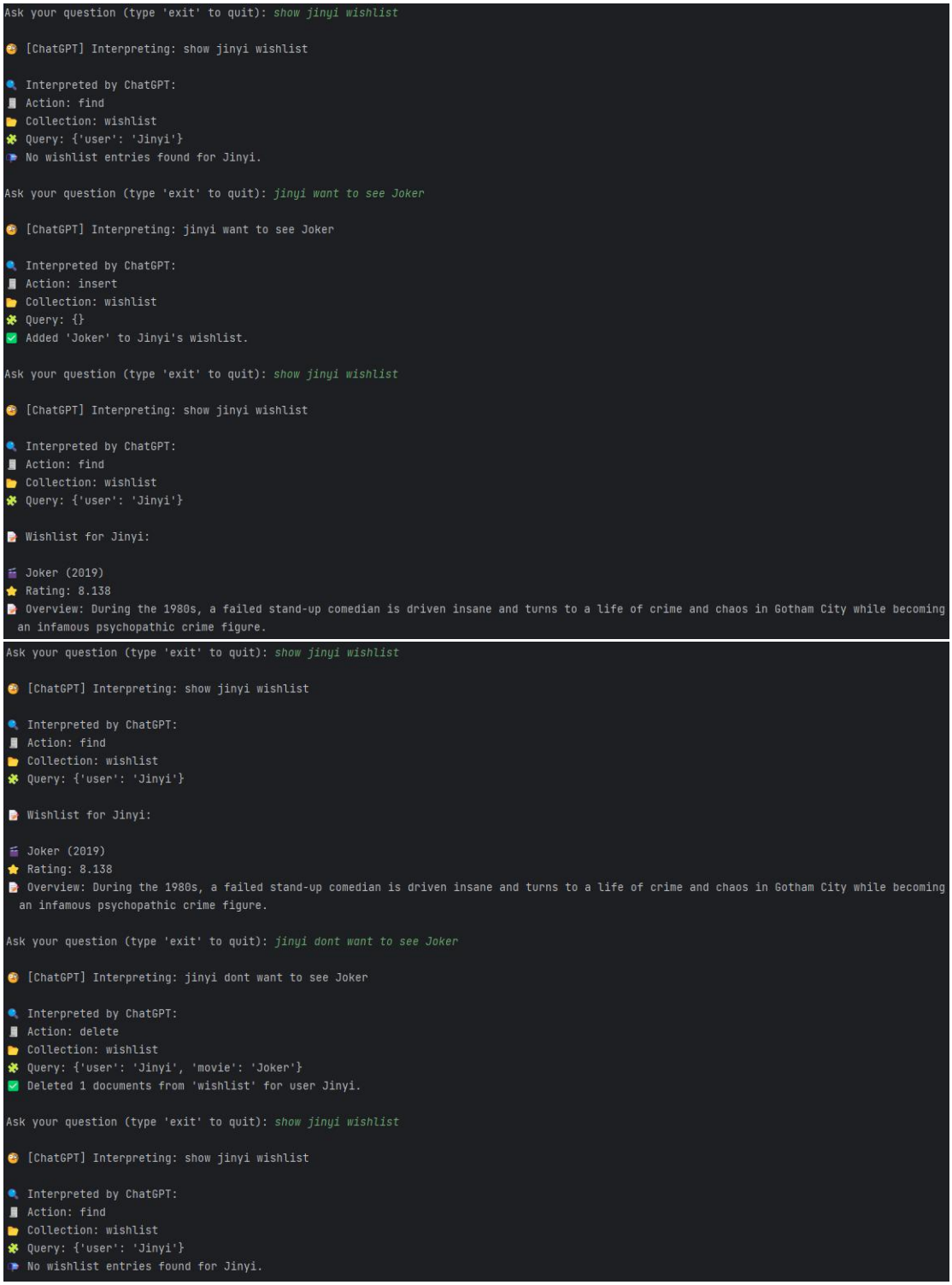

Figure 5. Data Modification: Insert and Delete Operations via Natural Language

5. Learning Outcomes

The development of ChatDB offered a valuable opportunity to combine natural language processing, NoSQL database design, and real-time system interaction into a unified, practical application. Through implementing and refining this project, several technical and conceptual

lessons were learned that extend beyond traditional coursework.

At the system level, one of the most important outcomes was learning how to construct a modular architecture that supports extensibility and dynamic behavior. Designing an interface that converts natural language into valid MongoDB queries required a deep understanding of both language models and query logic. The integration of ChatGPT as a natural language interpreter highlighted the importance of prompt design, entity extraction, and structured output formatting. Working with MongoDB's aggregation framework, especially $match, $group, $lookup, and $sort, also improved my ability to design complex queries and optimize data retrieval across collections. Additionally, implementing caching and fallback data retrieval from TMDb revealed the importance of system responsiveness and reliability in real-world applications.

On a personal level, the project significantly strengthened my ability to manage a full-stack, single-developer pipeline—from API integration and data ingestion to user interaction and query validation. Working alone required careful time management, iterative debugging, and maintaining clear documentation to ensure all components remained aligned. It also enhanced my problem-solving skills, especially in translating vague or ambiguous user input into precise backend logic. Through extensive testing and error handling, I developed a deeper sensitivity to how users phrase queries and how those variations affect the system's interpretation and behavior.

In summary, this project not only reinforced technical skills in Python, MongoDB, and API-driven development, but also deepened my understanding of how to build natural language systems that are robust, flexible, and user-centered.

6. Challenges Faced

One of the main challenges in this project was accurately converting natural language input into executable MongoDB queries. While straightforward filters such as genre or release year could be easily interpreted, complex queries involving multiple conditions—such as combining director, year, and rating—often resulted in parsing ambiguities. Ensuring the prompt to ChatGPT consistently yielded correct and structured output required multiple iterations of prompt engineering and error handling logic.

Another key difficulty was implementing multi-collection queries using MongoDB's $lookup. Since user-related operations involved linking user, movie, and watched_history

collections, the system had to construct reliable aggregation pipelines. Handling nested fields and preserving output clarity across joins required a deep understanding of MongoDB’s aggregation syntax and testing across varied data structures.

External API usage also introduced challenges. ChatGPT and TMDb both imposed rate limits and introduced latency. In early stages, repeated calls slowed system performance and made debugging inefficient. To address this, a local caching mechanism was introduced to store previous LLM results and movie metadata, reducing response time and dependency on external services.

Finally, handling vague or inconsistent user input was another recurring issue. Users often typed commands that were ambiguous, grammatically incomplete, or phrased in unexpected ways. To improve robustness, the system was refined to support synonym matching, default values, and fallback queries when no local results were found. This significantly improved usability and helped simulate a more conversational experience.

7. Conclusion

The ChatDB project demonstrates how large language models can be effectively integrated with NoSQL databases to enable natural, intuitive interactions with complex data systems. By allowing users to query and modify movie-related data using plain English, the system lowers the barrier to database access and offers a more user-friendly alternative to traditional query languages.

Through the integration of ChatGPT, MongoDB, and the TMDb API, this project successfully implemented a command-line interface capable of supporting dynamic querying, cross-collection joins, and data modification—all interpreted from natural language. The use of caching and fallback mechanisms further enhanced performance and reliability. The project also highlighted the importance of careful prompt design, modular query parsing, and robust error handling when building LLM-driven applications.

Overall, this system not only fulfills the core objectives of the DSCI 551 course project but also serves as a proof of concept for future tools that combine conversational AI with backend data systems. The modular design allows for future expansion into web interfaces, support for relational databases, or even voice-based querying, opening up many possibilities for real-world

deployment.

8. Future Scope

Although ChatDB currently operates through a command-line interface, one of the most immediate directions for future development is the integration of a web-based user interface. Building a frontend using frameworks like Flask or Streamlit would make the system more user-friendly, particularly for non-technical users. A web interface could also support additional features such as user authentication, session tracking, and persistent interaction history.

Another area for future enhancement is the introduction of alternative language models. While the current system relies on ChatGPT for query interpretation, future versions could incorporate open-source models such as DeepSeek or LLaMA to reduce dependency on third-party APIs. This would also make it possible to deploy the system offline or in enterprise environments with stricter data control requirements.

Lastly, the system could benefit from improved conversational continuity and memory. By retaining context across user inputs and offering follow-up clarifications, ChatDB could evolve into a more interactive and intelligent assistant. These improvements would further bridge the gap between database systems and natural human communication.