

Table of contents

Introduction.....	2
1. Team Member	
2. Project Details	
Planned Implementation.....	3-4
1. Timeline	
2. Task Progress	
Architecture Design	5-7
Implementation.....	7-13
1. Functionalities.....	
2. Tech Stack.....	
3. Implementation Screenshots	
Learning Outcomes.....	14
Challenges Faced.....	14-15
Individual Contribution (for multi-person team).....	15
Conclusion.....	15
Future Scope.....	15
References	15-16

Introduction

In this project, we will develop a Natural Language Interface (NLI) that connects to at least three distinct SQL and NoSQL databases. The NLI will support Create, Read, Update, and Delete (CRUD) operations using plain English commands, for example, users can ask, “How many rows are in this dataset?” or “Delete all entries from the year 2022.”

Our system will use a Large Language Model (LLM) API to convert natural language into executable queries. Users can interact with the database to view tables, apply filters, modify values, and more. They can explore schemas and data within databases, query, and modify data within databases. We will also build a full-stack web platform, allowing users to engage with the database through an intuitive “ChatGPT-style” interface called “ChatDB.”

The project includes a proposal, progress updates, and a final demo. Most of the work will focus on implementing the NLI and building a user-friendly front-end experience.

1. Team Members

Group	CHATDB 62		
Team Members	Quynh Tran	Riya Berry	Kevin Bui

2. Project details:

Team Name	ChatDB 62
Project Title	ChatDB for Large Scale Entertainment Datasets
Github Repo (Public)	https://github.com/rachelhotpot/CHATDB-FOR-LARGE-SCALE-ENTERTAINMENT-DATASETS
Google Drive Link	https://drive.google.com/drive/folders/1yjsbl_QOlPd9B_XC9IwOHVJ5ZoBAo2SZ?usp=drive_link

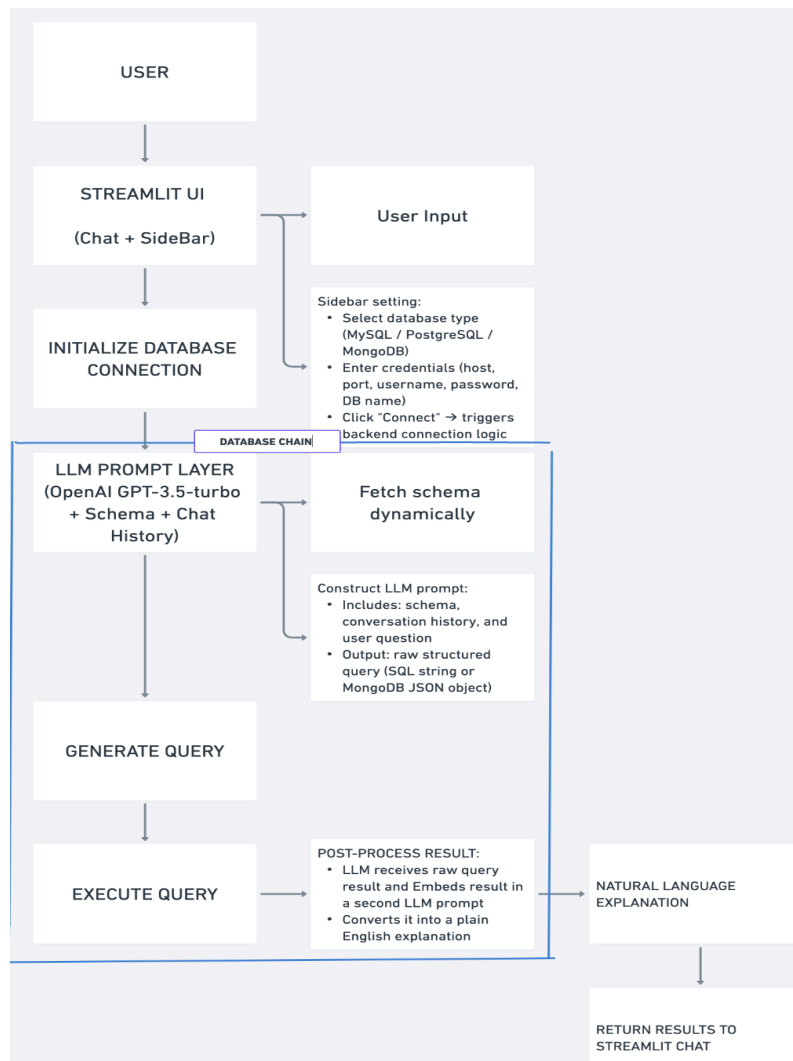
Planned implementation

1. Project Timeline and Task Progress

Task #	Task Name	Assigned Member(s)	Start Date	End Date	Status	Notes
1	Project Kickoff & Planning	Everyone	Feb 6, 2025	Feb 12, 2025	✓ Completed	Finalized scope, tools setup, Git initialized, database and LLM research
2	Proposal Submission	Riya Berry	Feb 7, 2025	Feb 7, 2025	✓ Completed	Submitted formal project proposal
3	Dataset Preparation	Riya Berry	Feb 12, 2025	Feb 17, 2025	✓ Completed	Cleaned using Pandas, exported to CSV/JSON, structured for SQL/NoSQL
4	SQL Database Implementation	Riya, Rachel	Feb 17, 2025	Feb 24, 2025	✓ Completed	MySQL server setup
5	NoSQL Database Implementation	Kevin Bui	Feb 17, 2025	Feb 24, 2025	✓ Completed	MongoDB, DynamoDB, Cassandra setup
6	LLM API Integration	Rachel	Feb 24, 2025	Mar 5, 2025	✓ Completed	Integrated Mistral API via Hugging Face
7	Backend Development	Kevin Bui	Mar 5, 2025	Mar 7, 2025	✓ Completed	API endpoints, Hugging Face integration, Firebase setup
8	Mid-Term Progress Report	Everyone	Mar 7, 2025	Mar 7, 2025	✓ Completed	Progress documented, tested Langchain integration
9	Tools & NLI Setup Check	Everyone	Mar 7, 2025	Mar 12, 2025	✓ Completed	Local setup instructions for Langchain, Streamlit, Firebase. Dataset cleanup ongoing
10	Testing & Debugging - Part 1	Everyone	Mar 19, 2025	Mar 19, 2025	✓ Completed	Focus on unit testing for query handling Changed LLM to OPENAI 3.5 with 743,444 tokens

11	Testing & Debugging - Part 2	Everyone	Mar 26, 2025	Mar 26, 2025	✓ Completed	Evaluate report structure, document challenges/solutions
12	Testing & Debugging - Part 3	Everyone	Apr 2, 2025	Apr 2, 2025	✓ Completed	Continued debugging and UI polish
13	Testing & Debugging - Part 4	Everyone	Apr 9, 2025	Apr 9, 2025	✓ Completed	Final round of bug fixes and polish
14	Final Testing & Demo Prep	Everyone	Apr 16, 2025	Apr 16, 2025	✓ Completed	Prepare for in-class demo
15	Demo Day	Everyone	Apr 21, 2025	Apr 21, 2025	✓ Completed	Live demo in class (Apr 21 and Apr 23). All members must present their contribution
16	Final Touch-Up	Everyone	Apr 24, 2025	Apr 24, 2025	✓ Completed	Final review of Rachel report version and codebase before submission
17	Final Report Submission	Rachel	May 9, 2025	May 9, 2025	✓ Completed	Submit comprehensive report with design, implementation, learnings & Google Drive link

Architecture Design

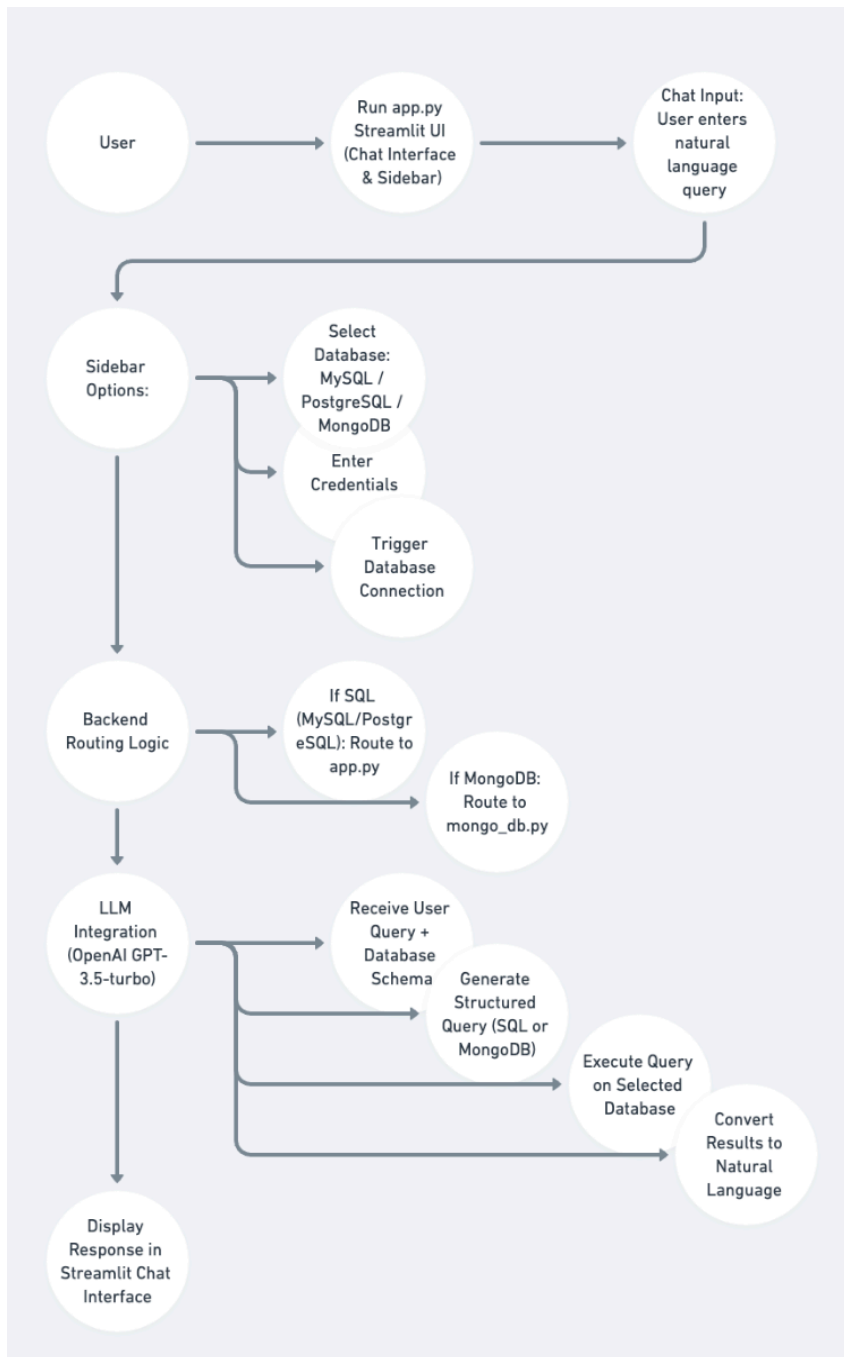


• Data architecture workflow (Backend _LLM)

1. Data is taken from Kaggle are cleaned and have 6 tables: *movie*, *rating*, *tag*, *genome_tags*, *genome_scores*, *link*
2. User input in NLQ (Natural Language Query):
 - The user submits a plain English question via Streamlit (e.g., "What are the top-rated movies in 2013?").
3. LLM (OpenAI GPT-3.5-turbo):
 - Receive the question + schema of the connected database (PostgreSQL/ MySQL/ MongoDB)
 - → Convert it into structured query:
 - SQL (for MySQL, and PostgreSQL)
 - Aggregation pipeline or find query (For MongoDB)

4. Query execution

- Query results are retrieved from the database and then LLM translation into natural language response with the context to display on Streamlit chat interface



- Website architecture (frontend & User Interaction):**

Our front-end is deployed using the Streamlit platform. It is a free platform and makes it easier to show our work results. Here are some instructions for how to engage with our site:

1. Type “streamlit run app.py” on your terminal

2. Streamlit pop up on the website
3. Sidebar lets users:
 - a. **Select database (MySQL / PostgreSQL / MongoDB)**
 - b. Enter credentials
 - c. Trigger database connection
4. Execute logic
 - a. SQL → executes logic via app.py
 - b. MongoDB → execute logic via mongo_db.py
5. User inputs their query
6. Backend routes it through the LLM → query → result → LLM (optional)
7. Output shown in real-time chat block

Implementation

1. Functionalities

Our application has the following functionalities:

- a. Natural Language Query
- b. Multiple Database support
- c. Query Execution
- d. Data Manipulation
- e. Real-time interaction

2. Tech Stack

- a. **Frontend:** Streamlit
- b. **Backend:** Python, LangChain
- c. **Databases:** MySQL, PostgreSQL, MongoDB
- d. **LLM integration:** OpenAI GPT-3.5-turbo via API and LangChain to build query and reasoning chains
- e. **Data Handling:** Pandas, pymongo, sqlalchemy, csv, json, re, dotenv
- f. **Environment management:** streamlit for local and cloud deployment, dotenv
- g. **Version Control:** Git & GitHub
- h. **OPENAI_API_KEY:** stored in this document [link](#)

3. Implementation Screenshots

a. MongoDB



What collections are in this database?



In this database, there are collections named 'link', 'genome_tags', 'rating', 'genome_scores', 'movie', 'rating-new', and 'tag'.

Show me the top 5 rated movies where user id is less than 5000 and the average rating is at most 4.5. Show me the MongoDB query and movie names.

The top 5 rated movies where the user id is less than 5000 and the average rating is at most 4.5 are:

1. Title: Unknown Woman, The (La Sconosciuta) (2006), Average Rating: 4.5
2. Title: Hunter, The (2011), Average Rating: 4.5
3. Title: Little Witches (1996), Average Rating: 4.5
4. Title: Ballad of a Soldier (Ballada o soldate) (1959), Average Rating: 4.5
5. Title: Pretty Poison (1968), Average Rating: 4.5

Here is the MongoDB query used to retrieve this information along with the movie names:

```
db.rating.aggregate([
  { $match: { userId: { $lt: 5000 } } },
  { $group: { _id: '$movieId', avgRating: { $avg: '$rating' } } },
  { $match: { avgRating: { $lte: 4.5 } } },
  { $sort: { avgRating: -1 } },
  { $limit: 5 },
  { $lookup: { from: 'movie', localField: '_id', foreignField: 'movieId',
  { $unwind: '$movie' },
  { $project: { title: '$movie.title', avgRating: 1 } }
])
```

The response from the database for this query includes the movie titles and their average ratings as shown above.

Insert a new movie with movie ID 9997, title 'Demo Movie', and genre 'Sci-Fi'.

To insert a new movie with movie ID 9997, title 'Demo Movie', and genre 'Sci-Fi' into the database, you can use the following MongoDB query:

```
db.movie.insertOne({
  'movieId': 9997,
  'title': 'Demo Movie',
  'genres': 'Sci-Fi',
  '_id': ObjectId('681d90d3dafe3790adcc5662')
})
```

The response from MongoDB after executing this query will be:

```
{'inserted_id': '681d90d3dafe3790adcc5662'}
```

This indicates that the new movie with the specified details has been successfully inserted into the database with the provided ID.



Update the genre of movie ID 9997 to 'Adventure'



To update the genre of movie ID 9997 to 'Adventure' in the database, you can use the following MongoDB query:

```
db.movie.updateOne(  
  {'movieId': 9997},  
  {'$set': {'genres': 'Adventure'}}  
)
```



After executing this query, the MongoDB response will indicate that 1 document was matched and 1 document was modified, confirming that the genre update for movie ID 9997 to 'Adventure' was successful.



Delete the movie with ID 9997.



To delete the movie with ID 9997 from the database, you can use the following MongoDB query:

```
db.movie.deleteOne({'movieId': 9997 })
```



This query will delete the movie with the specified ID from the database. The response from MongoDB will indicate that 1 document was successfully deleted.

b. MySQL:



The SQL query used to retrieve movies that have an average rating above 4.5 using the top 5000 user ids was:

```
SELECT movie.title, AVG(rating.rating) as avg_rating  
FROM movie  
JOIN rating ON movie.movieId = rating.movieId  
WHERE rating.userId <= 5000  
GROUP BY movie.title  
HAVING AVG(rating.rating) > 4.5  
LIMIT 5;
```

The results of the query show the following movies with their respective average ratings:

1. Welcome Home, Roscoe Jenkins (2008) - Average Rating: 5.0
2. Towelhead (a.k.a. Nothing is Private) (2007) - Average Rating: 5.0
3. Power of Nightmares, The: The Rise of the Politics of Fear (2004) - Average Rating: 4.75
4. Keith (2008) - Average Rating: 5.0
5. True Stories (1986) - Average Rating: 4.58



The SQL query used to add a new movie titled 'The Mandalorian & Grogu (2026)' in the genre 'Sci-Fi' with a movieid 12345 is:

```
INSERT INTO movie (movieId, title, genres)
VALUES (12345, 'The Mandalorian & Grogu (2026)', 'Sci-Fi');
```



The movie 'The Mandalorian & Grogu (2026)' has been successfully updated to include the genre 'Action'.



Show me the movie 'The Mandalorian & Grogu (2026)'.



The movie 'The Mandalorian & Grogu (2026)' has been successfully added to the database with the genres 'Sci-Fi' and 'Action'. The movie can be found with the movieid 12345 and has the genres Sci-Fi and Action.



Delete the movie 'The Mandalorian & Grogu (2026)'. Show me the query.



To delete the movie 'The Mandalorian & Grogu (2026)', the following SQL query should be used:

```
DELETE FROM movie
WHERE movieId = 12345;
```

This query will remove the movie entry with the movieid 12345 from the database.

c. PostgreSQL



Give me a sample of 5 rows in movie. Show the query.



Sure! Here are 5 sample rows from the 'movie' table:

1. Title: Toy Story (1995) Genres: Adventure|Animation|Children|Comedy|Fantasy
2. Title: Jumanji (1995) Genres: Adventure|Children|Fantasy
3. Title: Grumpier Old Men (1995) Genres: Comedy|Romance
4. Title: Waiting to Exhale (1995) Genres: Comedy|Drama|Romance
5. Title: Father of the Bride Part II (1995) Genres: Comedy

The SQL query used to retrieve these rows is:

```
SELECT *
FROM movie
LIMIT 5;
```

These are just a few examples of the movies stored in the 'movie' table. Let me know if you need more information!



To find movies that have an average rating above 4.5 using the top 5000 user IDs, we used the SQL query:

```
SELECT m.title
FROM movie m
JOIN rating r ON m.movieid = r.movieid
WHERE r.userid IN (SELECT userid FROM rating ORDER BY userid LIMIT 5000)
GROUP BY m.title
HAVING AVG(r.rating) > 4.5
ORDER BY m.title
LIMIT 5 OFFSET 10;
```

The results of the query show the following movies:

1. Akeelah and the Bee (2006)
2. Alien: Resurrection (1997)
3. Aliens of the Deep (2005)
4. Amazing Grace (2006)
5. American Crime, An (2007)

These movies have an average rating above 4.5 based on the ratings from the top 5000 user IDs.



Add a new movie titled 'The Mandalorian & Grogu (2026)' in the genre 'Sci-Fi' with a movieid 12345. Show me the SQL query used.



The SQL query used to add a new movie titled 'The Mandalorian & Grogu (2026)' in the genre 'Sci-Fi' with a movieid of 12345 is:

```
INSERT INTO movie (movieid, title, genres)
VALUES (12345, 'The Mandalorian & Grogu (2026)', 'Sci-Fi');
```

This query will insert the new movie into the 'movie' table with the specified details.



Update movie 'The Mandalorian & Grogu (2026)' to include the genre 'Action'. Show me the query.



The SQL query to update the movie 'The Mandalorian & Grogu (2026)' to include the genre 'Action' is:

```
UPDATE movie
SET genres = 'Sci-Fi|Action'
WHERE movieid = 12345;
```



This query will update the genres of the movie with the title 'The Mandalorian & Grogu (2026)' to include 'Action' in addition to 'Sci-Fi'.



Delete the movie 'The Mandalorian & Grogu (2026)'. Show me the query.



The SQL query to delete the movie 'The Mandalorian & Grogu (2026)' is:

```
DELETE FROM movie
WHERE movieid = 12345;
```

This query will remove the movie with the title 'The Mandalorian & Grogu (2026)' from the 'movie' table.

4. APP Layout and directory structure

CHATDB-FOR-LARGE-SCALE-ENTERTAINMENT-DATASETS/

|--requirements.txt # all the requirement for running our code

|--README.md

|--code/

 |--app.py # Used to create NLI and generate queries for MySQL/PostgreSQL

 |--mongo_db.py # Used to create NLI and generate queries for MongoDB

 |--dataset.sql # Used to upload data to MySQL

 |--dataset_psql.sql # Used to upload data to PostgreSQL

|--reports

 |--Draft- Group Proposal.pdf

 |--Mid Progress Report.pdf

 |--551_ Group Proposal Final.pdf

 |--CHATDB_Final_Report.pdf

Learning Outcomes

1. Full-Stack Application Setup

Through this project, we learned how to build a complete full-stack system starting from the front-end using Streamlit to handling back-end logic with Python. We were able to connect user inputs from the interface directly to database queries and AI-powered responses using a language model. This gave us a solid understanding of how each layer interacts and how to structure the system for real-time communication. It also gave us greater insight into how large-scale LLM applications like OpenAI, DeepSeek and Perplexity are implemented and function behind the scenes.

2. Multi-Database Integration

Working with multiple databases like MySQL, PostgreSQL, and MongoDB gave us practical experience in managing different data models. We gained a better understanding of how relational and non-relational databases differ in structure, performance, and syntax. This hands-on exposure helped us see when to use each type based on the data and the query requirements. It also strengthened our understanding of data engineering and ETL principles taught in the course.

3. Project Management and Version Control

Even though it was a relatively small-scale project, collaborating as a team taught us a lot about managing time, dividing responsibilities, and staying organized. Using Git and GitHub for version control helped us coordinate changes, track progress, and handle updates without stepping on each other's work. We also set up weekly meetings to check in on progress, conducted meetings with an organized agenda, and then followed up on our respective action items.

4. API Integration and Token Management

Integrating LLM APIs into our backend gave us insight into how API authentication works, how to securely manage keys, and how different LLM services handle the request (like OpenAI and Hugging Face) and choosing what is the best for our project. We also became more aware of token usage limits and how to plan for cost and efficiency when working with large scale models.

Challenges Faced

a. Dataset Size

We changed the dataset twice and ended up using one with multiple tables in order to demonstrate seamless join and aggregation operations. The large size caused delays when running complex queries like join and inserts in both SQL and MongoDB. To improve performance with limited time we had in demo day, we limited the processing to the first 500-1000 rows which helped reduce wait time without losing meaning results

b. OPENAI Token Usage and Mongo Alias Cost

We used OPENAI GPT- 3.5 and bought a large amount of tokens, but finished with over 700,000 tokens unused. For MongoDB, one teammate had to buy a paid add-on to enable Mongo Atlas because of the size of our data – which added extra setup cost.

Individual Contribution

Team Members	Quynh Tran	Riya Berry	Kevin Bui
Contributions	<ul style="list-style-type: none">• Created and edited Final report• Setting up LLM• Research resources• Created Mid-progress report	<ul style="list-style-type: none">• Build MongoDB code, MongoDB databases• Streamlit Integration• Created questions & prompts for Demo Day• Facilitated the presentation of our project during Demo Day	<ul style="list-style-type: none">• Responsible for entire whole front-end user interface• Relational Schema Design & Ingestion<ul style="list-style-type: none">◦ Set up both MySQL and PostgreSQL instances .◦ Created SQL+postgreSQL scripts

Conclusion

Over the course of this semester, we developed an interactive Natural Language Interface that combines front-end, backend, NoSQL and SQL Databases, and LLM integration to create an interactive database ChatBot called “ChatDB” – similar to the widely used “ChatGPT.” **Our NLI allows users to explore schemas and data within databases, execute natural language queries, and insert/delete/and update requests in natural language.**

This project allowed us to hone our Data Engineering, SQL, NoSQL, virtual environment, project management, and collaboration skills to create a real-world data science application with large-scale impact. Although challenges arose, we emerged with significant learning outcomes in terms of **Full-Stack Application Setup, Multi-Database Integration, Project Management, and API integration.** All of these skills will prove incredibly invaluable in our professional and academic data science journeys. We are so grateful for this opportunity – thank you for taking the time to read through our report.

Future Scope

If we have more time and budget, we definitely use more large scale datasets. Additionally, we would switch many LLM models based on the latency, accuracy needs and cost, and latency. To maintain privacy for the user, we will have user authentication to control who is using our app and improve the query accuracy by fine- tuning the prompts, reducing the error and improving translation from natural language to database specific syntax.

References

- Dataset from [Kaggle](#)
- OpenAI:
 - <https://platform.openai.com/>
- LangChain Docs:
 - https://github.com/TestZucky/mongo_db_langchain/blob/main/mongo_db_langchain_main.ipynb
- MongoDB:
 - <https://cloud.mongodb.com/v2#/org/67ddc5d814db796e809d0351/projects>
 - <https://www.oneschema.co/blog/import-csv-mongodb>
- Tutorials
 - <https://www.youtube.com/watch?v=YqqRkuizNN4>
 - <https://www.youtube.com/watch?v=fUx1>