# CSC3170 Project: Database for Models and Datasets (Draft)

- Notation: different parts to complete.
    - [s] secure
    - [d] database schema
    - [f] frontend
    - [i] data insight
    - [?] other todo list

## 1. Introduction and motivation

### 1.1. Introduction

- Our project is a database for machine learning models and datasets.
    - **Basic database operations:** It allows users to browse the information about the models and datasets, upload and download models and datasets.
    - **Schema:** Apart from the basic schemas such as dataset, model, user, we also included schemas that are especially helpful for machine learning developers, such as tables describing the modular structures of different architecture of models (CNN, RNN, Transformer).
    - **GUI:** A beautifully designed graphic user interface is implemented, where users and administrators can perform multiple types of operations.
    - **LLM:** An LLM agent is implemented, to translate user's natural language query into SQL language. User can also customize their query by selecting different tables and different fields.
    - **Security:** Methods are implemented to protect data security,

### 1.2. Motivation

- We are motivated by huggingface, one of the most influential platform in the AI community that facilitates the sharing and collaboration of machine learning models and datasets.

### 1.3. How to run our code

- **Step 1-3 has to be done ONLY when running it at the first time; if it's not the first time, you can skip 1-3, and also can skip 4 if you don't need to initialize the database.**
- [q] *update this part after startup.py is finished.*

1. Install dependencies according to `requirement.txt` [?]
2. Create an `.env` file at the root directory of the project, and add the following lines to it (repalce `$your_api_key` and `$your_base_url` with your own values):

```
# -----database-----
 DB_USERNAME=root
 DB_PASSWORD=123
 DB_HOST=0.0.0.0
 DB_PORT=3306
 TARGET_DB=openmodelhub
# -----agent----
API_KEY=$your_api_key
BASE_URL=$your_base_url
```

3. Test connection by running `database/db_connection_check.py`.
4. Initialize the database with the records stored in `database/records/demo.json`, by running:

```
database/load_data.py
```

- then you'll be asked to choose a .json file stored in `database/records` to intialize it; just choose `demo.json`.

5. Run the GUI:

```
streamlit run frontend/app.py
```

6. Login as common user or admin

- Login to admin with **username: admin, password: admin**.
- After logging in as admin, you can see the list of all users in the page `user management`. Note that some users are admin, too, as indicated on the page.
- Every user's password is admin.
- You can register your own user, too.

# 2. Design and implementation

## 2.0. Project Structure

- our project is composed of the following components:
    1. Database.
    2. Data.
    3. Frontend.
    4. Agent.
    5. Security.
    6. Data analysis.

## 2.1. Database

- [d][THE WHOLE PART needs fact-checking!! whether my description is accurate?]

**Schema Design**

- Our database follows the relational model and the 4th normal form.
- Our schema are as follows:
- [d] [please insert a markdown format table here to show the schema. can be generated from our slides.]
- [?] llm optimized design

**Implmentation**

- In `database/database_schema.py`, schemas are represented by python classes.
- In `database/database_interface.py`, we have encapsulated interfaces to perform SQL operations safely. Therefore, in other programs where we have to execute SQL, we can call an encapsulated functions instead of executing the SQL operations directly.

## 2.2. Data

**Initialization**

- We created a set of records to initialize our database; although more records can be inserted to or deleted from the database during use. It is stored in `database/records/demo.json`, and can be run by `database/load_data.py`, as indicated previously.
- The records consist of:
    1. 12 affiliations;
    2. 28 users from these affiliations;
    3. 100 datasets;
    4. 92 models.
- The models' names, corresponding architecture, media type, train method (fine-tuned or pre-trained) are real; the dataset's names and media types are real, because they are copied from models and datasets that are actually posted to huggingface. However, some other attributes, such as parameter number and authors, are made up.

**Upload and Download**

- `database/load_data.py` can initialize the database by inserting records stored in json formats, containing entities among `affiliation`, `user`, `dataset`, `model`.
- [f] [should explain how to download and what programs are responsible.]

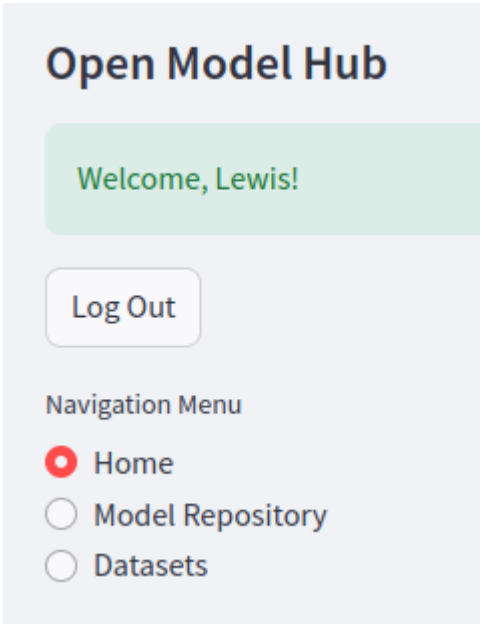
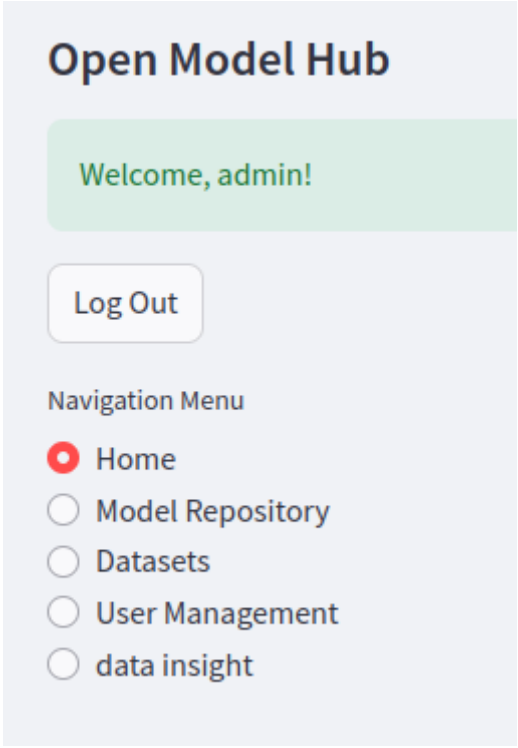
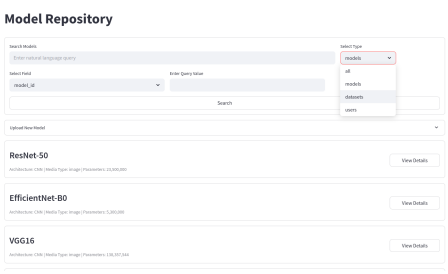
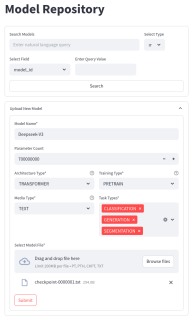
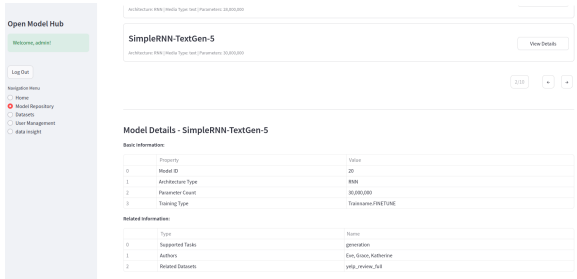
## 2.3. Frontend

**login/regsiter**

1. common user login
2. common user register and login

3. admin login: has some pages that common users don't have.
    - username: admin; password: admin.

- The pages visible to a common user / an admin is different.

| type | user | admin |
|---|---|---|
| sidebar |  |  |
| **LLM assisted search, with specifying the entity in the drop-down box** | **upload model** | **click "view details", and 2 tables representing the detailed information of that model will be displayed. paging are implemented for improved user experiment.** |
|  |  |  |

### Home page

- can export and download ata.

### Model/Dataset Repository page

- the following screenshots are from the model page; but the dataset page is very similar.

**(Admin Privilege) User Management / Data Insight**

| 4. (Admin Privilege) User Management | 5. (Admin Privilege) Data Insights | data insight, page 2 |
| --- | --- | --- |

create/edit user

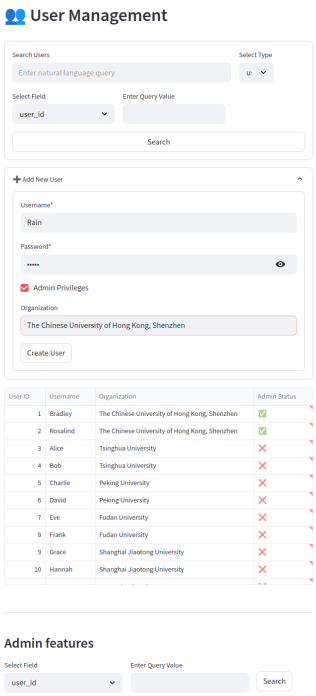
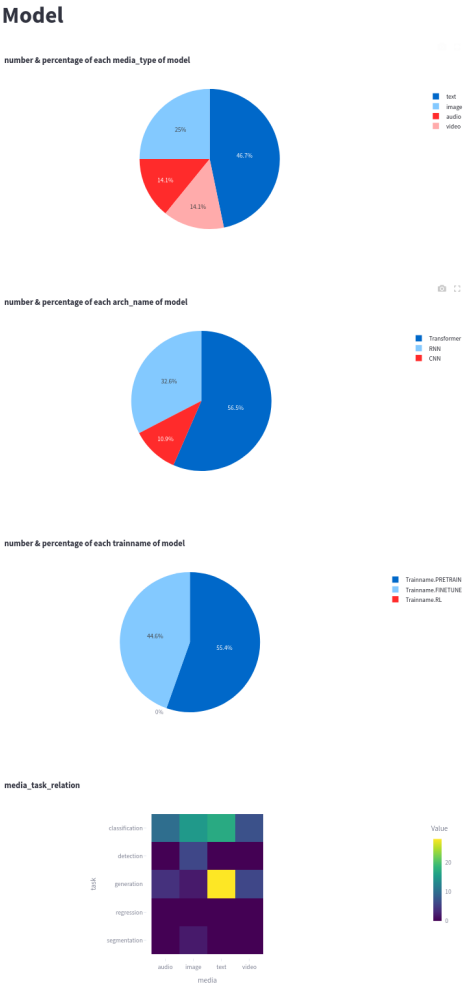
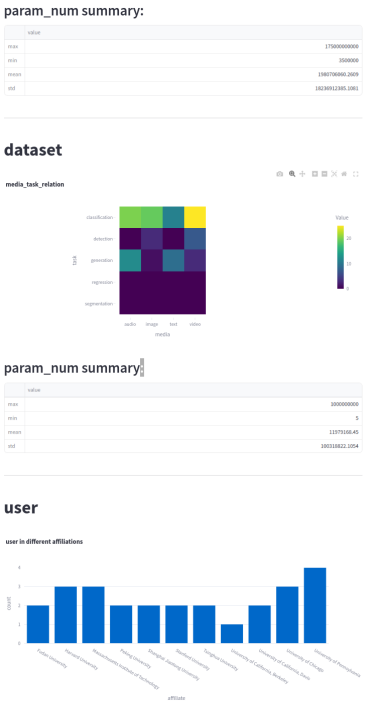



illustration of the analysis on the data in the database.




Also illustration.




## 2.4. Agent

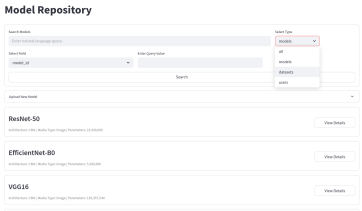
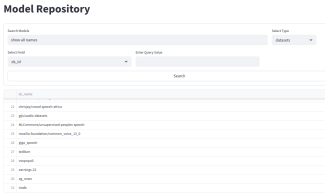
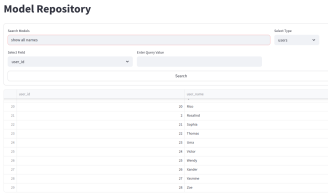
### Implementation

- We incorporated gpt-4o as an LLM agent that translates user's natural language input into SQL queries.
- **Input/Output**
  - Input includes a natural language query, and a integer specifying the type of entity user is asking for. A corresponding string will be appended to the natural language query. This integer is by default 0, indicating no specific constraints.
  - Output: a dictionary, consisting of:
    - an error code indicating whether a grammatically correct sql is generated;
    - a SQL query generated
    - the result of the SQL query

- **System prompt**
  - **Schema:** In the system prompt, we describe our database, the integrity constraints, and other information required.
  - **Synonyms:** In practice, we find it necessary to add some synonyms to help agent understand user's needs in this context. For example, if user asks for a `langauge model`, user is referring to `models where media_type includes 'text'`
  - **Instance type:** The constraints on the type of entity user's asking for is also indicated in the system prompt.
- **2-stage error-detection leveraging agent's self-correction**:
  - After the SQL is created, it will be executed to check its grammatical correctness, instead of directly returning the SQL.
  - **If incorrect, agent will perform another attempt to generate SQL, based on the previous failure.** However, if it fails again, no more attempts will be made.

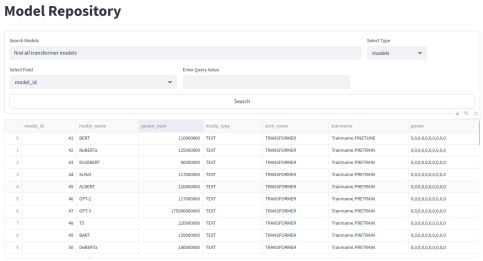
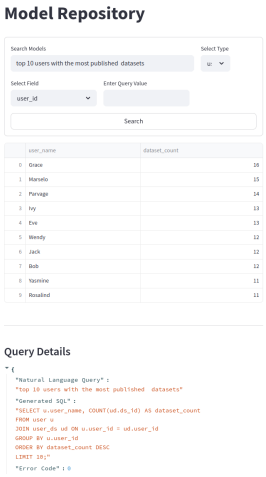
**Demonstration: using LLM assisted search in the GUI**

- Feature: generate query of a specific entity.

| entity type | model | dataset | user |
|---|---|---|---|
| query: show all names |  |  |  |

- other queries

| Query | Find all transformer models | (same as previous) | top 10 users with the most published datasets |
|---|---|---|---|

| Query | Find all transformer models | (same as previous) | top 10 users with the most published datasets |
|---|---|---|---|
| Result | Result can be represented in a table.<br>  | Can also view the corresponding SQL query.<br>  | More complicated queries can also be executed.<br>  |

### 2.5. Security

- [s]

### 2.6. Data Insight

- [i]

## 3. Conclusion and self-evaluation

### 3.1. Conclusion

- We has completed task [?] indicated in the project guideline.
- [?] mention detailed implementation here.

### 3.2. Self-Evaluation

- Work division is as follows: (members' names follows alphabetical order)

**Yimeng Teng**

- Implemented the entire `agent` part. Generated test cases to evaluate and refine it.
- Collaborated with Linyong Gan to generate `demo.json`, which contains sufficient amounts of records for initializing the database.
- Collaborated with Wentao Lin in implementing a data loader that load json files and insert records to the database. Designed the first version and help completed the final version.
- Participated in the formulation of the database schema (but not the implementation).

## 4. References

- https://huggingface.co/
- Feistel, H. (1973). Cryptography and computer privacy. Scientific american, 228(5), 15-23.
- Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.


# 5. Appendices

[?] what to include