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:** A variety of schemas is implemented. Apart from the basic schemas such as dataset, model, user, we also included schemas that are especially helpful for machine learning developers, such as the layer structure 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. As the number of machine learning models and datasets continues to grow rapidly, there is a pressing need for a structured and efficient way to manage these resources.
- When using a machine learning database with a considerable amount of data, it may be challenging for users to find a model or dataset that suits their requirement by traditional seache methods, or by directly inputting SQL, which may be technically challenging for general users.
- Therefore, we implement an agent to assists the user's queries, enhancing flexibility for users to perform customized operations, as well as efficiency especially when the user's need is too complicated to be manually written into SQL.
- Furthermore, the increasing concern over data security and privacy necessitates more robust solutions to protect valuable information. [s]

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:
  i. Database.
  ii. Data.
  iii. Frontend.
  iv. Agent.
  v. Security.

vi. 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:
  i. 12 affiliations;
  ii. 28 users from these affiliations;
  iii. 100 datasets;
  iv. 92 models.

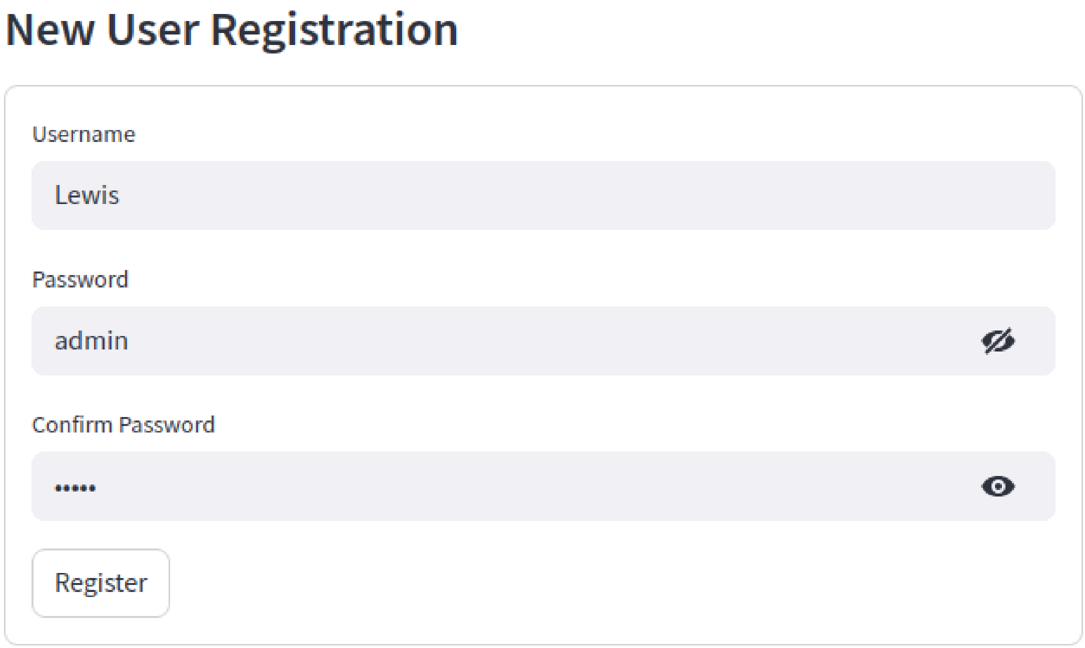
- The models' names, corresponding architecture, media type, train method (fine-tuned or pre-trained) are real; the dataset's names and media types are real, because they are copied from models and datasets that are actually posted to huggingface. However, some other attributes, such as parameter number and authors, are made up.

Upload and Download

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

2.3. Frontend

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



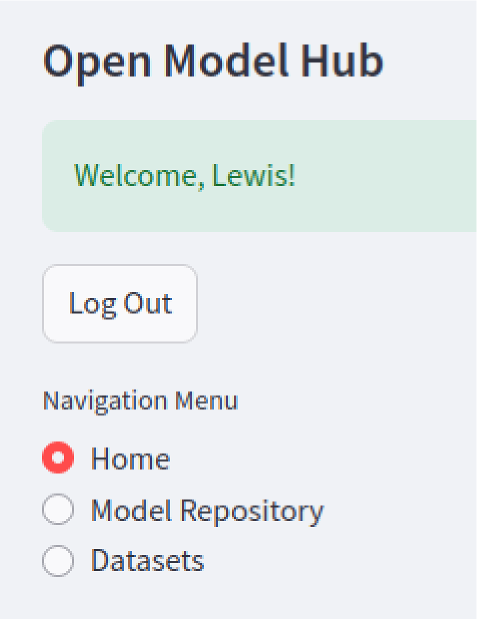
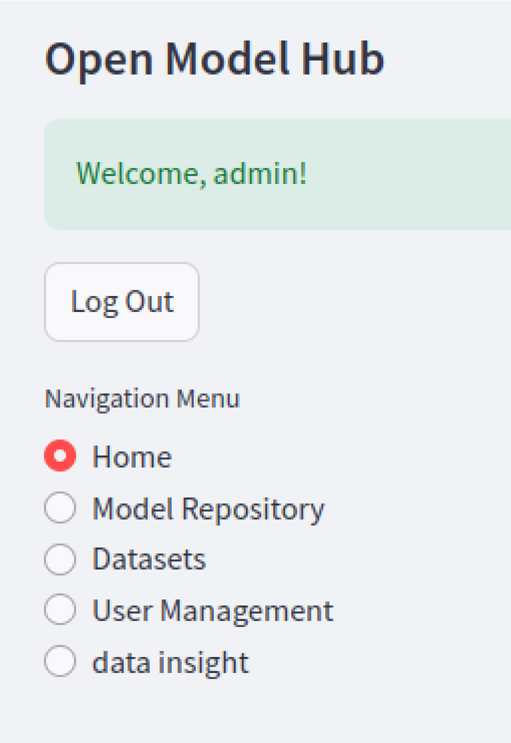
New User Registration

Username: Lewis

Password: admin

Confirm Password: •••••

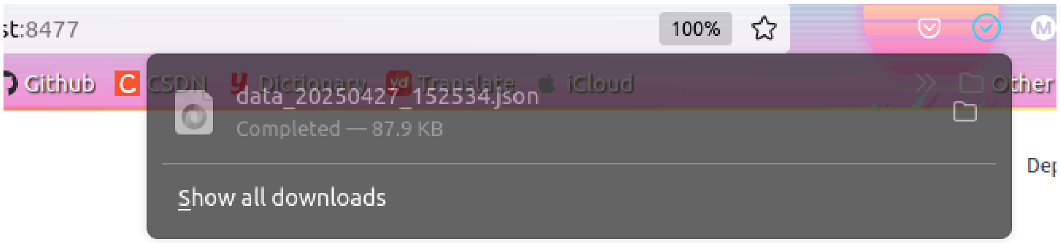
   3. admin login: has some pages that common users don't have.
      - username: admin; password: admin.

type	user	admin
sidebar	 Open Model Hub Welcome, Lewis! Log Out Navigation Menu ● Home ○ Model Repository ○ Datasets	 Open Model Hub Welcome, admin! Log Out Navigation Menu ● Home ○ Model Repository ○ Datasets ○ User Management ○ data insight

ii. page types

a. Home

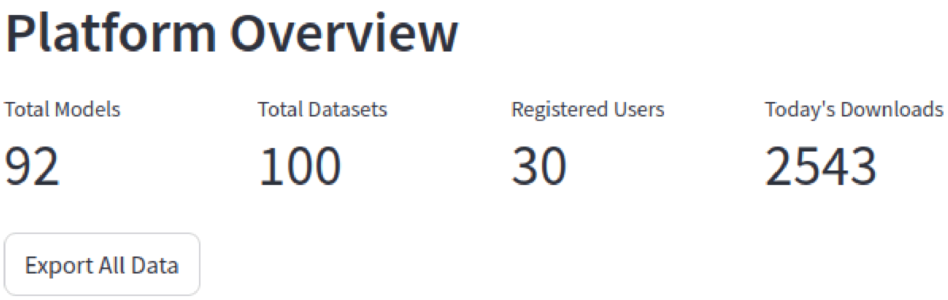
- export and download data



data_20250427_152534.json

Completed — 87.9 KB

Show all downloads



Platform Overview

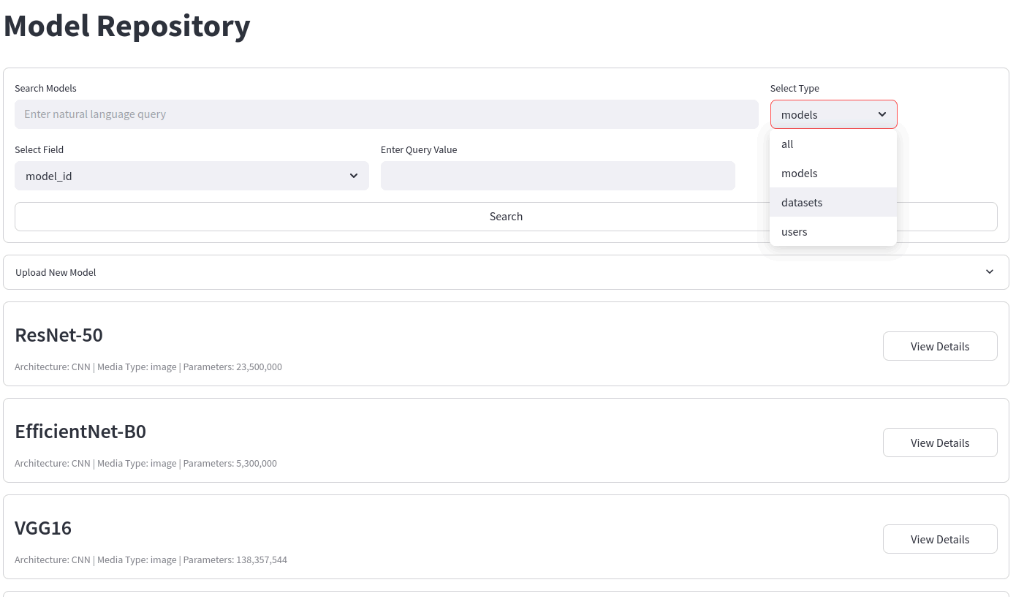
Total Models	Total Datasets	Registered Users	Today's Downloads
92	100	30	2543

Export All Data

-

b. Model Repository

- search with/without specifying the instance in the drop-down box



Model Repository

Select Type: models

Select Field: model_id

ResNet-50

Architecture: CNN | Media Type: image | Parameters: 23,500,000

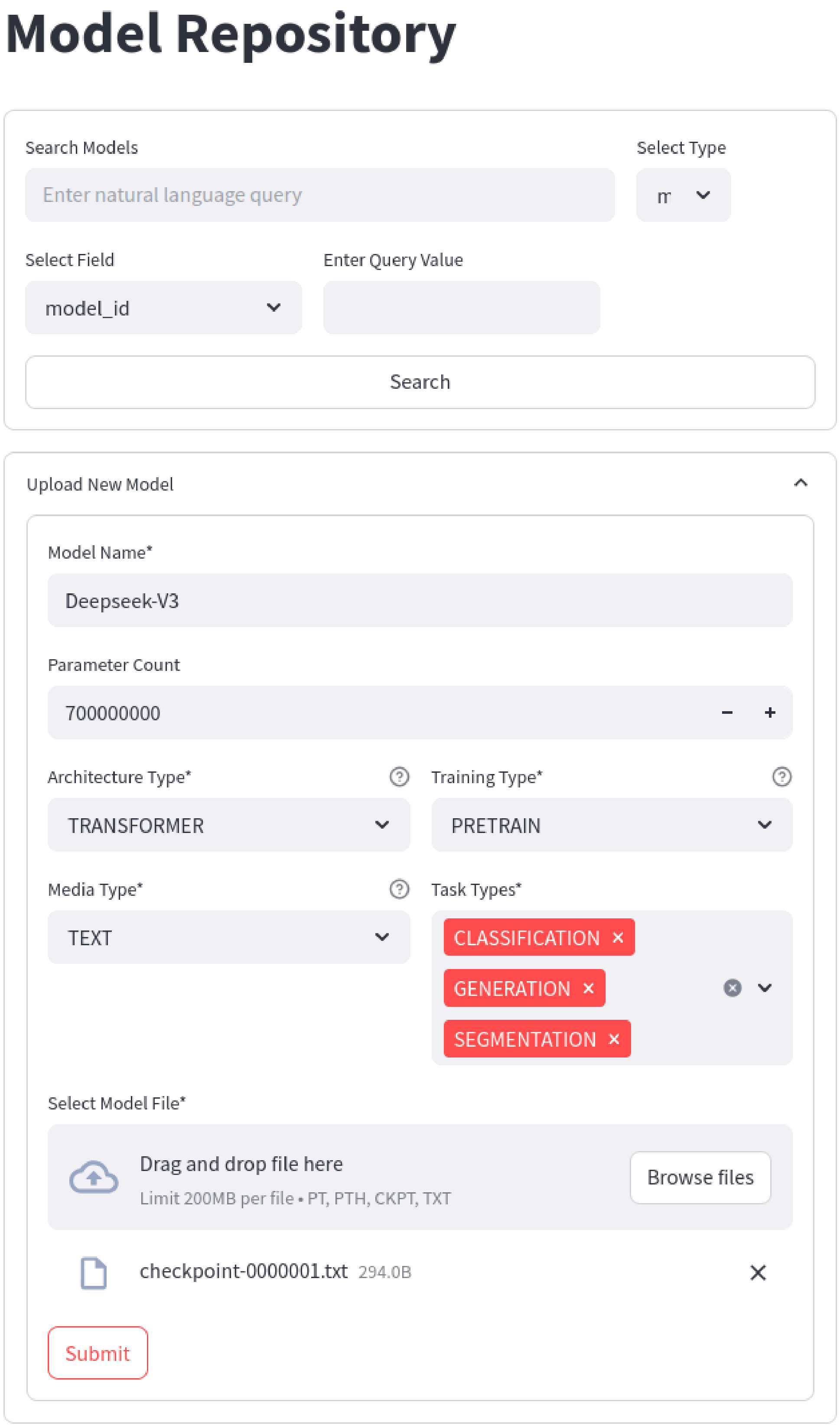
EfficientNet-B0

Architecture: CNN | Media Type: image | Parameters: 5,300,000

VGG16

Architecture: CNN | Media Type: image | Parameters: 138,357,544

- **LLM assisted search: refer to a following part**
- upload model



Model Repository

Search Models: Enter natural language query

Select Type: m

Select Field: model_id

Enter Query Value:

Upload New Model

Model Name*: Deepseek-V3

Parameter Count: 700000000

Architecture Type*: TRANSFORMER

Training Type*: PRETRAIN

Media Type*: TEXT

Task Types*: CLASSIFICATION × GENERATION × SEGMENTATION ×

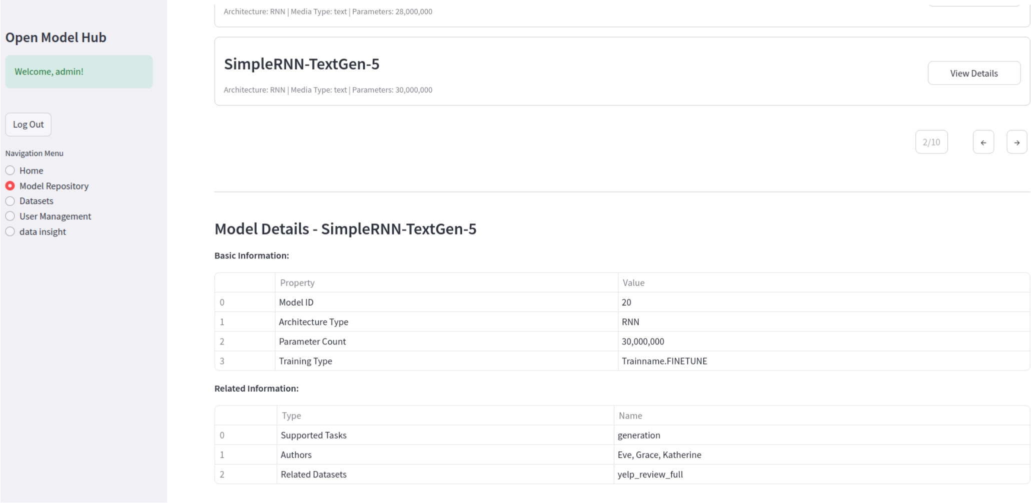
Select Model File*

Drag and drop file here

Limit 200MB per file • PT, PTH, CKPT, TXT

checkpoint-0000001.txt 294.0B

-
- click "view details", and 2 tables representing the detailed information of that model will be displayed.



Architecture: RNN | Media Type: text | Parameters: 28,000,000

SimpleRNN-TextGen-5

Architecture: RNN | Media Type: text | Parameters: 30,000,000

Model Details - SimpleRNN-TextGen-5

Basic Information:

	Property	Value
0	Model ID	20
1	Architecture Type	RNN
2	Parameter Count	30,000,000
3	Training Type	Trainname.FINETUNE

Related Information:

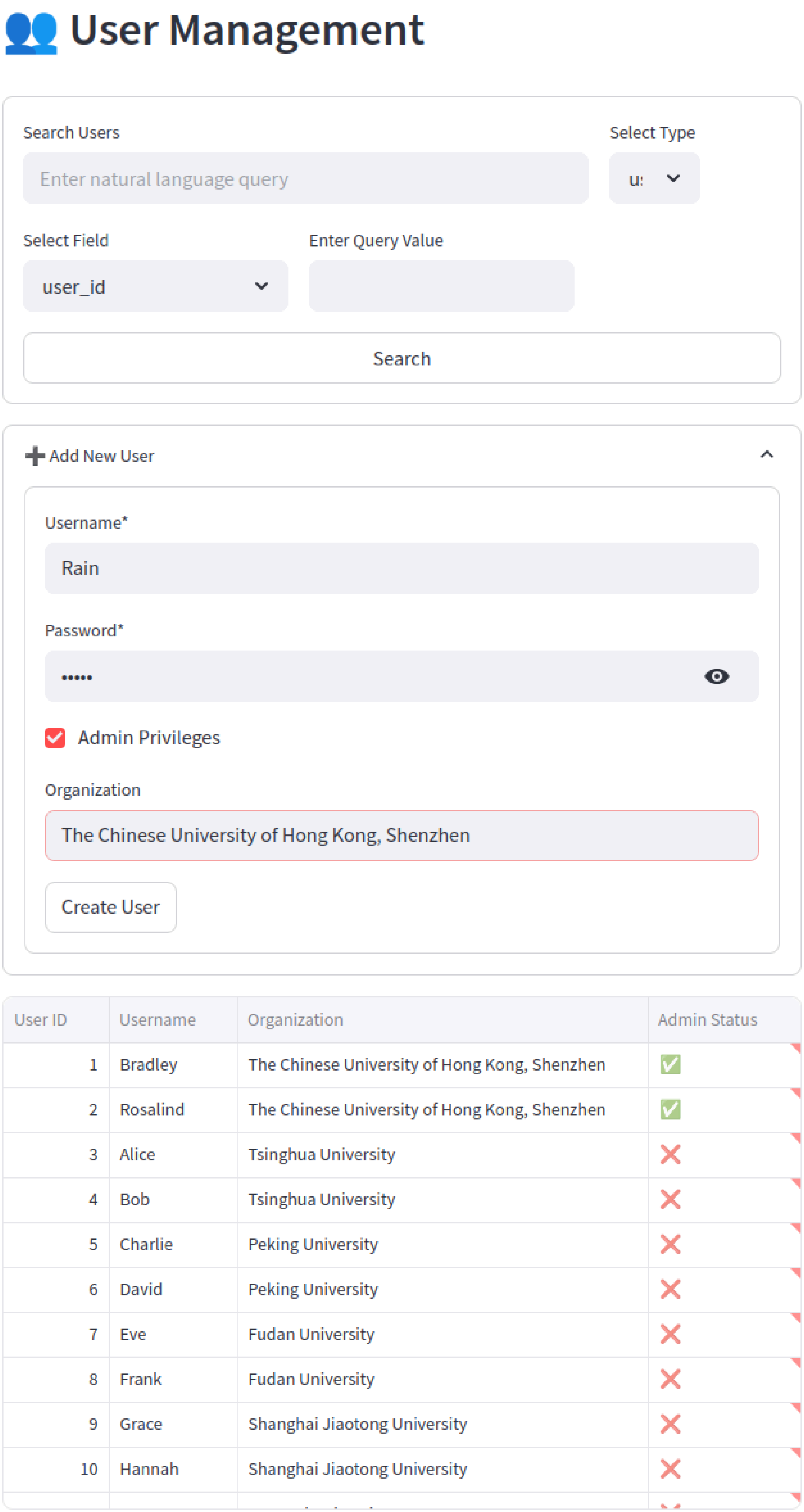
	Type	Name
0	Supported Tasks	generation
1	Authors	Eve, Grace, Katherine
2	Related Datasets	yelp_review_full

- paging are implemented for improved user experiment.

c. Datasets

d. **(Admin Privilege)** User Management

- create user



👥 User Management

Search Users

Enter natural language query

Select Type

u:

Select Field

user_id

Enter Query Value

Username*

Rain

Password*

•••••

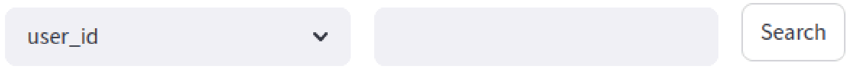
☑ Admin Privileges

Organization

The Chinese University of Hong Kong, Shenzhen

User ID	Username	Organization	Admin Status
1	Bradley	The Chinese University of Hong Kong, Shenzhen	✅
2	Rosalind	The Chinese University of Hong Kong, Shenzhen	✅
3	Alice	Tsinghua University	❌
4	Bob	Tsinghua University	❌
5	Charlie	Peking University	❌
6	David	Peking University	❌
7	Eve	Fudan University	❌
8	Frank	Fudan University	❌
9	Grace	Shanghai Jiaotong University	❌
10	Hannah	Shanghai Jiaotong University	❌

Admin features



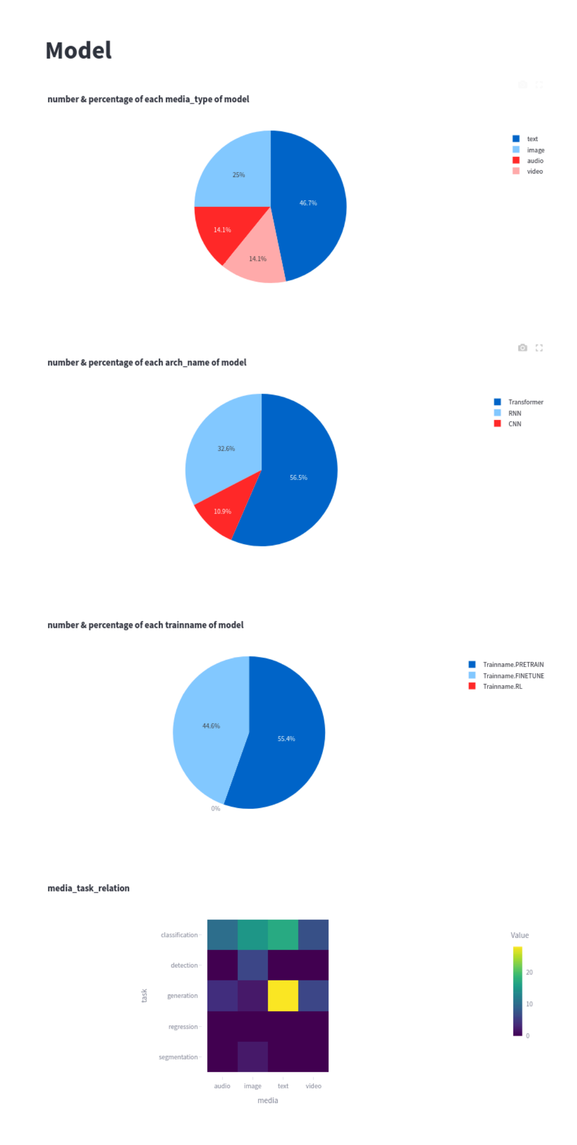
Select Field: user_id

Enter Query Value:

- edit user

e. **(Admin Privilege)** Data Insights

- illustration of the analysis on the data in the database.



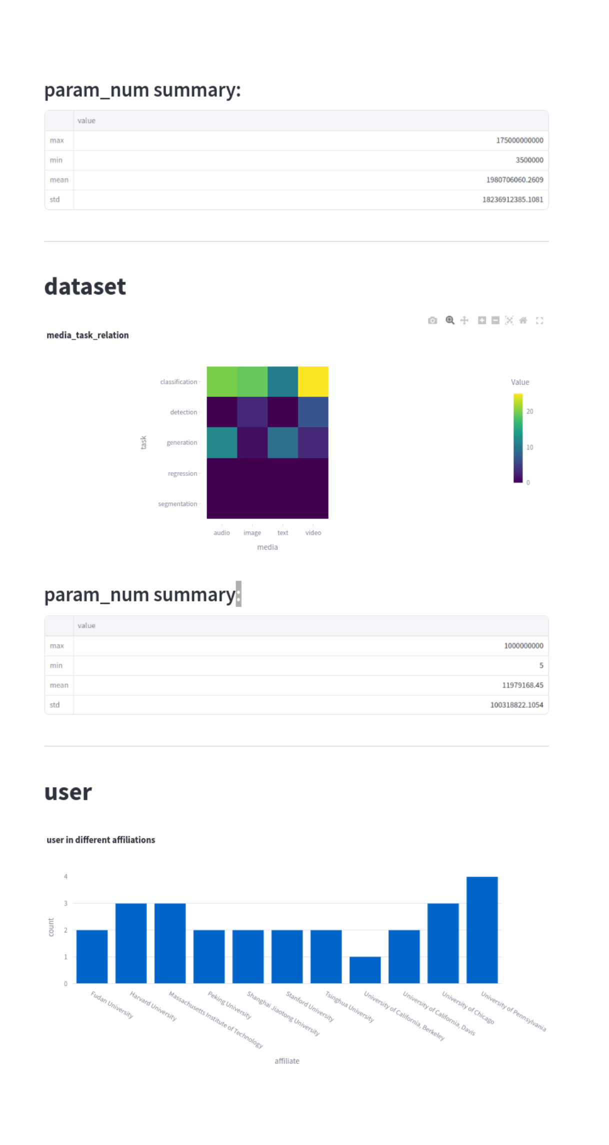
Model

number & percentage of each media_type of model

number & percentage of each arch_name of model

number & percentage of each trainname of model

media_task_relation



param_num summary:

	value
max	175000000000
min	3500000
mean	1980706060.2609
std	18236912385.1081

dataset

media_task_relation

param_num summary:

	value
max	1000000000
min	5
mean	11979168.45
std	100318822.1054

user

user in different affiliations

4
3
2
1
0
count
Fudan University
Harvard University
Massachusetts Institute of Technology
Peking University
Shanghai Jiaotong University
Stanford University
Tsinghua University
University of California, Berkeley
University of California, Davis
University of Chicago
University of Pennsylvania
affiliate

2.4. Agent

Implementation

- We incorporated gpt-4o as an LLM agent that translates user's natural language input into SQL queries.

- in `agent/agent_main.py` :

  ```
  async def natural_language_to_sql(nl_input: str) -> str:
      response = await client.chat.completions.create(
          model="gpt-4o",
          messages=[
              {"role": "system", "content": SYSTEM_PROMPT},
              {"role": "user", "content": nl_input}
          ],
          temperature=0,
      )
      return clean_sql_output(response.choices[0].message.content
  ```

- **Input/Output**
  - Input includes a natural language query, and a integer specifying the type of instance 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 gramatically correct sql is generated;
    - a SQL query generated
    - the result of the SQL query
    - corresponding code from `agent/agent_main` :

      ```
      async def query_agent(nl_input: str, verbose = False, sessi
          ret_dic = {
              'err': 0,
              'sql': '',
              'sql_res': ''
          }
          ...
          return ret_dic
      ```

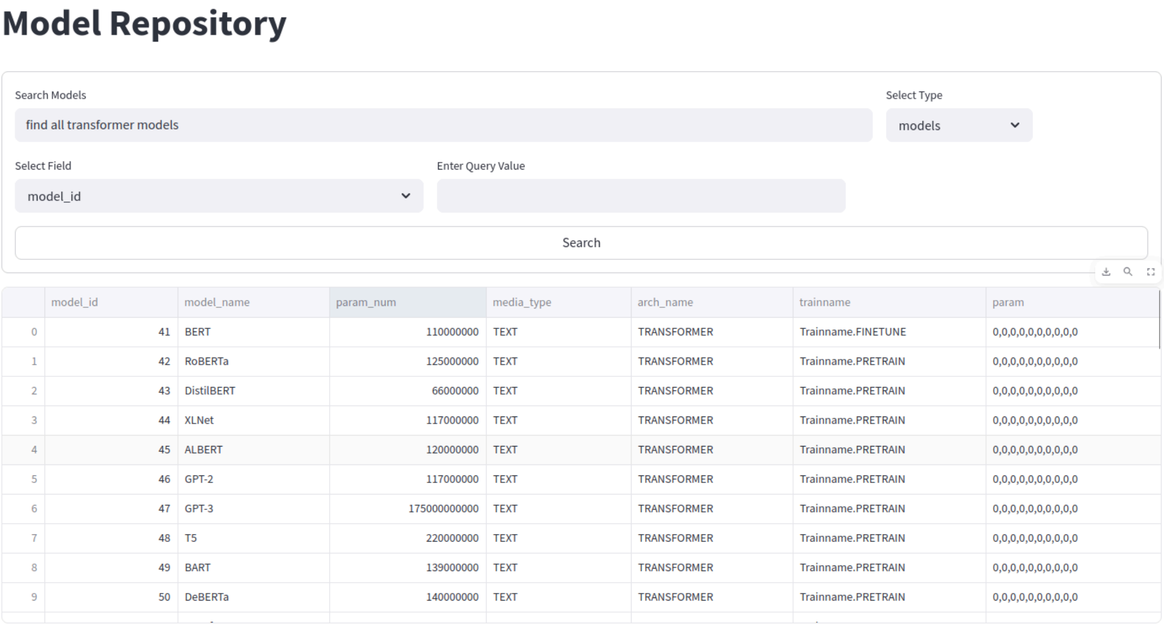
- **System prompt**
  - **Schema:** In the system prompt, we describe our database, the integrity constraints, and other information required.

    - **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 instance 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.
    - corresponding code from `agent/agent_main.query_agent()` :

```
# execute sql, attempt 1
result, error = await execute_sql(sql, session)
ret_dic["sql_res"] = result

if error:
    # generate sql, attempt 2
    print("\n⚠ Execution error, attempting to fix...")
    fixed_sql = await fix_sql_with_error(nl_input, sql, error)
    ret_dic['sql'] = fixed_sql
    if verbose: print("\n🔁 Fixed SQL:", fixed_sql)

    # execute sql, attempt 2
    result, error = await execute_sql(fixed_sql, session)
    ret_dic["sql_res"] = result

    if error:
        # define as failed if fix failed as well.
        ret_dic['err'] = 1
        if verbose: print("\n❌ Fix failed:", error)
    else:
        ret_dic['sql'] = fixed_sql
        if verbose: print("\n✅ Fix succeeded, results are as
else:
    if verbose: print("\n✅ Execution succeeded, results are a
```

Demonstration: using LLM assisted search in the GUI

1. search according to architecture: transfomer models
   - result in the table: ...



Model Repository

Search Models: find all transformer models

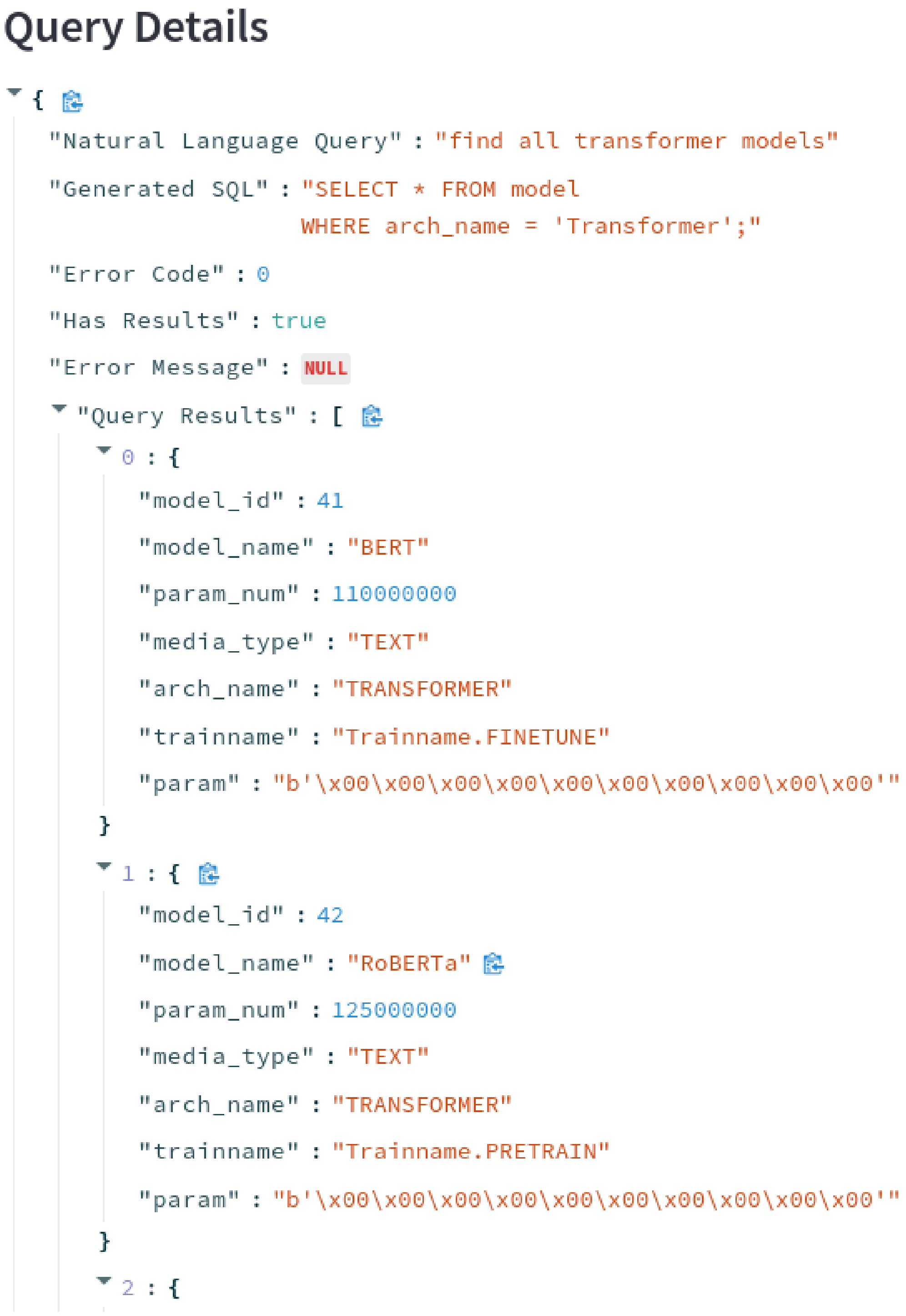
Select Type: models

Select Field: model_id

Enter Query Value:

	model_id	model_name	param_num	media_type	arch_name	trainname	param
0	41	BERT	110000000	TEXT	TRANSFORMER	Trainname.FINETUNE	0,0,0,0,0,0,0,0,0,0
1	42	RoBERTa	125000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0
2	43	DistilBERT	66000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0
3	44	XLNet	117000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0
4	45	ALBERT	120000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0
5	46	GPT-2	117000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0
6	47	GPT-3	175000000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0
7	48	T5	220000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0
8	49	BART	139000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0
9	50	DeBERTa	140000000	TEXT	TRANSFORMER	Trainname.PRETRAIN	0,0,0,0,0,0,0,0,0,0

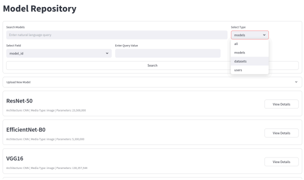
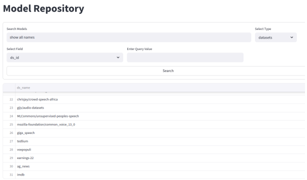
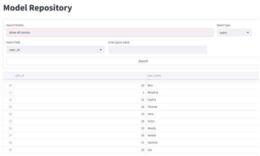
- sql query is also available:



Query Details

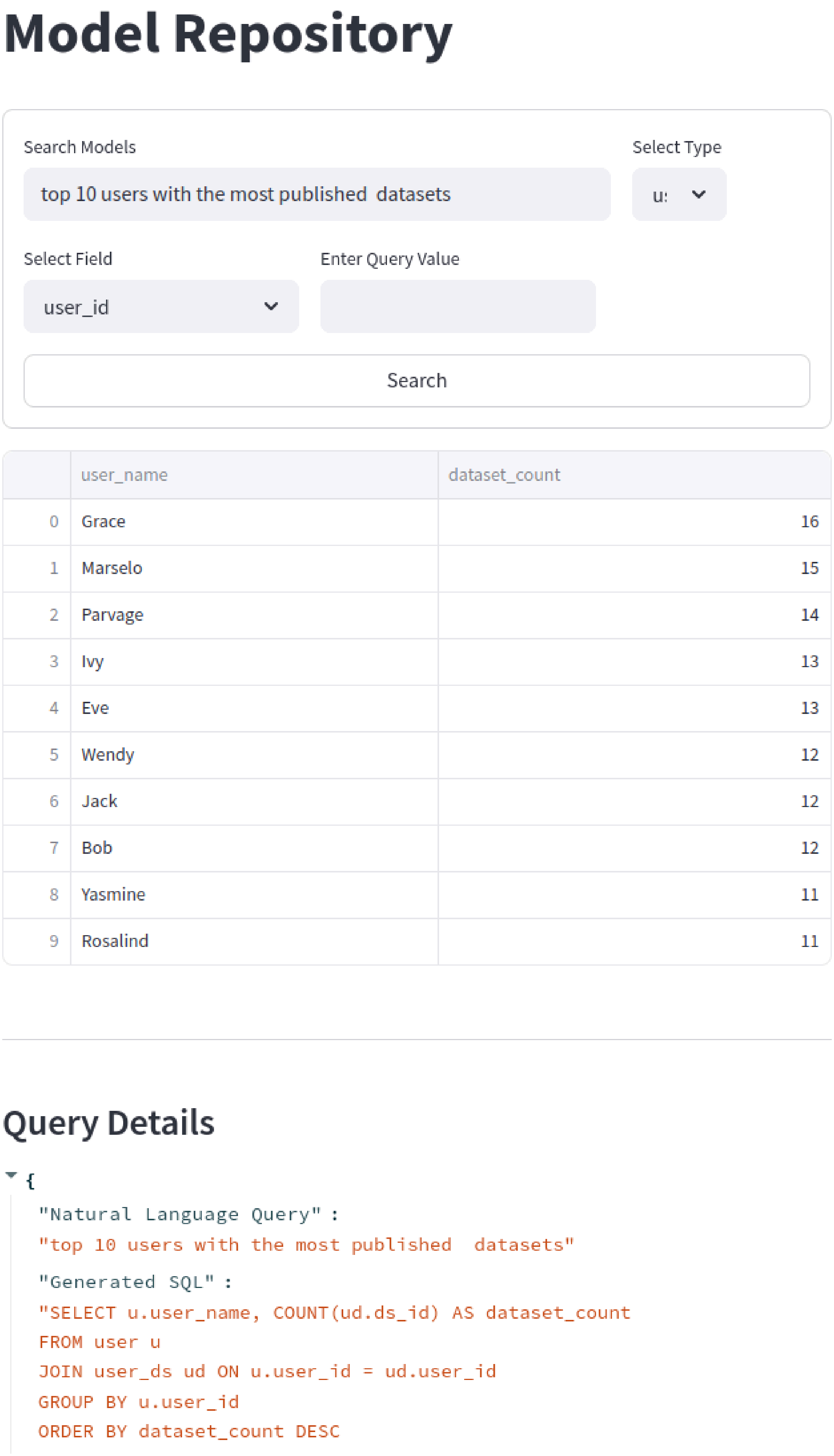
```
{
  "Natural Language Query" : "find all transformer models"
  "Generated SQL" : "SELECT * FROM model
                     WHERE arch_name = 'Transformer';"
  "Error Code" : 0
  "Has Results" : true
  "Error Message" : NULL
  "Query Results" : [
    0 : {
      "model_id" : 41
      "model_name" : "BERT"
      "param_num" : 110000000
      "media_type" : "TEXT"
      "arch_name" : "TRANSFORMER"
      "trainname" : "Trainname.FINETUNE"
      "param" : "b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'"
    }
    1 : {
      "model_id" : 42
      "model_name" : "RoBERTa"
      "param_num" : 125000000
      "media_type" : "TEXT"
      "arch_name" : "TRANSFORMER"
      "trainname" : "Trainname.PRETRAIN"
      "param" : "b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'"
    }
    2 : {
```

2. compare classified and not classified

instance type	**model**	**dataset**	**user**
query: show all names	 Model Repository Search Models Enter natural language query Select Type models all models datasets users Select Field model_id Enter Query Value Search Upload New Model ResNet-50 Architecture: CNN | Media Type: image | Parameters: 23,500,000 View Details EfficientNet-B0 Architecture: CNN | Media Type: image | Parameters: 5,300,000 View Details VGG16 Architecture: CNN | Media Type: image | Parameters: 138,357,544 View Details	 Model Repository Search Models show all names Select Type datasets Select Field ds_id Enter Query Value Search ds_name 22 chrisjay/crowd-speech-africa 23 gijs/audio-datasets 24 MLCommons/unsupervised-peoples-speech 25 mozilla-foundation/common_voice_13_0 26 giga_speech 27 tedlium 28 voxpopuli 29 earnings-22 30 ag_news 31 imdb	 Model Repository Search Models show all names Select Type users Select Field user_id Enter Query Value Search user_id user_name 20 20 Rico 21 2 Rosalind 22 21 Sophia 23 22 Thomas 24 23 Uma 25 24 Victor 26 25 Wendy 27 26 Xander 28 27 Yasmine 29 28 Zoe

3. more complicated search: ranking

- query: top 10 users with the most published datasets



Model Repository

Search Models: top 10 users with the most published datasets

Select Type: u:

Select Field: user_id

Enter Query Value:

	user_name	dataset_count
0	Grace	16
1	Marselo	15
2	Parvage	14
3	Ivy	13
4	Eve	13
5	Wendy	12
6	Jack	12
7	Bob	12
8	Yasmine	11
9	Rosalind	11

Query Details

```
{
  "Natural Language Query" :
  "top 10 users with the most published  datasets"
  "Generated SQL" :
  "SELECT u.user_name, COUNT(ud.ds_id) AS dataset_count
  FROM user u
  JOIN user_ds ud ON u.user_id = ud.user_id
  GROUP BY u.user_id
  ORDER BY dataset_count DESC
```

```
LIMIT 10;"
"Error Code" : 0
```

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