Financial Hack Extraction and Validation Tool for Hintsly

Project Overview

This project is an AI-driven tool designed to extract and validate financial hacks from internet sources. It processes relevant information to build detailed descriptions of the hacks, including the necessaries resources, steps to implement and expected outcomes. The system also tags the content's style, topics, complexity, and target audience.

The code is structured to be used as the back-end of an API, which retrieves content from the web, processes it using AI models, and stores the results in a PostgreSQL database. This database forms a library of validated financial hacks, which can be used by front-end web applications or services.

Directory Structure

```
project_root/
├── data/                              # Directory for data storage (not
included in version control)
├── src/
│   ├── prompts/                       # Contains text prompt templates
used by models
│   ├── handle_db.py                   # Functions to interact with
PostgreSQL database (e.g., read/write queries)
│   ├── llm_models.py                  # Large Language Model (LLM)
functionalities
│   ├── process_and_validate.py        # Core functions to process and
validate a single hack
│   ├── process_from_db.py             # Functions to process hacks in
batches from the database
│   └── settings.py                    # Configuration settings (paths and
prompt templates)
├── .env                               # File to store environment
variables (API keys, database URL)
├── requirements.txt                   # List of Python dependencies
└── README.md                          # Project documentation
```

Key Files and Their Roles

- `data/`: (Optional) Directory for storing any local data files or processed outputs.
- `src/prompts/`: Stores prompt templates that guide the LLMs when generating or analyzing financial hack descriptions.
- `src/handle_db.py`: Provides functions to interact with a PostgreSQL database, enabling reading from and writing to various tables used in hack validation, analysis, and classification.

- `src/llm_models.py`: Defines the LLMs used for processing and validating text. This module handles text generation, embedding creation, and similarity searches.
- `src/process_and_validate.py`: Contains the core logic to process and validate a single hack, including verifying the hack, generating validation queries, and performing deep analyses.
- `src/process_from_db.py`: Facilitates batch processing of hacks stored in the database, utilizing the functions defined in `process_and_validate.py` and `handle_db.py`.
- `src/settings.py`: Stores project-wide configuration, such as paths to prompt templates and environment settings.

Dependencies and Environment Setup

1. **Clone the repository**

```
git clone https://github.com/LignumBlocks/Hintsly.git
cd project
```

2. **Set up a virtual environment**

```
python3 -m venv venv
source venv/bin/activate    # For Linux/Mac
# or
venv\Scripts\activate       # For Windows
```

3. **Install dependencies**

Install the required Python libraries using `pip`:

```
pip install -r requirements.txt
```

4. **Configure environment variables**

Create a `.env` file in the project root to store your API keys and database connection URL:

```
OPENAI_API_KEY=your_openai_api_key
GOOGLE_API_KEY=your_google_api_key
DATABASE_URL=your_postgres_db_url
```

How to Run the Project

The project is designed to process financial hacks from internet sources using pre-built models and store them in a PostgreSQL database.

1. **Hacks Verification**

This function checks if the transcription content in the table `transcriptions` contains a hack, for each unprocessed transcription.

```
from process_from_db import hacks_verification
hacks_verification()
```

2. **Generates validation queries**

This function generates validation queries based on the hack title and summary, for each verified hack that does not have queries yet.

```
from process_from_db import get_queries
get_queries()
```

3. **Hacks Validation**

Validates financial hacks by using validation sources, analyzing them with a RAG model, for each hack from the 'validation_sources' table that has not been validated yet. It checks each hack's legitimacy and stores the results in the `validation` table.

```
from process_from_db import validate_hacks
validate_hacks()
```

4. **Analyze Validated Hacks**

This function analyzes validated hacks, builds comprehensive descriptions, and stores the analysis in the `hack_descriptions` table, for each validated hack with no descriptions. It also structures the hack’s descriptions for easier retrieval.

```
from process_from_db import analyze_validated_hacks
analyze_validated_hacks()
```

5. **Classify Hacks**

Once the hacks are analyzed, this function classifies them based on several parameters that generate tags about the the content, style and target audience of the financial hack. Only for hacks with descriptions that have not been classified yet.

```
from process_from_db import classify_hacks
classify_hacks()
```

Functionality Overview

1. **LLM Models (`src/llm_models.py`)**

The core AI component leverages various large language models (LLMs) and tools like RAG with Chroma vector store and chat history to use them to process the hacks.

2. **Database Interaction (`src/handle_db.py`)**

This module defines the logic to interact with the PostgreSQL database. It provides utility functions such as `read_from_postgres` and `execute_in_postgres` to retrieve data and insert results into the database.

3. **Hack Processing (`src/process_and_validate.py`)**

The `process_and_validate.py` file contains the functions that load the corresonding prompt template, call the LLM model and clean the result for each one of the 5 tasks.

4. **process hacks from the database (`src/process_from_db.py`)**

This file retrieves for each task the data from the database, applies the corresponding `process_and_validate.py` function and saves the result to the database.

Troubleshooting

1. **Database Connection Errors**:
    - Ensure that the `DATABASE_URL` in your `.env` file is correct and your PostgreSQL database is running.
    - Check that the database schema matches the structure expected by the code.
2. **API Key Errors**:
    - If you're seeing issues related to API key usage, ensure that your `.env` file contains valid API keys for OpenAI and Google.
3. **Dependencies Not Installed**:
    - Make sure you have installed all dependencies from `requirements.txt` using the virtual environment.
