SmartHug

This project is a precursor to a comprehensive internal enterprise knowledge base, outlined by four key objectives to construct an efficient knowledge management system:

1. **Systematic Model Selection**: Developing a system for evaluating and selecting the most optimal vector encoding and language models for question answering. This system not only facilitates the assessment and selection of the best models for immediate use but also ensures the future adaptability of the system to incorporate newly emerging superior models.

2. **Article Segmentation Testing**: Optimizing the segmentation of articles to achieve efficient information processing and retrieval from vast text datasets, a critical factor for the knowledge base's effectiveness.

3. **Vector Database Evaluation**: Conducting thorough evaluations of vector databases' performance and scalability to confirm the infrastructure's capability to manage the extensive data integral to the knowledge base.

4. **Prototype Development**: Creating a working prototype to showcase the project's potential, stimulate stakeholder discussions, solicit feedback, and foster ongoing enhancements.

These objectives are designed to forge a scalable, adaptable, and high-quality knowledge management system, focusing on a robust selection process for models and ensuring the system's long-term viability through continuous innovation and improvement.

SmartHug offers a framework for building an enterprise knowledge base, streamlining the integration of encoding and language models with a universal interface. It simplifies model evaluation, starting with tests on six notable models, including chatGPT and 文心一言, suitable for both online and local use.

The system features advanced article segmentation, allowing users to customize chunk sizes and overlaps for ideal data management and retrieval from large datasets.

SmartHug also lets users test the Milvus vector database's performance by uploading many documents and conducting vector searches.

With Gradio technology, SmartHug brings these functions into an easy-to-use prototype, providing a platform for user feedback and ongoing updates. This approach highlights the system's efficiency in handling complex data and its potential to improve organizational knowledge management, focusing on adaptability and efficiency.

Moreover, SmartHug's design encourages the exploration and testing of new models through a clear interface, aiding in the discovery of models that best suit the knowledge base's needs. This process is key to developing a scalable and efficient knowledge base system.# Installation

Follow these steps to set up the project environment and dependencies on your system:

1. Configure Python Environment

First, ensure you have Conda installed on your system. If not, download and install it from the official Conda website. Once Conda is installed, create a new Python 3.10 environment:

```
conda create --name smarthug python=3.10
conda activate smarthug
```

2. Clone the Project Repository

Clone the project repository from GitHub to your local machine. Open a terminal and run:

```
git clone https://github.com/hyffffff/SmartHug/SmartHug.git
```

Make sure to replace `<your-username>` and `<project-name>` with the appropriate GitHub username and the name of the project repository.

3. Install PyTorch with CUDA

Install PyTorch with CUDA 11.8 support. This step is crucial for leveraging GPU acceleration (if available) for model computations. Run the following command:

```
conda install pytorch torchvision torchaudio cudatoolkit=11.8 -c pytorch
```

This command installs PyTorch along with torchvision and torchaudio for multimedia processing, all compatible with CUDA 11.8.

4. Install Project Dependencies

Navigate to your project directory and install the remaining Python dependencies using pip:

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

5. Install Milvus Server

The project utilizes Milvus, a highly scalable vector database, for managing and querying vectorized text data. Follow the official Milvus installation guide to set up Milvus on your system. The guide provides detailed instructions for Docker-based and standalone installations.

After completing these steps, your environment should be ready for running and testing the project's functionalities.

Please ensure your system meets all hardware and software prerequisites for the above installations, especially the requirements for running Milvus and CUDA-enabled PyTorch.

6 Configuration

To configure the environment for the project, you will need to set up a `.env` file that stores all the necessary environment variables, including API keys and model configurations. Follow the steps below to properly set up your environment:

1. Create a `.env` File

Navigate to the root directory of your project and create a new file named `.env`. This file will store all your secret keys and configurations.

2. Add API Keys

You will need to obtain API keys from various services to use different models and functionalities. Add the following lines to your `.env` file, replacing the placeholders with your actual API keys:

```
# API Keys
glmapi_key = 'YOUR_GLM_API_KEY'
erniebot.api_type = "aistudio"
erniebot.access_token = "YOUR_ERNIEBOT_ACCESS_TOKEN"
OPENAI_API_KEY="YOUR_OPENAI_API_KEY"
```

To obtain these keys:

- For `glmapi_key`, apply at https://open.bigmodel.cn/usercenter/apikeys.
- For `erniebot.access_token`, visit https://aistudio.baidu.com/index/accessToken.
- For `OPENAI_API_KEY`, register and obtain a key from https://openai.com/.

3. Network Configuration

If you're behind a proxy or need to specify no-proxy hosts, configure them as follows:

```
NO_PROXY=127.0.0.1,localhost
```

4. Model Configuration

Select your desired models for encoding and answering by setting their IDs in the `.env` file. Each model corresponds to a number as indicated in the comments:

```
ANSWERMODEL = 1  # Wenxin Yiyi 3.5
ENCODEMODEL = 2  # Simple Local
```

5. Additional Settings

Specify the name of your vector database collection and the document segmentation settings for chunk length and overlap length:

```
KBCollection = "KBCollection"
Chunk_length = 200
Overlap_length = 40
```

6. Save the `.env` File

After adding all the necessary configurations to the `.env` file, save and close the file. These settings will be automatically loaded by the application at runtime.

Make sure not to share your `.env` file or disclose its contents, as it contains sensitive API keys and configurations.

Running the Application

After setting up your environment and configuring the `.env` file, you can run the application with its Gradio interface to interact with the models in a user-friendly way.

Starting the Application

Launch the Gradio web interface by running the `smarthug.py` script:

```
python smarthug.py
```

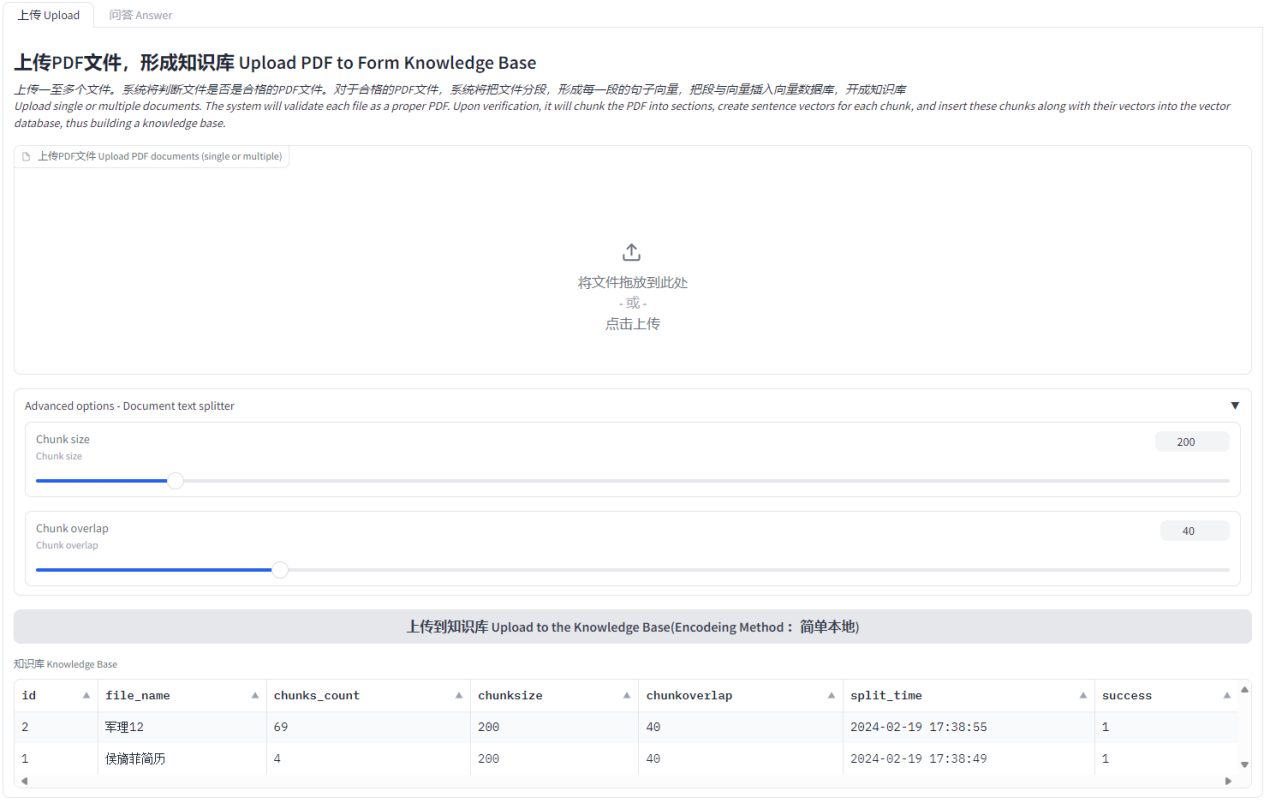
Gradio will start a web server and provide a URL to access the interface, usually something like `http://127.0.0.1:7860/`. Open this URL in your web browser to interact with the application.

Gradio Interface Overview

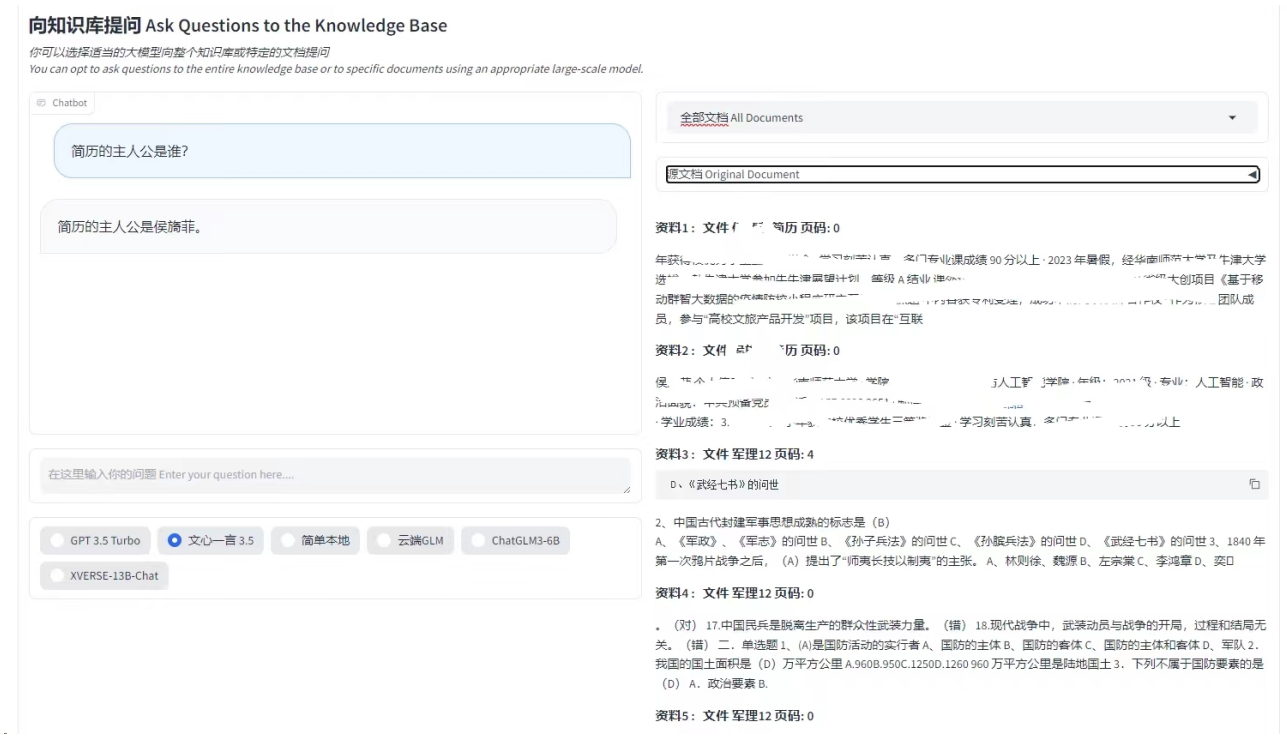
The Gradio interface is divided into two main tabs for ease of use:

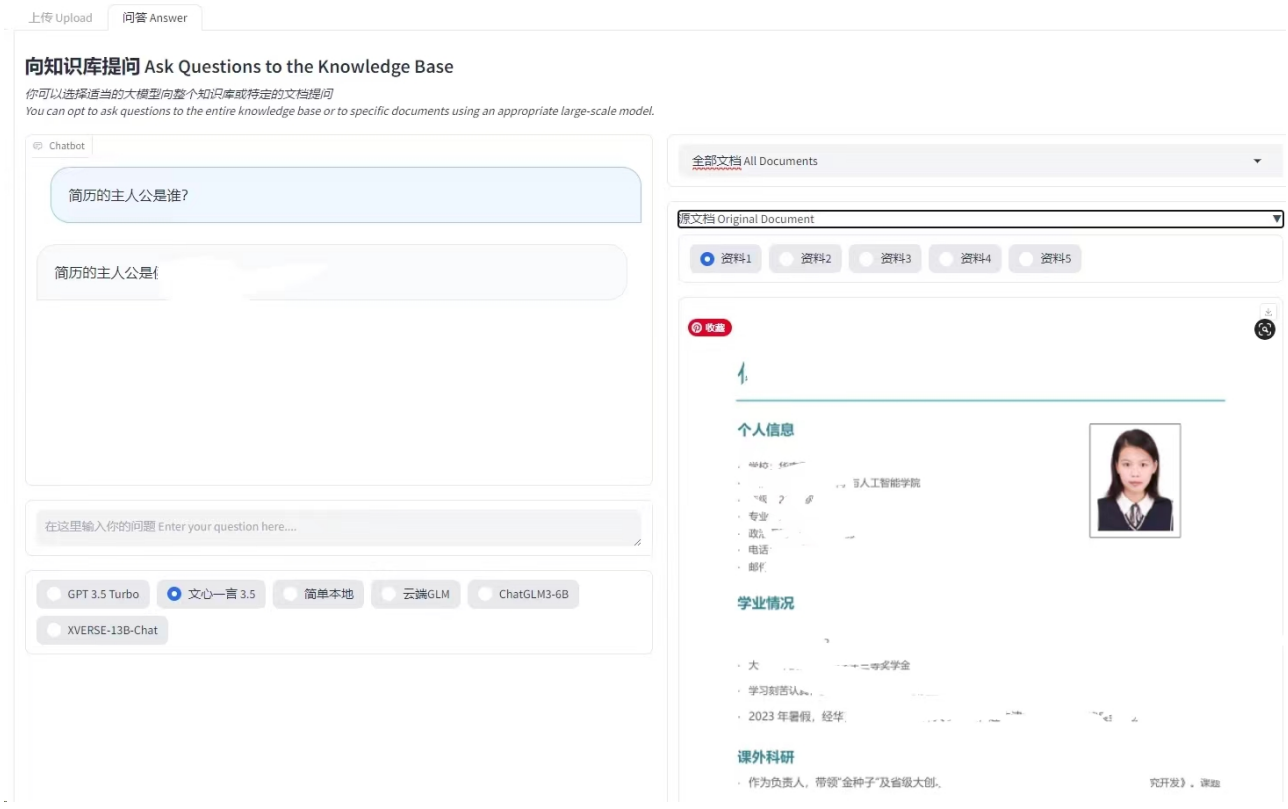
1. **Document Upload and Processing**: Users can upload PDF files, which are then processed to form the knowledge base. Advanced options allow for customization of document splitting to optimize text

segmentation.



2. **Knowledge Base Query and Answer**: This section enables users to ask questions and receive answers based on the knowledge base. The right side of the screen displays related text from the knowledge base, and users can view the original documents by clicking on "Original Document."





Features and Interactions

- **Flexible Model Selection**: Choose from various models for text encoding and question answering.
- **Customizable Text Segmentation**: Adjust chunk lengths and overlaps for optimal document processing.
- **Interactive Q&A**: Submit questions and receive contextually relevant answers.
- **Document Management**: Upload and view original documents for comparison and context.

Stopping the Application

To stop the application and the Gradio web server, press `Ctrl + C` in the terminal.