Figma LLM Library

Welcome to the Figma LLM Library - the ultimate way to take your Figma project to the next level.

Key Features

- **Versatile LLM Integration**: Connect effortlessly with leading LLM providers like Llama, ensuring broad compatibility and flexibility.
- **Efficient Embedding Storage**: Utilize the `EmbeddingStorage` class for optimal management of embeddings, enabling quick storage and retrieval.
- **Flexible Embedding Extraction**: The `EmbeddingModel` class provides a streamlined process for extracting embeddings, adaptable to various requirements.
- **Retrieval-Augmented Generation**: Elevate LLM outputs with RAG capabilities, enriching responses with contextually relevant information.
- **Customizable and Extensible**: A modular architecture allows for easy customization and extension to fit your specific needs.
- **Comprehensive Documentation**: Full access to detailed documentation and illustrative examples, empowering you to make the most of the library.

Project Structure

The library's structure is designed for ease of navigation and extendibility:

```
figma_llm/
├── embeds/              # Embedding handling
│   ├── db.py            # EmbeddingStorage class for embeddings management
│   └── extract.py       # EmbeddingModel class for extraction
├── models/              # LLM interaction
│   └── llm_manager.py   # Manages LLM interactions and RAG workflows
├── server/              # Server setup and web interface
│   ├── run.py           # Server startup script
│   ├── web_api.py       # Server startup script
│   └── templates/       # HTML templates for the web interface
└── utils/               # Utility functions and classes
    ├── config.py        # Configuration settings
    ├── distances.py     # Distance calculation utilities
    └── txt_chunk.py     # Text chunking utilities
├── app.py               # Application entry point
```

- `embeds/`: Contains modules for embedding storage and extraction.
  - `db.py`: Defines the EmbeddingStorage class for storing and retrieving embeddings.
  - `extract.py`: Defines the EmbeddingModel class for extracting embeddings from text.
- `models/`: Contains the LLM-related modules.

  - `llm_manager.py`: Defines the LLMManager class for interacting with the Llama library and managing the RAG workflow.
- `server/`: Contains the server-related modules.
  - `web_api.py`: Defines the API endpoints and routes for the web interface.
  - `run.py`: A script for running the web server.
  - `templates/`: Contains HTML templates for the web interface.
- `utils/`: Contains utility modules.
  - `config.py`: Defines the configuration classes for the application.
  - `distances.py`: Provides distance calculation functions for similarity search.
  - `txt_chunk.py`: Provides text chunking utilities.
- `app.py`: The main entry point of the application.

Getting Started

Installation

1. **Clone the repository**: `bash git clone https://github.com/yourusername/figma-llm.git cd figma-llm`
2. **Install dependencies**: `bash pip install -r requirements.txt`

Configuration

Customize your setup by adjusting the `Config` class in `figma_llm/utils/config.py`:

- `MODEL_PATH`: Path to the LLM model file.
- `DEFAULT_EMBEDDINGS`: Default settings for embedding models.

Running the Application

To launch the Figma LLM server:

```
python figma_llm/app.py --host localhost --port 5000
```

Access the web interface by navigating to http://localhost:5000 in your browser.

API Endpoints

Explore the versatile endpoints the Figma LLM server offers:

- **GET /**: Access the welcoming face of the web interface, designed to be intuitive and user-friendly, guiding you through the process of inputting prompts and viewing responses.
- **POST /**: Submit your linguistic queries through this endpoint. It expects a `prompt` parameter in the request form data, utilizing the power of LLM to generate insightful and contextually relevant responses.
- **POST /stream**: For continuous interaction with the LLM, this endpoint offers a streaming response capability. It's perfect for applications requiring real-time feedback, dynamically updating the response as more data becomes available.

Embedding and Storing Documents

Efficiently embedding and storing documents is a cornerstone of leveraging LLMs for enhanced language understanding. Here's how you can do it with the Figma LLM Library:

1. **Document Embedding**: Transform textual content into rich, meaningful embeddings that capture the essence and context of your documents. This process is streamlined and efficient, thanks to the `EmbeddingModel` class.
2. **Storing for Retrieval**: With the `EmbeddingStorage` class, these embeddings are not just processed but also systematically stored, making them readily available for future retrieval. This is essential for applications relying on quick access to pre-processed information.

```
documents = [
    "The capital of France is Paris.",
    "The Eiffel Tower is a famous landmark in Paris.",
]

for doc in documents:
    llm_manager.embed_and_store(doc)
llm_manager.embedding_storage._save_to_file()
```

This simple yet powerful functionality forms the backbone of applications that require rapid access to pre-computed linguistic insights, making your development process smoother and more effective.

Generating Responses with RAG

The Retrieval-Augmented Generation (RAG) feature is a game-changer, enhancing the quality of LLM-generated responses by providing relevant context. Here's how you can utilize RAG to create responses that are not only accurate but also rich in detail and relevance:

1. **Contextual Relevance**: By retrieving related embeddings from your stored documents, RAG ensures that the generated responses are not just generic but tailored to the specific context of the query.
2. **Leveraging Embeddings**: The embeddings act as a bridge between the query and the vast knowledge encapsulated in your dataset, allowing for responses that demonstrate a deep understanding of the subject matter.

```
query = "Tell me about a famous landmark in France."
top_similar = llm_manager.embedding_storage.find_top_n(llm_manager.embedding_model.embed(query),
n=2)

context = "\n".join([text for _, _, text in top_similar])
rag_response = llm_manager.generate_response(query, context=context)
```