r1-RAG-Agent Documentation

Ahmed Hoda

This project provides a framework for building a conversational agent leveraging Retrieval-Augmented Generation (RAG). The system uses PDF documents as a knowledge base, extracting content to answer user queries. The key components of this framework are the scripts `r1_smolagent_rag.py` and `ingest_pdfs.py`, which work together to load, process, store, and query document-based data.

---

1. `r1_smolagent_rag.py` - RAG Agent Configuration

Overview

This script initializes and configures the models and tools for the RAG-based conversational agent. It sets up environment variables, loads models from Hugging Face or OpenAI, and configures the reasoning model to enhance the agent's ability to generate relevant responses.

Key Components

1.1. Environment Configuration

- Loads environment variables from a `.env` file using the `dotenv` module.
- Critical environment variables include:
    - `REASONING_MODEL_ID`: Model ID for the reasoning model.
    - `TOOL_MODEL_ID`: Model ID for the tool-calling agent.
    - `HUGGINGFACE_API_TOKEN`: Hugging Face API token for model access.
    - `USE_HUGGINGFACE`: A flag determining whether to utilize Hugging Face models (`yes` by default).

1.2. Model Initialization

- The `get_model` function initializes models according to the environment configuration:
    - If `USE_HUGGINGFACE` is set to `yes`, it utilizes Hugging Face's `HfApiModel`.
    - Otherwise, it defaults to using a locally hosted `OpenAIServerModel` (via an Ollama server).

1.3. Reasoning Model

- The reasoning model is instantiated through the `CodeAgent` class from the `smolagents` module. This model enables context-based response generation, leveraging retrieved document data.

1.4. Vector Store and Embeddings

- A Chroma-based vector store is set up, with embeddings derived from Hugging Face's `sentence-transformers/all-mpnet-base-v2`.
- The vector store is persisted in the `chroma_db` directory.

1.5. RAG Tool Integration

- The `rag_with_reasoner` tool, defined with the `@tool` decorator, orchestrates the query-processing pipeline:
    - Takes a user query.
    - Retrieves relevant documents from the vector store.
    - Generates a response by reasoning over the context.

1.6. Primary Agent

- The `ToolCallingAgent` acts as the main conversational agent, w. Interacts with the `rag_with_reasoner` tool to process queries and deliver responses.

1.7. Gradio UI

- The script launches a Gradio interface (`GradioUI` from `smolagents`), enabling users to interact with the agent via a simple web interface.

---

2. `ingest_pdfs.py` - PDF Ingestion and Processing

Overview

This script processes PDF documents, splitting them into manageable chunks and storing them in a vector store for later retrieval by the RAG agent.

Key Components

2.1. Environment Configuration

- Loads environment variables from the `.env` file to configure PDF ingestion and vector storage.

2.2. PDF Processing

- The `load_and_process_pdfs` function loads PDFs from a specified directory using the `DirectoryLoader` and `PyPDFLoader`.
- PDFs are split into text chunks using `RecursiveCharacterTextSplitter` with a chunk size of 1000 characters and an overlap of 200 characters.

2.3. Vector Store Creation

- The `create_vector_store` function creates and persists a Chroma vector store.
    - If a vector store already exists, it is cleared before creating a new one.
    - The store is populated with embeddings generated using Hugging Face's `sentence-transformers/all-mpnet-base-v2`.

2.4. Main Processing Pipeline

- The `main` function defines the directories for input data (`data`) and output vector storage (`chroma_db`).
- It processes PDFs and stores the resulting embeddings in the vector store.

---

3. Workflow

1. **Ingest PDFs:**
    - Run `ingest_pdfs.py` to load, process, and chunk PDFs from the `data` directory, storing them in the `chroma_db` vector store.
2. **Set Up RAG Agent:**
    - Execute `r1_smolagent_rag.py` to initialize the RAG agent, which uses the vector store to retrieve and respond to user queries.
3. **Interact with the Agent:**
    - Use the Gradio UI to input queries. The agent responds based on the context provided by the ingested PDFs.

---

4. Dependencies

Python Packages:

- `smolagents`: Provides agent models and tools for conversational AI.
- `dotenv`: Loads environment variables from `.env`.
- `langchain_chroma`: LangChain integration for Chroma vector stores.
- `langchain_huggingface`: LangChain embeddings support via Hugging Face.
- `langchain_community.document_loaders`: Document loaders for PDF ingestion.
- `langchain.text_splitter`: Text splitting utilities for document chunking.

Required Environment Variables:

- `REASONING_MODEL_ID`: Reasoning model ID.
- `TOOL_MODEL_ID`: Tool-calling agent model ID.
- `HUGGINGFACE_API_TOKEN`: API token for Hugging Face models.
- `USE_HUGGINGFACE`: Whether to use Hugging Face models (`yes` by default).

---

5. Setup and Usage

1. Set Up Environment:

- Create a `.env` file with the necessary environment variables.

Install dependencies via `pip`:

- pip install -r requirements.txt

2. Ingest PDFs:

- Place your PDF documents in the `data` directory.

Run the ingestion script to process and store PDFs:

- python ingest_pdfs.py

3. Run the RAG Agent:

Start the agent and Gradio interface:

- python r1_smolagent_rag.py

4. Example Query:

- Input a query into the Gradio interface (e.g., "Compare and contrast the services offered by RankBoost and Omni Marketing").
- The agent retrieves relevant documents from the vector store and generates a contextual response.

6. Conclusion

This repository provides an efficient framework for building document-based conversational agents using Retrieval-Augmented Generation (RAG). By integrating PDF ingestion, vector storage, and advanced reasoning models, enables accurate and contextually relevant responses to user queries based on the content of documents.