RAG-based Chatbot Documentation

This documentation provides a comprehensive overview of the Retrieval-Augmented Generation (RAG)-based chatbot project. The chatbot leverages a FAISS vector store, SentenceTransformers for embeddings, and HuggingFace's transformers library for answering questions.

Overview

Project Objective

To create a chatbot that answers user queries by retrieving relevant knowledge from a dataset and generating responses using a Retrieval-Augmented Generation (RAG) pipeline.

Features

1. **Knowledge Base Retrieval**: Uses a dataset as the knowledge source.
2. **RAG Pipeline**: Combines similarity search with a question-answering model.
3. **Browser-Based Interface**: Deploys the chatbot via Streamlit.
4. **Efficient Reusability**: Saves the RAG pipeline to avoid repeated setup.

Technologies Used

1. **Python Libraries**:
    - **LangChain**: For document processing and FAISS integration.
    - **FAISS**: For fast similarity searches on vector embeddings.
    - **SentenceTransformers**: For generating embeddings from text.
    - **HuggingFace Transformers**: For the question-answering model.
    - **Streamlit**: For building a web-based chatbot interface.
2. **Dataset**:
    - A CSV file (`further_reduced_train.csv`) serves as the knowledge base.

Project Structure

```
rag_chatbot/
│
├── data/
│   └── further_reduced_train.csv      # Dataset file
│
├── src/
│   ├── __init__.py                    # Marks src as a package
│   ├── data_loader.py                 # Loads the dataset
│   ├── rag_pipeline.py                # RAG pipeline setup and
saving/loading
│   ├── chatbot.py                     # Command-line chatbot
│   └── app.py                         # Streamlit-based chatbot
│
├── .env                               # Optional: Environment
variables
├── requirements.txt                   # Python dependencies
├── README.md                          # Instructions to run the
project
└── .gitignore                         # Files to ignore in version
control
```

Setup Instructions

1. Prerequisites

- Python 3.8 or higher installed on your system.
- A virtual environment is recommended to avoid dependency conflicts.

2. Clone the Repository

```
git clone <repository_url>
cd rag_chatbot
```

3. Install Dependencies

Install the required Python libraries using `requirements.txt`:

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

4. Place the Dataset

Place the `further_reduced_train.csv` file in the `data/` directory.

5. Set Up the RAG Pipeline

Run the `rag_pipeline.py` script to process the dataset and save the RAG pipeline:

```
python src/rag_pipeline.py
```

6. Run the Chatbot

Command-Line Chatbot:

Run the chatbot in your terminal:

```
python src/chatbot.py
```

Streamlit App:

Launch the Streamlit-based chatbot in your browser:

```
streamlit run src/app.py
```

Visit the app at `http://localhost:8501`.

File Details

1. `data/further_reduced_train.csv`

- The dataset containing knowledge for the chatbot.
- Columns:
    - `text`: The main content used as the knowledge base.
    - `question`: Questions extracted from the content (for context).
    - `answer`: Corresponding answers to the questions.

2. `data_loader.py`

Loads and preprocesses the dataset for use in the RAG pipeline.

```
import pandas as pd

def load_data(file_path):
    """
    Loads a CSV dataset and prepares it for RAG.

    Args:
        file_path (str): Path to the CSV file.

    Returns:
        pd.DataFrame: Loaded dataset.
    """
    data = pd.read_csv(file_path)
    print(f"Loaded {len(data)} records from {file_path}.")
    return data
```

3. `rag_pipeline.py`

Handles the setup, saving, and loading of the RAG pipeline:

- Embeddings are created using `SentenceTransformer`.
- FAISS is used to store and query embeddings.

Key Functions:

- `setup_rag_pipeline`: Processes the dataset and saves the pipeline.
- `load_rag_pipeline`: Loads the saved pipeline for reuse.

4. `chatbot.py`

A terminal-based chatbot:

- Prompts the user for input.
- Retrieves relevant documents using FAISS.
- Generates answers with HuggingFace's question-answering model.

```
from rag_pipeline import load_rag_pipeline
from transformers import pipeline

def main():
    """
    Runs the chatbot in the terminal.
    """
    vectorstore = load_rag_pipeline("rag_pipeline")
    qa_model = pipeline("question-answering",
model="distilbert-base-uncased-distilled-squad")

    print("Chatbot is ready! Type 'exit' to quit.")
    while True:
        query = input("You: ")
        if query.lower() == "exit":
            print("Goodbye!")
            break

        docs = vectorstore.similarity_search(query, k=3)
        context = " ".join([doc.page_content for doc in docs])
        result = qa_model(question=query, context=context)
        print(f"Bot: {result['answer']}")
```

5. `app.py`

A Streamlit-based web app:

- Loads the RAG pipeline and QA model.
- Allows users to input questions and view answers in a browser interface.

```
import streamlit as st
from rag_pipeline import load_rag_pipeline
from transformers import pipeline

@st.cache_resource
def initialize_pipeline():
    vectorstore = load_rag_pipeline("rag_pipeline")
    qa_model = pipeline("question-answering",
model="distilbert-base-uncased-distilled-squad")
```

```
    return vectorstore, qa_model

def main():
    st.title("RAG-based Chatbot")
    vectorstore, qa_model = initialize_pipeline()
    query = st.text_input("Ask a question:")
    if query:
        docs = vectorstore.similarity_search(query, k=3)
        context = " ".join([doc.page_content for doc in docs])
        result = qa_model(question=query, context=context)
        st.write(f"Answer: {result['answer']}")
```

How It Works

1. **Pipeline Setup**:
    - `SentenceTransformers` generates embeddings for the dataset.
    - FAISS creates a vector index for efficient similarity search.
2. **Question-Answering**:
    - The user query retrieves the most relevant documents from FAISS.
    - HuggingFace's model generates an answer using the retrieved context.
3. **Web Interface**:
    - Streamlit provides a simple browser-based UI.

Customization

1. **Use a Custom Dataset**:
    - Replace `further_reduced_train.csv` with your own CSV file.
    - Ensure the dataset has a `text` column.
2. **Change the QA Model**:
    - Use a different HuggingFace model by replacing `distilbert-base-uncased-distilled-squad`.
3. **Enhance the UI**:
    - Customize Streamlit components (e.g., add logos, themes).

Troubleshooting

1. **Error: Missing Files**:
    - Ensure `further_reduced_train.csv` is in the `data/` folder.
    - Run `rag_pipeline.py` to create the saved pipeline.
2. **Performance Issues**:

- Use a more powerful embedding model (e.g., `all-MPNet-v2`) or increase FAISS vector dimensions.

3. **Streamlit Not Opening**:

Check for active Streamlit processes and restart:

```
streamlit run src/app.py
```

Conclusion

This RAG-based chatbot provides a reusable and efficient way to answer questions using a custom dataset. With the FAISS index and HuggingFace models, it delivers accurate and contextually relevant responses. The browser-based interface further enhances user experience.

Feel free to modify and expand the project based on your requirements. Let me know if you need further assistance!

Email : aravinditte0121@gmail.com