Documentation: AI-Powered Question Answering System with FAISS and LLMs

Introduction

This project is an AI-powered document-based question-answering system that integrates Flask for web-based user interaction, FAISS for efficient vector search, and a large language model (LLM) for response generation. The system enables users to upload documents, generate vector embeddings, and retrieve contextual answers based on user queries.

Features

- **User Authentication**: Secure login and registration using SQLite and password hashing.
- **Document Upload & Processing**: Users can upload PDF files, which are parsed into text chunks for indexing.
- **FAISS Vector Search**: Efficient similarity-based document retrieval using FAISS and SentenceTransformers.
- **LLM Integration**: Utilizes `ChatGroq` for intelligent, context-aware responses.
- **Automated Question Generation**: Generates topic-based questions using retrieved document context.
- **Grammar & Tokenization**: Implements `language_tool_python` for grammar correction and `nltk` for text tokenization.

Technologies Used

- **Flask**: Web framework for handling HTTP requests and user interaction.
- **FAISS**: Fast search and similarity matching for vectorized document storage.
- **SQLite**: Lightweight database for user authentication.
- **LangChain**: Framework for retrieval-augmented generation (RAG) and LLM pipelines.
- **Sentence Transformers**: Efficient sentence embeddings for document similarity.
- **LanguageTool & NLTK**: Grammar correction and NLP processing.
- **OpenAI ChatGroq**: LLM for text generation and answering queries.

Code Explanation

1. User Authentication

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        hashed_password = generate_password_hash(password)
        conn = sqlite3.connect('users.db')
        cursor = conn.cursor()
        cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)",
                       (username, hashed_password))
        conn.commit()
        conn.close()
        return redirect(url_for('login'))
    return render_template('register.html')
```

- Uses SQLite to store hashed passwords for security.
- Ensures unique usernames with exception handling.

2. Document Processing and Vector Store Creation

```
@app.route('/upload', methods=['GET', 'POST'])
def upload():
    if 'username' not in session:
        return redirect(url_for('login'))
    if request.method == 'POST':
        pdf_file = request.files.get('pdf_file')
        save_path = os.path.join(UPLOAD_FOLDER, pdf_file.filename)
        pdf_file.save(save_path)
        loader = PyPDFLoader(save_path)
        data = loader.load()
        text_splitter = RecursiveCharacterTextSplitter(separators=['\n\n', '\n', '.', ','],
chunk_size=2000, chunk_overlap=200)
        docs = text_splitter.split_documents(data)
        embeddings = SentenceTransformerEmbeddings(model_name="all-MiniLM-L6-v2")
        vectorstore = FAISS.from_documents(docs, embeddings)
        with open("vectorstore.pkl", "wb") as f:
            pickle.dump(vectorstore, f)
```

- Extracts text from PDFs and splits it into manageable chunks.

- Converts text into dense vector embeddings.
- Stores embeddings using FAISS for fast retrieval.

3. Querying and Retrieval from FAISS

```
@app.route('/ask', methods=['GET', 'POST'])
def ask():
    if 'username' not in session:
        return redirect(url_for('login'))
    if request.method == 'POST':
        question = request.form['question']
        result = chain.invoke({"question": question})
        return render_template('ask.html', question=question, answer=result['answer'])
```

- Takes user input and retrieves the most relevant document segments.
- Passes retrieved context to the LLM for response generation.

Advantages Over Existing Methods

1. FAISS vs. Traditional RNN-Based Retrieval

Feature	FAISS + LLM	RNN-Based Retrieval
Speed	Extremely fast due to vector indexing	Slower due to sequential processing
Scalability	Handles large document repositories efficiently	Limited scalability
Accuracy	Uses transformer-based embeddings for high accuracy	Lower accuracy due to vanishing gradient problem

2. Sentence Transformers vs. Standard Word Embeddings

- Unlike traditional RNN-based word embeddings, **SentenceTransformers** generate contextualized embeddings that better capture semantic meaning.
- Uses **cosine similarity** to retrieve the most relevant passages with higher precision.

3. Retrieval-Augmented Generation (RAG)

- RAG allows **on-the-fly knowledge integration** from document embeddings, making the system more accurate and dynamic.

- Traditional RNN-based QA systems rely on **pre-trained knowledge**, leading to outdated or limited responses.

Conclusion

This project enhances document-based question-answering systems using FAISS and LLMs, offering **fast, scalable, and contextually accurate** responses compared to traditional RNN-based retrieval methods. It is an **advanced AI-powered system** capable of handling vast document repositories while maintaining **high efficiency and accuracy**.