PROJECTS

Retrieval Augmented Gen

Retrieval-Augmented Generation (RAG) is an artificial intelligence framework that enhances the performance of large language models (LLMs) by integrating them with external knowledge sources. Instead of relying solely on data the model was trained on, RAG retrieves relevant and up-to-date information from databases, documents, or other knowledge bases and incorporates this information into its generated responses

This approach addresses common limitations of LLMs, such as outdated knowledge and the risk of generating inaccurate or generic answers (often called “hallucinations”). By grounding responses in authoritative, current data, RAG improves accuracy, relevance, and transparency.

IITB,Mumbai

```python
def rag_chatbot(query, db):
    docs = retrieve_docs(query, db)
    context = "\n\n".join([doc.page_content for doc in docs])
    answer = generate_answer_groq(query, context)
    return answer
```

RAG Q&A SYSTEM

```python
import streamlit as st
import requests

import requests

GROQ_API_KEY = "i removed mine for security reasons, please set your own"

def generate_answer_groq(query, context, model="llama3-8b-8192"):
    url = "https://api.groq.com/openai/v1/chat/completions"
    headers = {
        "Authorization": f"Bearer {GROQ_API_KEY}",
        "Content-Type": "application/json"
    }

    system_prompt = "You are a helpful assistant that answers questions based only on the provided context."
    user_prompt = f"Context:\n{context}\n\nQuestion: {query}\nAnswer:"

    payload = {
        "model": model,
        "messages": [
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_prompt}
        ],
        "temperature": 0.2,
        "max_tokens": 300
    }

    response = requests.post(url, headers=headers, json=payload)

    try:
        data = response.json()
        if "choices" in data:
            return data["choices"][0]["message"]["content"]
        else:
            return f"⚠️ Unexpected response: {data}"
    except Exception as e:
        return f"❌ Error: {e} | Raw response: {response.text}"


```

Classrooms of Sunset Valley School,

This book demonstrates a Retrieval-Augmented Generation (RAG) system designed to extract and answer questions from PDF documents. The workflow starts by loading and splitting the PDF text into overlapping chunks using PyPDF2 and LangChain’s.Recursive CharacterTextSplitter, which helps retain context. These text segments are then further indexed with FAISS and MiniLM embeddings, enabling efficient similarity searches. When a user submits a query, the system retrieves the most relevant document sections and generates accurate, context-aware answers using Groq’s Llama3 API, which is limited to using only the provided context. An example shows the system successfully answering technical questions from an assignment PDF.