# \*\*Generative AI Project: RAG-based Conversational Assistant for Standards and Design Guidelines\*\*

## Project Overview

This project implements a Retrieval-Augmented Generation (RAG) conversational assistant for querying standards and design guidelines. It leverages the Groq Llama-3.3-70b-Specdec language model for generating detailed responses and uses HuggingFace's sentence-transformers/all-MiniLM-L6-v2 for creating document embeddings. The system processes PDF documents, stores embeddings in FAISS, and supports chat history-aware contextual queries, providing precise answers based on the uploaded documents.

import os

from dotenv import load\_dotenv

from langchain\_community.document\_loaders import PyPDFDirectoryLoader

from langchain\_groq import ChatGroq

from langchain\_huggingface.embeddings import HuggingFaceEmbeddings

from langchain\_text\_splitters import RecursiveCharacterTextSplitter

from langchain\_community.vectorstores import FAISS

from langchain\_core.prompts import ChatPromptTemplate, MessagesPlaceholde

from langchain.chains import create\_history\_aware\_retriever

from langchain.chains.combine\_documents import create\_stuff\_documents\_chain

from langchain.chains import create\_retrieval\_chain

from langchain\_community.chat\_message\_histories import ChatMessageHistory

from langchain\_core.chat\_history import BaseChatMessageHistory

from langchain\_core.runnables.history import RunnableWithMessageHistory

import os

from dotenv import load\_dotenv

load\_dotenv()

os.environ['HF\_TOKEN'] = os.getenv("HF\_TOKEN")

groq\_api\_key = os.getenv("GROQ\_API\_KEY")

### \*\*1 : Load PDF documents\*\*

\* Purpose: Load PDF documents from the directory for further processing.

\* Explanation: The PyPDFDirectoryLoader extracts text from all PDFs in the specified directory.

### \*\*2: Initialize Groq LLM\*\*

\* Purpose: Use the Groq LLM for generating responses based on retrieved information.

\* Explanation: The Groq LLM is configured with the API key to enable communication.

# Step 2: Initialize Groq LLM

from langchain\_groq import ChatGroq

llm = ChatGroq(model = "Llama-3.3-70b-Specdec", groq\_api\_key = groq\_api\_key)

### \*\*3: Initialize embeddings\*\*

\* Purpose: Generate vector embeddings for the documents.

\* Explanation: sentence-transformers/all-MiniLM-L6-v2 converts textual data into numerical vectors for similarity-based retrieval

### Step 3: Initialize embeddings

from langchain\_huggingface.embeddings import HuggingFaceEmbeddings

embeddings = HuggingFaceEmbeddings(model\_name ="sentence-transformers/all-MiniLM-L6-v2")

embeddings

### \*\*4: Splitting the documents\*\*

\* Purpose: Split large documents into smaller chunks for better retrieval accuracy.

\* Explanation: The splitter creates chunks with overlap, ensuring context continuity.

## Step 4: Splitting the documents

from langchain\_text\_splitters import RecursiveCharacterTextSplitter

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size = 10000, chunk\_overlap = 1000)

splits = text\_splitter.split\_documents(docs)

### \*\*5: Use FAISS for vector storage\*\*

\* Purpose: Store document embeddings in a vector database for efficient retrieval.

\* Explanation: FAISS is a fast similarity search library that indexes and retrieves vectors.

# Step 5: Use FAISS for vector storage

from langchain\_community.vectorstores import FAISS

vector\_store = FAISS.from\_documents(documents = splits, embedding = embeddings)

retriever = vector\_store.as\_retriever()

retriever

### \*\*6: Create prompt templates\*\*

\* Purpose: Define reusable prompt templates for the system and contextualized question reformulation.

\* Explanation: Prompts guide the LLM in generating specific responses based on provided context and history.

# Step 6: Create prompt templates

from langchain\_core.prompts import ChatPromptTemplate, MessagesPlaceholder

## contextualize\_question\_system\_prompt

contextualize\_question\_system\_prompt = (

    "Based on the chat history and the most recent user question, "

    "which may refer to previous context in the conversation, "

    "reformulate the question so it can be understood independently of the chat history. "

    "Do not provide an answer; simply rephrase the question if necessary, otherwise return it unchanged."

)

contextualize\_question\_prompt = ChatPromptTemplate.from\_messages(

   [

        ("system", contextualize\_question\_system\_prompt),

        MessagesPlaceholder("chat\_history"),

        ("human", "{input}")

    ]

)

## System prompt

system\_prompt = (

    "You are a highly knowledgeable assistant tasked with providing accurate answers to questions."

    "Utilize the given pieces of retrieved information to craft your response."

    "If the answer is not present in the provided context, indicate that you do not know."

    "Your responses should be detailed and comprehensive."

    "\n\n"

    "{context}"

)

question\_prompt = ChatPromptTemplate.from\_messages(

    [

        ("system", system\_prompt),

        MessagesPlaceholder("chat\_history"),

        ("human", "{input}")

    ]

)

### \*\*7: Create retriever and history-aware retriever\*\*

\* Purpose: Build a retriever that considers chat history when retrieving documents.

\* Explanation: The retriever ensures relevance by reformulating queries based on chat history.

# Step 7: Create retriever and history-aware retriever

from langchain.chains import create\_history\_aware\_retriever, create\_retrieval\_chain

history\_aware\_retriever = create\_history\_aware\_retriever(llm, retriever, contextualize\_question\_prompt)

history\_aware\_retriever

### \*\*8: combine documents\*\*

\* Purpose: Combine retrieved documents for processing by the LLM.

\* Explanation: The chain organizes retrieved chunks into a coherent structure for answering questions.

# Step 8:combine documents

from langchain.chains.combine\_documents import create\_stuff\_documents\_chain

question\_answer\_chain = create\_stuff\_documents\_chain(llm, question\_prompt)

### \*\*9: Create retriever chain\*\*

\* Purpose: Link the retriever and document combiner into a RAG pipeline.

\* Explanation: The retrieval chain integrates retrieval and answer generation.

# Step 9:Create retriever chain

from langchain.chains import create\_retrieval\_chain

rag\_chain = create\_retrieval\_chain(history\_aware\_retriever, question\_answer\_chain)

rag\_chain

### \*\*10: Manage chat history\*\*

\* Purpose: Maintain session-based chat history for context retention.

\* Explanation: ChatMessageHistory stores interactions by session ID

# Step 10: Manage chat history

from langchain\_community.chat\_message\_histories import ChatMessageHistory

from langchain\_core.chat\_history import BaseChatMessageHistory

store = {}

def get\_session\_history(session\_id: str) -> BaseChatMessageHistory:

    if session\_id not in store:

        store[session\_id] = ChatMessageHistory()

    return store[session\_id]

### \*\*11: Add Runnable with message history\*\*

\* Purpose: Integrate the RAG chain with message history for conversational capabilities.

\* Explanation: RunnableWithMessageHistory enables history-aware responses.

# Step 11: Add Runnable with message history

from langchain\_core.runnables.history import RunnableWithMessageHistory

conversational\_rag\_chain = RunnableWithMessageHistory(

    rag\_chain,

    get\_session\_history,

    input\_messages\_key = "input",

    history\_messages\_key = "chat\_history",

    output\_messages\_key = "answer"

)

### Sample Invocations

\* Purpose: Query the system with a hospital design-related question.

\* Explanation: Responses are generated based on retrieved and history-aware context

response\_1 = conversational\_rag\_chain.invoke(

    {"input": "What are the components and design standards with dimesions I should follow when I am designing Hospitals: specialunits?, give me output in detail"},

    config  = {

        "configurable": {"session\_id": "user\_1"}

    },

)

print("User\_1")

print("\n")

print(f'''response\_1: {response\_1["answer"]}''')

Response:

User\_1

response\_1: Designing hospital special units requires careful consideration of various components and design standards to ensure efficient and effective patient care. Here's a detailed overview of the key components and design standards with dimensions to follow:

\*\*1. Intensive Care Units (ICUs):\*\*

\* Bed unit basic space module: 11-15 m²/unit

\* Number of beds: 6-7 maximum per unit

\* Distance from control station or viewpoint to patient: minimal, so that equipment can be read and patient can be seen

\* Nurse/physician station: 4-10 m²

\* Support area for medication station, utility, and treatment: 11-15 m²

\* Amenities such as rest room, locker, and WC: 0.5-1.0 m² per bed

\* Design considerations:

+ Will patient be conscious, require privacy, toilet, constant nursing attention?

+ Will location or configuration of unit help or hinder patient's recovery?

+ Can staff see all patients easily? Is ratio patients/staff station appropriate?

\*\*2. Neurosurgical Units:\*\*

\* Ratio of population to beds: 1:100,000

\* Number of beds: 20-30 per unit

\* Operating theaters: 2-3 per unit, with a minimum size of 40-50 m² each

\* Design considerations:

+ Patient will be unconscious during surgery, requiring constant monitoring and life-support equipment

+ Staff will require easy access to patients, equipment, and support services

+ Infection control and separation of patients from noise are crucial

\*\*3. Spinal Injury Units:\*\*

\* Number of beds: 20-24 for stages 1 and 2, 28-32 for stage 3

\* Bed width: 1,000 mm (wider than normal)

\* Bed center distance: 3,500 mm

\* Nurse working room: similar to that for physical/mental handicapped units

\* Design considerations:

+ Patients will require 100% care, with nearly always doubly incontinent and aggressive towards staff and visitors

+ Occupational therapy, physiotherapy, and study form vital parts of rehabilitation

+ Patients may take months to recover, requiring a phased approach to care

\*\*4. Ward Design: General\*\*

\* Outlooks on ward sizes: subject to rapid change, with a trend away from classical ward types (e.g., 'Nightingale' and 'Rigs')

\* Preference for 2-4 bed wards, despite controversy surrounding very small wards

\* Design considerations:

+ Patients need audio and visual privacy during medical visits

+ Background noise and bed curtains can provide some privacy, but lights can disturb at night

+ Small wards can be peaceful for resting patients, but may not provide audio-privacy

\*\*5. Acute Ward Design:\*\*

\* Number of beds: 24-30 per ward

\* Breakdown of bed areas:

+ 3 x 8-B rooms + 4 x 1-B rooms

+ 4 x 6-B rooms + 4 x 1-B rooms

+ 2 x 12-B rooms + 4 x 1-B rooms

\* Design considerations:

+ Walking distance: keep short for nurses and ambulant patients

+ Observation: 50% of beds should be visible from the nurse station

+ Control: patients, particularly children, adolescents, and confused, need to be controlled

+ Noise: design for 40-45 dB by day and 35-40 dB at night in multi-B wards

\*\*6. Acute Ward Areas:\*\*

\* 1-B room: 9.3-10.0 m²

\* Multi-B bay (per bed): 0.75 m²

\* Day room (per bed): 0.5-1.0 m²

\* Lockers, etc. (per bed): 0.5-1.0 m²

\* Nurse station: 4.0-10.0 m²

\* Clean utility: 10.2-18.0 m²

\* Dirty utility and sluice: 14.0-16.0 m²

\* Treatment room: 10.25-12.0 m²

\* Assisted bathroom: 7.0-9.0 m²

\* Sisters' room: 7.0-9.0 m²

\* Doctors' room: 7.0-9.0 m²

\* Bathroom: 7.0 m²

\* WC with hand rinse basin: 2.75 m²

\* Ward kitchen: 20.0 m²

\* Flower bay: 2.0-2.5 m²

\* Circulation area: 5.0-8.0 m²

\*\*7. Mechanical and Electrical Services:\*\*

\* Mechanical ventilation: often noisy and unsatisfactory, full air-conditioning expensive to install and run

\* Natural ventilation: rely on for normal sites, away from air or traffic noise

\* Central work rooms: require mechanical ventilation and suffer from heat build-up in summer

\* Electrical services: ensure sufficient power and data points for equipment and staff

\*\*8. Infection Control:\*\*

\* Design considerations:

+ Patient will be unconscious during surgery, requiring constant monitoring and life-support equipment

+ Staff will require easy access to patients, equipment, and support services

+ Separation of patients from noise and infection control are crucial

\* Measures:

+ Anti-static flooring and flashproof electrical outlets

+ Isolation rooms for infectious patients

+ Easy-to-clean surfaces and equipment

\*\*9. Fire Safety:\*\*

\* Design considerations:

+ Fire-resistant materials and construction

+ Compartmentalization of wards and departments

+ Fire alarms, sprinkler systems, and smoke detectors

+ Emergency evacuation plans and routes

\*\*10. Accessibility:\*\*

\* Design considerations:

+ Wheelchair accessibility: minimum 1,200 mm wide corridors and doorways

+ Ramp gradients: 1:12 for easy access

+ Elevators: minimum 1,500 mm wide and 2,100 mm deep

+ Toilets: wheelchair-accessible, with emergency alarms and grab bars

These are some of the key components and design standards to follow when designing hospital special units. Remember to consult local building codes, regulations, and healthcare authorities for specific requirements and guidelines.

### Key Features

\* LLM Model: Groq Llama-3.3-70b-Specdec.

\* Embedding Model: HuggingFace sentence-transformers/all-MiniLM-L6-v2.

\* Vector Database: FAISS.

\* Chat History: Session-based chat for personalized interactions.

\* RAG Pipeline: Combines retrieval and generation for accurate, contextual responses.

Code for Webapp using streamlit

import streamlit as st

import os

from dotenv import load\_dotenv

from langchain\_community.document\_loaders import PyPDFDirectoryLoader, PyPDFLoader

from langchain\_groq import ChatGroq

from langchain\_huggingface.embeddings import HuggingFaceEmbeddings

from langchain\_text\_splitters import RecursiveCharacterTextSplitter

from langchain\_community.vectorstores import FAISS

from langchain\_core.prompts import ChatPromptTemplate, MessagesPlaceholder

from langchain.chains import create\_history\_aware\_retriever

from langchain.chains.combine\_documents import create\_stuff\_documents\_chain

from langchain.chains import create\_retrieval\_chain

from langchain\_community.chat\_message\_histories import ChatMessageHistory

from langchain\_core.chat\_history import BaseChatMessageHistory

from langchain\_core.runnables.history import RunnableWithMessageHistory

load\_dotenv()

os.environ["HF\_TOKEN"] = os.getenv("HF\_TOKEN")

embeddings = HuggingFaceEmbeddings(model\_name = "sentence-transformers/all-MiniLM-L6-v2")

### Setup Streamlit app

st.title("Converstional RAG with pdf uploads and chat history")

st.write ("Upload pdf's and chat with their content")

## Iput the Groq API key

api\_key = st.text\_input("Enter your Groq API key: ", type = "password")

## Check if groq api key is provided

if api\_key:

    llm = ChatGroq(model = "Llama-3.3-70b-Specdec", groq\_api\_key = api\_key)

    ## Chat interface

    session\_id = st.text\_input("Session ID", value = "default\_Session")

    ## Statefully magange the chat history

    if "store" not in st.session\_state:

        st.session\_state.store = {}

    uploaded\_files = st.file\_uploader("Chosse a PDF file", type = "pdf", accept\_multiple\_files = True)

    ## Process uploaded PDF's

    if uploaded\_files:

        documents = []

        for uploaded\_files in uploaded\_files:

            temp\_pdf = f"./temp.pdf"

            with open(temp\_pdf, "wb") as file:

                file.write(uploaded\_files.getvalue())

                file\_name = uploaded\_files.name

            loader = PyPDFLoader(temp\_pdf)

            docs = loader.load()

            documents.extend(docs)

        ## Splitting and creating the embeddings for the documents

        text\_splitter = RecursiveCharacterTextSplitter(chunk\_size = 10000, chunk\_overlap = 1000)

        splits = text\_splitter.split\_documents(documents)

        vector\_store = FAISS.from\_documents(documents = splits, embedding = embeddings)

        retriever = vector\_store.as\_retriever()

        ## Create the Prompts

        contextualize\_question\_system\_prompt = (

        "Based on the chat history and the most recent user question, "

        "which may refer to previous context in the conversation, "

        "reformulate the question so it can be understood independently of the chat history. "

        "Do not provide an answer; simply rephrase the question if necessary, otherwise return it unchanged."

        )

        contextualize\_question\_prompt = ChatPromptTemplate.from\_messages(

        [

                ("system", contextualize\_question\_system\_prompt),

                MessagesPlaceholder("chat\_history"),

                ("human", "{input}")

            ]

        )

        history\_aware\_retriever = create\_history\_aware\_retriever(llm, retriever, contextualize\_question\_prompt)

       ## System prompt

        system\_prompt = (

            "You are a highly knowledgeable assistant tasked with providing accurate answers to questions."

            "Utilize the given pieces of retrieved information to craft your response."

            "If the answer is not present in the provided context, indicate that you do not know."

            "Your responses should be detailed and comprehensive."

            "\n\n"

            "{context}"

        )

        question\_prompt = ChatPromptTemplate.from\_messages(

            [

                ("system", system\_prompt),

                MessagesPlaceholder("chat\_history"),

                ("human", "{input}")

            ]

        )

        question\_answer\_chain = create\_stuff\_documents\_chain(llm, question\_prompt)

        rag\_chain = create\_retrieval\_chain(history\_aware\_retriever, question\_answer\_chain)

        def get\_session\_history(session: str) -> BaseChatMessageHistory:

            if session\_id not in st.session\_state.store:

                st.session\_state.store[session\_id] = ChatMessageHistory()

            return st.session\_state.store[session\_id]

        conversational\_rag\_chain = RunnableWithMessageHistory(

            rag\_chain,

            get\_session\_history,

            input\_messages\_key = "input",

            history\_messages\_key = "chat\_history",

            output\_messages\_key = "answer"

        )

        user\_input = st.text\_input("Your question: ")

        if user\_input:

            session\_history = get\_session\_history(session\_id)

            response = conversational\_rag\_chain.invoke(

                {"input": user\_input},

                config = {

                    "configurable": {"session\_id": session\_id}

                },

            )

            st.write(st.session\_state.store)

            st.success(f'''Assistant: {response["answer"]}''')

            st.write(f'''Chat History: {session\_history.messages}''')

else:

    st.warning("Please enter the Groq API key")