

INFO7375 - Prompt Engineering & AI

Summer 2024



"Portfolio Web Application with RAG Chatbot"

Submitted To -

Prof. Nicholas Brown

Submitted By -

1. Anmol Valecha (002813410)

2. Kaushik Bhawsar (002704590)

Master of Science in Information Systems

College of Engineering

Northeastern University, Toronto

Summer 2024

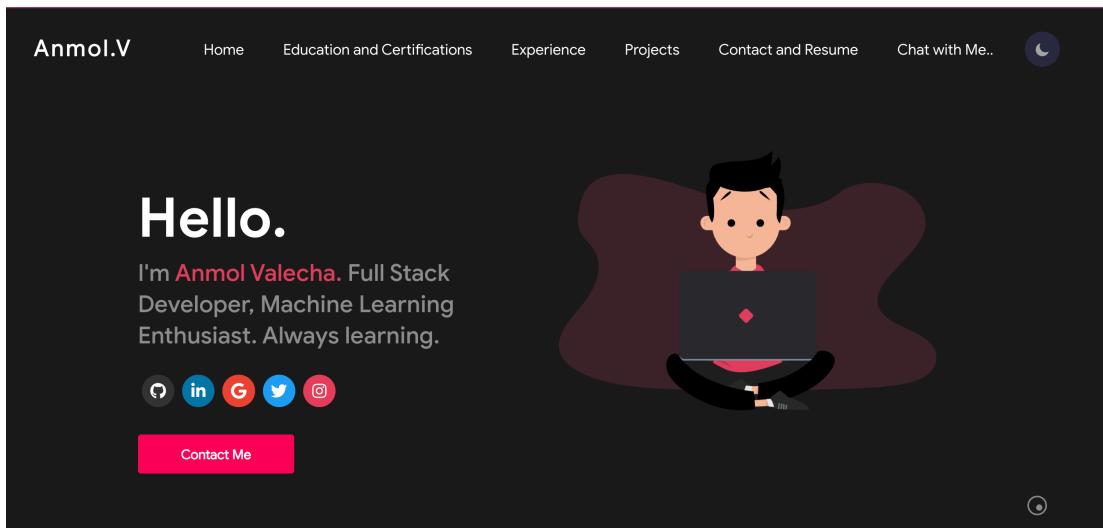
Table of Contents

Table of Contents.....	2
Introduction	4
Project Description	5
1. Scope Definition	5
Data Collection and Preprocessing.....	5
Preprocessing Steps.....	5
How Portfolio RAG Chatbot works?	7
1. Data Loading:.....	7
2. Text Chunking:.....	7
3. Language Model:.....	7
4. Similarity Matching:.....	7
5. Response Generation:	8
Project Architecture.....	8
Vector Database Implementation.....	8
1. Importing dependencies and API key.....	9
2. PDF Data Loading.....	9
3. Text Splitting	9
4. Text Embedding.....	9
5. Storing in Vector Database.....	10
Application Development	10
PART 1: Frontend Component:	10
PART 2: Chatbot Component:	13
1. Initialization of Environment and Dependencies	13
2. Setup of the Language Model and Vector Store	13
3. Preparation of the Retrieval System:	14
4. Integration of Custom Prompt Template:	14
5. Response Generation Mechanism:.....	15

6. Streamlit Interface Configuration:	15
7. User Interaction Flow:	17
Prompt Engineering for Mock Data	17
Prompt pattern.....	18
Evaluation Methodology.....	18
RAG Pipeline Evaluation	18
Evaluation Results.....	24
Improvement Strategies.....	24
Fine-tuning the LLM – using Unsloth	24
Testing Methodology	29
LLM Model Performance Comparison	32
Challenges and Overcome solutions.....	32
1. Dependency Installation Issues	32
2. Language Model Configuration.....	33
3. Streamlit Application Deployment.....	33
Deployment Plan:	33
Future Work:	35
Conclusion.....	35

Introduction

This final project **Portfolio web application with integrated RAG Chatbot** involves the development of a modern, responsive portfolio website with the integration of a personal assistant chatbot using advanced AI techniques. The primary objective is to create a professional online presence that not only showcases personal skills and projects but also provides an interactive and engaging experience for visitors. The relevance of this project lies in its demonstration of proficiency in frontend development, along with the application of generative AI concepts, particularly in Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs).



A screenshot of a chatbot interface titled "Explore Anmol's Portfolio of Excellence". On the left, there is a sidebar with a section for "AV_Bot - Anmol's Virtual Assistant" featuring a heart icon and a brief introduction. The main area shows a conversation with the bot. The user asks, "Hi there! I'm AV_Bot. How can I assist you in exploring Anmol's professional expertise and accomplishments? To get started, here are some questions you can ask me:". The bot responds with several questions: "What are Anmol's technical skills?", "What work experience does Anmol have in database management and full-stack development?", "Can you tell me about some notable projects Anmol has completed?", "Which certifications has Anmol earned?", and "What makes Anmol a strong candidate in the field of information systems?". At the bottom, there is a message input field with placeholder text "Your message" and a send button, along with a "Clear Chat" button and a "Manage app" link.

Project Description

The portfolio web application is designed to solve the problem of creating an effective online presence that stands out. It provides a platform to present individual achievements, skills, and projects in a manner that is both engaging and informative. The project includes the complete design and development of the portfolio website and the integration of a chatbot powered by LLM-based RAG techniques.

1. Scope Definition

The project encompasses the development of a responsive portfolio website with sections such as About, Projects, Skills, Education, Work Experience, Certifications, and Contact. Additionally, it includes the development and integration of a chatbot that uses state-of-the-art AI to provide instant, personalized responses to user queries.

- **Professional Experience:** Highlighted within the portfolio to demonstrate relevant industry experience and expertise, showcasing the practical application of skills in real-world scenarios.
- **Technical Skills:** Clearly listed and categorized in the portfolio, focusing on core competencies and technologies utilized in the project, such as frontend development and AI integration.
- **Projects and Achievements:** Detailed in a dedicated section, providing a comprehensive view of completed projects, including the integrated RAG chatbot, along with any notable accomplishments or awards.
- **Education and Certifications:** Included in the portfolio to validate academic background and professional qualifications, emphasizing certifications and courses relevant to the project's technical aspects.

Data Collection and Preprocessing

Data for the project is sourced from the portfolio website and the resume. The data collection process includes web scraping using BeautifulSoup to extract relevant text content from the website, and text extraction from the resume using PyPDF. The preprocessing steps involve cleaning the data to remove HTML tags, special characters, and irrelevant text, followed by vectorization using tools like Hugging Face Transformers. The processed data is then indexed and stored in a vector database like Chroma.

Preprocessing Steps

The preprocessing of the dataset involved several key procedures to optimize the data for indexing in the vector database and processing by the language model (LLM):

1. **Data Cleaning:** Removed any irrelevant information that did not pertain to the specific attributes, such as unrelated metadata or extraneous text.
2. **Handling Missing Data:** I identified and appropriately handled any missing values. This often involved imputing data based on similar items or removing entries that lacked essential information.
3. **Data Structuring:** Given the conversion from CSV to JSON to PDF, it was essential to maintain a structured format.

Code Snippet:

```

import requests
import pandas as pd
from bs4 import BeautifulSoup

url = "https://anmolvalechabotfolio.netlify.app/#/"
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

div = soup.find_all("div", class_="lb-grid")

def get_text_including_a_and_span_tags(g_tag):
    text_pieces = []
    for element in g_tag.descendants:
        if element.name in ["a", "span"]:
            text_pieces.append(element.get_text())
        elif isinstance(element, str):
            text_pieces.append(element)
    return "".join(text_pieces).strip()

current_question = None
current_answer = []

data = []
for div in div:
    div_children = div.find_all("div", class_="lb-txt-16 lb-text")
    for ps in div_children:
        ps = div.find_all("p")
        for p in ps:
            b = p.find("b")
            if b is not None and b.text[0] == "Q":
                if current_question:
                    data.append({"question": current_question, "answer": "".join(current_answer)})
                current_question = b.text
                current_answer = []
            current_answer.append(get_text_including_a_and_span_tags(p))

if current_question:
    data.append({"question": current_question, "answer": "".join(current_answer)})

df = pd.DataFrame(data)
df.to_csv("portfolio.csv", index=False)

```

Extracted Data:

Anmol is proficient in programming languages such as C++, Java, Python, and SQL.

Anmol has expertise in ETL/Data Testing, Web & Application Testing, Mobile Testing, Performance Testing, API Testing, Usability Testing, Regression Testing, Black-Box Testing, and Unit Testing.

Anmol is skilled in using Selenium WebDriver, TestNG, and BDD Cucumber for automation testing.

Anmol has experience with AWS, Databricks, and Azure DevOps.

Anmol is proficient in HTML, CSS, JavaScript, Bootstrap, Angular17, and React.

At TD Bank, Anmol worked as a Software Quality Engineering Intern, where they leveraged BDD Cucumber, Java, and Jenkins using Groovy to create automated regression pipelines, achieving 100% coverage. Anmol orchestrated data pipelines with Azure Data Factory, ensured seamless data processes, and earned TD appreciation e-cards for consistent, noteworthy contributions.

Anmol spearheaded quality assurance efforts, developed over 1500 automated multi-browser tests with Selenium and TestNG, and reduced manual workload by 60% through root cause analysis.

Anmol received a top performance rating of 5/5 and was honored with the 'Standing Ovation Award' in the FY 20-21 Walk of Fame.

Anmol worked as a Software Developer Intern, where they designed and implemented responsive websites using React, Angular, HTML, CSS, and Bootstrap. They enhanced interactivity by 30% and optimized load times by 20%.

Anmol has a Master of Science in Information Systems from Northeastern University, Toronto, with a GPA of 3.94/4.00, and a Bachelor of Technology in Information Technology from Shri Mata Vaishno Devi University, Jammu, with a CGPA of 9.0/10.00.

Anmol completed her Master's degree in Information Systems at Northeastern University, Toronto.

Anmol achieved a GPA of 3.94/4.00 in her Master of Science in Information Systems.

Anmol holds certifications such as Oracle Cloud Infrastructure (OCI) 2024 Generative AI Professional Certification and NPTEL IIT Kharagpur Database Management System Certification.

Anmol holds the Oracle Cloud Infrastructure (OCI) 2024 Generative AI Professional Certification.

Anmol obtained the Database Management System Certification from NPTEL IIT Kharagpur.

Anmol has worked on multiple personal projects, including designing and implementing responsive websites, and integrating AI-based chatbots to enhance interactivity.

Anmol emphasizes open communication and active collaboration with cross-functional teams, including developers, business analysts, and product owners. She contributes to sprint planning and backlog refinement.

Anmol adopts a systematic approach to problem-solving, starting with root cause analysis to identify the core issues. She then collaborates with her team to develop and implement solutions.

Anmol's automation efforts, particularly at Wipro, led to a dramatic reduction in manual workload by 60%. Her development of over 1500 automated multi-browser tests with Selenium and TestNG contributed significantly to this efficiency gain.

Anmol is committed to continuous learning, regularly participating in online courses, attending industry webinars, and reading up on the latest technologies. Her certifications in Oracle Cloud Infrastructure and Database Management System further validate her expertise in these domains.

How Portfolio RAG Chatbot works?

The Portfolio RAG system operates through a sequence of steps to process user queries and provide relevant responses:

1. Data Loading:

Utilizes **LangChain's PyPDFLoader** to read PDF document, extracting their text content. This step forms the foundation for creating a searchable database of portfolio-related information.

2. Text Chunking:

The extracted text is divided into smaller chunks using **LangChain's RecursiveCharacterTextSplitter**. This method ensures that the text is manageable and retains sufficient context, which is crucial for producing accurate embeddings.

3. Language Model:

Employs Hugging Face's Transformers, specifically the **sentence-transformers/all-MiniLM-L6-v2** model, to generate vector representations (embeddings) of the text chunks. These embeddings capture the semantic essence of the text, allowing for sophisticated query handling.

4. Similarity Matching:

When a query is received, the system performs a semantic similarity search using the **Chroma** vector database. This search identifies text chunks that are most semantically similar to the query, ensuring that the most relevant content is selected for response generation.

5. Response Generation:

The relevant text chunks are passed to the Hugging Face Hub Language Model. Three models were tested:

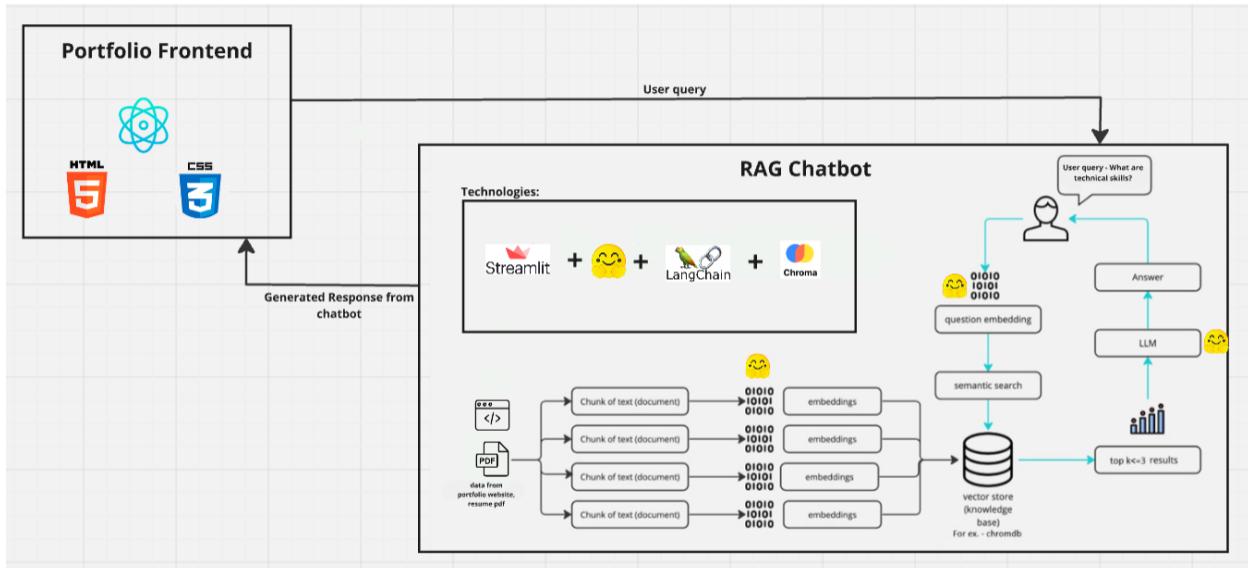
- meta-llama/Meta-Llama-3-8B-Instruct
- mistralai/Mistral-7B-Instruct-v0.3
- Fine Tuned - unsloth/meta-llama-3.1-8b-bnb-4bit

Project Architecture

The architecture of the project is divided into three main components:

- **Frontend Component:** Built using React, HTML, and CSS to ensure a modern and responsive user interface.
- **Chatbot Component:** The chatbot is powered by a RAG pipeline using LangChain, with a Hugging Face transformer for LLM, and Pinecone/Chroma for the vector database. The frontend for the chatbot is built using Streamlit or Flowise.

A diagram of the project architecture illustrates how these components interact, providing a clear overview of the system's workflow.



Vector Database Implementation

For Portfolio RAG Chatbot, We chose Chroma as the vector database due to its compatibility with LangChain and the specific requirements for handling fashion data. This decision supports efficient data retrieval based on semantic similarity, which is crucial for delivering accurate information. Below are the implementation steps:

1. Importing dependencies and API key

```
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma
import os
from dotenv import load_dotenv
from collections import OrderedDict

# Load environment variables from .env file
load_dotenv()

os.environ["HUGGINGFACEHUB_API_TOKEN"] = os.getenv("HUGGINGFACEHUB_API_TOKEN")
```

2. PDF Data Loading

I used the PyPDFLoader from LangChain to load the fashion data PDF. This step ensures that all the textual content needed for the vector database is accessible.

```
# Load the PDF
loader = PyPDFLoader("Anmol_Valecha_Resume.pdf")
documents = loader.load()
```

3. Text Splitting

With the RecursiveCharacterTextSplitter, I divided the loaded text into manageable chunks, optimizing them for the embedding process.

```
# Split the text
text_splitter = RecursiveCharacterTextSplitter(chunk_size=2000, chunk_overlap=300)
texts = text_splitter.split_documents(documents)
```

4. Text Embedding

Using HuggingFaceEmbeddings, specifically the sentence-transformers/all-MiniLM-L6-v2 model, I converted the text chunks into vector embeddings. This transformation is key to enabling semantic-based data retrieval.

```
# Initialize the embedding model
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

# Convert texts to embeddings
try:
    embeddings = embedding_model.embed_documents([doc.page_content for doc in texts])
    print("Vector Embeddings created successfully")
except Exception as e:
    print(f"Error creating vector embeddings: {e}")
```

5. Storing in Vector Database

The generated embeddings were stored in Chroma, set up with a directory to persist data. This database structures the data to facilitate quick and efficient similarity searches.

```
# Initialize Chroma vector store
vector_store = Chroma(embedding_function=embedding_model, persist_directory="data")

# Add documents to the vector store
vector_store.add_documents(documents=texts)
```

Application Development

PART 1: Frontend Component:

The frontend of the portfolio web application was meticulously crafted using React, HTML, and CSS to ensure a modern and responsive user interface. The design emphasizes user experience, with intuitive navigation and visually appealing layouts that cater to both desktop and mobile users. Each section of the portfolio—ranging from About to Projects and Skills—is seamlessly integrated, providing a cohesive and engaging user experience. The frontend is not only designed to showcase the user's skills and accomplishments but also to facilitate smooth interaction with the embedded chatbot.

Website url – <https://anmolvalechaprofilebot.netlify.app/#/>

The screenshot shows the homepage of Anmol Valecha's portfolio. The background is black with a large white 'Hello.' text on the left. To the right is a cartoon illustration of a person sitting at a desk with a laptop. The navigation bar at the top includes links for Home, Education and Certifications, Experience, Projects, Contact and Resume, Chat with Me.., and a dark mode switch. Below the 'Hello.' text is a bio: 'I'm **Anmol Valecha**. Full Stack Developer, Machine Learning Enthusiast. Always learning.' There are social media icons for GitHub, LinkedIn, Google+, Twitter, and Instagram. A 'Contact Me' button is at the bottom.

This screenshot shows the 'Here's what I do' section. The background is black. On the left, there is an illustration of a person sitting cross-legged, working on a laptop. To the right, the text 'Here's what I do' is displayed. Below it, the heading 'Full Stack Development' is shown. Underneath, there is a list of skills: '⚡ Building responsive website front end using ReactJS and Angular', '⚡ Developing mobile applications using Flutter and native Android', and '⚡ Backend development using NodeJS, ExpressJS and Flask'.

Education

Basic Qualification and Certifications

I actively participate in hackathons and other tech-related activities. Below are some of my major certifications.

Degrees Received

Northeastern University Sep 2023 – May 2025

M.S. in Information Systemse

⚡ Specialized in advanced coursework covering Data Management & Database Design, Application Engineering & Management.

⚡ Gained hands-on experience in Web Design and User Experience Engineering.

[Visit Website](#)

Anmol.V

Home Education and Certifications Experience Projects Contact and Resume Chat with Me..

Projects

My projects makes use of vast variety of latest technology tools. My best experience is to create Data Science projects. Below are some of my projects. Note that not all of the mentioned projects are on GitHub yet.

CineScope
An IMDB-like application.

Chat With Me
You can ask any question related to my Resume

RecipesAPI
Get recipes for hundreds and thousands of dishes

PART 2: Chatbot Component:

For the RAG ChatBot, we designed an intuitive and interactive user interface using Streamlit. This interface allows users to effortlessly submit natural language queries related to their profile, providing an easy and engaging way to access information about our candidacy. The design focuses on user experience, ensuring that interactions with the system are smooth and that users can quickly find the information they need.

Implementation steps:

1. Initialization of Environment and Dependencies

- Import necessary libraries such as Streamlit, LangChain, sqlite3 and dotenv for environment management.
- Load environment variables like the HuggingFace API token to authenticate and utilize HuggingFace's services.

```
from EvaluationMetrics import evaluate_metrics
import numpy as np
import streamlit as st
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import Chroma
from langchain_community.llms import HuggingFaceHub
from langchain.prompts import PromptTemplate
from langchain.chains import RetrievalQA
import warnings, os
warnings.filterwarnings("ignore")
__import__('sqlite3')
import sys
sys.modules['sqlite3'] = sys.modules.pop('pysqlite3')

data_directory = os.path.join(os.path.dirname(__file__), "data")

os.environ["HUGGINGFACEHUB_API_TOKEN"] = st.secrets["HUGGINGFACEHUB_API_TOKEN"]
```

2. Setup of the Language Model and Vector Store

- Initialize the Hugging Face Embeddings model (sentence-transformers/all-MiniLM-L6-v2) to process and embed textual data.
- Configure Chroma as the vector database to store these embeddings, which facilitates efficient retrieval based on semantic similarity.

```
21
22 # Load the vector store from disk
23 embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
24 vector_store = Chroma(embedding_function=embedding_model, persist_directory=data_directory)
25
```

3. Preparation of the Retrieval System:

Set up the RetrievalQA system from LangChain, incorporating the Hugging Face Hub Language Model. Three models were tested:

- **meta-llama/Meta-Llama-3-8B-Instruct**
- **mistralai/Mistral-7B-Instruct-v0.3**
- **kaushik01/Llama_3_1_finetuned_model** [unslloth/meta-llama-3.1-8b-bnb-4bit]

```
# Initialize the Hugging Face Hub LLM
hf_hub_llm1 = HuggingFaceHub(
    repo_id="meta-llama/Meta-Llama-3-8B-Instruct",
    model_kwarg={"temperature": 1, "max_new_tokens":1024},
)
hf_hub_llm2 = HuggingFaceHub(
    repo_id="mistralai/Mistral-8x7B-Instruct-v0.1",
    model_kwarg={"temperature": 0.01, "max_new_tokens":1024},
)
hf_hub_llm3 = HuggingFaceHub(
    repo_id="kaushik01/Llama_3_1_finetuned_model",
    model_kwarg={"temperature": 0.01, "max_new_tokens":1024},
)
```

- Configure the retriever within this system to fetch the top 3 most relevant entries from the vector database, ensuring responses are precise and relevant.

```
69     rag_chain = RetrievalQA.from_chain_type(
70         llm=hf_hub_llm,
71         chain_type="stuff",
72         retriever=vector_store.as_retriever(top_k=3), # retriever is set to fetch top 3 results
73         chain_type_kwarg={"prompt": custom_prompt})
74
```

4. Integration of Custom Prompt Template:

- Define a custom prompt template based on **Directive-based prompt pattern** and **Template prompt pattern** which guides the language model on how to generate appropriate responses based on the query context and user interaction directives.

```

prompt_template = """
You are Anmol's intelligent assistant with a vast knowledge base.
Your task is to read the given resume and answer the question asked based on the information provided in the resume.
While answering the question please adhere to the following guidelines:

1. Answer only the question asked: Use only the information provided in the resume. Do not add any extra information or make
2. Greetings and other general queries: For non-resume-related questions like greetings or general inquiries, respond appropri
3. Contact details: If asked for contact details, use the following: \n
   - Email: valecha.a@northeastern.edu \n
4. Frame your answers in such a way that they showcase Anmol's importance.
5. No pre-amble and post-amble is required, just answer the question.

Resume:
{context}

Question: {question}

Answer:
"""

```

5. Response Generation Mechanism:

- Implement a function `get_response` that takes a user's question, processes it through the RetrievalQA chain, and extracts the answer from the model's output. This ensures that the responses are not only accurate but also tailored to the user's specific inquiry.

```

68
69  rag_chain = RetrievalQA.from_chain_type(
70      llm=hf_hub_llm,
71      chain_type="stuff",
72      retriever=vector_store.as_retriever(top_k=3), # retriever is set to fetch top 3 results
73      chain_type_kwargs={"prompt": custom_prompt})
74
75  def get_response(question):
76      result = rag_chain({"query": question})
77      response_text = result["result"]
78      answer_start = response_text.find("Answer:") + len("Answer:")
79      answer = response_text[answer_start:].strip()
80      return answer

```

6. Streamlit Interface Configuration:

- Customize the Streamlit layout to remove unnecessary whitespace and add aesthetic elements like headers and sidebar messages.

```
# Streamlit app
# Remove whitespace from the top of the page and sidebar
st.markdown(
    """
        <style>
            .appview-container .main .block-container {{
                padding-top: {padding_top}rem;
                padding-bottom: {padding_bottom}rem;
            }}
        </style>""", format(padding_top=1, padding_bottom=1),
    unsafe_allow_html=True,
)

# st.header("Explore Anmol's Portfolio of Excellence", divider='grey')
st.markdown("""
    <h3 style='text-align: left; padding-top: 35px; border-bottom: 3px solid purple;'>
        | Explore Anmol's Portfolio of Excellence
    </h3>""", unsafe_allow_html=True)
```

- Create interactive elements such as chat input and buttons to clear the chat history, enhancing user interaction.

```
object > Anmol_ResumeBot > 🗂 ResumeBot.py > ...
st.header("Explore Anmol's Portfolio of Excellence", divider='grey')
.markdown("""
    <h3 style='text-align: left; padding-top: 35px; border-bottom: 3px solid purple;'>
        | Explore Anmol's Portfolio of Excellence
    </h3>""", unsafe_allow_html=True)

de_bar_message = """Hi there, I'm [Anmol](https://www.linkedin.com/in/anmol-valecha/), and I've developed this virtual assistant, **AV_Bot**,  

re are some key areas to consider:  

**Professional Experience**  

**Technical Skills**  

**Projects and Achievements**  

**Education and Certifications**  

el free to ask anything!
"""

th st.sidebar:
    st.title(':purple_heart: AV_Bot - Anmol\'s Virtual Assistant')
    st.markdown(side_bar_message)

initial_message = """
Hi there!👋 I'm AV_Bot.  

How can I assist you in exploring Anmol's professional expertise and accomplishments? To get started, here are some questions you can ask me:  

What are Anmol's technical skills?\n
What work experience does Anmol have in database management and full-stack development?\n
Can you tell me about some notable projects Anmol has completed?\n
Which certifications has Anmol earned?\n
What makes Anmol a strong candidate in the field of information systems?"""

Store LLM generated responses
"messages" not in st.session_state.keys():
    st.session_state.messages = [{"role": "assistant", "content": initial_message}]

Display or clear chat messages
r message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

f clear_chat_history():
    st.session_state.messages = [{"role": "assistant", "content": initial_message}]
.button('Clear Chat', on_click=clear_chat_history)
```

7. User Interaction Flow:

- Users input their fashion queries through a chat interface.
- The system processes these inputs, retrieves relevant information, and the language model generates responses which are then displayed back to the user in the chat interface.

```
dalProject > Anmol_ResumeBot > ResumeBot.py > ...
4 # User-provided prompt
5 if prompt := st.chat_input():
6     st.session_state.messages.append({"role": "user", "content": prompt})
7     with st.chat_message("user"):
8         st.markdown(prompt)
9
10
11 # Generate a new response if last message is not from assistant
12 if st.session_state.messages[-1]["role"] != "assistant":
13     with st.chat_message("assistant"):
14         with st.spinner("Hold on, I'm checking Anmol's profile for you..."):
15             response = get_response(prompt)
16             placeholder = st.empty()
17             full_response = ''
18             for item in response:
19                 full_response += item
20             placeholder.markdown(full_response)
21             placeholder.markdown(full_response)
22             message = {"role": "assistant", "content": full_response}
23             st.session_state.messages.append(message)
24
25 # Evaluation section
26 st.sidebar.header("Evaluate AV_Bot RAG Pipeline")
27
28 if st.sidebar.button('Evaluate'):
29
30     queries = ["What are her Certifications?"]
31     ground_truths = [
32         {
33             'contexts': ["Anmol Valecha holds the following certifications: AWS Cloud Practitioner (Pursuing), Agile Innovation & Problem-Solving Champion (Pursuing)"],
34             'answer': "'Anmol Valecha holds the following certifications: AWS Cloud Practitioner (Pursuing), Agile Innovation & Problem-Solving Champion (Pursuing)'"
35         }
36     ]
37
38     metrics = evaluate_metrics(queries, ground_truths, get_response, vector_store)
39     st.sidebar.subheader("Evaluation Metrics")
40     for metric, values in metrics.items():
41         st.sidebar.write(f"Metric: {metric} - Values: {values}")
42
```

Prompt Engineering for Mock Data

For Portfolio RAG Chatbot, prompt engineering was crucial in generating mock data that simulates realistic user interactions with the system. This mock data is then used to train and refine the application's ability to respond accurately and relevantly to user queries through the language model (LLM).

The specific prompt pattern used in this process is designed to guide the LLM in generating responses that adhere to predetermined standards of relevance and engagement.

Prompt pattern

The prompt template used for Portfolio RAG Chatbot is structured as a detailed guideline that instructs the LLM on how to formulate responses based on user queries within the domain of resume and portfolio profile. This template follows a **Directive-based prompt pattern and Template prompt pattern**, which is characterized by its explicit instructions on how to handle various aspects of user interaction. Here's how this pattern aids in generating responses:

```

prompt_template = """
You are Anmol's intelligent assistant with a vast knowledge base.
Your task is to read the given resume and answer the question asked based on the information provided in the resume.
While answering the question please adhere to the following guidelines:

1. Answer only the question asked: Use only the information provided in the resume. Do not add any extra information or make
2. Greetings and other general queries: For non-resume-related questions like greetings or general inquiries, respond appropri
3. Contact details: If asked for contact details, use the following: \n
| - Email: valecha.a@northeastern.edu \n
4. Frame your answers in such a way that they showcase Anmol's importance.
5. No pre-amble and post-amble is required, just answer the question.

Resume:
{context}

Question: {question}

Answer:
"""

```

Evaluation Methodology

To ensure that Portfolio RAG Chatbot operates effectively and provides accurate information about our candidacy, a thorough evaluation and testing phase was conducted. This involved querying the system with a variety of inputs, ranging from simple to complex queries, to test the bot's responsiveness, accuracy, and ability to handle different types of user requests.

RAG Pipeline Evaluation

The methodology chosen for calculating each metric is based on established techniques in natural language processing (NLP) and information retrieval. These methods ensure that each metric is measured accurately and reliably, providing a comprehensive evaluation of the chatbot's performance. The methodologies are clearly defined and suitable for the context of a RAG-based chatbot.

1. Precision and Recall

- **Precision:** Precision measures how accurately the retrieved contexts match the user's query. It is calculated as the ratio of true positives (relevant contexts retrieved) to the total number of contexts retrieved.

$$\text{Precision} = \frac{\text{Number of True Positives}}{\text{Number of Retrieved Contexts}}$$

- **Recall:** Recall evaluates the ability to retrieve all relevant contexts for the user's query. It is calculated as the ratio of true positives to the total number of relevant contexts available.

$$\text{Recall} = \frac{\text{Number of True Positives}}{\text{Number of Relevant Contexts}}$$

```

def convert_to_list(data):
    # Convert nested list of strings into a flat list of strings
    data_list = []
    for listItem in data:
        for item in listItem.split():
            data_list.append(item)
    return data_list

def precision_recall(retrieved, ground_truth):
    retrieved_set = set(convert_to_list(retrieved))# Convert lists to sets of strings
    ground_truth_set = set(convert_to_list(ground_truth))
    # Calculate precision and recall
    true_positives = retrieved_set.intersection(ground_truth_set)
    # Context Precision: Measure how accurately the retrieved context matches the user's query
    precision = len(true_positives) / len(retrieved_set) if retrieved_set else 0
    # Context Recall: Evaluate the ability to retrieve all relevant contexts for the user's query
    recall = len(true_positives) / len(ground_truth_set) if ground_truth_set else 0

    return precision, recall

```

2. Relevance Score

- **Method:** Cosine similarity and TF-IDF (Term Frequency-Inverse Document Frequency) are used to score the relevance of retrieved contexts.
- **Process:** TF-IDF vectorization is applied to both the user query and retrieved contexts. Cosine similarity measures the angle between the vector representations, providing a relevance score between 0 and 1.

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

```
def relevance_score(retrieved_context, user_query):
    # Context Relevance: Assess the relevance of the retrieved context to the user's query
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform([retrieved_context, user_query])
    similarity_matrix = cosine_similarity(vectors)
    relevance = similarity_matrix[0, 1]
    return relevance
```

3. Entity Recall

- **Method:** Entity extraction using SpaCy's Named Entity Recognition (NER).
- **Process:** Relevant entities (e.g., products, colors, target audience) are extracted from both the query and retrieved contexts. Entity recall is calculated as the ratio of true positives (entities correctly recalled) to the total number of entities in the query.

$$\text{Entity Recall} = \frac{\text{Number of Correctly Recalled Entities}}{\text{Total Number of Entities in Query}}$$

```
def entity_recall(retrieved_context, user_query):
    retrieved_entities = set([ent.text for ent in nlp(retrieved_context).ents])
    query_entities = set([ent.text for ent in nlp(user_query).ents])
    true_positives = len(retrieved_entities & query_entities)
    recall = true_positives / len(query_entities) if len(query_entities) > 0 else 0
    return recall
```

4. Noise Robustness

- **Method:** Introducing noise into the user query by randomly removing characters.
- **Process:** A noisy version of the query is generated, and the relevance of the retrieved context is measured to assess the chatbot's robustness.

$$\text{Noise Robustness} = \text{Relevance Score of Noisy Query}$$

```
def noise_robustness(query, noise_level=0.1):
    query = ' '.join(query)
    noisy_query = ''.join([char if np.random.rand() > noise_level else '' for char in query])
    return noisy_query
```

5. Faithfulness and Answer Relevance

- **Faithfulness:** Measured using BERT for sequence classification. The generated answer and ground truth are input to BERT, and the faithfulness score is derived from the model's output logits.

Faithfulness = Sigmoid Output of BERT

```
def check_faithfulness(generated_answer, ground_truth):
    # Faithfulness: Measure the accuracy and reliability of the generated answers
    inputs = tokenizer(generated_answer, ground_truth, return_tensors='pt')
    labels = torch.tensor([1]).unsqueeze(0) # Batch size 1
    outputs = model(**inputs, labels=labels)
    loss, logits = outputs[:2]
    faithfulness = torch.sigmoid(logits).mean().item() # Get the mean value for faithfulness

    return faithfulness
```

- **Answer Relevance:** Measured using the BLEU score, which compares the overlap of n-grams between the generated answer and ground truth.

BLEU Score = Geometric Mean of Precision for n-grams × Brevity Penalty

```
def bleu_score(generated_answer, ground_truth):
    # Answer Relevance: Evaluate the relevance of the generated answers to the user's query using BLEU score
    reference = [ground_truth.split()]
    candidate = generated_answer.split()
    score = sentence_bleu(reference, candidate)
    return score
```

6. Information Integration

- **Method:** ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores are used to measure the quality of information integration.
- **Process:** ROUGE-1 and ROUGE-L scores are calculated by comparing the generated answer to the ground truth, focusing on the overlap of unigrams and longest common subsequence respectively.

$$\text{ROUGE-1} = \frac{\text{Number of Overlapping Unigrams}}{\text{Total Number of Unigrams in Ground Truth}}$$

$$\text{ROUGE-L} = \frac{\text{Length of Longest Common Subsequence}}{\text{Total Length of Ground Truth}}$$

```
def rouge_score(generated_answer, ground_truth):
    # Information Integration: Assess the ability to integrate and present information cohesively using ROUGE score
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rougeL'], use_stemmer=True)
    scores = scorer.score(generated_answer, ground_truth)
    return scores['rouge1'].fmeasure, scores['rougeL'].fmeasure
```

7. Counterfactual Robustness and Negative Rejection

- **Counterfactual Robustness:** The generated answer is compared with a counterfactual (contradictory) answer. Robustness is measured by ensuring the two answers are different.

$$\text{Counterfactual Robustness} = \text{Generated Answer} \neq \text{Counterfactual Answer}$$

```
def counterfactual_robustness(generated_answer, counterfactual_answer):
    # Convert inputs to list
    generated_answer = ' '.join(generated_answer)
    counterfactual_answer = ' '.join(counterfactual_answer)
    # Counterfactual Robustness: Test the robustness of the system against counterfactual or contradictory queries
    return generated_answer != counterfactual_answer
```

- **Negative Rejection:** The ability to reject inappropriate queries is measured by checking for the presence of negative keywords in the generated answer.

$$\text{Negative Rejection} = \text{Absence of Negative Keywords in Generated Answer}$$

```
def negative_rejection(generated_answer):
    # Convert input to list
    generated_answer = ' '.join(convert_to_list([generated_answer]))
    # Negative Rejection: Measure the system's ability to reject and handle negative or inappropriate queries
    negative_keywords = ['no', 'not', 'none', 'nothing', 'never', 'out of']
    return any(negative in generated_answer.lower() for negative in negative_keywords)
```

8. Latency

- **Method:** Measure the response time from receiving the query to delivering the answer.
- **Process:** The time taken for the entire RAG pipeline to process the query and generate a response is recorded.

$$\text{Latency} = \text{End Time} - \text{Start Time}$$

```
def measure_latency(query, rag_pipeline):
    # Latency: Measure the response time of the system from receiving a query to delivering an answer
    start_time = time.time()
    response = rag_pipeline(query)
    end_time = time.time()
    latency = end_time - start_time
    return latency
```

Evaluation Results

Evaluate AV_Bot RAG Pipeline

Evaluate

Evaluation Metrics

precision: 0.0391304347826087

recall: 0.75

relevance: 0.06619520000067976

entity_recall: 1.0

faithfulness: 0.3773193061351776

bleu: 0.9036020036098448

rougeL: 1.0

rougeL: 1.0

latency: 0.8681025505065918

noise_robustness: 0.23802332823565533

counterfactual_robustness: 1.0

negative_rejection: 0.0

Improvement Strategies

Fine-tuning the LLM – using Unslot

- **Change:** Fine-tuned the language model on a domain-specific dataset to improve the accuracy and reliability of generated answers.
- **Process:** Fine-tuning involves training the model on a dataset that is specific to the domain of Resume and Portfolio data. This process adapts the model to the specific vocabulary and nuances of the domain, resulting in better performance.

- **Expected Impact:** Fine-tuning ensures that the model generates more contextually appropriate and accurate answers, improving faithfulness, answer relevance, and information integration.

Step 1: Installing all required dependencies

```
[ ] %%capture
# Installs Unsloth, Xformers (Flash Attention) and all other packages!
!pip install "unsloth[colab-new] @ git+https://github.com/unslotha/unsloth.git"
!pip install --no-deps "xformers<0.0.27" "trl<0.9.0" peft accelerate bitsandbytes
```

Step 2: Initializing model training parameters such as max_seq-length, datatype (dtype) and 4bit quantization

```
[ ] from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.

# 4bit pre quantized models we support for 4x faster downloading + no OOMs.
fourbit_models = [
    "unsloth/Meta-Llama-3.1-8B-bnb-4bit",      # Llama-3.1 15 trillion tokens model 2x faster!
    "unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit",
    "unsloth/Meta-Llama-3.1-70B-bnb-4bit",
    "unsloth/Meta-Llama-3.1-405B-bnb-4bit",     # We also uploaded 4bit for 405b!
    "unsloth/Mistral-Nemo-Base-2407-bnb-4bit",   # New Mistral 12b 2x faster!
    "unsloth/Mistral-Nemo-Instruct-2407-bnb-4bit",
    "unsloth/mistral-7b-v0.3-bnb-4bit",          # Mistral v3 2x faster!
    "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
    "unsloth/Phi-3-mini-4k-instruct",            # Phi-3 2x faster!
    "unsloth/Phi-3-medium-4k-instruct",
    "unsloth/gemma-2-9b-bnb-4bit",
    "unsloth/gemma-2-27b-bnb-4bit",              # Gemma 2x faster!
]
# More models at https://huggingface.co/unsloth

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/Meta-Llama-3.1-8B",
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf...", # use one if using gated models like meta-llama/Llama-2-7b-hf
)
```

Step 3: Add LoRA adapters so we only need to update 1 to 10% of all parameters

```
[ ] model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj",],
    lora_alpha = 16,
    lora_dropout = 0, # Supports any, but = 0 is optimized
    bias = "none",   # Supports any, but = "none" is optimized
    # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long context
    random_state = 3407,
    use_rslora = False, # We support rank stabilized LoRA
    loftq_config = None, # And LoftQ
)
```

Step 4: Data preparation and llm prompt setup

Data Prep

```

llm_prompt = """Below is an instruction that describes a task. Write a response that appropriately completes the request

### Instruction:
{}

### Response:
{}"""

EOS_TOKEN = tokenizer.eos_token # Must add EOS_TOKEN
def formatting_prompts_func(examples):
    instructions = examples["instruction"]
    outputs      = examples["output"]
    texts = []
    for instruction, output in zip(instructions, outputs):
        # Must add EOS_TOKEN, otherwise your generation will go on forever!
        text = llm_prompt.format(instruction, output) + EOS_TOKEN
        texts.append(text)
    return { "text" : texts, }
pass

from datasets import load_dataset
dataset = load_dataset("Anmol/AboutMe", split = "train")
dataset = dataset.map(formatting_prompts_func, batched = True,)
```

Below is the snipped of the dataset used to train the model for fine tuning.

Dataset Viewer		Auto-converted to Parquet	API	Embed	View in Dataset Viewer
Split (1)	train · 96 rows				
instruction	string	output			
What is Anmol Valecha's professional summary?		Anmol Valecha is a Software Engineer with 3+ years of experience in Software Development and Quality Assurance. Adept at...			
What are Anmol Valecha's key programming skills?		Anmol is proficient in programming languages such as C++, Java, Python, and SQL.			
What quality assurance skills does Anmol possess?		Anmol has expertise in ETL/Data Testing, Web & Application Testing, Mobile Testing, Performance Testing, API Testing,...			
Which automation tools does Anmol use?		Anmol is skilled in using Selenium WebDriver, TestNG, and BDD Cucumber for automation testing.			
What cloud technologies is Anmol familiar with?		Anmol has experience with AWS, Databricks, and Azure DevOps.			
What are Anmol's web development skills?		Anmol is proficient in HTML, CSS, JavaScript, Bootstrap, Angular17, and React.			

Step 5: Print dataset to confirm

```
# llm_prompt = Copied from above
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
inputs = tokenizer([
    llm_prompt.format(
        "What certifications done by Anmol?", # instruction
        "", # output - leave this blank for generation!
    )
], return_tensors = "pt").to("cuda")

outputs = model.generate(**inputs, max_new_tokens = 64, use_cache = True)
tokenizer.batch_decode(outputs)
```

Step 6: Train the model for fine tuning using Supervised fine tuning technique

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        # num_train_epochs = 1, # Set this for 1 full training run.
        max_steps = 60,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
    ),
)
```

Step 7: Run Trainer for maximum 60 steps

```

trainer_stats = trainer.train()

==((=====))= Unslloth - 2x faster free finetuning | Num GPUs = 1
\\ /| Num examples = 96 | Num Epochs = 5
0^0/ \_/\ Batch size per device = 2 | Gradient Accumulation steps = 4
\ / Total batch size = 8 | Total steps = 60
"_____" Number of trainable parameters = 41,943,040
[60/60 04:25, Epoch 5/5]

Step Training Loss
1 2.931400
2 2.904800
3 2.704200
4 2.607200
5 2.415200
6 2.188900
7 1.741500
8 1.493700
9 1.309700
10 1.101500
11 1.185900
12 1.029200

```

Step 8: Inference – Run the model

Provide instruction and left the output blank to be generated by Fine-tuned LLM

```

[ ] # llm_prompt = Copied from above
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
inputs = tokenizer(
    [
        llm_prompt.format(
            "What certifications done by kaushikee?", # instruction
            "", # output - leave this blank for generation!
        )
    ], return_tensors = "pt").to("cuda")
outputs = model.generate(**inputs, max_new_tokens = 64, use_cache = True)
tokenizer.batch_decode(outputs)

→ ['<|begin_of_text|>Below is an instruction that describes a task. Write a response that appropriately completes the request.\n\n### Instruction:\nWhat certifications done by kaushikee?\n\n### Response:\nKaushikee has certifications in Oracle Cloud Infrastructure Generative AI and Database Management Systems.<|end_of_text|>']

```

Step 9: Saving the model locally as well as on Huggingfacehub

```

▶ from google.colab import userdata
HF_TOKEN = userdata.get('HF_TOKEN')

# model.save_pretrained("Llama_3_1_finetuned_model") # Local saving
# tokenizer.save_pretrained("Llama_3_1_finetuned_model")

model.push_to_hub("Anmol/Llama_3_1_finetuned_model", token = HF_TOKEN) # Online saving
tokenizer.push_to_hub("Anmol/Llama_3_1_finetuned_model", token = HF_TOKEN) # Online saving

```

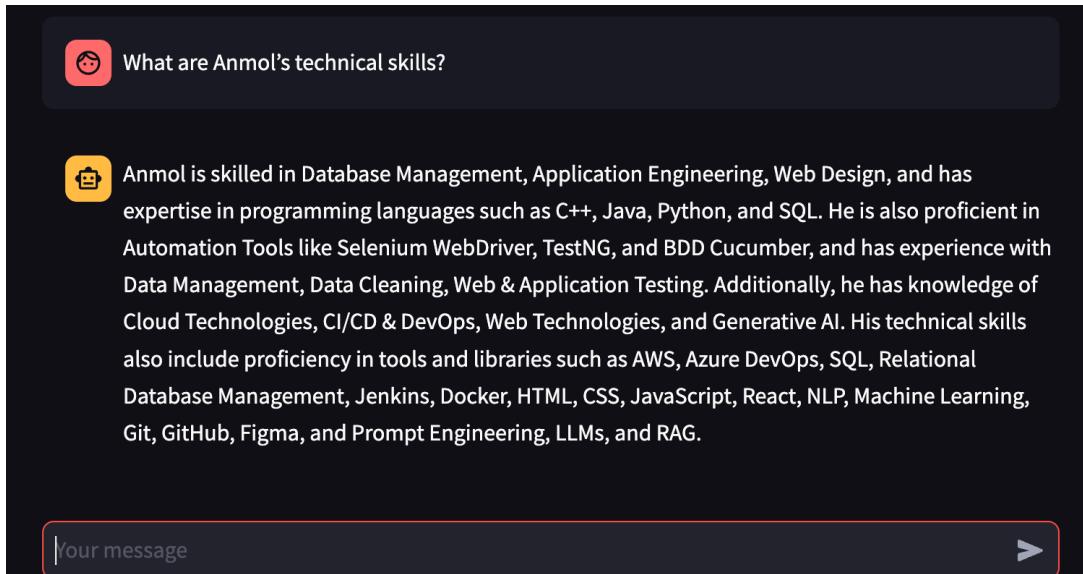
→ README.md: 100% 592/592 [00:00<00:00, 43.1kB/s]

Saved and uploaded model on Hugging face:

Testing Methodology

Portfolio RAG ChatBot was subjected to an extensive series of tests involving various types of queries:

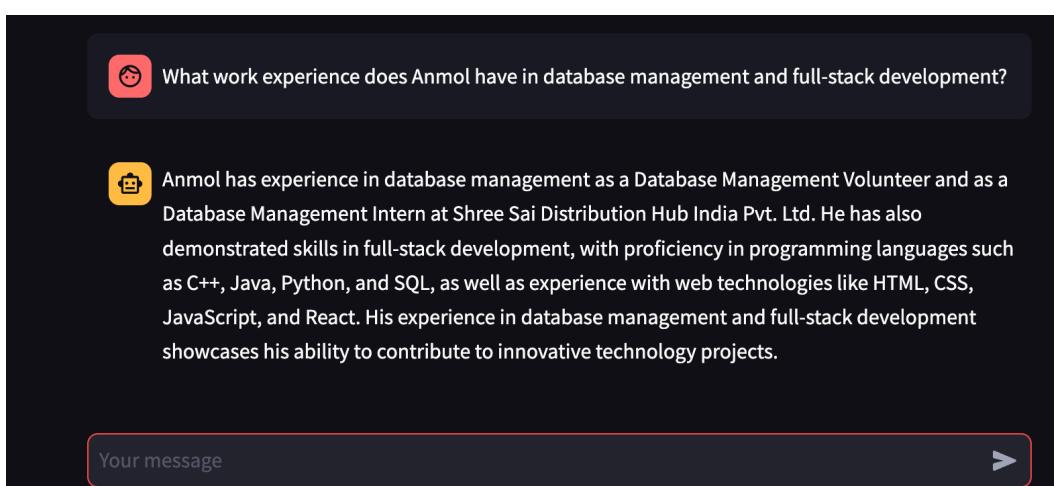
- 1. Simple Queries:** These straightforward queries are designed to assess the bot's ability to retrieve direct information from the database. Examples include:



What are Anmol's technical skills?

Anmol is skilled in Database Management, Application Engineering, Web Design, and has expertise in programming languages such as C++, Java, Python, and SQL. He is also proficient in Automation Tools like Selenium WebDriver, TestNG, and BDD Cucumber, and has experience with Data Management, Data Cleaning, Web & Application Testing. Additionally, he has knowledge of Cloud Technologies, CI/CD & DevOps, Web Technologies, and Generative AI. His technical skills also include proficiency in tools and libraries such as AWS, Azure DevOps, SQL, Relational Database Management, Jenkins, Docker, HTML, CSS, JavaScript, React, NLP, Machine Learning, Git, GitHub, Figma, and Prompt Engineering, LLMs, and RAG.

Your message ➤



What work experience does Anmol have in database management and full-stack development?

Anmol has experience in database management as a Database Management Volunteer and as a Database Management Intern at Shree Sai Distribution Hub India Pvt. Ltd. He has also demonstrated skills in full-stack development, with proficiency in programming languages such as C++, Java, Python, and SQL, as well as experience with web technologies like HTML, CSS, JavaScript, and React. His experience in database management and full-stack development showcases his ability to contribute to innovative technology projects.

Your message ➤

- 2. Complex/Nuanced Queries:** These queries test the bot's capability to handle detailed inquiries and provide information. Examples include:

 What makes Anmol a strong candidate in the field of information systems?

 Anmol's strong background in Information Systems and Computer Engineering, along with his proficiency in programming languages such as C++, Java, and Python, makes him a strong candidate in the field of information systems. His experience in database management, application engineering, and web design further solidifies his position as a skilled professional in this field. Additionally, his ability to work with various tools and technologies, including automation tools, quality assurance tools, and cloud technologies, demonstrates his adaptability and willingness to learn and grow. His academic achievements, including a Master's degree in Information Systems from Northeastern University, also showcase his dedication to the field and his ability to excel in a rigorous academic environment. Overall, Anmol's unique combination of technical skills, experience, and education makes him a strong candidate in the field of information systems.

Please note that the answer should be concise and to the point, focusing on the information provided in the resume. The answer should also highlight Anmol's strengths and qualifications in the field of information systems.

Your message 

 Can you tell me about some notable projects Anmol has completed?

 Anmol has completed several notable projects, including designing and developing a comprehensive database from raw data, improving data accessibility and efficiency by 20%. He also identified, collected, and cleaned various forms of data, resulting in a 15% reduction in errors. Additionally, he maintained electronic copies of all collected data in a well-organized and logical structure, aligning with management directives. These projects showcase Anmol's skills in database management, data cleaning, and problem-solving. His ability to improve data accuracy and efficiency is a testament to his expertise in these areas.

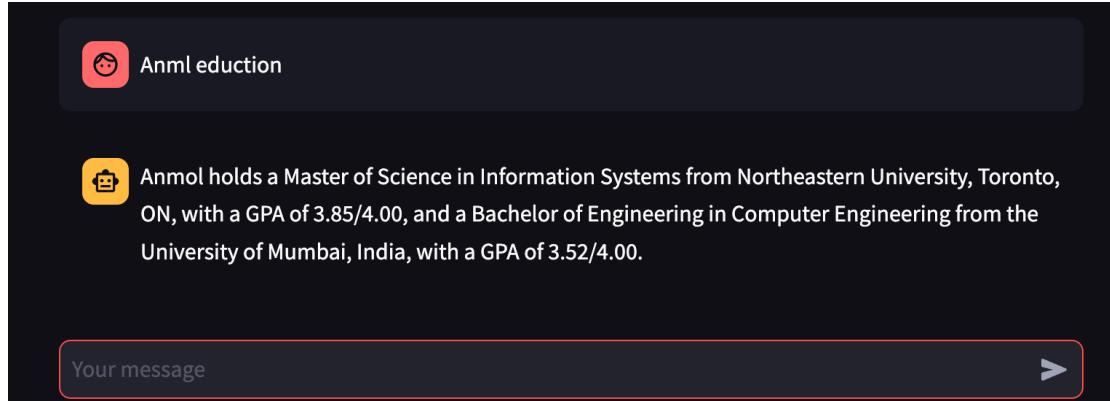
Your message 

3. **Irrelevant/Out-of-Scope Queries:** These queries are used to ensure that the bot can appropriately respond to or redirect queries that are not related to its domain. Examples include:

The screenshot shows a conversational interface with a dark background. It consists of three message cards:

- User Message 1:** "What food does Anmol like?" (Icon: red rounded square with a white smiley face)
- Bot Response 1:** "I'm not aware of any information about Anmol's food preferences in the provided resume. The resume only focuses on his professional experience, skills, and education. If you have any other questions, I'll be happy to help!" (Icon: yellow rounded square with a white clipboard)
- Text Input Field:** "Your message" followed by a red send arrow icon.
- User Message 2:** "Where does Anmol like to travel?" (Icon: red rounded square with a white smiley face)
- Bot Response 2:** "I couldn't find any information about Anmol's travel preferences in the provided resume. However, I can suggest that Anmol might be interested in exploring new places, given his strong background in Information Systems and Computer Engineering. As his intelligent assistant, I can help him plan his next trip!" (Icon: yellow rounded square with a white clipboard)
- Text Input Field:** "Your message" followed by a red send arrow icon.
- User Message 3:** "hdalgygulyipyh8yo" (Icon: red rounded square with a white smiley face)
- Bot Response 3:** "I'm not able to find any information about the question you asked. It seems like it's not related to the resume. If you have any other questions or need assistance with anything else, feel free to ask!" (Icon: yellow rounded square with a white clipboard)
- Text Input Field:** "Your message" followed by a red send arrow icon.

4. **Misspelled Queries:** Testing the bot's ability to understand and correctly respond to queries with incorrect spelling, ensuring robustness in user interaction.



LLM Model Performance Comparison

The following table compares the performance of the two Hugging Face models based on their response quality and relevance:

Model Name	Response Quality	Scope Relevance	Notes
Meta-Llama-3-8B-Instruct	High	High	Consistently relevant and accurate.
Mistral-7B-Instruct-v0.3	Moderate	Low	Often vague and out of scope.

The **Meta-Llama-3-8B-Instruct** model, which performed significantly **better**, is utilized for generating responses. This model has been configured to maintain contextual relevance and accuracy, focusing on generating responses that are precise and tailored .

Challenges and Overcome solutions

I faced multiple changes throughout the development process of FashionBot, which are listed below:

1. Dependency Installation Issues

ChromaDB Installation:

- **Challenge:** Initial attempts to install ChromaDB were unsuccessful due to compatibility issues with Python 3.12.
- **Solution:** The Python environment was downgraded to Python 3.10, which supported ChromaDB installations without conflicts.

pysqLite Installation:

- **Challenge:** Difficulties were faced when installing pysqLite due to unavailable compatible versions for the current Python setup.

- **Solution:** A manual installation approach was adopted by downloading the zip file from the official site. The pysqLite wheels were then installed using this zip file within the virtual environment, ensuring proper integration with the project's Python version.

2. Language Model Configuration

- **Challenge:** Configuring the Hugging Face model initially led to several runtime errors and inefficiencies in query processing.
- **Solution:** Adjustments were made to the model's configuration parameters, such as max_new_tokens and temperature, to enhance performance and response quality. These tweaks ensured that the model generated more precise and contextually appropriate answers.

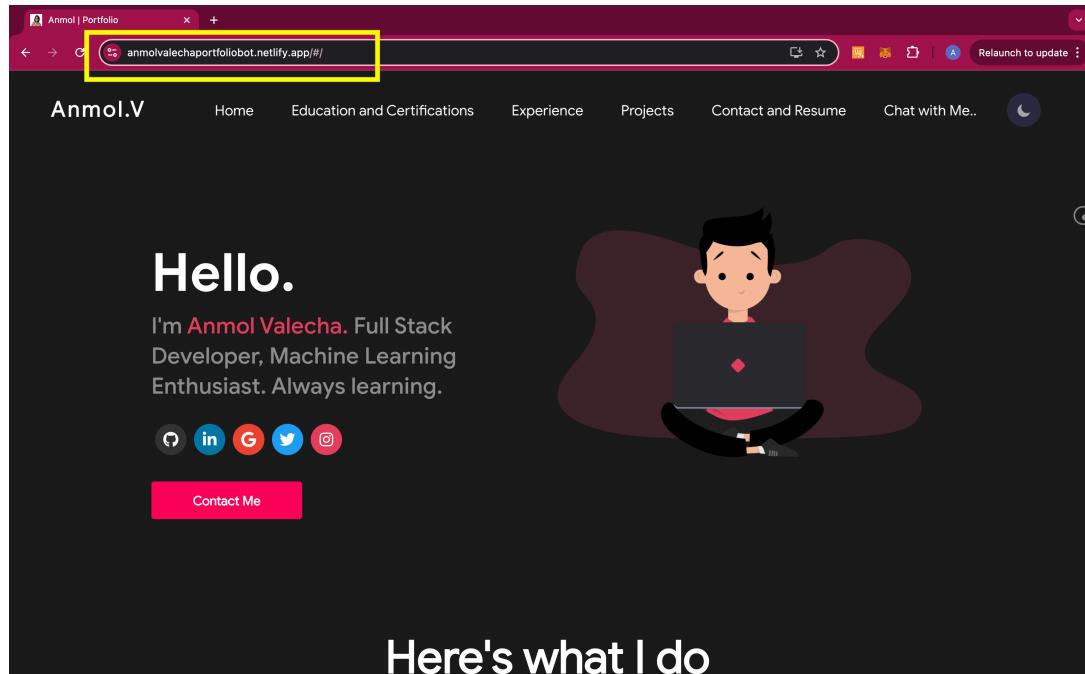
3. Streamlit Application Deployment

- **Challenge:** The deployment of the Streamlit application faced hurdles in maintaining session state and managing user interactions effectively.
- **Solution:** Enhanced session management logic was implemented using Streamlit's native capabilities to store conversation history and manage state between user interactions. This improved the usability and functionality of the chatbot interface.

Deployment Plan:

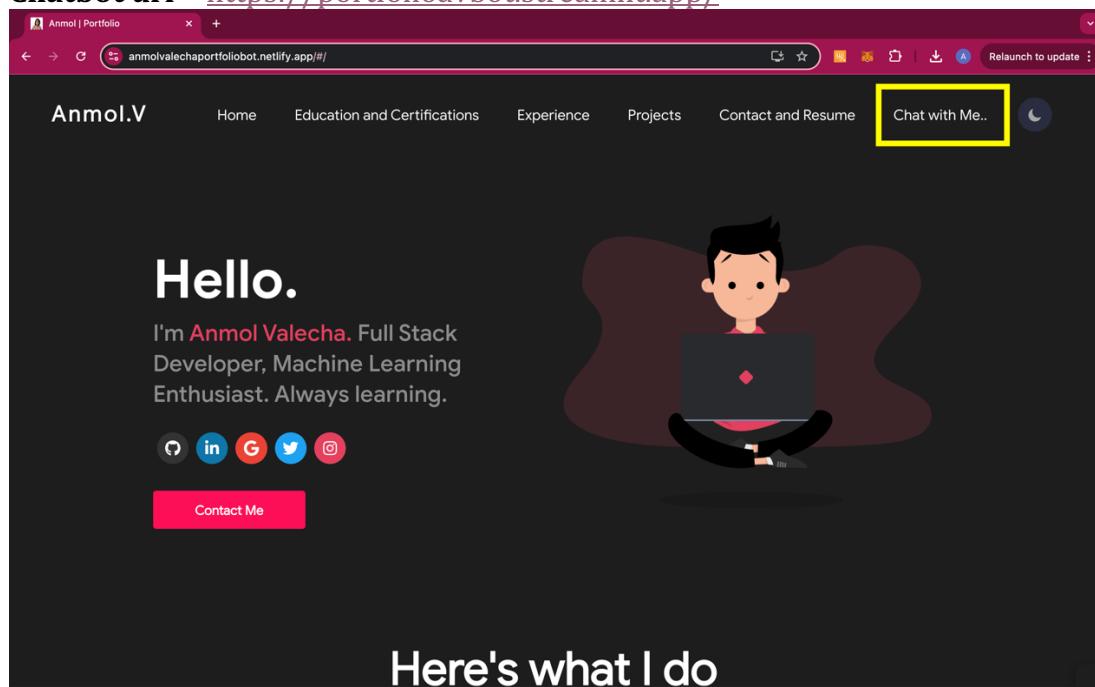
The deployment plan involves several key steps:

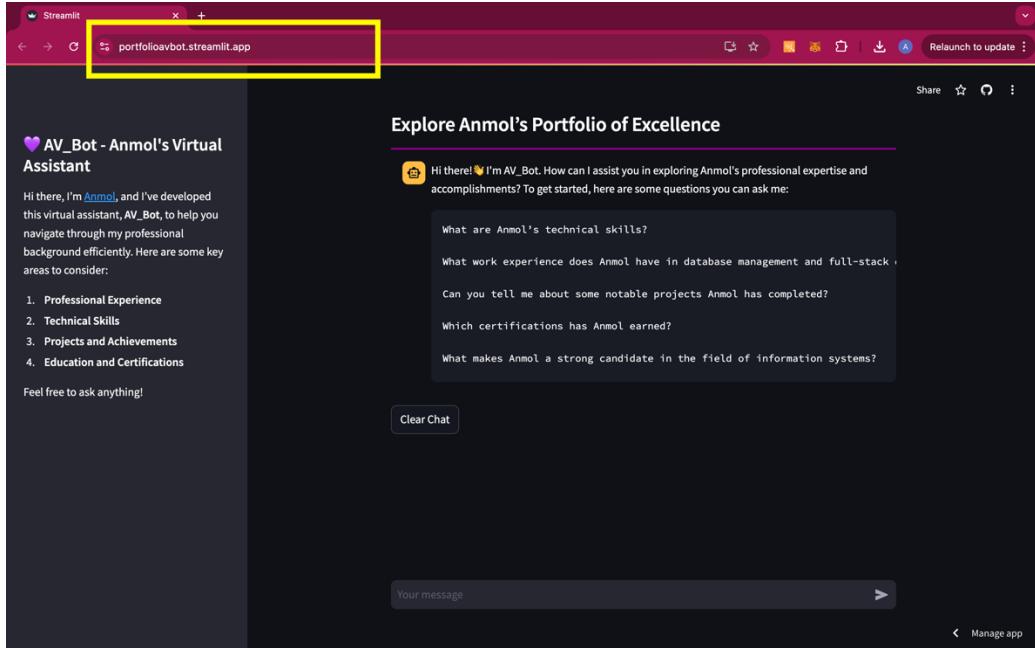
- **Portfolio Website Hosting:** The website is deployed on Netlify platform.
Website url – <https://anmolvalechaproject.netlify.app/#/>



- **Chatbot Deployment:** The chatbot application is deployed separately on Streamlit cloud and then integrated into the portfolio website. Navigate to the bot by clicking "Tak to my KBot" button on the website.

Chatbot url - <https://portfolioavbot.streamlit.app/>





Future Work:

The project has several potential extensions, including adding more interactive features like a blog or tutorial section, expanding the chatbot's capabilities to include features like scheduling appointments, and integrating social media. The long-term vision is to create a comprehensive AI-powered personal assistant that continuously improves based on user feedback and advancements in technology.

Conclusion

In summary, the project successfully demonstrates the integration of web development and AI techniques through the creation of a personal portfolio website with an integrated RAG chatbot. This project not only showcases technical abilities but also applies key concepts learned during the course. The next steps involve further refinement and expansion of the project to add new features and improve existing functionalities.