RAG-BASED CHATBOT ON AWS EC2

Devapangu Abhishek(21bds016), Hosur Sai Kartik(21bds021),
Nischay Kondai(21bds045), Pratik Raj(21bds047)

***Abstract*— Our RAG-based chatbot application utilizes Streamlit and Natural language processing libraries like Google Generative AI to extract information from PDFs and offer conversational responses. Users can upload PDFs, ask questions, and receive real-time answers. Key features include text extraction, vector embeddings, similarity search, and conversational AI. Hosting it on an AWS EC2 instance ensures scalability and reliability. We specify instance details like names and tags, security groups, and storage volumes, and then launch instances within a virtual private cloud. Through this project, we aim to provide users with an intuitive and efficient platform for extracting insights from PDF documents through natural language interaction.**

***Key Words*: Text Extraction, Conversational AI, AWS EC2**

I. INTRODUCTION

In today's digital world, a lot of information is frequently found in PDF documents, making it difficult for users to effectively extract insights. Our research provides a web-based chat application that aims to solve this issue by enabling smooth extraction of text from PDF files and then give responses to the queries relating to the text in the file. Using Streamlit and natural language processing tools such as Google Generative AI and Langchain, our application enables users to ask questions, upload PDF files, and get real-time conversational responses in the User Interface supported by Streamlit. Utilising technologies like vector embeddings, text extraction, and similarity search, our platform improves PDF content's usability and accessibility. This study offers a comprehensive analysis of the development process, key features, and overall objective of providing customers with a simple and easy-to-use platform for natural language interaction-based insight extraction from PDF documents.

II. MODEL USED FOR THE CHATBOT

We needed some basic functionalities in our RAG (Retrieval Augmented Generation) based chatbot to provide better experience and responses relating to the input PDF file. Some of the functions used in our chatbot using the imported libraries are as follows:

- **Reading text from the input file:** Using the PyPDF2 library, we have created get_pdf_text function which takes all the PDF files through a file uploader in the UI, to read the text content from PDF files uploaded by users and concatenate it for further processing. The function iteratively extracts text through each document and concatenates it for further steps. It returns the final consolidated string as the output, which will be used further in the process.
- **Segmenting the string into chunks:** It is possible that there may be some resource constraints when the string generated from a larger file is too long and thus difficult to process.The large string output from the previous function thus needs to be divided into smaller text chunks. By segmenting the string into chunks, the model ensures fine analysis and parallel processing, thus keeping the efficiency similar even in case of larger input files. Hence, we have created get_pdf_chunk function to break down the string into smaller chunks for better results and performance.
- **Converting text chunks into vector embeddings and Storing it for analysis:** One of the most important functions in the application is get_vector_store(text_chunks), which allows text chunks taken out of PDF documents to be converted into vector embeddings and then stored for quick retrieval and analysis. The first step is to initialise an instance of the GoogleGenerativeAIEmbeddings class, which creates vector embeddings from text data using Google's Generative AI model. This initialization process makes sure that the embeddings come from an advanced language model that can capture subtle semantic details. The produced embeddings are then used to create a vector store using the FAISS (Facebook AI Similarity Search) library. The FAISS class's from_texts() method makes this process easier by using the supplied text chunks to create a vector store, with the embeddings acting as the text data's underlying representation. Finally, the constructed vector store is saved to the local filesystem using the save_local() method, ensuring that the embeddings are preserved for future use in tasks such as similarity search operations. Overall, this function simplifies the process of preparing text data for analysis, enhancing the application's ability to process and understand textual content effectively.
- **Setting Up Conversational Question-Answering Chain:** Setting Up Conversational Question-Answering Chain initiates a chain of conversational questions and answers within the program. It begins by defining a prompt template, providing instructions on how to formulate answers based on the provided topic and context. The template guides the model to utilize the context to deliver comprehensive answers to questions, explicitly clarifying if the answer is unavailable. Subsequently, a ChatGoogleGenerativeAI model, named "gemini-pro", is configured with a temperature value of 0.3 to control response creativity. Utilizing the defined prompt template, an instance

of PromptTemplate is constructed, specifying input variables "context" and "question" to direct the conversational chain. Finally, the configured model and prompt template instantiate the question-answering chain, specifying "stuff" as the chain type for general question-answering tasks. This configured chain is then returned, equipped to handle user inquiries and provide conversational responses based on the predefined guidelines.

- **Building the UI:** And at the end, in the main function, we built the UI of the chatbot and called all the necessary functions listed above.
- **Generating reponses:** Finally, the input from the user needs to be read, processed and then an appropriate response needs to be generated. For this, we have created the user_input function to which user_question parameter is passed. The input is passed by the user via the Streamlit user interface. The function initializes a GoogleGenerativeAIEmbeddings object. This object is responsible for generating embeddings (vector representations) of text data using the specified model. Next, the function loads the vector store (index) from a local file named "faiss_index" using the FAISS library. The embeddings object initialized earlier is passed as an argument, allowing the vector store to utilize the same embedding model for similarity calculations. After loading the vector store, the function performs a similarity search using the user's question as the query. This search aims to identify relevant documents or chunks of text from the vector store that are similar to the user's query. The results of the similarity search are stored in the docs variable, which likely contains the relevant text chunks retrieved from the vector store based on their similarity to the user's question. Now, a conversational chain is initialised, which is responsible for processing the user's question and generating a response based on the provided context and any available information in the text documents. Now, the response from the conversational chain is printed on the Streamlit UI.

III. DEPLOYING CHATBOT ON AWS EC2 INSTANCE

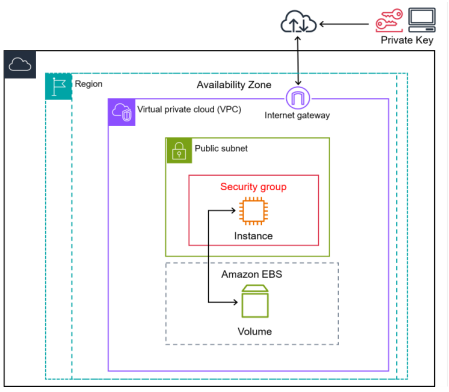
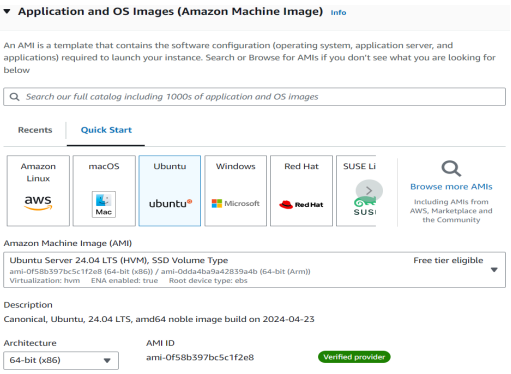


Fig. 1: EC2 basic Architecture

Amazon EC2 (Elastic Compute Cloud) is a web service provided by Amazon Web Services (AWS) that allows you to rent virtual computers, known as instances, to run your applications. An EC2 instance is essentially a virtual machine in the cloud, providing computing resources such as CPU, memory, storage, and networking capabilities. When launching an EC2 instance, several technical specifications which we need to set, to ensure a smooth and secure operation. The specifications are as follows:

- Firstly, there are names and tags. These are labels that you need to assign to EC2 instances for organizational purposes, aiding in efficient identification and management, particularly in large-scale deployments.
- Next, the AMI (Amazon Machine Image) serves as the foundation of your instance. It's a template containing the operating system, software packages, and configuration settings necessary for launch. In our project, we had used the Ubuntu 24.04 LTS (Long Term Support) AMI template. The persistent storage volumes for EC2 instances is provided by EBS (Elastic Block Store). When you launch an EC2 instance using an Ubuntu AMI, you have the option to attach one or more EBS volumes to the instance. These volumes serve as the primary storage for your data, applications, and operating system files. While the Ubuntu AMI provides the template for the instance's operating system and software configuration, EBS volumes provide the storage capacity needed to store data and applications running on the instance.



- Instance type is also crucial. It defines the computing resources available to your EC2 instance, including CPU, memory, storage, and network performance. Choosing the right type is essential for meeting the chatbot's workload requirements.
- Key pair login files are fundamental for secure access to the instance. When launching an EC2 instance, you specify a key pair consisting of a public and private key. The private key is used for secure SSH (Secure Shell) access, enhancing the instance's security. We need to save the key-pair file and access the instance everytime through SSH command.
- Network settings, including placement within a virtual private cloud (VPC), subnet, and security groups, determine the instance's network configuration. Security

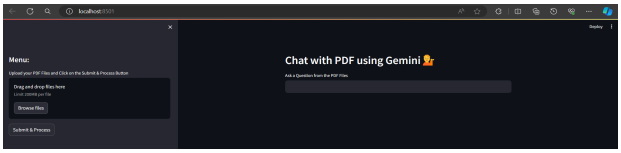
groups act as virtual firewalls, controlling inbound and outbound traffic to the instance. For our project, we had edited and added new security group rule of 'Custom TCP' type, setting the eligible port number and source type to 'Anywhere'.

Deploying the chatbot on an EC2 instance had the following advantages:

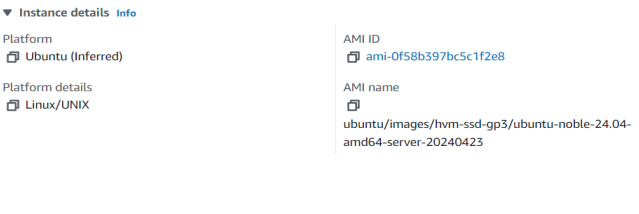
- **Scalability:** EC2 offers the flexibility to easily scale computing resources up or down based on demand. If your chatbot experiences a sudden surge in traffic, you can quickly add more instances to handle the increased load. Conversely, you can scale down during periods of lower demand to optimize costs.
- **Customization:** EC2 instances allow for full customization of the computing environment. You have control over the operating system, software configurations, and security settings, enabling you to tailor the environment to the specific requirements of your chatbot.
- **Reliability:** AWS ensures high availability and reliability of EC2 instances through redundancy and fault tolerance mechanisms. Your chat canbot benefit from the resilience of AWS infrastructure, minimizing downtime and ensuring consistent performance.

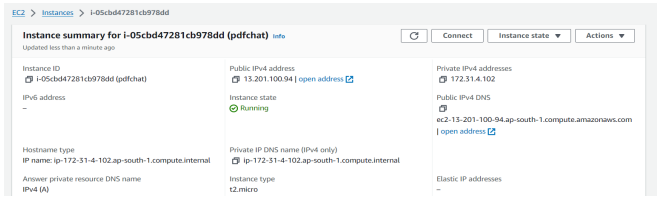
IV. RESULTS

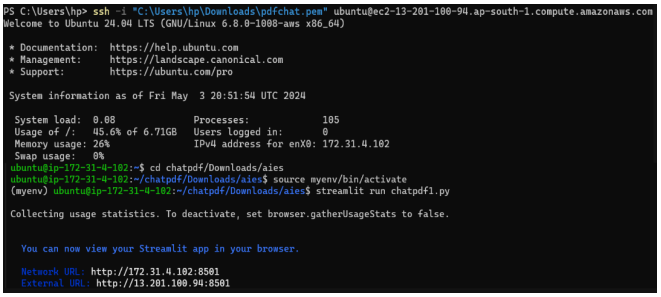
We successfully built our chatbot on Streamlit UI using NLP packages like Langchain and Google GenerativeAI.

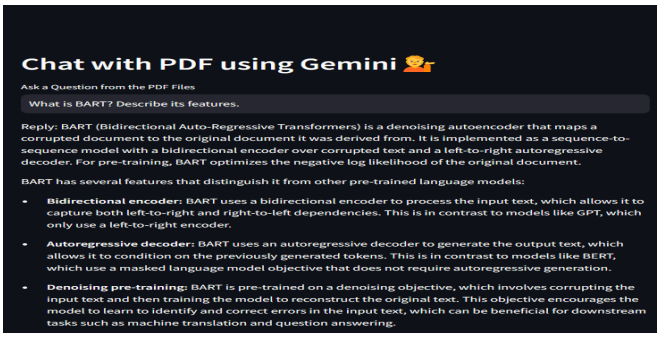


After ensuring the chatbot has been working properly on local, we successfully created an EC2 instance and then accessed the created instance with the saved the pem key file using SSH command in the terminal. Then, a virtual environment was successfully activated which was able to run the chatbot py file. We used the external URL to open the deployed chatbot, in which we came out to be successful.









V. CONCLUSION

The successful deployment of our web-based chat application on AWS EC2 helps in providing users with an intuitive platform for interacting with PDF documents. By using Streamlit and advanced language processing libraries, including Google Generative AI, we have created a seamless experience for users to upload PDFs, ask questions, and receive conversational responses in real-time. The deployment on AWS EC2 ensures scalability, reliability, and accessibility, allowing users to access the application from anywhere, anytime. The integration of FAISS for efficient similarity search and storage of vector embeddings further enhances the application's capabilities. Overall, our project aims to empower users to extract valuable insights from PDF documents through natural language interaction, and the deployment on AWS EC2 provides a robust infrastructure to support.

VI. ACKNOWLEDGEMENT

Special thanks to Dr. Animesh Chaturvedi for giving us this opportunity to work on a popular language processing tools such as Google Generative AI and cloud services such as Amazon Web Services(AWS).

REFERENCES

- **Rahul Saini & Rachna Behl. (2020).**An Introduction to AWS—EC2 (Elastic Compute Cloud)
- **Lalit Kumar, Pooja & Pradeep Kumar. (2021).** Amazon EC2: (Elastic Compute Cloud) Overview
- **Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini & Hervé Jégou. (2024).** The Faiss library
- **Adith Sreeram A.S. & Pappuri Jithendra Sai. (2023).** An Effective Query System Using LLMs and LangChain
- **Amazon EBS Documentation**