

# **INTERNSHIP PROJECT** **REPORT**

**Natural Language Processing and Large Language  
Models**

*By*

**Dhruv Mehrotra**

1/07/2024 to 15/09/2024

**Centre of Excellence for Data Analytics (CEDA)**  
**National Informatics Centre Services Incorporated**  
**(NICS)**

*Under the Guidance of*

**Shree Kumar Jyoti**

**Senior Manager, NICS**

# INDEX

<b>ABSTRACT</b>	<b>3</b>
<b>PREREQUISITES</b>	<b>4</b>
<b>Code Management and Development</b>	<b>4</b>
Docker:	4
Version Control System:	7
Python	11
<b>MACHINE LEARNING PROJECTS</b>	<b>16</b>
<b>Creating generative chatbots using NLP</b>	<b>16</b>
Abstract	16
Problem Statement	16
Methodology	17
Significance:	18
Code	18
Code Explanation	22
Scope	23
<b>Handled use cases</b>	<b>24</b>
Use case 1: <b>Automated Responses:</b> Answer frequently asked questions and resolve common issues.	24
Use case 2: <b>Ticketing &amp; Escalation:</b> Handle basic inquiries and escalate complex issues to human agents.	26
Use case 3: <b>24/7 Availability:</b> Provide round-the-clock support without human intervention.	29
<b>Conclusion</b>	<b>31</b>

## Abstract:

Data science is a multidisciplinary field that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data. It involves a systematic process of collecting, processing, analyzing, and interpreting vast amounts of data to inform decision-making, uncover patterns, and predict future trends. With the exponential growth of data generated by modern technologies, data science has become an essential tool across various industries like finance etc.

This report presents an exploration of Python which is a versatile and powerful programming language that has become a cornerstone in various fields of computing, particularly in data science and machine learning. Known for its simplicity and readability, Python offers a gentle learning curve, making it accessible to both beginners and experienced developers.

Natural Language Processing (NLP) focuses on enabling machines to comprehend, process, and respond to human language. **Generative chatbots** use NLP and AI to generate realistic, context-aware conversations. **Large Language Models (LLMs)** like GPT are advanced AI systems trained on extensive datasets, allowing them to produce human-like text. LLMs power generative chatbots by enabling them to create responses that are contextually accurate, flexible, and coherent, enhancing user interactions.

Author: \_\_\_\_\_

Dhruv Mehrotra

Certified by: \_\_\_\_\_

Shree Kumar Jyoti

(Industrial Coordinator)

# PREREQUISITES

## Code Management and Development

### Docker:

Initially, software(s) came with low portability, and that was because of multiple dependencies encountered whilst their creation, for example use of framework catering to a certain language, which makes these software inappropriate for execution on multiple systems or machines. Thus, came a need to effectively use software created on one machine on another, which led to the rise of virtual machines. Virtual machines also known as a Virtual Box is defined as a computer resource that functions like a physical computer and makes use of software resources only instead of using any physical computer for functioning, running programs, and deploying the apps.

A virtual machine is a program that emulates a complete computer and imitates dedicated hardware. It shares physical hardware resources with other users but isolates the operating system. The end user has the same experience on a Virtual Machine as they would have on dedicated hardware; in simple terms a guest operating system was to be installed for each application that is to be run on host OS.

This although eliminated the problem with dependencies, created another nuisance of **overheads**. Installing separate OS for every application was indeed gruesome and not cost efficient. This in turn paved a way for docker.

Docker is a tool that allows developers to easily deploy their applications in a box called container to run on the host operating system. The key benefit of Docker is that it allows users to package an application with all of its dependencies into a standardized unit for software development.

Unlike virtual machines, containers do not have high overhead and hence enable more efficient usage of the underlying system and resources.



Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data centre, the public cloud, or even a developer's personal laptop. This gives developers the ability to create predictable environments that are isolated from the rest of the applications and can be run anywhere. Containers take a different approach in comparison to virtual machines by leveraging the low-level mechanics of the host operating system, containers provide most of the isolation of virtual machines at a fraction of the computing power.

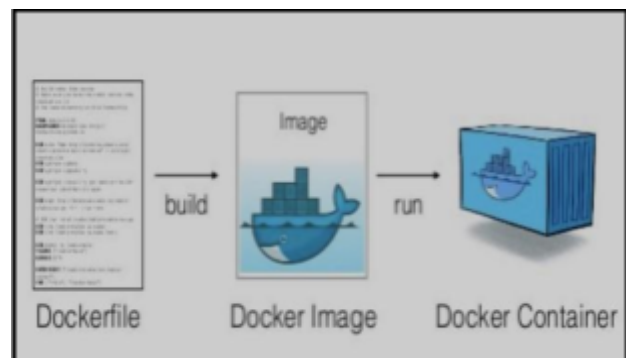
### **Basic terminology**

**Docker Image:** It is a file, composed of multiple layers, used to execute code in a Docker container. Docker image is a blueprint of our application including the dependencies and execution parameters needed to run a software. They are a set of instructions used to create docker containers. Docker Image is an executable package of software that includes everything needed to run an application. This image informs how a container should instantiate, determining which software components will run and how.

**Docker File:** docker file is a blueprint that tells docker how to configure the environment that runs our application i.e. docker image. It contains all commands to execute to build a docker image.

**Container:** a container is a runtime instance of docker image. It is created from Docker images and runs the actual application. A Docker container consists of

- A Docker image
- An execution environment
- A standard set of instructions



**Docker Daemon:** The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system which clients talk to. In a nutshell, the Docker daemon manages local resources including containers and images, while the Docker Registry acts as a repository for storing and distributing.

**Docker Registry:** Docker Registry is a remote service that stores Docker images and other related information. The registry acts as a central hub through which docker images can be accessed by users. Users can also upload their own images to the registry. DockerHub is one of popular public registry, they can also be privately managed.

**Docker Hub:** This is a centralized resource for working with Docker and its components. It provides the following services:

- A registry to host Docker images
- User authentication
- Automated image builds and workflow tools such as build triggers and web hooks
- Integration with GitHub and Bitbucket
- Security vulnerability scanning

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

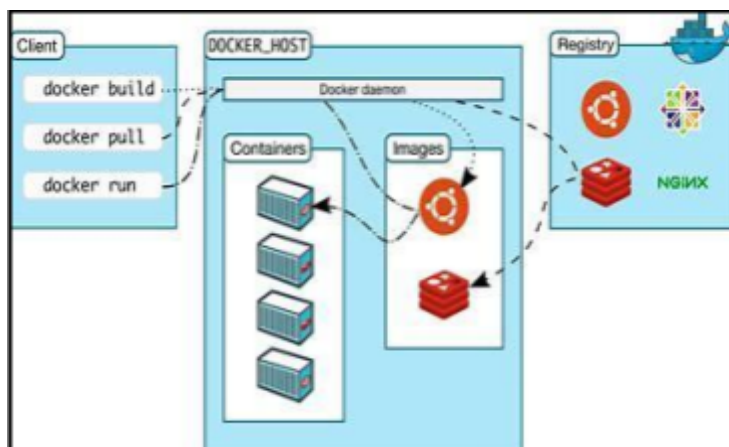
## **Why use docker?**

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data centre, a cloud provider, or a hybrid of the two.

## **Docker architecture:**

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



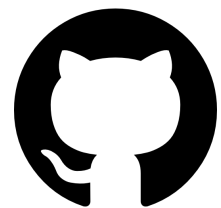
## **Version Control System:**

**Git:** Git is a distributed version control system that allows you to track changes in source code during software development. It supports multiple developers to work together on a project without affecting others, and keep the code under control by helping in sharing and maintaining.

Each developer in Git has a full copy of the entire repository with history on his local machine. Instead of making a compromise and choosing one or the other, these design choices mean that developers can work offline but still enjoy redundancy and reliability when handling project history.

In simple words, Git files are snapshots which belong to a tiny file system. Whenever you commit (save the state of your project), Git takes a picture, or records what all my files look like at that time and stores a reference to this snapshot. If files have not changed, Git doesn't store the file again but a link to the previous identical file.

GitHub is a web-based platform that uses Git for version control and provides additional features to facilitate collaboration and project management



### **Introduction to GitHub**

GitHub is a web-based platform that uses Git for version control, collaboration, and project management. It is a widely used tool for developers to host, manage, and share code. Founded in 2008, GitHub has grown to be the largest host of source code in the world, providing a collaborative environment for open-source projects and private repositories.

### **The Need for GitHub**

1. **Version Control:** Version control systems (VCS) are essential for managing changes to code over time. GitHub uses Git, a distributed version control system, to track changes, allowing multiple developers to collaborate on a project without overwriting each other's work.
2. **Collaboration:** GitHub facilitates collaboration by providing tools for code review, issue tracking, and project management. Developers can work together on the same project, submit pull requests, and merge changes seamlessly.
3. **Backup and Central Repository:** By hosting code in the cloud, GitHub ensures that code is backed up and accessible from anywhere. It serves as a central repository where the latest version of the project can be found.
4. **Open-Source Contributions:** GitHub has a massive community of developers who contribute to open-source projects. It allows anyone to contribute to a project, fostering innovation and the development of new technologies.



## **What GitHub Does?**

1. **Hosting Code:** GitHub hosts code repositories, providing a central location for projects. Repositories can be public (open to everyone) or private (restricted access).
2. **Version Control:** With Git's version control system, GitHub tracks all changes made to code, allowing developers to revert to previous versions if necessary.
3. **Collaboration Tools:** GitHub provides tools such as pull requests, issues, and project boards to facilitate collaboration. Pull requests enable developers to propose changes to a codebase, while issues and project boards help in tracking tasks and bugs.
4. **Code Review:** GitHub's pull request system allows for code review before changes are merged into the main branch. This ensures that code quality is maintained and potential issues are caught early.
5. **Continuous Integration and Deployment:** GitHub integrates with various CI/CD tools, enabling automated testing and deployment of code. This ensures that code changes do not break the build and are deployed smoothly.

## **How GitHub is Operated**

1. **Creating an Account:** To start using GitHub, you need to create an account on the GitHub website. After registration, you can create repositories, contribute to projects, and collaborate with others.
2. **Creating a Repository:** Repositories are the main entities in GitHub where your project's code and files are stored. You can create a new repository through the GitHub interface by clicking on the "New" button under the "Repositories" tab.
3. **Cloning a Repository:** To work on a repository locally, you need to clone it.
4. **Making changes to repository:** Making changes in GitHub consists of two steps: 1. Add 2. Commit. We first add a change then we commit it to make the changes permanent.
5. **Pushing Changes:** To upload your changes to GitHub, use the push command
6. **Pull Requests:** To propose changes to a repository, you can create a pull request. This allows other developers to review your code before it is merged into the main branch.

### Important Commands:

git init: Initialize a new Git repository.

git clone [url]: Clone an existing repository.

git add [file]: Stage changes to a file.

git commit -m "message": Commit staged changes with a message.

git push [remote] [branch]: Push changes to a remote repository.

git pull [remote]: Fetch and merge changes from a remote repository.

git status: Show the status of changes.

git log: Show commit history.

git fetch: Download objects and refs from another repository.

git pull: Fetch from and integrate with another repository or a local branch.

git push: Update remote refs along with associated objects.

git remote -v: Show the URLs of remote repositories.

git branch: List branches.

git branch [branch-name]: Create a new branch.

git checkout [branch-name]: Switch to a branch.

git merge [branch-name]: Merge a branch into the current branch.

## **Python**

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability and ease of use, with its syntax allowing programmers to express concepts in fewer lines of code compared to languages such as C++ or Java.

### **Key features of Python include:**

1. **Ease of Learning and Use:** Python's straightforward syntax makes it an excellent choice for beginners. It allows developers to write clear and logical code for small and large-scale projects.
2. **Interpreted Language:** Python is an interpreted language, which means that code is executed line by line, making debugging and testing easier.

3. **Extensive Standard Library:** Python comes with a comprehensive standard library that supports many common programming tasks, including file I/O, system calls, and internet protocols.
4. **Cross-Platform Compatibility:** Python runs on many platforms, including Windows, macOS, Linux, and Unix.
5. **Dynamic Typing:** Python uses dynamic typing, which means that you don't have to declare the type of variable. The type is decided at runtime.
6. **Support for Multiple Programming Paradigms:** Python supports various programming paradigms, including procedural, object-oriented, and functional programming.
7. **Extensive Support for Third-Party Modules:** Python has a rich ecosystem of third-party libraries and frameworks that extend its capabilities. Notable examples include Django and Flask for web development, NumPy and pandas for data science, and TensorFlow and PyTorch for machine learning.

Python is widely used in various fields, such as:

1. **Web Development:** Using frameworks like Django, Flask, and Pyramid.
2. **Data Science and Analytics:** With libraries such as pandas, NumPy, SciPy, and Matplotlib.
3. **Artificial Intelligence and Machine Learning:** Utilizing frameworks like TensorFlow, Keras, and PyTorch.
4. **Automation and Scripting:** Automating repetitive tasks and processes.
5. **Scientific Computing:** Used in research and development with tools like Jupyter Notebooks.
6. Python's combination of simplicity, readability, and extensive support for various programming paradigms and libraries makes it a popular choice for both beginners and experienced developers in many domains.

## **Package installer in Python**

PIP is a package manager for Python packages, or modules if you like. A package contains all the files you need for a module. Modules are Python code libraries you can include in your project.

To install a package you can simply write `pip install module_name`. Similarly for uninstalling a package write `pip uninstall module_name`

# **MACHINE LEARNING PROJECTS**

## **Title: Creating a generative chatbot using Natural Language Processing**

### **Abstract**

This project involves developing an AI-driven generative chatbot that integrates Natural Language Processing (NLP) with a Retrieval-Augmented Generation (RAG) pipeline. The chatbot utilizes NLP to understand and process user inputs, generating coherent and contextually appropriate responses. The RAG pipeline enhances the chatbot's capabilities by retrieving relevant information from external databases or knowledge sources, ensuring that responses are both accurate and up-to-date. This combination of generative AI and information retrieval allows the chatbot to handle complex queries more effectively, offering improved relevance and precision. The result is a highly capable conversational agent that can be applied across various domains, including customer support, healthcare, and education, providing users with a richer, more informed interaction experience.

### **Problem Statement**

The National Informatics Centre Services Inc. (NICSI) website serves a diverse range of users seeking information on government services, policies, and resources. However, the current system lacks an efficient, real-time, and interactive solution for handling user queries, leading to delays in information retrieval and a poor user experience.

The goal of this project is to develop a generative chatbot utilizing Natural Language Processing (NLP) with a Retrieval-Augmented Generation (RAG) pipeline. This chatbot will provide accurate, dynamic, and context-specific responses by accessing a vast repository of government documents, FAQs, and user data. By doing so, it aims to enhance user engagement, reduce the workload on customer service teams, and deliver real-time information effectively.

## Methodology

- **Data Collection:** the data collection process involves **web scraping** and converting the scraped data into **PDF format** as follows:

1. **Web Scraping:**

- **Objective:** Extract relevant information from NICSI's website or other government portals.
- **Method:** Use of web scraping tool (BeautifulSoup) to automatically gather text, documents, and data from web pages.
- **Process:** Scrape HTML content to extract information such as FAQs, policies, and service details, which are then stored in a structured format.

2. **Converting to PDF:**

- **Objective:** Compile the scraped data into a usable format for the chatbot.
- **Method:** Use tools like Python's **pdfkit** or **ReportLab** to convert the structured data into PDFs.
- **Process:** Format the data into a well-organized PDF document, including clear sections and searchable text, to create a comprehensive resource for the chatbot to reference.

This approach ensures that the chatbot has access to structured and easily retrievable information, enhancing its ability to provide accurate and timely responses.

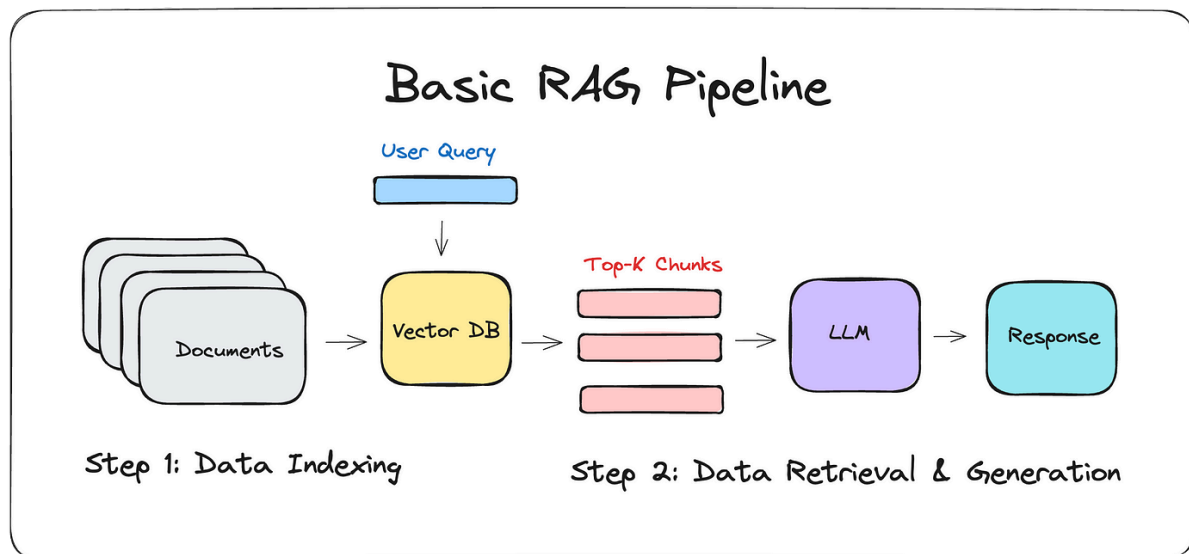
- **RAG Methodology:** The Retrieval-Augmented Generation (RAG) pipeline is a hybrid architecture that combines information retrieval with generative language modeling to improve the performance of chatbots. For the NICSI project, the RAG pipeline functions in two key steps:

1. **Retrieval Step:**

- When a user asks a question, the pipeline first retrieves relevant documents, FAQs, or data from NICSI's internal knowledge base, government documents, or other related sources.
- The retrieval system uses traditional search techniques, such as sparse retrieval (e.g., BM25) or dense retrieval (e.g., embeddings-based methods), to find relevant passages of text from the knowledge base.

## 2. Generation Step:

- After retrieving the most relevant documents, the system passes both the query and the retrieved information to a generative model.
- The model (e.g., an NLP model like GPT or a similar LLM) then generates a response by incorporating both the query and the retrieved documents. This enables it to generate more accurate, contextually grounded, and relevant responses, rather than relying solely on the model's internal knowledge.

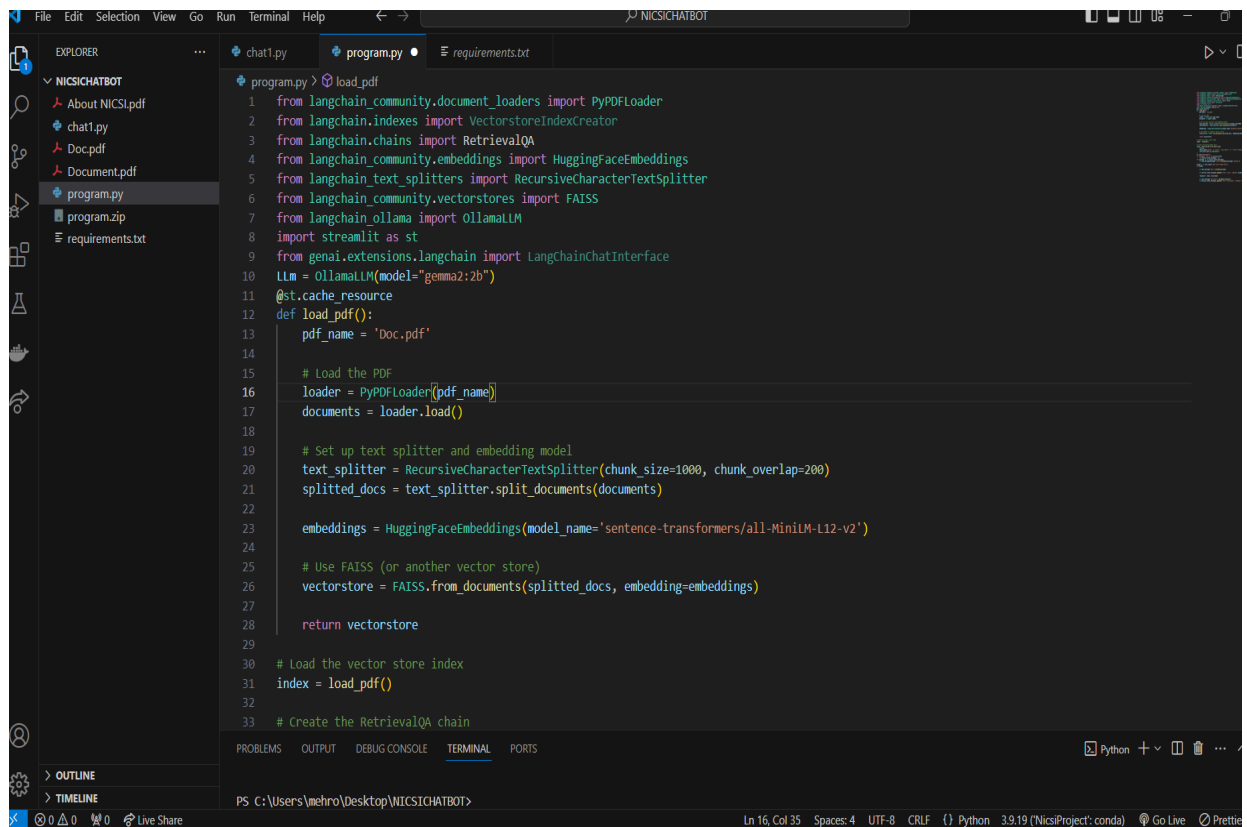


- **Model Selection:** Gemma 2B is an ideal choice for a generative chatbot due to its balance of performance, cost-efficiency, and scalability. With 2 billion parameters, it delivers accurate, context-aware responses, ensuring natural conversations. It offers up-to-date knowledge, handles complex queries, and is more efficient than larger models like GPT-3, making it suitable for enterprise-grade, high-traffic chatbot applications.

### Significance:

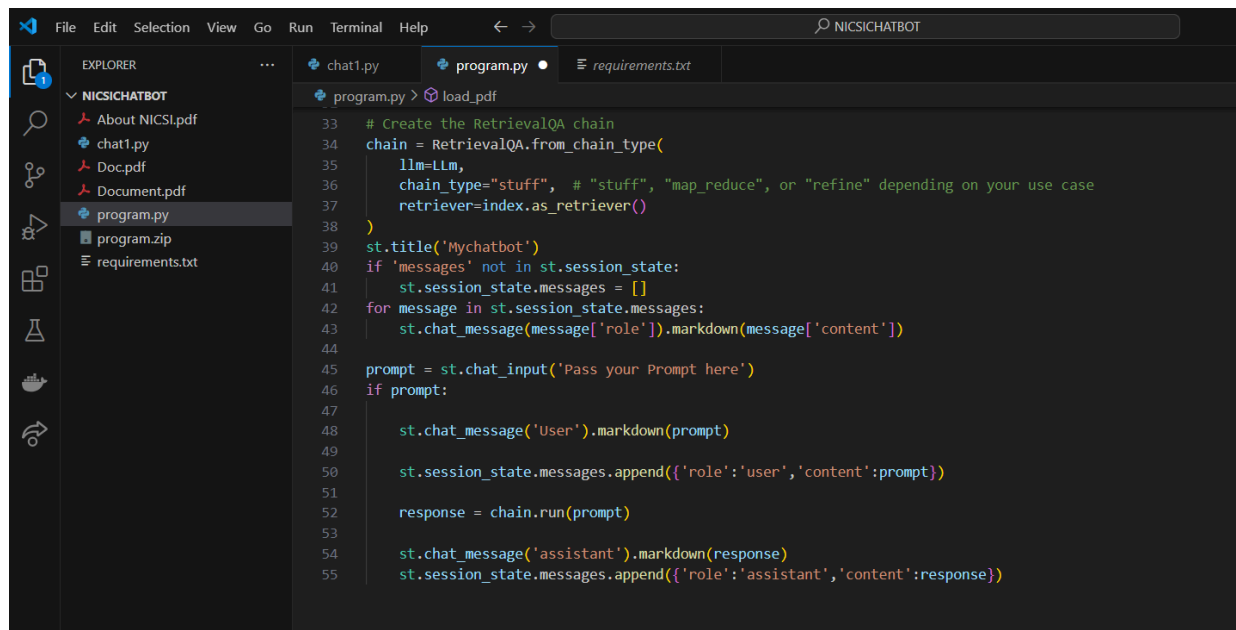
The significance of this project is to revolutionize user interactions by providing accurate, real-time responses to queries. Leveraging NLP and RAG, the chatbot can dynamically pull relevant information from extensive government databases, offering personalized and context-aware answers. This solution enhances user experience by delivering quick, efficient support, reducing response times, and minimizing reliance on manual intervention. It streamlines public access to government services and information, making the system scalable, more efficient, and able to meet the growing demands of users.

## Code:



The screenshot shows a VS Code editor window with the file explorer on the left and the editor on the right. The file explorer shows a project named 'NICSICHATBOT' with files: 'About NICSIL.pdf', 'chat1.py', 'Doc.pdf', 'Document.pdf', 'program.py', 'program.zip', and 'requirements.txt'. The editor is open to 'program.py' and shows the 'load\_pdf' function. The function imports necessary libraries, loads a PDF, splits it into chunks, creates embeddings, and stores them in a FAISS vector store. It also creates a RetrievalQA chain.

```
1 from langchain_community.document_loaders import PyPDFLoader
2 from langchain.indexes import VectorstoreIndexCreator
3 from langchain.chains import RetrievalQA
4 from langchain_community.embeddings import HuggingFaceEmbeddings
5 from langchain_text_splitters import RecursiveCharacterTextSplitter
6 from langchain_community.vectorstores import FAISS
7 from langchain_ollama import OllamaLLM
8 import streamlit as st
9 from genai.extensions.langchain import LangChainChatInterface
10 llm = OllamaLLM(model="gemma2:2b")
11 @st.cache_resource
12 def load_pdf():
13     pdf_name = 'Doc.pdf'
14
15     # Load the PDF
16     loader = PyPDFLoader(pdf_name)
17     documents = loader.load()
18
19     # Set up text splitter and embedding model
20     text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
21     splitted_docs = text_splitter.split_documents(documents)
22
23     embeddings = HuggingFaceEmbeddings(model_name='sentence-transformers/all-MiniLM-L12-v2')
24
25     # Use FAISS (or another vector store)
26     vectorstore = FAISS.from_documents(splitted_docs, embedding=embeddings)
27
28     return vectorstore
29
30 # Load the vector store index
31 index = load_pdf()
32
33 # Create the RetrievalQA chain
```





The screenshot shows the same VS Code editor window, but now showing the chat logic in 'program.py'. The code creates a RetrievalQA chain, sets up a session state for messages, and handles user input and assistant responses using Streamlit's state management.

```
33 # Create the RetrievalQA chain
34 chain = RetrievalQA.from_chain_type(
35     llm=llm,
36     chain_type="stuff", # "stuff", "map_reduce", or "refine" depending on your use case
37     retriever=index.as_retriever()
38 )
39 st.title('Mychatbot')
40 if 'messages' not in st.session_state:
41     st.session_state.messages = []
42 for message in st.session_state.messages:
43     st.chat_message(message['role']).markdown(message['content'])
44
45 prompt = st.chat_input('Pass your Prompt here')
46 if prompt:
47
48     st.chat_message('User').markdown(prompt)
49
50     st.session_state.messages.append({'role': 'user', 'content': prompt})
51
52     response = chain.run(prompt)
53
54     st.chat_message('assistant').markdown(response)
55     st.session_state.messages.append({'role': 'assistant', 'content': response})
```

## Outputs:

# NICSI ChatBot

 nicsi

 Based on the text provided, here is a summary of NICSI's key aspects and services:


**NICSI Overview:**

- **Established in 1995:** As a section-25 company under the Companies Act, 2013.
- **Government Focus:** Focused on IT solutions for government initiatives.
- **Large Scale Operations:** Successfully completed over 25,000 projects and have a turnover of more than Rs. 1477 crores (FY-2021-22).
- **Vision:** To be a technology leader in India and other developing nations by providing cost-effective solutions.

**NICSI Core Services:** A wide range of services that are delivered through purchase from high-quality vendors, listed on NICSI/GeM:


- **Data Analytics Center (CEDA)**
- **Productization & International Promotion**
- **Data Centre Services**
- **Software and Website Development**
- **Manpower Services**
- **IT Consultancy**
- **Rollout Services**


Pass your Prompt here >

 The provided text mentions a number of NICSI national projects, including:

- **IRAD** (Integrated Rural Development Applications)
- **Diksha**
- **NATIONAL KNOWLEDGE network**
- **NIC Cloud Services**
- **e-Procurement**

Let me know if you'd like more information about these specific projects.

 tell me more details about it

 This document provides information on the National Informatics Centre Services Incorporated (NICSI), a company specializing in IT solutions and infrastructure development within India and internationally.

Here's a breakdown of key points:

**Focus Areas:**



- **Government Projects & Public Sector:** NICSI boasts a strong track record serving government agencies, state governments, and public organizations. They consistently deliver over 1000 projects annually.
- **ICT Solutions:** Their services span software development, network infrastructure (including the National Knowledge Network), cloud services, and e-governance initiatives.
- **International Presence:** NICSI has successfully implemented solutions for various countries

Pass your Prompt here >



## Codeflow

### Using Python 3.9.19

1. Environment Setup
  - Install Platforms: Install Ollama from <https://ollama.com>
  - Pull Gemma 2b model:
2. Install requirements.txt file for further dependencies. Link is mentioned below.  
[requirements.txt](#)
3. (Chatbot developed online with the help of api keys)  
 rag2.ipynb
4. app.py (source code for running colab file)  
[app.py](#)
5. dataset  
 NICS (3).pdf

## Code Explanation

### 1. Pulling models:

To get started with Ollama, first visit the [Ollama website](#) and download the appropriate version for your operating system. Follow the installation instructions specific to your OS to complete the setup. Once installed, verify that Ollama is working correctly by running **ollama --version** in your terminal, which should display the version number. Next, to download the Gemma2b model, use the command **ollama pull gemma2:2b**. After the model has been pulled successfully, you can run it with the command **ollama run gemma2:2b**. These steps will ensure that Ollama and the Gemma2b model are installed and operational.

## 2. Install the requirements.txt file

After installation of models successfully. we will install the dependencies listed in the requirements.txt.

## 3. Codeflow

Our code defines a chatbot interface that processes a PDF document to enable question-answering functionality using a pre-trained language model and vector search with embeddings. The description of code is as follows:

**PyPDFLoader**: Used to load PDF documents.

**VectorstoreIndexCreator**: Typically helps in creating an index for the documents.(Used when we are using multiple pdfs)

**RetrievalQA**: A chain that handles the question-answering task by retrieving relevant documents and generating answers.

**HuggingFaceEmbeddings**: Converts text into vector embeddings using the Hugging Face **sentence-transformers** model.

**RecursiveCharacterTextSplitter**: Splits large documents into smaller chunks, improving retriever performance.

**FAISS**: A vector store that allows for fast and efficient similarity searches over embeddings.

**OllamaLLM**: A language model that will be used for generating responses (here, the **gemma2:2b** model).

**Streamlit**: Used for building a web-based user interface.

## 3. Language Model (LLM):

The variable **LLm** is initialized with **OllamaLLM**, which uses a pre-trained model **gemma2:2b** to generate text responses based on input.

## 4. PDF Loading and Processing (@st.cache\_resource decorated function load\_pdf):

**load\_pdf**: Loads and processes a PDF document named Doc.pdf.

- **Step 1**: Use **PyPDFLoader** to read the PDF content into documents.

- **Step 2:** A **RecursiveCharacterTextSplitter** splits the document into smaller chunks of up to 1000 characters, with a 200-character overlap between chunks.
- **Step 3:** Converts the split text chunks into embeddings using a Hugging Face embedding model (**all-MiniLM-L12-v2**).
- **Step 4:** The embeddings are stored in a **FAISS** vector store, allowing efficient retrieval of similar chunks later. The function returns this vector store.

## 5. Vector Store Index:

The **index** variable stores the vector store created by calling the **load\_pdf()** function. This is the search index for retrieving relevant document chunks based on a user query.

## 6. RetrievalQA Chain:

A **RetrievalQA chain** is created using the OllamaLLM model and the FAISS-based vector retriever:

- The chain type is specified as "stuff", meaning that all relevant chunks are retrieved and "stuffed" into the LLM for generating answers.

## 7. Streamlit UI Setup:

**st.title:** Displays the title of the app, "NICS ChatBot".

**Session State (st.session\_state):**

- Initializes an empty list for messages to store the conversation history between the user and the chatbot.
- Each message (both user and assistant) is displayed in the chat interface using **st.chat\_message**.

## 8. Handling User Input:

**st.chat\_input:** Provides an input box for the user to enter a prompt or question.

**Message Logging:**

- If the user enters a prompt, it's logged and displayed in the chat interface as a "User" message.
- The user's prompt is appended to **st.session\_state.messages**.

## 9. Running the Chain:

The **chain.run(prompt)** call is made using the user's prompt to retrieve relevant document chunks and generate a response using the language model.

The assistant's response is displayed in the chat interface and appended to **st.session\_state.messages**.

## Scope

**Healthcare:** Assist with medical inquiries by retrieving information from medical databases and generating advice or information about symptoms, treatments, and medication.

**Education:** Support learning by retrieving educational content and generating explanations or answers to students' questions.

**Finance:** Offer financial advice and information by retrieving data from financial sources and generating personalized financial recommendations or explanations.

## Handled use cases:

### Use case 1:Automated Responses:Answer frequently asked questions and resolve common issues

#### **Objective:**

The primary objective of implementing automated responses is to enhance efficiency and user satisfaction by quickly addressing frequently asked questions (FAQs) and resolving common issues. This can lead to improved customer service, reduced workload on human agents, and a more streamlined support process.

## **Problems that can be Addressed:**

### **1. High Volume of Inquiries:**

- Automated responses can handle a high volume of repetitive inquiries efficiently, reducing the burden on human support agents.

### **2. 24/7 Availability:**

- Automated systems can provide round-the-clock support, ensuring users can get answers to their questions outside of regular business hours.

### **3. Consistency in Responses:**

- Automated systems ensure that responses to FAQs are consistent, reducing discrepancies that can arise from human error.

### **4. Scalability:**

- Automated responses can easily scale to accommodate growing numbers of inquiries without the need for proportional increases in human resources.

## **Potential Challenges:**

### **1. Accuracy of Responses:**

- Ensuring that automated responses are accurate and relevant is crucial. Incorrect or misleading information can frustrate users and potentially lead to larger issues.

### **2. Understanding Context:**

- Automated systems might struggle to understand the context of user queries, leading to inappropriate or off-target responses.

### **3. Handling Complex Queries:**

- Automated responses may not effectively handle complex or nuanced questions that require human judgment or expertise.

### **4. Maintaining Up-to-Date Information:**

- Regular updates to the database or knowledge base are necessary to ensure that the information provided is current and relevant.

## **Future Recommendations:**

### **1. Enhance Natural Language Understanding (NLU):**

- Invest in advanced NLU technologies to improve the system's ability to understand and respond to user queries accurately.

### **2. Implement Continuous Learning:**

- Use machine learning algorithms to continuously learn from user interactions and improve the accuracy and relevance of responses over time.
- 3. Regularly Update Knowledge Base:**
    - Ensure that the knowledge base is regularly updated with the latest information, FAQs, and common issues to keep responses accurate.
  - 4. Provide Escalation Options:**
    - Include options for users to escalate their queries to human agents if the automated system cannot resolve their issues or if they prefer human interaction.

By addressing these challenges and implementing these recommendations, you can create a robust automated response system that efficiently answers FAQs and resolves common issues, leading to a better overall user experience and streamlined support operations.

## **Use case 2: Ticketing & Escalation Handling Basic Inquiries and Escalating Complex Issues**

### **Objective**

To design a chatbot system that can effectively manage basic customer inquiries and efficiently escalate complex issues to human agents. The system aims to streamline support processes, reduce response times for simple queries, and ensure that intricate problems receive the appropriate human attention.

### **Potential Challenges**

#### **1. Accurate Issue Classification:**

- **Challenge:** Ensuring the chatbot correctly identifies which issues are basic and which are complex.
- **Impact:** Misclassification can lead to incorrect handling of queries, either overloading human agents with simple issues or failing to address complex problems adequately.

#### **2. Contextual Understanding:**

- **Challenge:** Maintaining context and understanding the nuances of customer queries to provide relevant responses.

- **Impact:** Poor context handling may result in irrelevant or incorrect responses, frustrating users.

### **3. Seamless Escalation:**

- **Challenge:** Ensuring a smooth transition from the chatbot to human agents, including transferring relevant information and maintaining the conversation flow.
- **Impact:** Inefficient escalation can disrupt the customer experience and delay resolution.

### **4. Training and Adaptability:**

- **Challenge:** Continuously updating the chatbot's knowledge base and training it to handle new types of inquiries and issues.
- **Impact:** An outdated or poorly trained chatbot may struggle with emerging issues or new customer concerns.

### **5. User Expectations:**

- **Challenge:** Managing user expectations regarding response times and the chatbot's capabilities.
- **Impact:** Users may become frustrated if they perceive the chatbot as inadequate or slow in resolving issues.

## **Problems That Can Be Addressed**

### **1. Reduced Response Time:**

- **Solution:** Automate responses to common and straightforward inquiries, which reduces the time customers spend waiting for answers and increases overall efficiency.

### **2. Improved Support Efficiency:**

- **Solution:** By handling routine inquiries, the chatbot allows human agents to focus on more complex and value-added tasks, improving overall support productivity.

### **3. Consistent Responses:**

- **Solution:** Provide standardized answers to frequently asked questions, ensuring consistency and accuracy in the information given to customers.

### **4. Enhanced Customer Experience:**

- **Solution:** Deliver faster, more reliable support for basic inquiries, which can enhance customer satisfaction and loyalty.

#### **5. Effective Issue Tracking:**

- **Solution:** Implement robust tracking and logging of escalated issues to analyze trends, improve support processes, and refine the chatbot's capabilities.

### **Future Recommendations**

#### **1. Enhanced NLP Capabilities:**

- Invest in improving the chatbot's natural language processing capabilities to better understand and respond to a broader range of inquiries and context.

#### **2. Dynamic Escalation Triggers:**

- Develop adaptive escalation triggers based on the complexity of the query and user sentiment to improve the accuracy of issue classification and escalation.

#### **3. Integration with CRM Systems:**

- Integrate the chatbot with Customer Relationship Management (CRM) systems to provide human agents with comprehensive context and history of customer interactions.

#### **4. Continuous Learning:**

- Implement machine learning techniques to continuously train and update the chatbot based on user interactions and feedback to improve its performance over time.

#### **5. Feedback Mechanism:**

- Introduce a feedback mechanism for users to rate their chatbot interactions and escalations, providing valuable insights for ongoing improvements.

#### **6. User Education:**

- Develop resources and guidance to educate users on how to interact effectively with the chatbot and what to expect during escalation, managing their expectations and improving the overall experience.

### **Use case 3:24/7 Availability: Provide round-the-clock support without human intervention**

#### **Objective:**



The primary objective of providing 24/7 availability for chatbots is to ensure that users receive continuous support and assistance at any time of the day or night without requiring human intervention. This enhances user satisfaction, reduces wait times, and ensures consistent service quality.

### **Problems that can be Addressed:**

1. **Reduced Wait Times:** Users get immediate responses, eliminating the delays associated with traditional human support.
2. **Increased Availability:** Support is available around the clock, addressing issues outside of regular business hours.
3. **Consistency:** Ensures uniform service quality without the variability that can occur with human agents.
4. **Cost Efficiency:** Reduces the need for large support teams and associated operational costs.

### **Potential Challenges:**

#### **1. Quality of Responses:**

- **Challenge:** Ensuring that the chatbot delivers high-quality, accurate, and contextually appropriate responses at all times.
- **Solution:** Regularly update and train the chatbot with diverse data to improve its understanding and response capabilities.

#### **2. Handling Complex Queries:**

- **Challenge:** Addressing complex or nuanced queries that may require human judgment or specialized knowledge.
- **Solution:** Implement escalation procedures where the chatbot can flag complex issues for human review or provide options for users to contact human support.

#### **3. Scalability:**

- **Challenge:** Managing increased demand and user interactions during peak hours or unexpected spikes in usage.
- **Solution:** Utilize scalable cloud infrastructure to handle varying loads and ensure performance remains stable.

#### **4. Data Privacy and Security:**

- **Challenge:** Safeguarding user data and ensuring compliance with privacy regulations.
- **Solution:** Implement robust data protection measures, encryption, and adhere to relevant privacy laws and standards.

## **Future Recommendations:**

### **1. Advanced Natural Language Processing (NLP):**

- **Recommendation:** Invest in advanced NLP models to improve the chatbot's ability to understand and generate human-like responses.
- **Benefit:** Enhances the chatbot's effectiveness in handling complex queries and maintaining a natural conversational flow.

### **2. Integration with Human Agents:**

- **Recommendation:** Implement seamless handoff mechanisms to human agents for cases that the chatbot cannot resolve.
- **Benefit:** Provides users with a fallback option for complex issues while ensuring that they receive appropriate assistance.

### **3. User Feedback Mechanism:**

- **Recommendation:** Incorporate feedback collection features to gather user input on the chatbot's performance and areas for improvement.
- **Benefit:** Allows for ongoing refinement based on user experiences and preferences.

### **4. Personalization:**

- **Recommendation:** Use personalization techniques to tailor responses based on user profiles, previous interactions, and preferences.
- **Benefit:** Creates a more engaging and relevant user experience, improving overall satisfaction.

## **Conclusion**

In this project, we developed a generative chatbot for the NICSI website using Natural Language Processing (NLP) and Retrieval-Augmented Generation (RAG) techniques. The chatbot leverages NLP models to understand user queries and generate responses, while RAG combines this with a retrieval mechanism to provide accurate, fact-based information from NICSI's knowledge base. This hybrid approach enhances the relevance of responses, improving the user experience on the website.

By integrating NLP and RAG, the chatbot offers a more efficient and user-friendly way for visitors to access information about NICSI services. This project demonstrates the potential of AI in improving digital interactions and sets the stage for future enhancements such as increased accuracy, knowledge base expansion, and multilingual support.