iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** V **Month of publication:** May 2024

DOI: https://doi.org/10.22214/ijraset.2024.61195

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue V May 2024- Available at www.ijraset.com

Gemini MultiPDF Chatbot: Multiple Document RAG Chatbot using Gemini Large Language Model

Mohd Kaif[1], Sanskar Sharma[2], Asst. Prof. Dr. Sadhana Rana[3]
Computer Science and Engineering SRMCEM Lucknow, India

Abstract: ***The Gemini MultiPDF Chatbot represents a groundbreaking advancement in natural language processing (NLP) by integrating Retrieval-Augmented Generation (RAG) techniques with the Gemini Large Language Model. This innovative chatbot is designed to handle multiple document retrieval and generation tasks, leveraging the extensive knowledge base of the Gemini model. By harnessing RAG methods, the chatbot enhances its ability to acquire, comprehend, and generate responses across diverse knowledge sources contained within multiple PDF documents. The integration of Gemini's powerful language understanding capabilities with RAG facilitates seamless interaction with users, offering comprehensive and contextually relevant responses. This paper presents the design, implementation, and evaluation of the Gemini MultiPDF Chatbot, demonstrating its effectiveness in navigating complex information landscapes and delivering high-quality conversational experiences.***

Keywords: Retrieval Augment Generation, FAISS Index, LangChain, Large Language Models (LLMs).

I. INTRODUCTION

In recent years, we've seen big leaps in how computers understand human language, thanks to cool advancements like Retrieval-Augmented Generation (RAG). These methods blend two powerful techniques: one for finding information and another for putting that info into words. Now, imagine combining RAG with a super-smart language model like Gemini. That's where the Gemini MultiPDF Chatbot comes in. It's like having a conversation with a buddy who's really good at finding and explaining stuff in multiple PDF documents. This introduction sets the stage for exploring how this chatbot works, why it's special, and how it can make dealing with complex info a whole lot easier for all of us.

Gemini models are adept at various NLP tasks such as text summarization, sentiment analysis, and language translation. By leveraging the strengths of dual-encoder architecture, Gemini models have demonstrated superior performance across a wide range of NLP benchmarks, making them a preferred choice for researchers and practitioners seeking state-of-the-art solutions in natural language processing.

II. METHODOLOGY

A. Gemini Model Introduction

Gemini, as described by [1], employs a cutting-edge multimodal architecture. Built upon Transformer decoders, it's meticulously optimized to deliver efficient and dependable performance, especially when scaled. Utilizing Google's potent TPU hardware, Gemini undergoes robust training and execution processes. With an impressive capability to process context lengths of up to 32,000 tokens, its reasoning skills are notably enhanced. Attention mechanisms play a pivotal role in intensifying the intricate analysis performed by the model. By seamlessly integrating text, graphics, and sounds, Gemini harnesses distinct visual symbols and direct voice analysis. Robust reliability features are incorporated to mitigate hardware malfunctions and data distortion during rigorous training sessions. Gemini's ability to comprehend and draw inferences from diverse information is significantly expanded, evidenced by its exceptional benchmark scores and groundbreaking performance in exams. This model sets formidable benchmarks in multimodal AI research and applications.[1]

B. Retrieval Augment Generation

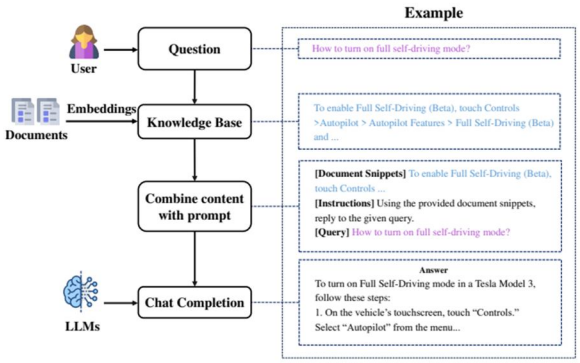


Fig. 1. The workflow of Retrieval-Augmented Generation (RAG).

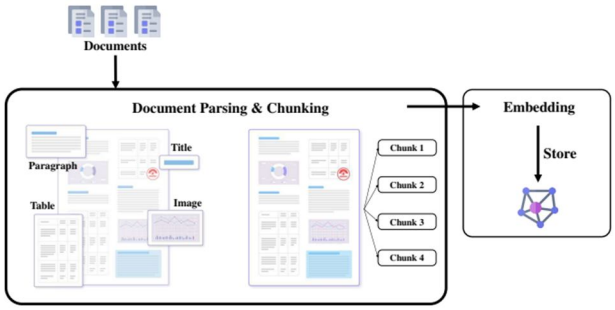


Fig. 2. The process of converting PDFs into retrievable contents.

1) Document Parsing & Chunking. It involves extracting paragraphs, tables, and other content blocks, then dividing the extracted content into chunks for subsequent retrieval.[2]
2) Embedding. It transforms text chunks into real-valued vectors and stores them in a database.[3]

C. Using Langchain to implement Faiss index.

In the realm of artificial intelligence, particularly when dealing with vast amounts of data, efficient retrieval of similar items becomes paramount. This is where Faiss indexing steps in, offering a powerful toolkit for lightning-fast similarity search. Developed by Facebook AI, Faiss stands for Facebook AI Similarity Search.

At its core, Faiss deals with data represented as vectors – numerical arrays that capture the essence of an object. Imagine a collection of images, each encoded as a vector reflecting its color distribution, textures, and shapes. Given a new image (another vector), Faiss helps us find images most similar to it – perhaps those depicting the same object from different angles.

We have used langchain a python library to implement faiss indexing to make vector store for Gemini Model to get the context.

Here is the code snippets for doing the same –

```
# read all pdf files and return text
def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        pdf_reader = PdfReader(pdf)
        for page in pdf_reader.pages:
            text += page.extract_text()
    return text
# split text into chunks
def get_text_chunks(text):
    splitter = RecursiveCharacterTextSplitter(
```

```
        chunk_size=10000, chunk_overlap=1000)
    chunks = splitter.split_text(text)
    return chunks
# creating vector store
def get_vector_store(chunks):
    embeddings = GoogleGenerativeAIEmbeddings(
model="models/embedding-001")
    vector_store = FAISS.from_texts(chunks, embedding=embeddings)
    vector_store.save_local("faiss_index")
```

D. Streamlit for creating user interface

Streamlit is a Python framework designed specifically to help data scientists and machine learning engineers quickly develop and share interactive web apps. Unlike traditional web development, Streamlit requires minimal coding knowledge beyond Python itself. This allows data professionals to focus on the core functionality of their applications, such as data visualization, model deployment, or creating user input interfaces, without getting bogged down in complex front-end development like HTML, CSS, and Javascript. Streamlit streamlines the process by converting Python code into beautiful and functional web apps in minutes, making it a valuable tool for rapidly prototyping and deploying data-driven applications.

Usage in our codebase –

```
import streamlit as st
st.set_page_config(
        page_title="Gemini PDF Chatbot",
        page_icon="□"
    )
    # Sidebar for uploading PDF files
    with st.sidebar:
        st.title("Menu:")
        pdf_docs = st.file_uploader(
            "Upload your PDF Files and Click on the Submit & Process Button", accept_multiple_files=True)
        if st.button("Submit & Process"):
            with st.spinner("Processing..."):
                raw_text = get_pdf_text(pdf_docs)
                text_chunks = get_text_chunks(raw_text)
                get_vector_store(text_chunks)
                st.success("Done")
    # Main content area for displaying chat messages
    st.title("Chat with PDF files using Gemini□")
    st.write("Welcome to the chat!")
    st.sidebar.button('Clear Chat History', on_click=clear_chat_history)
    # Chat input
    # Placeholder for chat messages
    if "messages" not in st.session_state.keys():
        st.session_state.messages = [
            {"role": "assistant", "content": "upload some pdfs and ask me a question"}]
    for message in st.session_state.messages:
        with st.chat_message(message["role"]):
            st.write(message["content"])
    if prompt := st.chat_input():
        st.session_state.messages.append({"role": "user", "content": prompt})
        with st.chat_message("user"):
```

```
        st.write(prompt)
    # Display chat messages and bot response
    if st.session_state.messages[-1]["role"] != "assistant":
        with st.chat_message("assistant"):
            with st.spinner("Thinking..."):
                response = user_input(prompt)
                placeholder = st.empty()
                full_response = ''
                for item in response['output_text']:
                    full_response += item
                    placeholder.markdown(full_response)
                placeholder.markdown(full_response)
        if response is not None:
            message = {"role": "assistant", "content": full_response}
            st.session_state.messages.append(message)
```

III. RESULTS

The results of our research highlight the significant impact of integrating a PDF parser with Gemini in enhancing the accuracy and relevance of responses generated by Large Language Models (LLMs) through Retrieval-Augmented Generation (RAG).

A. Improved Response Accuracy

Our Project demonstrate a notable improvement in the accuracy of responses generated by Gemini when assisted by a proficient PDF parser. By effectively extracting and integrating structured information from documents into prompts, the PDF parser enhances the contextual understanding of the model, leading to more accurate and informative responses also it save cost and effort of the user.

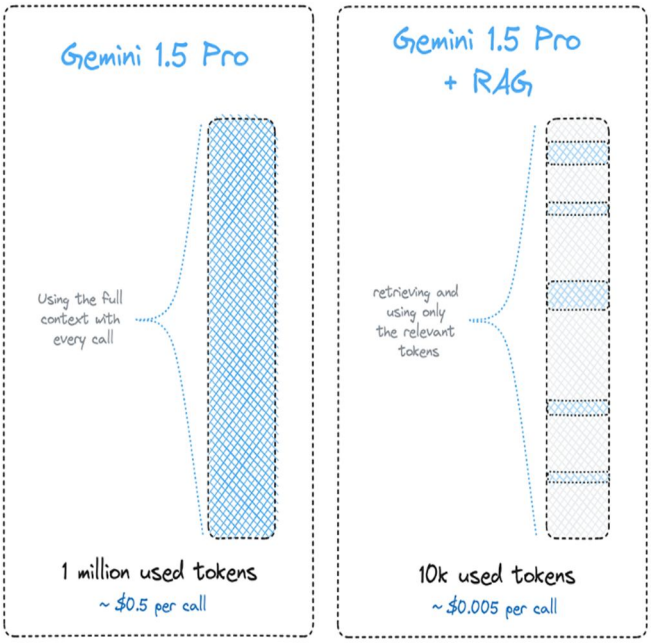


Fig. 3. Cost comparison of Base Gemini model vs Gemini with RAG

B. Enhanced Relevance of Data

The integration of a PDF parser with Gemini enriches the quality and relevance of the data fed into the model. This process ensures that the model is provided with pertinent information from documents, thereby enabling it to produce responses that are contextually appropriate and coherent.[3]

C. Screenshot of the actual project

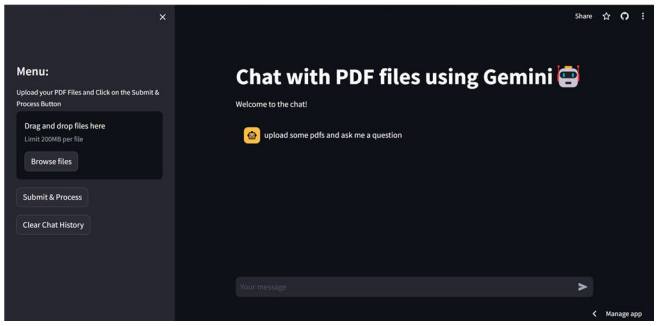


Fig. 4.Homepage of https://gmultichat.streamlit.app/

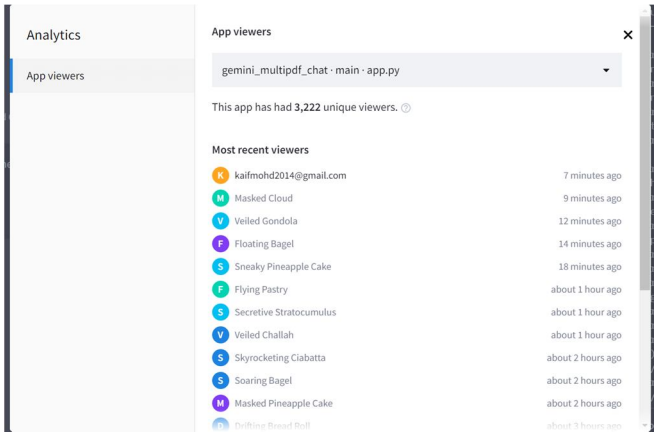


Fig. 5.Total users till date who used the application

IV. DISCUSSION

Although pre-trained language models (LLMs) have great promise, their actual strength rests in their ability to be finely tuned.[4]

A. Domain-specific Fine-tuning

Fine-tuning LLMs to specific domains unlocks their ability to understand and excel in specialized tasks. Here are some prominent techniques:

1) *Curriculum Learning:* Curriculum Learning is a strategy that aims to effectively create domain-specific abilities in a model by progressively raising the complexity of the training data[5].
2) *Knowledge Distillation:* Information distillation involves transferring information from a pre-trained model that is specialised to a certain domain to the LLM. This process may speed up learning and help overcome the problem of limited data availability.
3) *Data Augmentation*: The process of creating artificial data that is relevant to a certain domain enhances the training dataset, resulting in a more resilient and adaptable model.
4) These methods have clearly enhanced the efficiency of completing tasks in several fields, including medical diagnostics and financial forecasting. Nevertheless, a crucial obstacle is determining the ideal equilibrium between safeguarding the LLM's overall knowledge and obtaining specialised skill in a certain field.

B. Dynamic Fine-tuning

LLMs must possess the ability to constantly adjust to new facts and growing demands because to the dynamic nature of information. Dynamic fine-tuning methods tackle this difficulty by:

1) *Online Learning*: Updating the model with new data points in small increments helps it stay up-to-date and improve its knowledge base in real-time.
2) *Meta-learning*: Giving the LLM the capability to learn how to learn from a small number of examples enables it to quickly adapt to new tasks and domains.

3) Federated Learning: The process of training a model on numerous devices with different data distributions, allowing the model to acquire specific local knowledge while ensuring privacy.

Dynamic fine-tuning has great potential for situations that include continuous data streams and quickly changing knowledge needs. For example, it has shown efficacy in the real-time analysis of emotions expressed in social media data and in tailoring language translation to individual preferences[6].

TABLE I. COMPARISION OF VARIOUS TECHNIQUES FOR INCREASING KNOWLEDGE BASE OF LLM

Technique	Description	Advantages	Disadvantages
Pre-training	Training an LLM on a large corpus of unlabelled text data	Enables the LLM to acquire diverse and general knowledge from various domains	Requires a lot of computational resources and time; may introduce biases or errors from the data
Fine-tuning	Adapting an LLM to a specific task or domain by providing labelled data	Improves the LLM's performance and knowledge retention for the target task or domain	May cause catastrophic forgetting of previous knowledge; requires task-specific data and supervision
Retrieval-augmented generation	Enhancing an LLM's generation with external data sources	Allows the LLM to access and utilize relevant information beyond its context size	Depends on the quality and availability of the external data sources; may introduce noise or inconsistency
Self-improvement	Using an LLM to generate and evaluate its own solutions for a task	Enables the LLM to learn from its own reasoning and feedback; reduces the need for human supervision	May be prone to errors or biases; requires careful design of the self-improvement mechanism

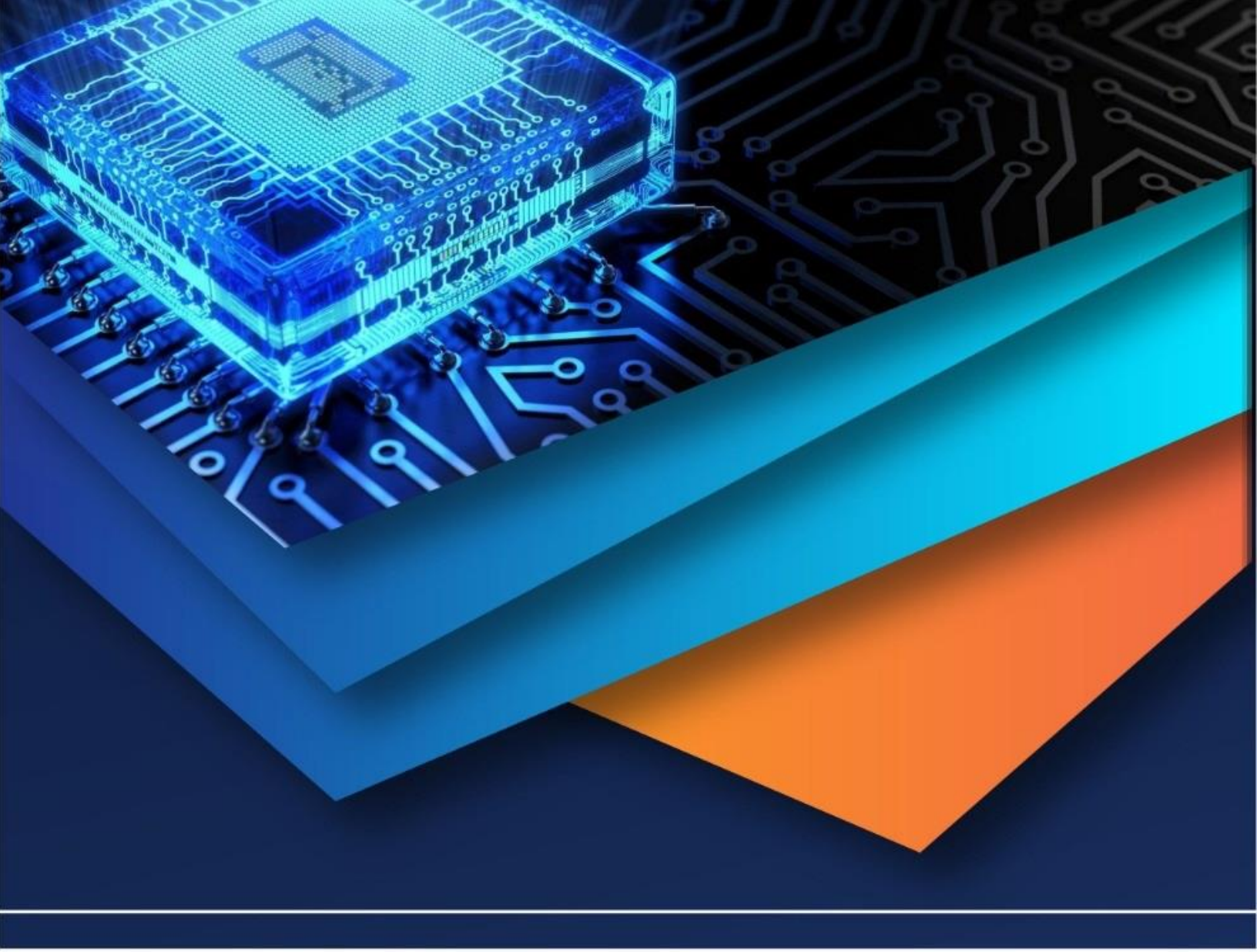
V. CONCLUSION

Imagine giving Gemini, our powerful language model, a helping hand. By using a skilled PDF parser, we can unlock even more precise and relevant responses. This works like a perfect partnership: the parser extracts key information from documents, feeding Gemini with high-quality data. The better the data, the sharper and more accurate Gemini's responses become.

Our next step? We're diving deep into different parsing methods based on deep learning. This lets us explore the fascinating link between the quality of document parsing and how well Gemini performs Retrieval-Augmented Generation (RAG). Early signs suggest some open-source parsing tools might not quite meet the high bar needed for top-notch RAG results within Gemini.

REFERENCES

[1] G. Gemini Team, " Gemini: A Family of Highly Capable Multimodal Models, 2024.".
[2] S. Siriwardhana, R. Weerasekera, E. Wen, T. Kaluarachchi, R. † Rajib, and S. Nanayakkara, "Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering", doi: 10.1162/tacl.
[3] W. Yu, "Retrieval-augmented Generation across Heterogeneous Knowledge."
[4] K. Rangan and Y. Yin, "A Fine-tuning Enhanced RAG System with Quantized Influence Measure as AI Judge," Feb. 2024, [Online] Available: http://arxiv.org/abs/2402.17081
[5] J. Liuska, "Bachelor's Thesis- ENHANCING LARGE LANGUAGE MODELS FOR DATA ANALYTICS THROUGH DOMAIN SPECIFIC CONTEXT CREATION," 2024
[6] [Y. Chang et al., "A Survey on Evaluation of Large Language Models," Jul. 2023, [Online]. Available: http://arxiv.org/abs/2307.03109



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY