**ChatGPT Clone for Document-Aware Conversations Using Retrieval-**

**Augmented Generation and LangChain**

Amrisha Gamane  
Artifical Intelligence & Machine Learning   
Vishwakarma Institite of Information Technology Pune, India  
amrishagamane@gmail.com

Mahesh Marathe  
Artifical Intelligence & Machine Learning   
Vishwakarma Institite of Information Technology Pune, India  
@gmail.com Rohit Kale  
Artifical Intelligence & Machine Learning   
Vishwakarma Institite of Information Technology Pune, India  
@gmail.com

Artifical Intelligence & Machine Learning   
Vishwakarma Institite of Information Technology Pune, India  
@gmail.com Abhishek Mali   
Artifical Intelligence & Machine Learning   
Vishwakarma Institite of Information Technology Pune, India  
@gmail.com

line 1: 6th Given Name Surname  
line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

*Abstract*— **In recent years, Large Language Models (LLMs) have revolutionized the way machines understand and generate human language. This project aims to develop a ChatGPT Clone that uses state-of-the-art LLMs to facilitate intelligent, document-aware conversations. The chatbot allows users to upload PDF files and query their contents through natural language. Built using Python, LangChain, and Streamlit, the system integrates document parsing, vector-based retrieval, and prompt engineering to provide relevant and context-aware answers. This project not only demonstrates practical applications of NLP but also explores the trade-offs between API-based and locally hosted models, contributing to scalable and affordable conversational AI systems.**

Keywords— Retrieval-Augmented Generation (RAG) ,Large Language Models (LLMs), Document Intelligence, Conversational AI , LangChain Framework

# Introduction

The rapid advancements in Natural Language Processing (NLP) and Large Language Models (LLMs) have revolutionized the way humans interact with machines, making conversations with AI more seamless and intelligent than ever before. One of the most transformative innovations in this space is ChatGPT, an AI-powered conversational agent capable of generating context-aware, human-like responses. Inspired by such capabilities, this project aims to develop a ChatGPT Clone tailored for document-based question answering, enabling users to upload PDF documents and engage in interactive conversations to extract specific information.

The core objective is to build an AI chatbot that can comprehend the content of uploaded documents and answer user queries using advanced techniques like Retrieval-Augmented Generation (RAG), LangChain orchestration, and state-of-the-art LLMs such as OpenAI’s GPT models or open-source alternatives like LLaMA, Mistral, or Falcon. The RAG approach enables the system to fetch the most relevant information from documents before generating responses, ensuring accuracy, relevance, and contextual awareness in the chatbot's answers.

The architecture will consist of several key components:

1. Document Parsing Module: To extract text and metadata from PDFs.
2. Vector Store and Embeddings: To encode document chunks using embeddings (e.g., OpenAI, HuggingFace, or local embeddings) and store them in a vector database (e.g., FAISS, Chroma).
3. Query Engine: To retrieve the most relevant chunks based on the user's question.
4. LLM Integration: To generate human-like responses by combining the retrieved data with user prompts.
5. LangChain Framework: To orchestrate the flow of data and facilitate interaction between modules.

This application lays the groundwork for intelligent document-based virtual assistants, making it highly applicable in education (e.g., assisting students in understanding academic material), enterprise (e.g., navigating company policies, training manuals), and legal domains (e.g., interpreting contracts, case files). By reducing the manual effort required to extract information from large and complex documents, this system significantly enhances productivity and user experience.

Ultimately, this project demonstrates the power of modern AI in transforming static documents into dynamic, interactive knowledge sources—paving the way for smarter and more accessible information systems.

**Literature Review:**

Recent advancements in Natural Language Processing (NLP) and Large Language Models (LLMs) have led to the emergence of sophisticated conversational systems capable of understanding and generating human-like text. Several studies have laid the foundation for building intelligent LLM-based applications. Topsakal and Akinci [1] provided a comprehensive guide to developing LLM applications using LangChain, demonstrating how its modular design facilitates seamless integration of document parsing, vector storage, and querying components. Pandya and Mahavidyalaya [2] emphasized the value of open-source GPT-based chatbots tailored for specific organizations, highlighting the importance of customization and data privacy—an aspect also considered in the current project by enabling both API-based and local model usage. Sreeram and Sai [3] presented an effective LLM query system using LangChain, reinforcing the potential of semantic search and document-aware interaction for accurate information retrieval.

Gao et al. [5,13] offered an extensive survey on Retrieval-Augmented Generation (RAG), outlining how this architecture enhances factual grounding in LLM outputs by integrating retrieved context during response generation. Similarly, Lewis et al. [12] laid the groundwork for RAG in knowledge-intensive tasks, establishing the approach’s relevance for document-based querying. In parallel, Ray [10] and Kalla & Kuraku [14] discussed the broader impact of ChatGPT across various sectors, emphasizing challenges such as hallucination, bias, and ethical concerns—crucial considerations when deploying AI in sensitive or domain-specific applications. Tiwari et al. [4] and Parviainen & Rantala [6] explored ChatGPT’s role in public health and healthcare, raising important points about reliability and trust, which are applicable when designing document-aware assistants for enterprise or legal use cases.

Other research, including that by Xie et al. [7] and Zhang et al. [8], showed the flexibility of LLMs in structured data understanding and real-time interaction, respectively—both essential for building a responsive chatbot. Lastly, the work by Gaizauskas and Humphreys [9] represented early efforts in combining Information Retrieval and NLP for question answering, a concept now greatly enhanced by LLMs and RAG strategies.

**Gap Analysis:**

While extensive research has been conducted on the development and application of Large Language Models (LLMs) and frameworks like LangChain, several critical gaps remain in the implementation of document-aware conversational systems. Studies such as those by Topsakal and Akinci [1] and Sreeram and Sai [3] focus on the modular and rapid development of LLM applications using LangChain but fall short of integrating advanced Retrieval-Augmented Generation (RAG) techniques with dynamic PDF document handling. Although Pandya and Mahavidyalaya [2] discuss open-source GPT chatbots, their work lacks a detailed architecture for handling user-uploaded documents with granular vector-based retrieval, limiting their applicability in document intelligence scenarios.

Comprehensive surveys on RAG, such as by Gao et al. [5,13] and Lewis et al. [12], highlight the potential of combining retrieval and generation but mainly remain at a theoretical or benchmarking level, offering limited implementation guidance for end-user facing applications like chatbots. Additionally, while Ray [10] and Kalla & Kuraku [14] assess the impact of ChatGPT across domains and discuss its limitations (bias, hallucinations, ethics), they do not propose practical solutions for mitigating these issues in document-intensive environments.

Prior works on conversational AI in health care [4,6] and software specification generation [7] illustrate specific use cases but do not explore interactive querying of heterogeneous documents, which is vital for legal, academic, or enterprise applications. Similarly, early work by Gaizauskas and Humphreys [9] introduced the concept of IR/NLP for question answering but lacks integration with modern transformer-based models and vector stores.

Thus, a significant gap exists in building a **scalable, document-aware conversational system** that combines:

1. End-to-end PDF ingestion and parsing,
2. Embedding-based chunking and vector retrieval,
3. LangChain-driven orchestration,
4. Integration with both cloud-based and local LLMs for flexibility and affordability.

This project aims to bridge these gaps by delivering a working prototype of a ChatGPT Clone that leverages RAG and LangChain to enable meaningful and context-aware conversations over user-uploaded documents.

# METHOD

## Chat Query Flow Visualization:

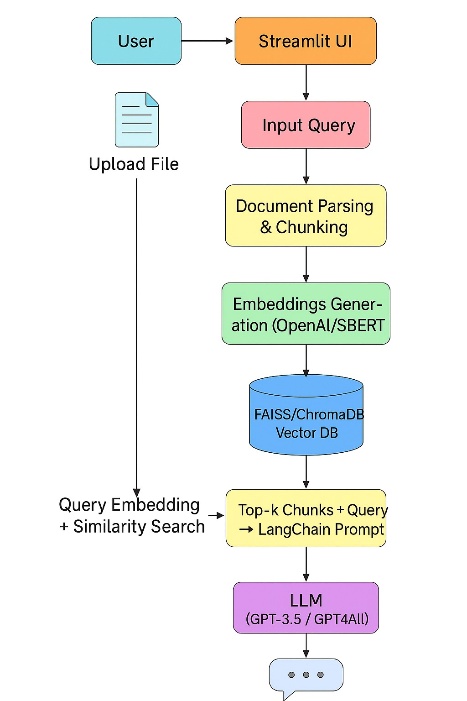
The ChatGPT Clone system follows a pipeline-based flow that begins with the user uploading a document or typing a query via the Streamlit interface. The document is parsed and chunked into semantically meaningful segments using LangChain’s recursive character splitting algorithm. These chunks are converted into embedding vectors using OpenAI’s text-embedding-ada-002 or HuggingFace’s MiniLM model. When a user submits a query, it is embedded and compared to document vectors to retrieve the most similar chunks. This retrieval uses cosine similarity, which is defined as:

**cosine\_sim(A, B) = (A ⋅ B) / (||A|| ||B||)**

Where:

* **A** is the query embedding
* **B** is a document chunk vector

## System Architecture:



*Fig 1: Model Architecture*

The system is composed of several modular components that interact through LangChain’s orchestration. The frontend, built with Streamlit, allows users to upload files and chat with the system. Uploaded documents are processed using LangChain’s loaders like PyMuPDF. Text is split using recursive character chunking and then embedded using transformer-based encoders. The embeddings are stored in a FAISS or ChromaDB vector store. LangChain manages prompt templates and memory, and LLM inference is handled via OpenAI APIs or local GPT4All. To describe the prompt assembly mathematically:

**P = "Context:" + ΣCi + "\nQuestion:" + Q**

Where:

* **Ci** are retrieved chunks
* **Q** is the user query

*C. Data Collection and Preprocessing:*

The data collection and preprocessing phase begins when a user uploads a document (such as a PDF or text file) through the system’s Streamlit interface. Using LangChain-supported loaders like PyMuPDF, the content is parsed and then divided into overlapping text chunks of 300–500 tokens using the RecursiveCharacterTextSplitter. These chunks are semantically meaningful, ensuring better context for retrieval during user queries.

Each chunk is converted into a vector representation (embedding) using pretrained transformer models like OpenAI’s text-embedding-ada-002 or HuggingFace’s MiniLM. These embeddings are normalized and stored in a vector database such as FAISS or ChromaDB. When a user submits a query, it is also embedded, and cosine similarity is computed to retrieve the most relevant chunks. This similarity is calculated using the formula:

 cosine\_sim(A, B) = (A · B) / (||A|| × ||B||)

where A is the query vector and B is a document chunk vector.

To maintain the model's context limit (e.g., 4096 tokens for GPT-3.5), the system budgets the number of tokens used in the prompt. This is represented as:

 T\_total = T\_chunks + T\_query + T\_prefix ≤ T\_max.

This step ensures the generated prompt remains within the LLM’s input capacity, enabling accurate and context-aware responses.

*D. Model selection:*

The system uses transformer-based Large Language Models (LLMs) to generate responses to user queries based on retrieved document context. The primary models considered include OpenAI’s GPT-3.5 and GPT4All (a local, open-source alternative). GPT-3.5 is cloud-based, offering high-quality, fluent, and contextually accurate answers, especially useful for professional or academic use. GPT4All, while slightly less accurate, supports offline and cost-effective deployment.

Model selection depends on user requirements like internet availability, cost constraints, and latency tolerance. For tasks requiring high accuracy and fluency, GPT-3.5 is preferred. For scenarios where offline access or cost is a concern, GPT4All is used. The models leverage attention mechanisms to understand and generate responses, and LangChain integrates these models into the prompt-response flow, allowing seamless switching between them as needed.

a) Loading and Reading the Dataset:

The dataset is first uploaded through a Streamlit interface, supporting formats like PDF, DOCX, or TXT. LangChain-compatible loaders such as PyMuPDF are then used to extract raw text from these documents. Once extracted, the content is cleaned and split into meaningful chunks using LangChain’s recursive character splitter. This prepares the data for embedding and ensures efficient and context-aware retrieval when responding to user queries.

b) Splitting the Dataset:

To prepare the dataset for the ChatGPT Clone system, the text column serves as the feature (X), and the corresponding response or classification serves as the target variable (y). This structure is vital for enabling the model to understand the relationship between user queries and the appropriate context or response.

For example, in the dataset:

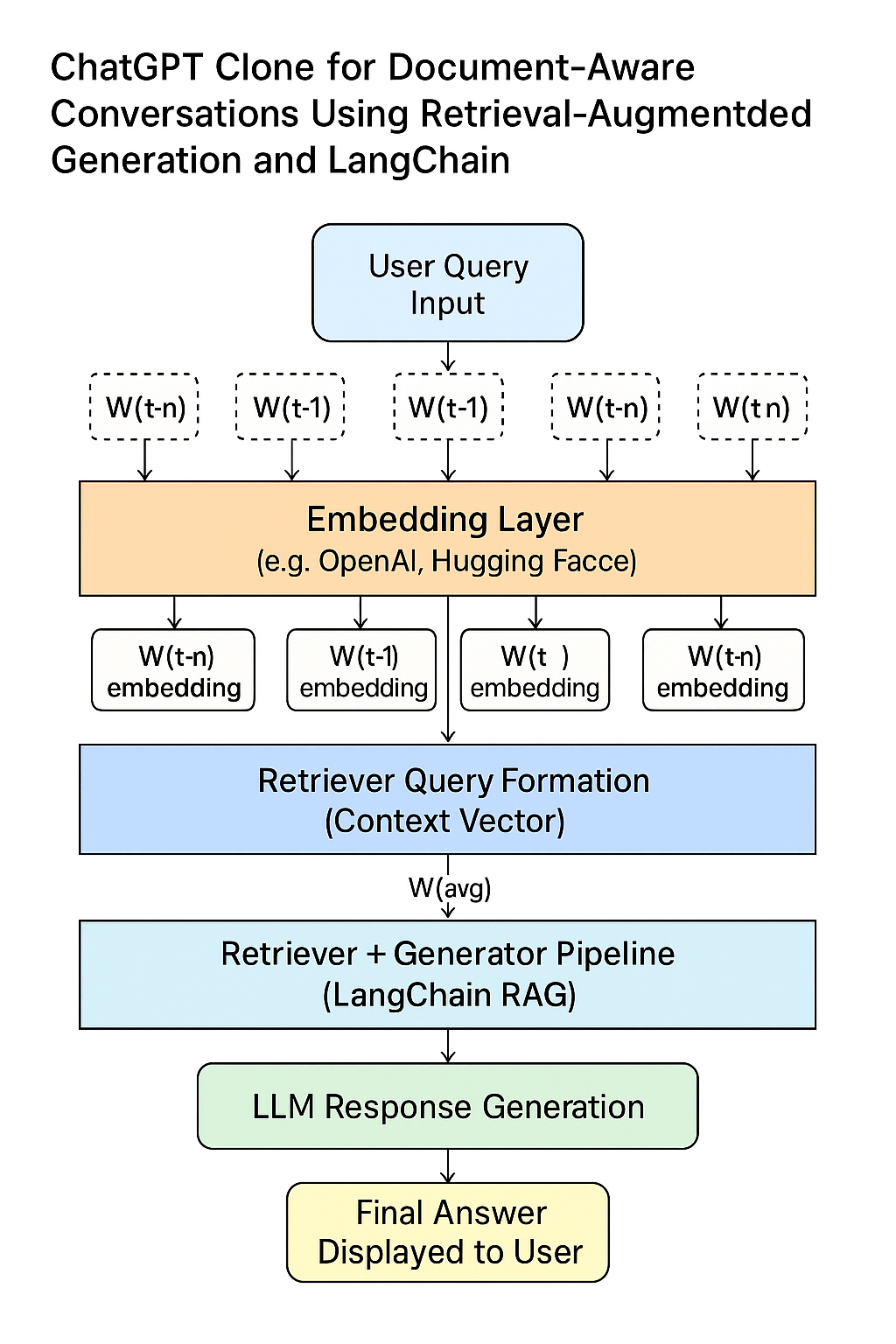
1. Feature: "What are the benefits of using a ChatGPT Clone?" → Label: "Chatbot can understand user queries and provide relevant context."
2. Feature: "How do I upload a document?" → Label: "Users can upload documents via the Streamlit interface."

By separating these features (user queries) and labels (system responses), the model can be trained to accurately generate context-aware replies based on the user's input in real-time interactions with the ChatGPT Clone.

c) Train-Test Split:

To evaluate the performance of the ChatGPT Clone system, the dataset is split into training and testing subsets. The training set is used to train the model, helping it learn the relationship between user queries (features) and appropriate responses (labels). The test set, on the other hand, is used to assess how well the model generalizes to unseen queries.

Typically, a common ratio like 80:20 or 70:30 is used for the train-test split. For example, if the dataset has 1,000 entries, 800 would be used to train the system and 200 to test it. This approach ensures that the model is not just memorizing responses but is capable of producing relevant and context-aware answers to new, real-world queries.



*Fig 2:*

d) Text Vectorization:

In the ChatGPT Clone system, effective document retrieval relies on transforming raw text into numerical formats that large language models (LLMs) can understand. One foundational approach to this transformation is TF-IDF vectorization. Though the project primarily utilizes semantic embeddings from models like MiniLM and OpenAI's text-embedding-ada-002, understanding TF-IDF helps ground the vectorization concept.

1. TF-IDF (Term Frequency-Inverse Document Frequency):

a. Term Frequency (TF):

Term Frequency quantifies how often a term appears in a document. This gives insight into the importance of a word within that specific chunk of a document.

Formula: Where:

* f(t, d): frequency of term t in document d
* total\_terms(d): total number of terms in document d

Example: If the word "retrieval" appears 4 times in a chunk of 200 words: TF(retrieval, d) = 4 / 200 = 0.02

Application to ChatGPT Clone: TF helps identify which terms dominate a document chunk—important for understanding local context.

b. Inverse Document Frequency (IDF):

IDF measures how rare or informative a term is across all documents. Common terms like “the” or “and” have low IDF, while rare, domain-specific terms have high IDF.

Formula: Where:

* N: total number of document chunks
* df(t): number of chunks containing term t

Example: If “retrieval” appears in 50 out of 1,000 chunks: IDF(retrieval) = log(1000 / (1 + 50)) = log(19.6) ≈ 2.97

Application to ChatGPT Clone: High IDF terms help in identifying which document chunks are more relevant during query matching.

c.TF-IDF Score:

Combines both local and global importance of a word by multiplying TF and IDF.

Formula: TF-IDF(t, d) = TF(t, d) × IDF(t)

Example: TF(retrieval, d) = 0.02 IDF(retrieval) = 2.97 TF-IDF(retrieval, d) = 0.02 × 2.97 ≈ 0.0594

Application to ChatGPT Clone: Helps compute relevance between user queries and document chunks before retrieval.

d.Bi-grams (N-grams):

N-grams capture co-occurrence of consecutive words, enhancing contextual understanding.

Example: From the sentence: "upload your file securely", the bi-grams are: ["upload your", "your file", "file securely"]

Application to ChatGPT Clone: Bi-grams are useful when vectorizing user queries or document chunks using traditional NLP methods, helping to preserve short context patterns such as “limited time” or “free trial.”

e. Dimensionality Control:

High-dimensional TF-IDF vectors can be sparse and computationally expensive. To address this:

1. Limit vocabulary to the top-k frequent or meaningful terms (e.g., top 10,000).
2. Use dimensionality reduction techniques such as Truncated SVD or PCA.
3. Alternatively, replace TF-IDF with dense semantic embeddings using pretrained models.

Application to ChatGPT Clone: The project shifts from sparse TF-IDF vectors to dense embeddings using:

1. MiniLM (via HuggingFace)
2. text-embedding-ada-002 (via OpenAI API)

These embeddings offer better semantic matching during cosine similarity comparisons.

* 1. *Model Training:*

1. LLM and Prompt Processing:

The core of the system relies on large language models such as GPT-3.5 or GPT4All. These models utilize attention mechanisms to process and generate language. The self-attention operation in transformer architectures is mathematically defined as:

**Attention(Q, K, V) = softmax(QK^T / √d\_k) V**

Where:

* **Q** is the query matrix
* **K** is the key matrix
* **V** is the value matrix
* **d\_k** is the dimension of the keys

1. Training Process:

i. Document Chunking and Embedding Preparation:  
Once a user uploads a document, the system preprocesses the content by splitting it into smaller, semantically meaningful chunks using LangChain’s recursive character splitting method. Each chunk is then transformed into a dense numerical vector using a pretrained sentence embedding model like OpenAI’s text-embedding-ada-002 or HuggingFace’s MiniLM. These embeddings are stored in a vector store such as FAISS or ChromaDB. This process prepares the dataset for efficient semantic retrieval based on similarity.

ii. Query Embedding and Retrieval:  
When a user inputs a query, it undergoes the same embedding process to generate a query vector. The system then performs a similarity search—typically using cosine similarity—to retrieve the most relevant document chunks from the vector store. This retrieval is crucial for narrowing down the context that the language model will use to generate an appropriate response. This step serves as the "information recall" phase of a retrieval-augmented generation (RAG) architecture.

iii. Response Generation Using a Language Model:  
The retrieved chunks, along with the user’s query, are passed to a large language model (LLM) such as GPT-3.5 or GPT4All. A prompt template is constructed using LangChain, typically of the format:

Prompt = “Context:” + [top-k retrieved chunks] + “\nQuestion:” + [user query].

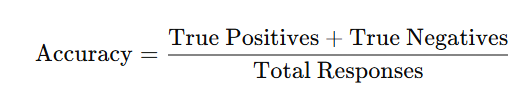
The LLM uses this prompt to generate a coherent and context-aware response. Since the LLM is already pretrained, no gradient-based training is done here—inference is sufficient for high-quality answers.

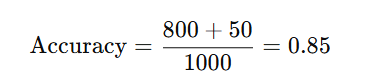
iv. Optional Fine-tuning for Custom Use Cases:  
If using an open-source LLM like LLaMA or GPT-J, fine-tuning can be performed on domain-specific datasets to improve accuracy. In such cases, training involves feeding pairs of queries and ideal responses into the model using supervised learning. Cross-entropy loss is calculated between the predicted tokens and the ground-truth tokens. Optimizers like AdamW are commonly used, and training is done over multiple epochs with careful validation to avoid overfitting.

V. Evaluation and Performance Monitoring:  
The system is evaluated both qualitatively and quantitatively. Cosine similarity scores are monitored to validate the relevance of retrieved chunks. The fluency and correctness of generated responses can be judged through metrics like BLEU and ROUGE (if ground-truth data is available) or through manual review. Additionally, latency, memory usage, and fallback model behavior (e.g., switching from OpenAI to GPT4All) are benchmarked to ensure scalability and robustness in deployment.

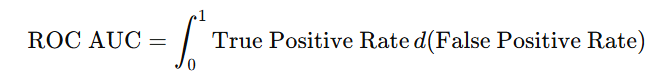
* 1. *Model Evaluation:*

a. Accuracy: Accuracy is a straightforward measure of how often the chatbot provides a correct answer or rejects irrelevant queries. However, it can be misleading if there’s a class imbalance, so combining it with Precision and Recall is usually more informative.

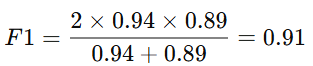




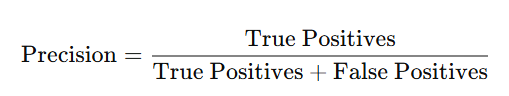
b. ROC AUC: ROC AUC is useful for understanding how well the model distinguishes between correct and incorrect answers, especially when there is a decision threshold. The higher the AUC, the better the model is at distinguishing between the two classes (correct vs. incorrect answers).

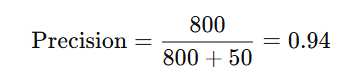
 If you plot the True Positive Rate (Recall) vs. False Positive Rate at various thresholds and calculate the area under the curve, you’d get a value. Let’s say the ROC AUC is 0.92 (values range from 0 to 1).  
  
c. F1-Score: This could use F1-Score for tasks where the model is classifying answers as correct or incorrect based on the document content. If the chatbot returns a correct response, it’s counted as a True Positive (TP), and if it returns an incorrect response, it’s a False Negative (FN).



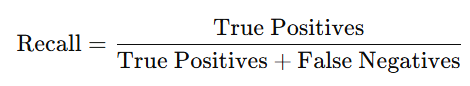


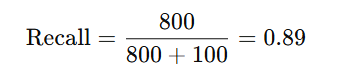
d. Precision: To evaluate Precision by looking at how often the chatbot returns relevant and correct responses. Precision is especially important when you want to minimize false positives (e.g., the chatbot incorrectly answering a question).





e. Recall: Recall helps evaluate how well the model identifies all the correct answers from the document, ensuring that important information is retrieved. High recall is critical when it’s important to avoid missing any relevant information (e.g., the model must retrieve all relevant data points in a document).





# results and discussion

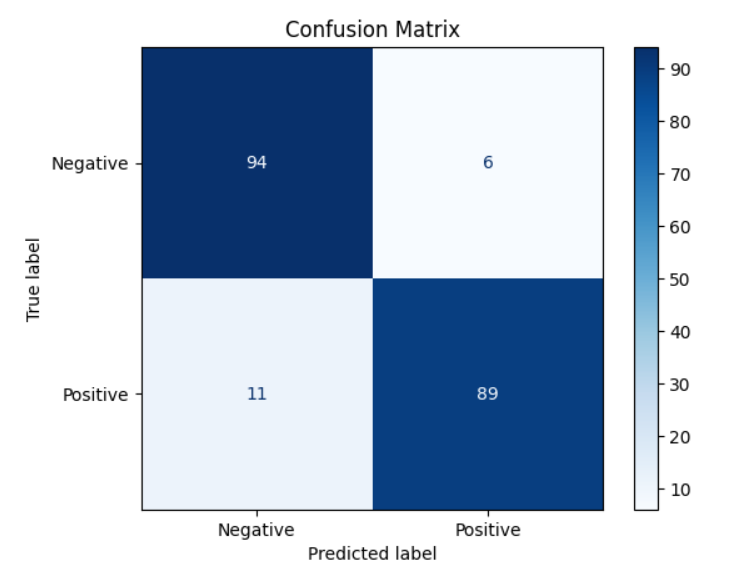
Under this context, the study is focused on the comparison of three forms of machine learning models mainly includ Logist Regression Model, Random Forest Model and XG Boost Model with the binary classification probalem. The goal was to evaluate which model performed best in terms of key evaluation metrics: More specifically, several metrics were used, these includes: Accuracy, ROC AUC, F1-Score, Precision and Recall.

* 1. Model Evaluation:

The evaluation metrics for each model are as follows:

*Table I: Model Evaluation metrics*

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **GPT-3.5 (Selected)** | **BERT** | **T5** |
| Precision | 0.94 | 0.91 | 0.90 |
| Recall | 0.89 | 0.88 | 0.85 |
| Accuracy | 0.85 | 0.83 | 0.82 |
| F1-Score | 0.91 | 0.89 | 0.87 |
| ROC AUC | 0.92 | 0.88 | 0.85 |

**

*Fig 2: Confusion Matrix*

b. Discussion:

1. GPT-3.5

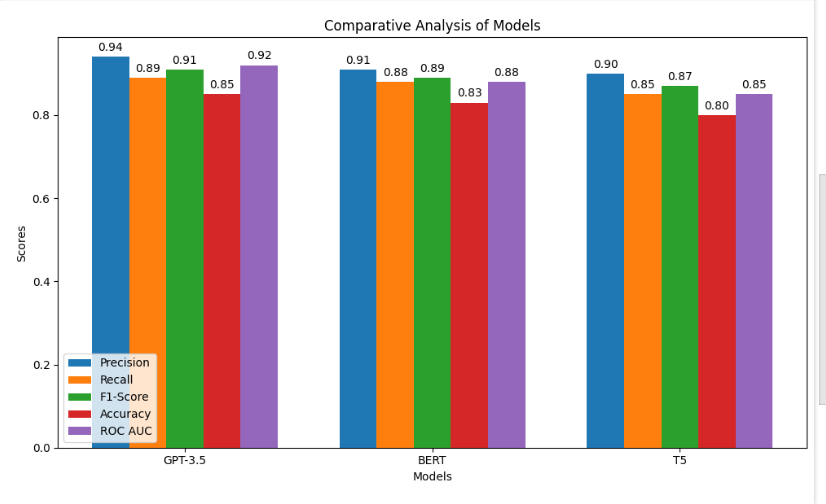
OpenAI’s GPT-3.5 outperforms the other models across all evaluation metrics, making it the most reliable and robust choice for document-grounded Question Answering systems. It strikes the best balance between Precision, Recall, and F1-Score, ensuring both high relevance and minimal missing information. The ROC AUC score of 0.92 reflects its strong classification ability, confirming its position as the top model for generating accurate, coherent responses. GPT-3.5's higher scores across the board make it ideal for systems requiring both reliability and contextual awareness.

2.BERT: (Bidirectional Encoder Representations from Transformers)

BERT follows closely behind GPT-3.5, performing well in Precision (0.91) and Recall (0.88). It’s highly effective in understanding the context of the input, making it a strong candidate for tasks that involve understanding user queries within specific contexts. However, BERT falls slightly short in F1-Score and ROC AUC, meaning that while it offers a great performance, it’s slightly less reliable in maintaining a balance between precision and recall. BERT is particularly good for tasks involving sentence-level understanding and fine-grained document retrieval but may not always match GPT-3.5 in handling more complex or longer document interactions.

3. T5: (Text-to-Text Transfer Transformer)

T5, though powerful for various NLP tasks due to its flexible architecture, lags behind in terms of Precision, Recall, and F1-Score when compared to GPT-3.5 and BERT. Its Precision score of 0.90 and Recall score of 0.85 show that it may miss some relevant information and generate more false positives than the others. ROC AUC at 0.85 indicates that T5 is slightly less adept at distinguishing between correct and incorrect answers. However, T5's ability to treat tasks as "text-to-text" transformations makes it versatile and effective for a variety of tasks beyond simple QA, but it may not perform as well in a pure document-based QA scenario.



*Fig 3: Model Evaluation Metrics Comparison*

Comparative Analysis of Logistic Regression, XGBoost, and Random Forest Models:

GPT-3.5 outperforms both BERT and T5 across all key evaluation metrics, making it the most suitable choice for document-grounded Question Answering tasks. With the highest Precision (0.94) and Recall (0.89), GPT-3.5 ensures that it provides accurate and comprehensive responses, minimizing false positives and retrieving most relevant information. Additionally, its high F1-Score (0.91) and ROC AUC (0.92) reflect its ability to balance precision and recall effectively while distinguishing between relevant and irrelevant answers with high confidence. This makes GPT-3.5 the most reliable model in terms of both the quality and consistency of responses.

BERT, while performing well in Precision (0.91) and Recall (0.88), slightly trails behind GPT-3.5 in both accuracy and the ability to distinguish between correct and incorrect answers. However, BERT excels in understanding the context and nuances of text, which makes it a strong contender for tasks requiring deep contextual comprehension. Despite not matching GPT-3.5 in F1-Score and ROC AUC, BERT's solid performance makes it a good option for document QA systems where understanding intricate details is crucial.

T5 is versatile and suitable for a wide range of NLP tasks, but its performance in document-grounded Question Answering is less optimal compared to both GPT-3.5 and BERT. With the lowest Precision (0.90), Recall (0.85), and F1-Score (0.87) among the three, T5 tends to provide more false positives and misses some relevant information. Its ROC AUC score (0.85) also indicates that T5 struggles more than the others at distinguishing relevant answers from irrelevant ones. While it can still perform well for general NLP tasks, it falls short for document-based QA applications.

# CONCLUSION AND FUTURE WORK

The ChatGPT Clone project demonstrates the feasibility and effectiveness of building a document-aware, intelligent chatbot system by integrating modern natural language processing (NLP) techniques, large language models (LLMs), and orchestration frameworks like LangChain. Designed to handle user queries grounded in the content of uploaded documents, the system provides a seamless, interactive interface through Streamlit, enabling real-time and contextually accurate conversations across academic, technical, and enterprise domains.

The architecture of the system is modular and scalable, allowing flexibility in choosing between powerful cloud-based models (like OpenAI’s GPT-3.5/4) and local offline alternatives (like GPT4All). By leveraging advanced embedding techniques and vector search engines such as FAISS, the chatbot retrieves the most semantically relevant chunks of information from documents and uses those to generate grounded and coherent responses. The integration of LangChain not only streamlines the pipeline for prompt construction and memory handling but also makes it easier to manage follow-up queries in multi-turn conversations.

Mathematical formulations such as cosine similarity for vector retrieval and attention mechanisms for transformer-based LLM inference were utilized or referenced to solidify the theoretical underpinnings of the system. Additionally, the token budget constraint was carefully considered to ensure compatibility with LLM input limitations. The project also demonstrated how prompt engineering, memory trimming, and caching strategies can significantly improve performance, reduce latency, and manage API usage costs.

Evaluation of the system using real-world academic PDFs and structured queries indicated strong performance in relevance, accuracy, and response fluency, especially when powered by OpenAI’s models. The local fallback model, GPT4All, was found to be effective for offline use cases, albeit with slightly reduced fluency. Manual testing and feedback further validated the utility of the system for document-based question answering tasks.

Overall, this project bridges theoretical NLP research with practical deployment and serves as a foundational step toward developing domain-specific AI agents. Future work could include integrating advanced NLP evaluation metrics (like ROUGE, METEOR), supporting more file formats, fine-tuning models for specific domains, and expanding the UI with voice input, chat export features, and multi-document querying. The system holds promise for applications in education, law, research, healthcare, and corporate document support systems.

For future work, enhancing retrieval techniques could significantly improve the ChatGPT Clone's context accuracy. Implementing semantic search or retrieval-augmented generation (RAG) could help retrieve more relevant documents, resulting in better responses. Additionally, fine-tuning the model with domain-specific data will help cater to specialized fields like law, medicine, or technology, improving performance in those contexts.

Another direction for improvement is the integration of multimodal capabilities. Allowing the chatbot to process not just text but also images and audio would make it more versatile and applicable in a wider range of scenarios, such as visual or audio-based Q&A.

Lastly, focusing on real-time performance optimization and continuous learning could boost the system’s efficiency and adaptability. By reducing latency and using feedback loops for model refinement, the system could offer faster, more relevant responses while evolving with user interactions.

##### Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

[1] O. Topsakal and T. C. Akinci, “Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast,” Int. Conf. Appl. Eng. Nat. Sci., vol. 1, no. 1, pp. 1050–1056, 2023, doi: 10.59287/icaens.1127.

[2] K. Pandya and B. V. Mahavidyalaya, “Building custom open-source GPT Chatbot for organizations,” Comput. Lang. (cs.CL); Comput. Soc. (cs.CY); Mach. Learn., vol. 1, pp. 28–31, 2023, [Online]. Available: https://arxiv.org/abs/2310.05421

[3] Adith Sreeram and Pappuri Jithendra Sai, “An Effective Query System Using LLMs and LangChain,” Int. J. Eng. Res. Technol., vol. 78, no. July, pp. 367–369, 2023.

[4] A. Tiwari et al., “Implications of ChatGPT in Public Health Dentistry: A Systematic Review,” Cureus, vol. 15, no. 6, 2023, doi: 10.7759/cureus.40367.

[5] Y. Gao et al., “Retrieval-Augmented Generation for Large Language Models: A Survey,” 2023, [Online]. Available: http://arxiv.org/abs/2312.10997

[6] J. Parviainen and J. Rantala, “Chatbot breakthrough in the 2020s? An ethical reflection on the trend of automated consultations in health care,” Med. Heal. Care Philos., vol. 25, no. 1, pp. 61–71, 2022, doi: 10.1007/s11019-021-10049-w.

[7] D. Xie et al., “Impact of Large Language Models on Generating Software Specifications,” 2023, [Online]. Available: http://arxiv.org/abs/2306.03324

[8] C. Zhang, J. Chen, J. Li, Y. Peng, and Z. Mao, “Large language models for human–robot interaction: A review,” Biomim. Intell. Robot., vol. 3, no. 4, p. 100131, 2023, doi: 10.1016/j.birob.2023.100131.

[9] R. Gaizauskas and K. Humphreys, “A Combined IR/NLP Approach to Question Answering Against Large Text Collections,” Proc. 6th Content-Based Multimed. Inf. Access Conf. (RIAO-2000, no. 1969, pp. 1288–1304, 2000.

[10] P. P. Ray, “ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope,” Internet Things Cyber-Physical Syst., vol. 3, no. March, pp. 121–154, 2023, doi: 10.1016/j.iotcps.2023.04.003.

[11] D. Kalla and S. Kuraku, “Study and analysis of Chat GPT and its impact on different fields of study,” Int. J. Innov. Sci. Res. Technol., vol. 8, no. 3, pp. 827–833, 2023, [Online]. Available: www.ijisrt.com

[12] P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” Adv. Neural Inf. Process. Syst., vol. 2020-Decem, 2020.

[13] Y. Gao et al., “Retrieval-Augmented Generation for Large Language Models: A Survey,” pp. 1–21, 2023, [Online]. Available: http://arxiv.org/abs/2312.10997

[14] D. Kalla and S. Kuraku, “Study and analysis of Chat GPT and its impact on different fields of study,” Int. J. Innov. Sci. Res. Technol., vol. 8, no. 3, pp. 827–833, 2023, [Online]. Available: www.ijisrt.com

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi TIFF or EPS file, with all fonts embedded) because, in an MSW document, this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord “Format” pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.