

## **Applicative Project Report**

### **Deep chat: Customizable Chatbot with API and LLM Integration**



#### **Student Id's**

- 1. Mohitha Bandi (22WUO0105037)**
- 2. Pailla Bhavya (22WUO0105020)**
- 3. T. Harshavardhan Reddy(2WUO0105023)**
- 4. Thumma Manojna Reddy(22WUO0104134)**

#### **Under Supervision of**

**Dr. Ram Murat Singh**

**Assistant Professor**

**Woxsen University**

**Hyderabad**

## Contents

|                                                    |    |
|----------------------------------------------------|----|
| 1. Abstract .....                                  | 3  |
| 2. Introduction .....                              | 3  |
| 2.1 Purpose .....                                  | 3  |
| 2.2 Background .....                               | 3  |
| 2.3 Project Scope .....                            | 3  |
| 2.4 Definitions, Acronyms, and Abbreviations ..... | 4  |
| 3. Overall Description .....                       | 5  |
| 3.1 Product Perspective .....                      | 5  |
| 3.2 Product Functions .....                        | 5  |
| 3.3 User Characteristics .....                     | 5  |
| 3.4 Constraints .....                              | 6  |
| 3.5 Assumptions and Dependencies .....             | 7  |
| 4. Specific Requirements .....                     | 7  |
| 4.1 Functional Requirements .....                  | 7  |
| 4.2 Non-Functional Requirements .....              | 8  |
| 4.3 External Interface Requirements .....          | 9  |
| 5. Literature Survey .....                         | 9  |
| 6. Working .....                                   | 10 |
| 6.1 Aim .....                                      | 10 |
| 6.2 Proposed System .....                          | 10 |
| 6.3 Methodology .....                              | 11 |
| 6.4 Workflow .....                                 | 11 |
| 7. Software Requirements .....                     | 13 |
| 8. Results .....                                   | 13 |
| 9. Future work .....                               | 16 |
| 10. Conclusion .....                               | 17 |
| 11. References .....                               | 17 |

### 1. Abstract

The increasing demand for intelligent, user-friendly, and highly customizable chatbot solutions has given rise to the development of advanced conversational systems powered by large language models (LLMs). Deep Chat is a lightweight, browser-hosted, and highly extensible chatbot framework designed to overcome the limitations of traditional chatbots by enabling seamless API integration, local model hosting, and multimodal interactions.

This project combines cutting-edge technologies such as Retrieval-Augmented Generation (RAG), ChromaDB for document-based semantic search, and Low-Rank Adaptation (LoRA) for fine-tuning lightweight language models like TinyLlama. Deep Chat supports integration with various APIs including OpenAI, Google, and Spotify, enabling users to perform diverse tasks through a single unified interface. The chatbot is equipped with advanced features like speech-to-text (STT), text-to-speech (TTS), document understanding, file upload, and contextual Q&A, all accessible via a browser-based interface with minimal configuration.

The primary objective of this system is to empower developers, researchers, and businesses with a customizable chatbot that is easy to deploy and versatile in usage. Whether it's for educational support, customer service, internal knowledge access, or interactive learning, Deep Chat is built to adapt across use cases while offering high performance and modularity.

### 2. Introduction

#### 2.1 Purpose

The purpose of this document is to outline the complete software requirements specification (SRS) for the project titled "Deep Chat: Customizable Chatbot with API and LLM Integration." This document aims to serve as a guideline for the design, development, testing, and deployment phases of the system. It details the system's functionalities, constraints, and interfaces to ensure all stakeholders have a clear understanding of the system's objectives and deliverables.

#### 2.2 Background

With the growing demand for intelligent and adaptive digital assistants, many businesses and developers are seeking highly flexible chatbot solutions that can be easily integrated into their websites and applications. Most existing chatbots are either overly simple or tightly bound to specific APIs and services. They lack modularity, multi-model support, and contextual awareness across different content formats like documents or voice inputs.

To address these limitations, "Deep Chat" is developed as an innovative, browser-hosted, lightweight, and customizable chatbot framework. It supports integration with various APIs such as OpenAI, Google, and Spotify; allows for voice-based interaction; provides document understanding through RAG (Retrieval-Augmented Generation); and supports hosting of fine-tuned local LLMs like TinyLlama with LoRA optimization.

#### 2.3 Project Scope

This system will act as a unified, embeddable chatbot solution that can be customized by developers and used by end-users for diverse conversational tasks, ranging from document-

## Deep chat: Customizable Chatbot with API and LLM Integration

based Q&A to API testing. The chatbot can be embedded into any web page, supports real-time speech-to-text and text-to-speech functionalities, file uploads, and vector search via ChromaDB. It is designed to work cross-platform and can serve educational institutions, customer support systems, data-centric organizations, and even research environments.

### 2.4 Definitions, Acronyms, and Abbreviations

- **AI (Artificial Intelligence):** The simulation of human intelligence by machines to perform tasks such as learning, reasoning, and problem-solving.
- **API (Application Programming Interface):** A set of defined rules and protocols that allow different software applications to communicate with each other.
- **LLM (Large Language Model):** A powerful AI model trained on vast amounts of textual data to generate and understand human-like language.
- **RAG (Retrieval-Augmented Generation):** A technique that retrieves relevant data from a knowledge source and uses it to enhance the generation capabilities of a language model.
- **LoRA (Low-Rank Adaptation):** A parameter-efficient fine-tuning method used to adapt large pre-trained models with reduced computational requirements.
- **TTS (Text-to-Speech):** A technology that converts written text into spoken audio output.
- **STT (Speech-to-Text):** A voice recognition system that converts spoken language into written text.
- **ChromaDB:** An open-source vector database optimized for storing and querying dense vector embeddings, often used in AI and NLP systems.
- **UI/UX (User Interface / User Experience):** The design and interaction elements that define how users interact with a system and how satisfying that experience is.
- **Whisper:** A speech-to-text transcription model developed by OpenAI, known for its multilingual and highly accurate transcription capabilities.
- **TinyLlama:** A lightweight, fine-tuned large language model ideal for resource-constrained environments, offering high performance with fewer computational resources.
- **Embedding:** A mathematical representation of text or other data in a high-dimensional space used for tasks like search, classification, and semantic analysis.
- **OpenAI:** An AI research organization that provides cutting-edge models like GPT for natural language understanding and generation.

### 3. Overall Description

#### 3.1 Product Perspective

Deep Chat is a standalone, browser-based chatbot designed to be easily integrated into websites or used independently. Unlike traditional chatbots, it offers high customizability with support for multiple APIs, local LLM hosting (e.g., TinyLlama), and document-based Q&A using RAG and ChromaDB. It enables multimodal interaction with text, voice (speech-to-text and text-to-speech), and file uploads.

The system is modular, platform-independent, and developer-friendly, allowing seamless integration with services like OpenAI, Google, and Spotify. It acts as both an intelligent assistant and a flexible tool for various domains such as education, customer service, and internal knowledge access.

#### 3.2 Product Functions

The Deep Chat system provides the following key functionalities:

- **Conversational Interface:** Offers a dynamic chat interface for users to interact via text and voice, simulating human-like conversations using integrated language models.
- **Multimodal Support:** Enables Speech-to-Text (STT) and Text-to-Speech (TTS) for voice-based communication, enhancing accessibility and user experience.
- **Document-Based Q&A (RAG):** Allows users to upload documents (PDFs, text, etc.) and ask context-specific questions, which are answered using Retrieval-Augmented Generation and ChromaDB.
- **Custom API Integration:** Supports integration with external APIs (e.g., OpenAI, Google Search, Spotify) to fetch dynamic data or perform actions like music retrieval, knowledge queries, etc.
- **Local Model Hosting:** Provides an option to host and interact with fine-tuned local LLMs such as TinyLlama, optimized using LoRA for lightweight deployment.
- **File Upload and Interaction:** Users can upload files, and the chatbot can extract, process, and generate responses based on the file content.
- **Browser-Based Deployment:** Fully accessible via modern web browsers with minimal setup, ensuring cross-platform compatibility without external installations.
- **Customizable UI & Behavior:** Developers can customize the UI, APIs, and response behavior based on their application needs, making it flexible for different use cases.

#### 3.3 User Characteristics

The intended users of Deep Chat span across various technical and non-technical backgrounds. Understanding their characteristics helps in designing an intuitive, accessible, and functional system.

## Deep chat: Customizable Chatbot with API and LLM Integration

- **Developers & Integrators:**
  - Technical users with knowledge of APIs and front-end/back-end technologies.
  - They will integrate Deep Chat into websites, configure custom APIs, or host local models.
  - Familiar with JavaScript, HTML, Python, or API documentation.
- **End Users (Non-Technical):**
  - General users interacting with the chatbot for assistance, information retrieval, or task execution.
  - May include students, customers, employees, or visitors on a website.
  - Require a clean, intuitive interface with simple prompts and voice/text support.
- **Administrators:**
  - Manage configurations, monitor logs, and maintain performance.
  - May handle uploading new documents for Q&A, setting permissions, or managing API keys.

All user types benefit from the system's modular design and user-friendly UI, enabling smooth adoption and integration regardless of technical proficiency.

### 3.4 Constraints

- **Browser Dependency:**The system is designed to be browser-based, so it relies on modern web browsers with support for JavaScript, WebRTC, and microphone/file access.
- **Internet Requirement (for API Access):**Access to external APIs like OpenAI, Google, or Spotify requires a stable internet connection. Offline capabilities are limited to locally hosted models.
- **Performance Limitations on Low-End Devices:**Hosting LLMs or running heavy document processing locally may not perform optimally on resource-constrained devices.
- **Data Privacy & Security:**File uploads, API calls, and voice data may include sensitive user information. It's essential to handle data with secure protocols (e.g., HTTPS, encrypted storage).
- **Limited Multilingual Support (Dependent on Models):**Multilingual capabilities are dependent on the language support of the selected LLM or speech model (e.g., Whisper or TinyLlama).
- **Model Size & Hosting Constraints:**Large models like GPT require remote access or GPU hosting. Locally hosted models are limited to lightweight versions (e.g., TinyLlama) due to hardware constraints.

## Deep chat: Customizable Chatbot with API and LLM Integration

- **Customization Requires Technical Skills:** Deep customizations such as integrating new APIs or modifying model behavior require knowledge of front-end/backend development.
- **Third-Party API Rate Limits:** APIs used (OpenAI, Google, etc.) may impose rate limits or usage costs, which can affect scalability.

### 3.5 Assumptions and Dependencies

- **User Has Internet Access:** It is assumed that users have a stable internet connection when accessing features that depend on external APIs (e.g., OpenAI, Google Search).
- **Modern Web Browser Availability:** The system depends on users having access to a modern browser (Chrome, Firefox, Edge, Safari) that supports JavaScript, WebRTC, and HTML5 features.
- **Third-Party API Availability:** The system assumes continuous availability and stability of third-party services like OpenAI, Google, and Spotify. Downtime in these services can affect chatbot performance.
- **Pre-configured API Keys:** Developers are expected to set up valid API keys or access tokens for integrated services. The system depends on proper configuration of these credentials.
- **Local Hosting Environment (Optional):** If using TinyLlama or other local models, it is assumed that the deployment environment has the required dependencies installed (Python, hardware compatibility, etc.).
- **Security Measures in Place:** It is assumed that appropriate security measures (HTTPS, CORS settings, access control) are implemented during deployment for safe file handling and API interaction.
- **Users Will Interact Ethically:** The system assumes that users will not intentionally try to misuse the chatbot for offensive, malicious, or unintended purposes.

## 4. Specific Requirements

### 4.1 Functional Requirements

- **Chat Interface**
  - Provides a web-based chat interface for users.
  - Supports both text input and voice input (speech-to-text).
- **LLM Integration**
  - Integrates with OpenAI API or locally hosted TinyLlama model.
  - Allows switching between local and cloud-based models.
- **API Integration**
  - Supports third-party API integration (e.g., Google Search, Spotify).
  - Enables real-time API data fetching during conversations.
- **File Upload and Processing**
  - Allows users to upload files (PDF, CSV).
  - Extracts content and enables question answering based on documents using Retrieval-Augmented Generation (RAG).



## Deep chat: Customizable Chatbot with API and LLM Integration

- **Speech-to-Text (STT)**
  - Converts voice input from users into text using Whisper or browser-native APIs.
- **Text-to-Speech (TTS)**
  - Converts chatbot's text responses into voice using TTS engines (e.g., ElevenLabs, browser-native options).
- **Vector Database Integration**
  - Uses ChromaDB to store document embeddings.
  - Retrieves relevant document chunks during question answering.
- **Customizability**
  - Allows developers to configure chatbot behavior, personality, APIs, and UI settings.
  - Supports modular addition of new features and integrations.
- **Browser Compatibility**
  - Fully functional in modern browsers without requiring external software installations.
- **Security and Permissions**
  - Handles file uploads securely.
  - Allows only authorized developers to modify backend/API configurations.

### 4.2 Non-Functional Requirements

#### Performance

- The chatbot should respond to user inputs within 2–3 seconds under normal conditions.
- Document-based responses should be delivered in under 5 seconds for typical files.

#### Scalability

- The system should support multiple simultaneous users.
- Ability to switch between local and cloud-based LLMs based on system resources.

#### Usability

- Interface should be user-friendly and easy to navigate.
- Voice input/output should include visual indicators and clear instructions.

#### Reliability & Availability

- Cloud-hosted features should maintain at least 99% uptime.
- Local deployment should run consistently on compatible machines.

#### Portability

- The application should run on all major platforms (Windows, Linux, macOS).
- Codebase and configuration should be easily portable between environments.

#### Maintainability

- The code should follow modular design for easy updates.



## Deep chat: Customizable Chatbot with API and LLM Integration

- Configuration (e.g., API keys) should be editable without frontend changes.

### Security

- All file uploads should be scanned/sanitized to prevent malicious content.
- API keys and credentials must be secured and hidden from the client side.

## 4.3 External Interface Requirements

### 4.3.1 User Interface

- Web-based interface with minimalist UI
- Input fields for text and voice
- Upload buttons for document handling
- Live chat history view

### 4.3.2 Hardware Interfaces

- Microphone and camera input
- Local file system access for uploads

### 4.3.3 Software Interfaces

- OpenAI API
- Whisper and TTS APIs
- Spotify for Spotify interaction
- ChromaDB for RAG and embeddings

### 4.3.4 Communication Interfaces

- Secure HTTPS communication
- WebSocket integration for real-time messaging

## 5.Literature Survey

Over the past few decades, chatbot technology has undergone a significant transformation—from simple rule-based systems to advanced conversational agents powered by large language models (LLMs). Early examples like ELIZA (1966) and ALICE (1995) relied on fixed rules and pattern matching, making them incapable of understanding context or generating dynamic responses. These systems laid the groundwork for more interactive and intelligent chatbots. The emergence of retrieval-based systems marked a major advancement, as they could return relevant responses by searching a fixed knowledge base. However, they still struggled with flexibility and natural conversation flow.

## Deep chat: Customizable Chatbot with API and LLM Integration

The breakthrough in natural language processing came with the introduction of transformer-based generative models like OpenAI's GPT family, Google's Bard, and Anthropic's Claude. These models are capable of understanding and generating human-like text, offering coherence, contextual awareness, and adaptability. Alongside this evolution, voice interaction became more prominent through the development of accurate Speech-to-Text (STT) models like Whisper by OpenAI, and realistic Text-to-Speech (TTS) engines such as Google TTS, Amazon Polly, and ElevenLabs.

To further enhance chatbot performance, Retrieval-Augmented Generation (RAG) emerged as a powerful approach. It combines information retrieval from external sources with text generation, allowing chatbots to provide grounded, accurate, and document-aware responses. Tools like ChromaDB and FAISS support this by enabling efficient storage and retrieval of document embeddings. This is particularly useful for custom use cases such as answering questions based on uploaded PDFs or organizational documents.

In parallel, the rise of lightweight and open-source LLMs—such as TinyLlama, Alpaca, and Mistral—has made it feasible to run advanced models locally, reducing reliance on cloud APIs and improving privacy and cost control. Additionally, modern chatbot systems are increasingly being designed to integrate with third-party APIs, enabling dynamic functionalities such as real-time search, weather reports, and media control.

Building on all these advancements, the Deep Chat project aims to combine the best of these technologies into a single customizable, browser-based chatbot. By integrating LLMs, RAG, voice features, and third-party APIs, Deep Chat offers a highly modular and intelligent conversational assistant suitable for a wide range of applications.

## 6. Working

### 6.1 Aim

The aim of this project is to design and develop an intelligent, browser-based chatbot system named Deep Chat that is highly customizable and integrates with both local and cloud-hosted Large Language Models (LLMs). It should support multimodal interaction including text, voice, and document-based queries, and allow developers to easily plug in external APIs and extend its capabilities for various real-world applications.

### 6.2 Proposed System

The proposed system, Deep Chat, is a customizable and modular chatbot platform that merges the power of LLMs with additional intelligent features such as speech-to-text (STT), text-to-speech (TTS), API integration, document-based query answering (RAG), and developer configuration. The system is designed to run directly in a web browser, ensuring ease of deployment, accessibility, and minimal setup.

Key components of the system include:

- **LLM Integration:** Switchable between TinyLlama (local) and RAG.
- **Document Interaction:** Users can upload documents (PDF, DOCX, TXT), which are embedded and stored in a vector database (ChromaDB) for contextual querying using RAG.

## Deep chat: Customizable Chatbot with API and LLM Integration

- **Speech Support:** Voice inputs are transcribed using STT (Whisper or native APIs) and bot responses can be vocalized via TTS engines.
- **Third-party API Support:** Developers can plug in APIs like Google Search, Spotify, or custom tools to enhance the chatbot's intelligence.
- **Developer-Friendly:** Offers configuration panels and flexible architecture for feature toggling, API key injection, and UI customization.

### 6.3 Methodology

The development of Deep Chat follows a modular and iterative methodology, ensuring flexibility, ease of integration, and performance. The methodology can be summarized in the following phases:

- **Requirement Analysis**
  - Identify key functionalities like text/voice chat, document Q&A, and LLM connectivity.
  - Analyze target users (developers, end-users, educators, etc.).
- **Design & Architecture**
  - Design frontend (chat UI, controls) and backend (LLM handler, API manager, vector DB).
  - Define modular components for each feature (speech, file handling, APIs, LLM selection).
- **Implementation**
  - Develop frontend in modern web stack (React/HTML5/CSS3).
  - Integrate LLMs via OpenAI API or local inference using TinyLlama.
  - Embed RAG pipeline using ChromaDB and HuggingFace models.
  - Build speech modules using Whisper and ElevenLabs/browser-native TTS.
- **Testing**
  - Perform unit testing, integration testing, and UI testing.
  - Validate responses, document querying accuracy, and multi-feature stability.
- **Deployment**
  - Deploy system locally or host via a lightweight browser-based server.
  - Ensure compatibility across major browsers.

### 6.4 Workflow

#### 1. User Opens Chat Interface

- The user accesses the chatbot through a web browser interface.

#### 2. Input Mode Selection

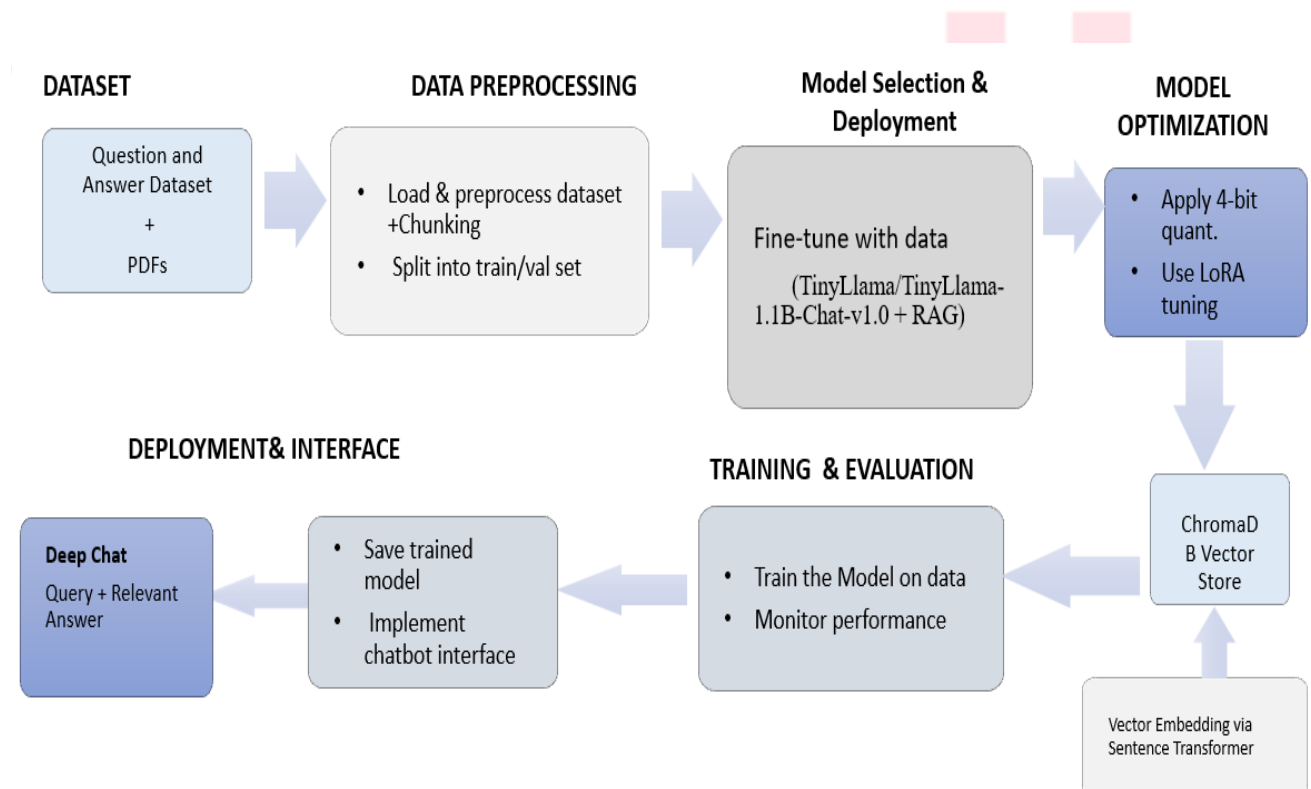
- The user can choose between text input or voice input.
- For voice, STT converts spoken words into text.

#### 3. Chatbot Processing

- The chatbot processes the input using either:
  - A cloud-based LLM (e.g., OpenAI GPT)

## Deep chat: Customizable Chatbot with API and LLM Integration

- A locally hosted TinyLlama model
4. **Optional Document Context (RAG)**
    - If a document is uploaded:
      - It's parsed and converted into vector embeddings.
      - Relevant content is retrieved from ChromaDB and used to augment the response.
  5. **API Integration (Optional)**
    - If the input involves an external data request, the chatbot calls the appropriate API and merges results with the LLM response.
  6. **Response Generation**
    - The final response is generated and sent to the user.
    - Optionally, the response is spoken using TTS.
  7. **User Interaction Continues**
    - The conversation continues in a loop until ended by the user.



## 7. Software Requirements

### Programming Language:

- Python

### Libraries:

- Transformers – Load and fine-tune TinyLlama
- SentenceTransformers – Generate document embeddings
- LangChain – Build the RAG pipeline (split, embed, retrieve, query)
- Pandas – Read and preprocess the CSV dataset
- ChromaDB / FAISS – Vector store for fast semantic search
- PyPDF – Extract text from PDFs

### Tools:

- Streamlit – Build the chatbot UI (with built-in CSS support)
- Ollama – Run local LLMs in your environment

### Operating System:

- Windows

## 8. Results

| Step | Training Loss |
|------|---------------|
| 10   | 1.434100      |
| 20   | 0.193600      |
| 30   | 0.140000      |
| 40   | 0.106800      |
| 50   | 0.087100      |
| 60   | 0.066900      |
| 70   | 0.057800      |
| 80   | 0.043900      |
| 90   | 0.038400      |
| 100  | 0.035600      |

## Deep chat: Customizable Chatbot with API and LLM Integration

Answer:  
Use the following pieces of context to answer the question at the end. If you don't know the answer, just say that you don't know, don't try to make up an answer.

universities in the US was used for this study. The different ways in which LLM powered LLMs can make their way assisting the educational system are mentioned in the paper.

universities in the US was used for this study. The different ways in which LLM powered LLMs can make their way assisting the educational system are mentioned in the paper.

Another aspect came out to be on using the LLMs in the field of education was the inexperience of the educational institutions in working with the similar kind of technology in the past. Though LLMs have the potential to enhance the language education but its successful implementation relies on thoughtful and skillful integration by the educational institutions. Also, the LLM service providers have not fully disclosed the privacy and data protection

### ➤ RAG +Llama

### RAG Chain Question Answering

Enter a URL and a query to get answers from the RAG chain.

url

<https://www.geeksforgeeks.org/what-is-retrieval-augmented-generation-rag/>

question

what is RAG?

Clear Submit

output

Retrieval-Augmented Generation (RAG) is a technique used in natural language processing that combines the strengths of generation and retrieval models. In RAG, a generator model (such as GPT) is augmented with external knowledge retrieved from a massive dataset, allowing it to generate more accurate and informative responses.

The key benefits of RAG include:

1. **"Improved accuracy"**: By incorporating real-time data, RAG can provide more up-to-date and relevant information.
2. **"Domain-specific knowledge"**: RAG can integrate domain-specific knowledge, making it suitable for tasks that require specialized expertise.
3. **"Scalability"**: RAG eliminates the need for extensive retraining on new domains or topics, making it highly scalable.
4. **"Cost-effective"**: Retrieving external data is more efficient than fine-tuning large models for specific tasks.

However, RAG also faces some challenges, including:

1. **"Complexity"**: Combining generation and retrieval models can be complex and require careful tuning.
2. **"Data quality and bias"**: The quality of the external data can impact the accuracy and fairness of

Flag

Use via API · Built with Gradio · Settings

### RAG Question Answering Chatbot

Enter a URL and a query to get answers from the RAG.

url

<https://www.geeksforgeeks.org/what-is-retrieval-augmented-generation-rag/>

question

Can you give me the html code for the login page

Clear Submit

output

I'd be happy to help you with that! However, I noticed that you didn't provide any specific requirements or constraints for the login page HTML code.


Assuming you want a basic login form with username and password fields, here's an example of HTML code:

```
'''html
<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
<style>
body {
font-family: Arial, sans-serif;
}

.login-container {
width: 300px;
margin: 40px auto;
padding: 20px;
border: 1px solid #ccc;
border-radius: 5px;
}
```

Flag

Use via API · Built with Gradio · Settings




# Deep chat - RAG-Document Summarizer & Interactive QueryBot

Document Summarizer


FAQ Chatbot

## Upload & Summarize Documents


Upload a PDF or TXT file


 Drag and drop file here  
Limit 200MB per file • PDF, TXT


Browse files

 Overview\_default\_database.pdf 19.1KB

×



 This document has already been processed!

 Stored Summary:

 Summary:

The provided text describes a database with six tables:


1. customers: stores information about customers, including their ID, name, and country.
2. employees: stores information about employees, including their ID, first name, last name, salary, and office location.
3. offices: stores information about office locations, including the ID, city, state or province, and country.
4. orders: stores information about customer orders, including the order ID, customer ID, employee


  Show Saved Documents

Select a document:

Database Overview.pdf

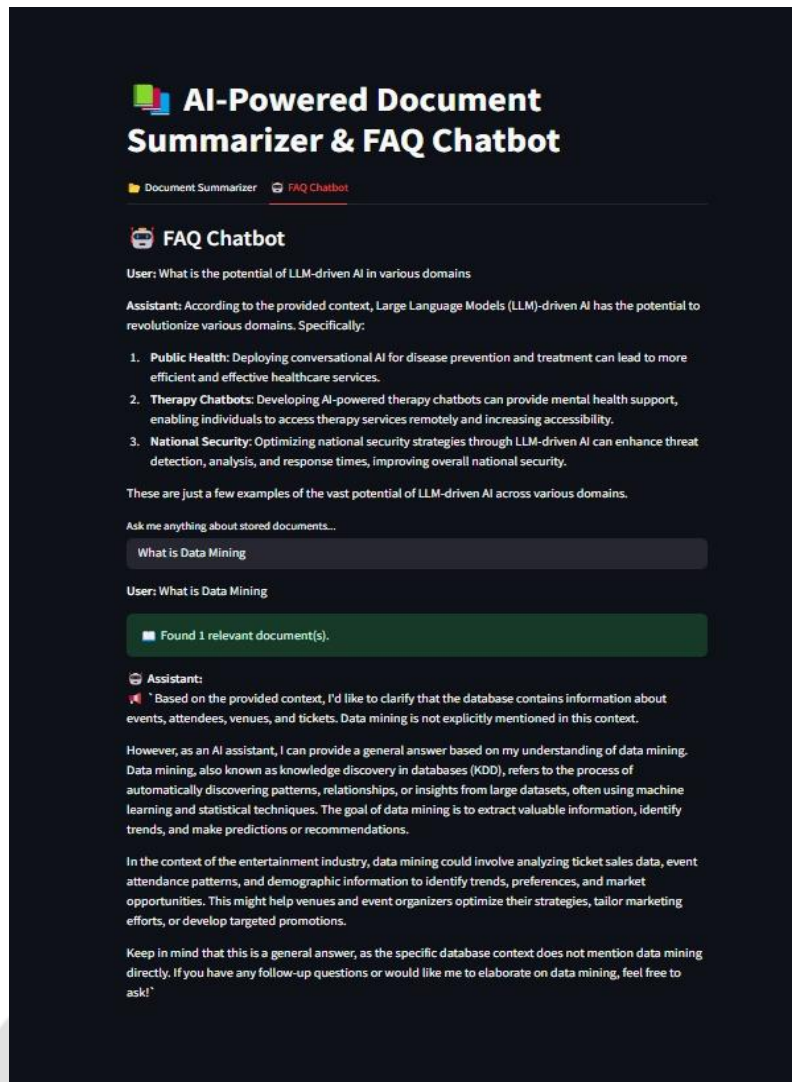
▼

 View Summary

 Delete Document

15





## 9. Future work

- **Image Summarization:**
  - Convert diagrams or screenshots into short, readable descriptions.
  - This helps users understand visuals without needing to analyze them deeply.
- **Ask Questions on Tables and Images:**
  - Let users query data from tables or visuals in documents.
  - Makes interaction with complex documents easier and more natural.
- **Show Graphs from Tables:**
  - Automatically turn table data into charts and graphs.
  - This helps users quickly visualize trends or comparisons.
- **Support for Images and Tables Together:**
  - Understand content from mixed formats like images, tables, and text.
  - Ideal for documents like reports or research papers.
- **Run on Mobile or Browser:**
  - Make the chatbot lightweight enough to run on phones or in browsers.

- No installation or heavy backend required for use.

## 10. Conclusion

The Deep Chat project demonstrates the successful design and implementation of a highly customizable, browser-based chatbot system that integrates advanced AI capabilities with user-friendly modularity. By leveraging both cloud-hosted and locally hosted LLMs, the chatbot provides flexible and intelligent responses suited for various domains. The inclusion of document-based querying through Retrieval-Augmented Generation (RAG), support for speech-to-text and text-to-speech interactions, and the ability to integrate third-party APIs sets Deep Chat apart as a versatile and powerful conversational platform.

This project not only addresses the limitations of traditional chatbot systems—such as limited context awareness and rigid input modes—but also creates a framework that developers and organizations can build upon to meet specific needs. The modular architecture, ease of configuration, and lightweight browser deployment make Deep Chat an ideal choice for both individual users and enterprise-level applications.

In conclusion, Deep Chat embodies the next generation of conversational agents, blending cutting-edge AI technology with practical usability, accessibility, and adaptability. It stands as a testament to what is possible when language models, speech processing, and dynamic content retrieval are harmonized into a single interactive platform.

## 11. REFERENCES:

1. M. S. Salim et al., "LLM based QA chatbot builder: A generative AI-based chatbot builder for question answering," *SoftwareX*, vol. 29, Article 102029, February 2025.
2. B. Johnson et al., "Unleashing LangChain, RAG, and Performance-Optimized LLM," *ScienceDirect*, 2024.
3. C. Lee et al., "A Complete Survey on LLM-based AI Chatbots," *ResearchGate*, 2024.
4. E. Chen et al., "An experimental hybrid customized AI and generative AI chatbot," *Springer*, 2024.
5. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... & Narang, S. (2023). *Llama 2: Open foundation and fine-tuned chat models*. Meta AI.