PROJECT: CODE GENEI AI

TECH Chatbot

1. Introduction

This project began as a simple chatbot interface and evolved into an advanced, intelligent system capable of analyzing user input, processing uploaded documents, and performing OCR (Optical Character Recognition) on images. Initially, the chatbot was designed to take user text input and respond using an AI model. Over time, features like file upload support, text extraction from multiple formats (PDF, Word, Excel, PowerPoint, TXT), and image-to-text conversion using OCR were added.

The chatbot now integrates **Ollama** (a local LLM inference server) and **DeepSeek API** (cloud-based AI service), allowing users to switch between AI backends seamlessly. This upgrade makes it a powerful, multi-modal chatbot capable of analyzing structured and unstructured data, documents, and images — all from a single interface.

2. Process / Procedures

Step-by-Step Workflow

1. **User Interaction**: The user provides input via text or uploads a file (PDF, DOCX, TXT, XLSX, PPTX, or image).
2. **File Processing**:The system detects the file type, extracts text using specialized functions, and passes it to the chatbot along with the user query.
3. **NLP Processing**:The extracted text and user prompt are sent to the selected backend (Ollama or DeepSeek) for natural language processing and response generation.
4. **Response Delivery**:The chatbot displays the AI-generated answer along with any insights derived from the file content.

3. NLP Explanation

3.1 What is NLP?

Natural Language Processing (NLP) is a field of AI that enables machines to understand, interpret, and generate human language. In this chatbot, NLP is used to analyze user input, extract meaning, and generate contextually relevant responses.

3.2 How NLP Works in This Project

- **Input Processing:** User query + extracted file text are combined into a single prompt.
- **Language Model Inference:** Prompt is sent to the selected AI model (Ollama or DeepSeek).
- **Contextual Understanding:** Model interprets the meaning and context of the message.
- **Response Generation:** AI generates a coherent, contextually accurate response.

4. Frontend

This project uses **Gradio** as the frontend framework.

Gradio Overview

- **Purpose:** Gradio is a Python library for quickly creating web-based interfaces for ML models.
- **Activity in Project:**
    - Provides a **chat interface** for user interaction.
    - Enables **file upload functionality** for multiple formats.
    - Displays chat history and supports clearing/resetting conversations.
    - Allows backend selection (Ollama / DeepSeek) dynamically.

5. Ollama – Local AI Inference

What is Ollama?

Ollama is a local inference engine that allows running large language models (LLMs) like Gemma, LLaMA, or Mistral on your machine without sending data to the cloud.

How It Is Useful

- **Privacy:** Data stays on your computer.
- **Offline Capability:** Works even without internet.
- **Low Latency:** Faster responses compared to cloud APIs.

6. Integration with Ollama

The chatbot integrates with Ollama via its REST API:

- **Backend Selection:** Users can choose "Ollama" in the UI.
- **Request Flow:** User input is sent to `http://localhost:11434/api/generate` with the model name (`gemma3:1b`).
- **Response Handling:** The API returns the generated text, which is displayed in the chatbot interface.

This ensures smooth interaction with the locally running model.

7. DeepSeek API Integration

DeepSeek API acts as a cloud-based AI backend.

- **Key Benefit:** Enables powerful language understanding and reasoning using DeepSeek's hosted model.
- **Request Flow:** A POST request is made to `https://api.deepseek.com/v1/chat/completions` with the user query and API key.
- **Fallback Option:** Users can switch between local (Ollama) and cloud (DeepSeek) backends as per preference.

8. OCR (Optical Character Recognition)

What is OCR?

OCR converts printed or handwritten text in images into machine-readable text. This project uses **Tesseract OCR** for image-to-text extraction.

Integration in Chatbot

- **File Upload Support:** Users can upload image files.
- **Processing:** Pytesseract extracts text from images.
- **Chat Enhancement:** Extracted text is passed to the NLP model, enabling analysis of scanned documents, receipts, handwritten notes, etc.

This feature significantly **upgrades the chatbot** to handle multimodal inputs, making it capable of analyzing not just typed queries but also content from real-world images.

9. Technologies Used

- **Programming Language – Python:**

  Python was chosen as the primary programming language due to its extensive libraries, simplicity, and strong ecosystem for AI, NLP, and file handling tasks.

- **Frontend Framework – Gradio:**

Gradio provides an intuitive and interactive web interface, enabling seamless user interaction with the chatbot, file upload, and real-time display of AI responses.

- **AI Backends – Ollama & DeepSeek API:**

  Ollama offers a **local inference engine** to run large language model (LLMs) privately, while DeepSeek API provides a **cloud-based NLP service** for enhanced reasoning and scalability.

- **OCR Engine – Tesseract (via Pytesseract):**

  Tesseract OCR extracts text from images, making the chatbot capable of analyzing scanned documents, handwritten notes, and other image-based content.

- **Libraries & Tools:**
  - **PyPDF2:** Extracts text from PDFs efficiently.
  - **python-docx:** Reads and processes Microsoft Word documents.
  - **pandas:** Handles Excel files, allowing tabular data extraction.
  - **python-pptx:** Reads PowerPoint slides for text extraction.
  - **Pillow:** Opens and manipulates images before OCR processing.
  - **requests:** Facilitates API communication with Ollama and DeepSeek servers.

11. Source Code

```
import os
import io
import base64
import tempfile
import mimetypes
import requests
import numpy as np
import pandas as pd
import pytesseract
import docx
import PyPDF2
from PIL import Image
```

```
from pptx import Presentation
import gradio as gr

# ============================================================
# API CONFIGURATIONS
# ============================================================
OLLAMA_URL = "http://localhost:11434/api/generate"
OLLAMA_MODEL = "gemma3:1b"  # ✅ Ollama model

DEEPSEEK_URL = "https://api.deepseek.com/v1/chat/completions"
DEEPSEEK_MODEL = "deepseek-chat"
DEEPSEEK_API_KEY = os.getenv("DEEPSEEK_API_KEY")

# ============================================================
# FILE PROCESSING FUNCTIONS
# ============================================================
def extract_text_from_pdf(file_path):
    """Extract text from a PDF file."""
    try:
        with open(file_path, "rb") as file:
            pdf_reader = PyPDF2.PdfReader(file)
            return "\n".join(page.extract_text() for page in pdf_reader.pages)
    except Exception as e:
        return f"❌ PDF Error: {str(e)}"


def extract_text_from_docx(file_path):
    """Extract text from a Word document."""
    try:
        doc = docx.Document(file_path)
        return "\n".join(paragraph.text for paragraph in doc.paragraphs)
    except Exception as e:
        return f"❌ DOCX Error: {str(e)}"


def extract_text_from_txt(file_path):
    """Extract text from a plain text file."""
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            return file.read().strip()
    except Exception as e:
        return f"❌ TXT Error: {str(e)}"


def extract_text_from_excel(file_path):
    """Extract text from Excel file (all sheets)."""
    try:
```

```
        excel_file = pd.ExcelFile(file_path)
        text = ""
        for sheet_name in excel_file.sheet_names:
            df = pd.read_excel(file_path, sheet_name=sheet_name)
            text += f"--- Sheet: {sheet_name} ---\n{df.to_string()}\n\n"
        return text.strip()
    except Exception as e:
        return f"❌ Excel Error: {str(e)}"


def extract_text_from_ppt(file_path):
    """Extract text from PowerPoint slides."""
    try:
        prs = Presentation(file_path)
        text = ""
        for slide_number, slide in enumerate(prs.slides, start=1):
            text += f"--- Slide {slide_number} ---\n"
            for shape in slide.shapes:
                if hasattr(shape, "text"):
                    text += shape.text + "\n"
            text += "\n"
        return text.strip()
    except Exception as e:
        return f"❌ PowerPoint Error: {str(e)}"


def extract_text_from_image(file_path):
    """Extract text from image using OCR."""
    try:
        pil_image = Image.open(file_path)
        extracted_text = pytesseract.image_to_string(pil_image).strip()
        return extracted_text or "⚠️ No text could be extracted from the image."
    except Exception as e:
        return f"❌ OCR Error: {str(e)}"


def process_uploaded_file(file):
    """Detect file type and extract text accordingly."""
    if not file:
        return "No file uploaded."

    file_path = file.name
    mime_type, _ = mimetypes.guess_type(file_path)

    try:
        if mime_type:
            if mime_type == "application/pdf":
```

```
            return extract_text_from_pdf(file_path)
        elif mime_type == "application/vnd.openxmlformats-
officedocument.wordprocessingml.document":
            return extract_text_from_docx(file_path)
        elif mime_type == "text/plain":
            return extract_text_from_txt(file_path)
        elif mime_type in ["application/vnd.ms-excel",
                           "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"]:
            return extract_text_from_excel(file_path)
        elif mime_type in ["application/vnd.ms-powerpoint",
                           "application/vnd.openxmlformats-
officedocument.presentationml.presentation"]:
            return extract_text_from_ppt(file_path)
        elif mime_type.startswith("image/"):
            return extract_text_from_image(file_path)

    # Fallback: check file extension
    ext = os.path.splitext(file_path)[1].lower()
    if ext == ".pdf":
        return extract_text_from_pdf(file_path)
    elif ext == ".docx":
        return extract_text_from_docx(file_path)
    elif ext == ".txt":
        return extract_text_from_txt(file_path)
    elif ext in [".xlsx", ".xls"]:
        return extract_text_from_excel(file_path)
    elif ext in [".pptx", ".ppt"]:
        return extract_text_from_ppt(file_path)
    elif ext in [".png", ".jpg", ".jpeg", ".bmp", ".tiff", ".webp"]:
        return extract_text_from_image(file_path)
    else:
        return f"⚠️ Unsupported file type: {ext}"

  except Exception as e:
    return f"❌ File processing error: {str(e)}"

# ============================================================
# CHATBOT LOGIC
# ============================================================
def ollama_reply(message):
  """Send a message to the Ollama local LLM."""
  try:
    response = requests.post(
      OLLAMA_URL,
      json={"model": OLLAMA_MODEL, "prompt": message, "stream": False}
    )
    if response.status_code == 200:
      return response.json().get("response", "⚠️ No response from Ollama.")
```

```
        return f"⚠️ Ollama API Error: {response.text}"
    except Exception as e:
        return f"❌ Error connecting to Ollama: {str(e)}"


def deepseek_reply(message):
    """Send a message to DeepSeek API."""
    try:
        headers = {"Authorization": f"Bearer {DEEPSEEK_API_KEY}"}
        response = requests.post(
            DEEPSEEK_URL,
            headers=headers,
            json={"model": DEEPSEEK_MODEL,
                  "messages": [{"role": "user", "content": message}]}
        )
        if response.status_code == 200:
            return response.json()["choices"][0]["message"]["content"]
        return f"⚠️ DeepSeek API Error: {response.text}"
    except Exception as e:
        return f"❌ Error connecting to DeepSeek: {str(e)}"


def chatbot_reply(message, history, backend, file=None):
    """Generate reply using the selected backend (Ollama / DeepSeek)."""
    file_content = ""
    if file is not None:
        file_content = process_uploaded_file(file)
        if file_content.startswith(("❌", "⚠️")):
            return file_content

    final_message = message
    if file_content and message.strip():
        final_message = f"{message}\n\n📄 File content:\n{file_content}"
    elif file_content:
        final_message = f"📄 File content:\n{file_content}\n\nPlease analyze this content."

    if backend == "Ollama":
        return ollama_reply(final_message)
    elif backend == "DeepSeek":
        return deepseek_reply(final_message)
    return "⚠️ Invalid backend selected."

# ============================================================
# UI LAYOUT (GRADIO)
# ============================================================
with gr.Blocks(css="""
#chat-history {height: 400px; overflow-y: auto;}
```

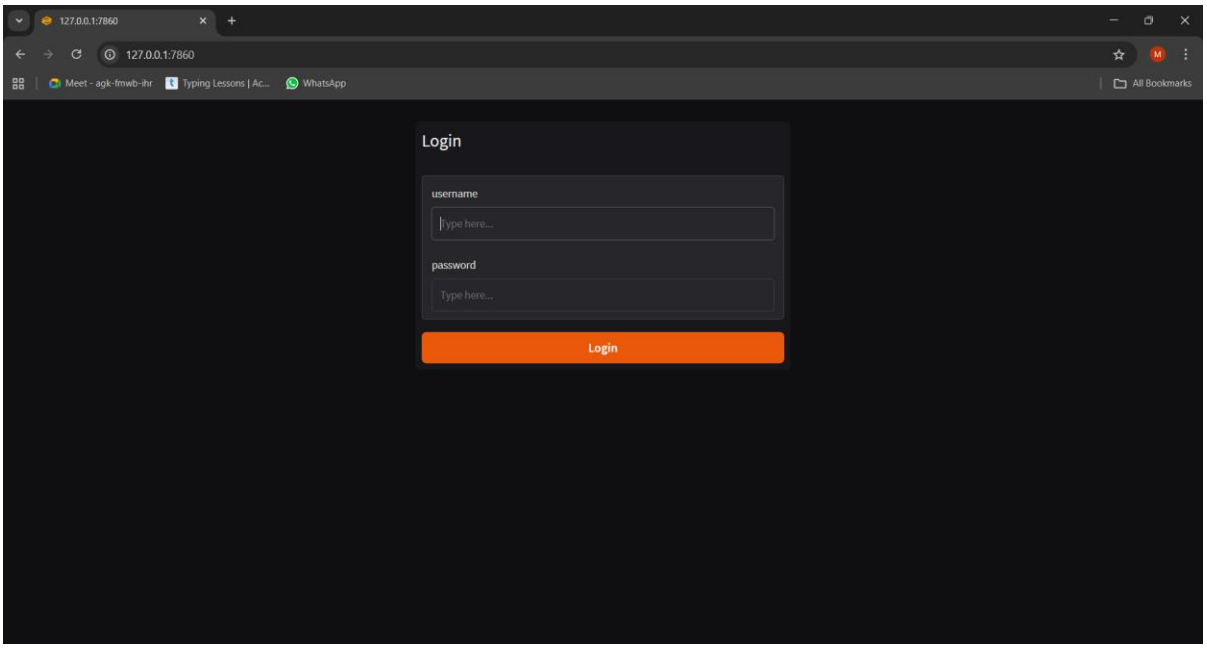
```
.file-upload-section {border: 1px solid #ccc; padding: 10px; border-radius: 5px; margin-bottom: 15px;}
.supported-files {font-size: 12px; color: #666; margin-top: 5px;}
""") as demo:
    gr.Markdown("## 🤖 TECH Chatbot")

    with gr.Row():
        with gr.Column(scale=2):
            gr.Markdown("### 📜 Chat History")
            chat_history = gr.State([])
            chat_titles = gr.Textbox(value="Chat 1", label="Current Chat", interactive=False)
            history_box = gr.Textbox(label="Saved Chats", elem_id="chat-history")
            backend_selector = gr.Radio(["Ollama", "DeepSeek"], value="Ollama", label="Choose AI Backend")

        with gr.Column(scale=3):
            with gr.Group(elem_classes="file-upload-section"):
                gr.Markdown("### 📁 Upload File")
                file_upload = gr.File(
                    label="Upload any file for analysis",
                    file_types=[
                        ".txt", ".pdf", ".docx", ".xlsx", ".xls",
                        ".pptx", ".ppt", ".png", ".jpg", ".jpeg", ".bmp", ".tiff", ".webp"
                    ],
                    file_count="single"
                )
                gr.Markdown("""
                <div class="supported-files">
                Supported files: PDF, Word (.docx), Excel (.xlsx, .xls), PowerPoint (.pptx, .ppt),
                Text (.txt), Images (.png, .jpg, .jpeg, .bmp, .tiff, .webp)
                </div>
                """)

            chatbot = gr.Chatbot(label="Chat Window", height=400, type="messages")
            user_input = gr.Textbox(label="Type your question here...", placeholder="Ask me something... or upload a file above for analysis")

            with gr.Row():
                submit_btn = gr.Button("Send", variant="primary")
                clear_btn = gr.Button("Clear Chat")
                clear_file_btn = gr.Button("Clear File")

    # Functions for UI
    def user_submit(message, history, titles, backend, file):
        response = chatbot_reply(message, history, backend, file)
        file_indicator = " + 📁" if file else ""
        history = history + [
```

```
            {"role": "user", "content": message + file_indicator},
            {"role": "assistant", "content": response}
        ]
        return history, history, "\n".join([f"Chat {i//2 + 1}" for i in range(len(history)//2)]), "", None

    def clear_chat():
        return [], [], "Chat 1", None

    def clear_file():
        return None

    # Button actions
    submit_btn.click(user_submit, [user_input, chatbot, chat_history, backend_selector,
file_upload],
                     [chatbot, chat_history, history_box, user_input, file_upload])
    user_input.submit(user_submit, [user_input, chatbot, chat_history, backend_selector,
file_upload],
                      [chatbot, chat_history, history_box, user_input, file_upload])
    clear_btn.click(clear_chat, outputs=[chatbot, chat_history, history_box, file_upload])
    clear_file_btn.click(clear_file, outputs=[file_upload])

# ============================================================
# RUN APP
# ============================================================
if __name__ == "__main__":
    demo.launch(
        server_name="127.0.0.1",
        server_port=7860,
        debug=True,
        auth=[("admin", "mypassword")]
```

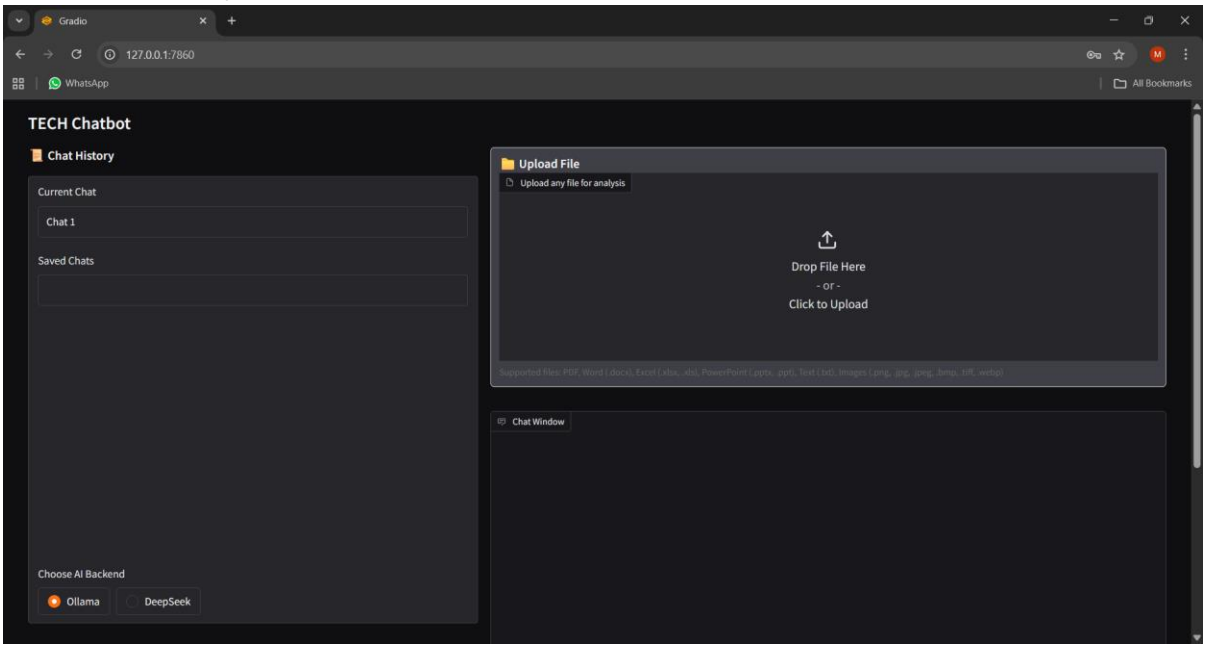
12. Screenshots

To clearly demonstrate the functionality of the project, the following screenshots are recommended:

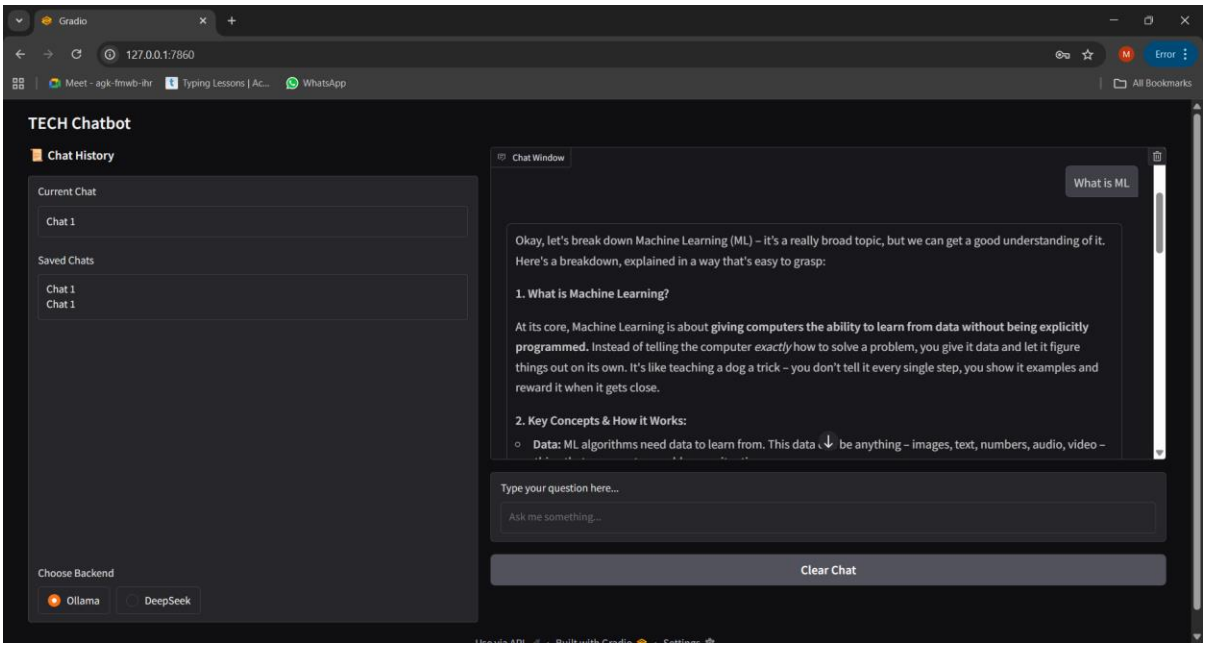
- **Login Page:** Showcasing the initial authentication interface for user access.



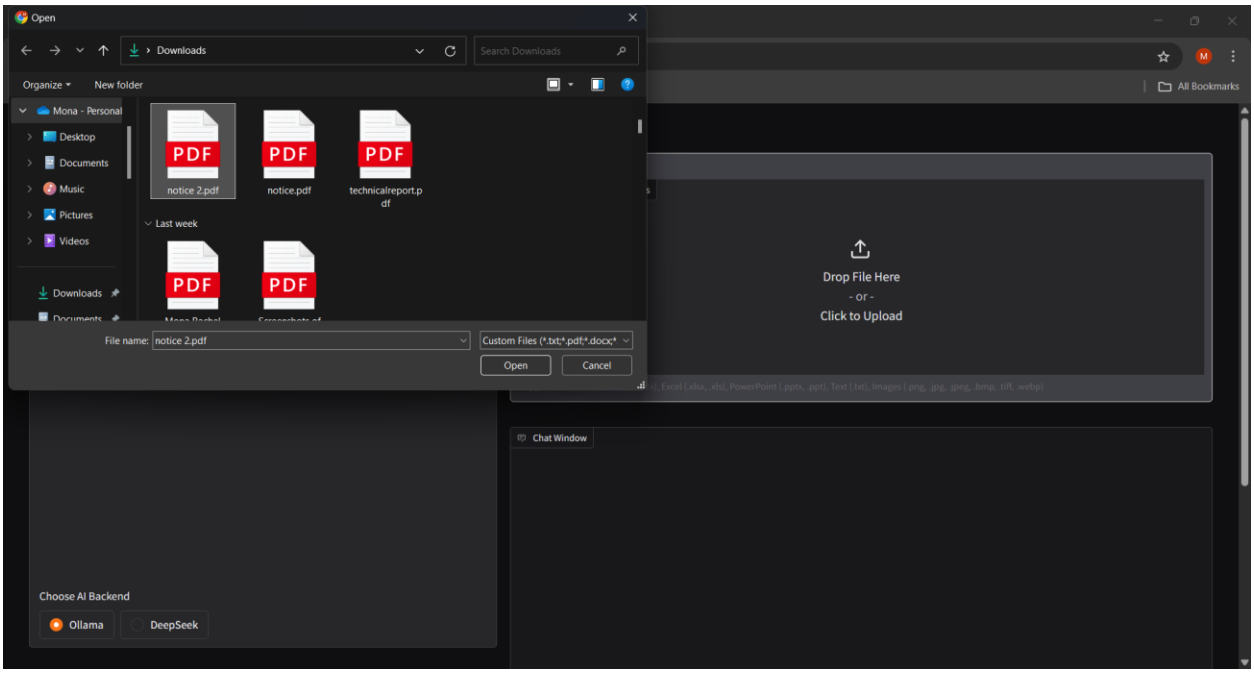
- **Homepage Interface:** Displaying the chatbot layout with file upload section and chat history.

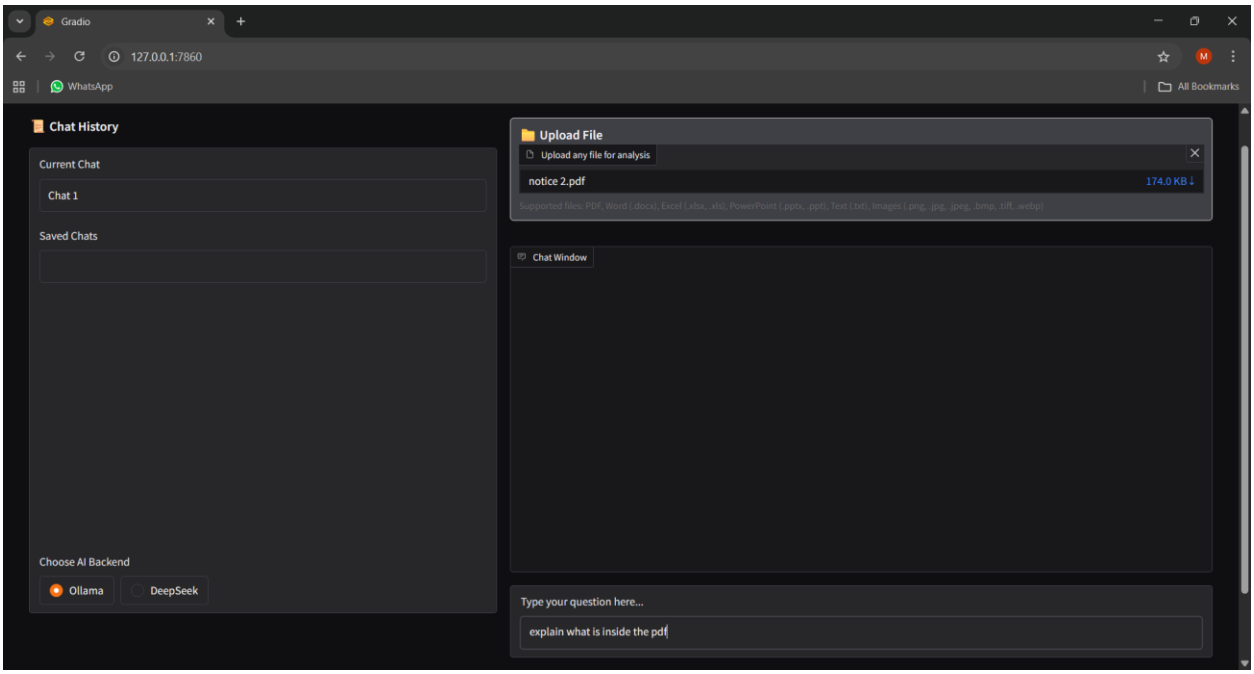


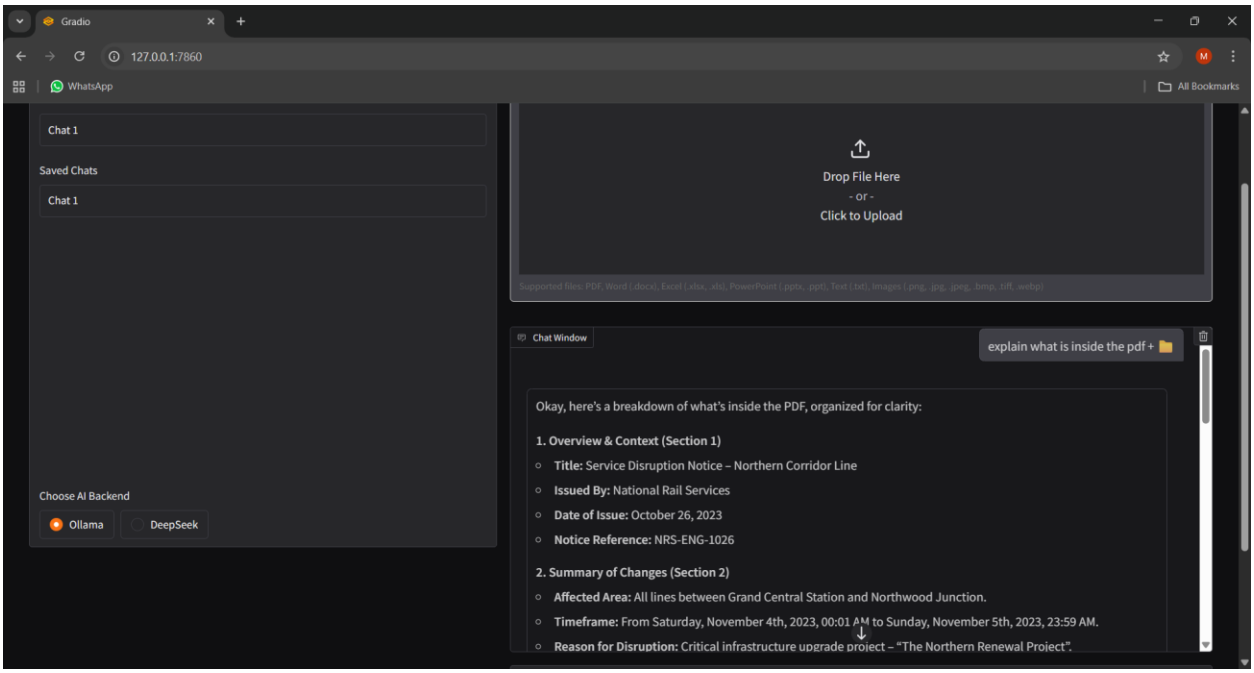
- **Initial Ollama Working:** Screenshot of the first successful response generated by the local Ollama LLM.



- **OCR Example:** Image upload process with extracted text and chatbot analysis.


Open
Downloads
Search Downloads
Organize
New folder
Mona - Personal
Desktop
Documents
Music
Pictures
Videos
Downloads
notice 2.pdf
notice.pdf
technicalreport.pdf
Last week
File name: notice 2.pdf
Custom Files (*.txt;*.pdf;*.docx;*
Open
Cancel
All Bookmarks
Drop File Here
- or -
Click to Upload
Chat Window
Choose AI Backend
Ollama
DeepSeek


Gradio
127.0.0.1:7860
WhatsApp
All Bookmarks
Chat History
Current Chat
Chat 1
Saved Chats
Upload File
Upload any file for analysis
notice 2.pdf
174.0 KB
Chat Window
Choose AI Backend
Ollama
DeepSeek
Type your question here...
explain what is inside the pdf



13. Advancements

Planned improvements and future enhancements include:

- **Voice Interaction:** Adding speech-to-text and text-to-speech for a hands-free experience.
- **Document Summarization:** Automatic summarization of lengthy documents and reports.
- **Database Integration:** Saving chat history, file data, and insights for future reference.
- **Multi-Language Support:** Expanding OCR and NLP to work with global languages.
- **Custom UI Upgrade:** Migrating to React or Streamlit for a richer, more scalable frontend experience.
- **Fine-Tuned Models:** Integrating domain-specific LLMs for specialized industries (legal, medical, academic).

14. Result / Conclusion

This project successfully demonstrates the development of a **robust, AI-driven, multi-modal chatbot** capable of understanding and responding to natural language queries, processing and analyzing both structured and unstructured documents, and extracting text from images using OCR. By integrating both **local inference (Ollama)** and **cloud-based reasoning (DeepSeek)**, it provides flexibility, scalability, and privacy-conscious execution. The system serves as a strong foundation for building advanced document intelligence solutions, digital assistants, and workflow automation tools. With further enhancements such as summarization, database integration, and multi-language support, this project can be scaled to support **enterprise-level use cases**, intelligent search, and data-driven decision-making.

GitHub Repository: 🔗 https://github.com/Rachel-0864/Tech-Chatbot.git