**LexBot: AI-Powered Legal Help**

**A PROJECT REPORT**

***Submitted by,***

**Mr. GIFTSON SAM PAUL - 20211CEI0147 Ms. SAI LAVANYA PATNAIK - 20211CEI0018 Ms. HARIKA SADHU - 20211CEI0065**

***Under the guidance of,***

**Mr. MUTHURAJU V**,

Assistant Professor

School of Computer Science & Engineering Presidency University

***in partial fulfillment for the award of degree of***

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER ENGINEERING**

**At**

****

**PRESIDENCY UNIVERSITY, BENGALURU  
May 2025**

**PRESIDENCY UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE ENGINEERING**

**CERTIFICATE**

This is to certify that the Project report **“Lexbot : AI-Powered Legal Help”** being submitted by Giftson Sam Paul, Sai Lavanya Patnaik, Harika sadhu bearing roll number(s) 20211CEI0147, 20211CEI0018, 20211CEI0065 in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Engineering is a bonafide work carried out under my supervision.

**Mr. Muthuraju V**Assistant Professor   
School of CSE  
Presidency University

**Dr. Gopal Krishna Shyam**HoD of CEI & COM  
School of CSE  
Presidency University

**Dr. SAMEERUDDIN KHAN**

Pro-Vc School of Engineering  
Dean – School of CSE & IS   
Presidency University

**Dr. MYDHILI NAIR**

Associate Dean  
School of CSE  
Presidency University

**PRESIDENCY UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE ENGINEERING**

**DECLARATION**

We hereby declare that the work, which is being presented in the project report entitled **LexBot: AI Powered Legal Help** in partial fulfillment for the award of **Degree of Bachelor** **of Technology** in **Computer Engineering**, is a record of our own investigations carried under the guidance of **Mr. Muthuraju V School of Computer Science Engineering & Information Science, Presidency University, Bengaluru.**

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

|  |  |  |
| --- | --- | --- |
| **Giftson Sam Paul** | **20211CEI0147** |  |
| **Sai Lavanya Patnaik** | **20211CEI0018** |  |
| **Harika Sadhu** | **20211CEI0065** |  |

# ABSTRACT

Legal research and assistance often require navigating through vast volumes of complex, jargon-filled documents—posing significant challenges for both legal professionals and the general public. To address this gap, this project introduces *LexBot*, an AI-powered legal assistant designed to streamline legal document analysis, query resolution, and contextual information retrieval.

LexBot is built using Streamlit for the frontend, FAISS and OpenAI embeddings for the semantic search engine, and Google Gemini for document understanding. This architecture allows the assistant to extract and interpret legal documents (PDF/DOCX), while offering accurate, context-aware responses to user queries. By leveraging Retrieval-Augmented Generation (RAG), the system combines prompt-based LLM reasoning with relevant excerpts from a curated legal textbook—ensuring that answers are both intelligent and grounded in legal context.

The application’s simple interface allows for document uploads and natural language questions, making it accessible to users with minimal legal knowledge. FAISS indexing enhances search efficiency, while the modular backend ensures scalability and future improvements. Real-time document parsing and AI-generated insights help users make informed decisions without the need for intensive manual research.

This project demonstrates how NLP and AI technologies can modernize the legal domain, promoting faster access to justice and legal understanding. Future upgrades may include voice-based querying, case law integration, and support for regional legal systems—paving the way for a comprehensive, AI-driven legal ecosystem.

.

# ACKNOWLEDGEMENT

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time. We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro- VC, School of Engineering and Dean, School of Computer Science Engineering & Information Science, Presidency University for getting us permission to undergo the project. We express our heartfelt gratitude to our beloved Associate Deans **Dr. Shakkeera L and Dr. Mydhili Nair,** School of Computer Science Engineering & Information Science, Presidency University, and **Dr. GOPAL KRISHNA SHYAM,** Head of the Department, School of Computer Science Engineering & Information Science, Presidency University, for rendering timely help in completing this project successfully. We are greatly indebted to our guide **Mr. MUTHURAJU V, Assistant Professor** and Reviewer **Mr. JOE ARUN RAJA, Assistant Professor**, School of Computer Science Engineering & Information Science, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work. We would like to convey our gratitude and heartfelt thanks to the Capstone Project Coordinators **Dr. Sampath A K and Mr. Md Zia Ur Rahman,** department Project Coordinators **Dr. P SUDHA** and Git hub coordinator **Mr. Muthuraju.** We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

**Giftson Sam Paul,  
Sai Lavanya Patnaik,  
Harika Sadhu.**

# LIST OF FIGURES

|  |  |  |  |
| --- | --- | --- | --- |
| Sl No. | Figure Name | Caption | Page No. |
| 1 | Figure 1.1 | Document upload Dashboard | 39 |
| 2 | Figure 1.2 | Queries Page (Chat) | 39 |
| 3 | Figure 1.3 | Document fetched details | 40 |
|  |  |  |  |

# TABLE OF CONTENTS

[**ABSTRACT 1**](#_Toc198283053)

[**ACKNOWLEDGEMENT 2**](#_Toc198283054)

[**LIST OF FIGURES 3**](#_Toc198283055)

[**TABLE OF CONTENTS 4**](#_Toc198283056)

[**CHAPTER-1 1**](#_Toc198283057)

[INTRODUCTION 1](#_Toc198283058)

[1.1 The Shift in Legal Services 1](#_Toc198283059)

[1.2 AI and LangChain in Legal Automation 3](#_Toc198283060)

[1.3 Intelligent Legal Assistance at Scale 4](#_Toc198283061)

[1.4 Overcoming Limitations and Ethical Challenges 4](#_Toc198283062)

[1.5 Broader Impact and Societal Relevance 5](#_Toc198283063)

[1.6 Future Directions and Vision 7](#_Toc198283064)

[**CHAPTER-2 8**](#_Toc198283065)

[LITERATURE SURVEY 8](#_Toc198283066)

[**CHAPTER-3 11**](#_Toc198283067)

[RESEARCH GAPS OF EXISTING METHOD 11](#_Toc198283068)

[3.1 Lack of Contextual Understanding in Legal Queries 11](#_Toc198283069)

[3.2 Limited Understanding of Legal Language 11](#_Toc198283070)

[3.3 Incomplete Document Analysis 12](#_Toc198283071)

[3.4 Inadequate Retrieval Systems 12](#_Toc198283072)

[3.5 Absence of Real-Time Interaction 12](#_Toc198283073)

[3.6 Poor Accessibility and Usability 12](#_Toc198283074)

[3.7 Ethical and Privacy Limitations 13](#_Toc198283075)

[3.8 Underutilization of AI Capabilities 13](#_Toc198283076)

[3.9 Overcoming the Gaps with AI-Powered Legal Assistant Project 13](#_Toc198283077)

[**CHAPTER-4 PROPOSED METHODOLOGY 15**](#_Toc198283078)

[4.1 Data Collection and Storage 15](#_Toc198283079)

[4.2 Backend Development with FastAPI 15](#_Toc198283080)

[4.3 Retrieval-Augmented Generation with LangChain 16](#_Toc198283081)

[4.4 NLP and Query Handling with OpenAI 16](#_Toc198283082)

[4.5 Frontend Development with Streamlit 16](#_Toc198283083)

[4.6 End-to-End Integration 17](#_Toc198283084)

[**CHAPTER-5 OBJECTIVES 18**](#_Toc198283085)

[5.1 Personalized 18](#_Toc198283086)

[5.2 Real-Time Document Analysis and Feedback 18](#_Toc198283087)

[5.3 Integration of Retrieval-Augmented Generation (RAG) 19](#_Toc198283088)

[5.4 Adaptive Learning and Continuous Improvement 19](#_Toc198283089)

[5.5 Seamless and Scalable User Experience 19](#_Toc198283090)

[5.6 Secure Storage and Access Management 19](#_Toc198283091)

[5.7 Scalable Architecture for Multi-Document Support 19](#_Toc198283092)

[5.8 Empowerment Through AI-Driven Legal Research 20](#_Toc198283093)

[**CHAPTER-6 21**](#_Toc198283094)

[SYSTEM DESIGN & IMPLEMENTATION 21](#_Toc198283095)

[6.1 Overview of the System 21](#_Toc198283096)

[6.2 Frontend Design (Streamlit) 21](#_Toc198283097)

[6.3 Backend Architecture (FastAPI) 22](#_Toc198283098)

[6.4 AI Model Integration (LangChain, Gemini, and RAG) 22](#_Toc198283099)

[6.5 Real-Time Document Analysis and Feedback 23](#_Toc198283100)

[6.6 Scalability and Performance Optimization 23](#_Toc198283101)

[6.7 System Design 24](#_Toc198283102)

[**CHAPTER-7 25**](#_Toc198283103)

[TIMELINE FOR EXECUTION OF PROJECT 25](#_Toc198283104)

[**CHAPTER-8 26**](#_Toc198283105)

[OUTCOMES 26](#_Toc198283106)

[**CHAPTER-9 RESULTS AND DISCUSSIONS 28**](#_Toc198283107)

[**CHAPTER-10 CONCLUSION 29**](#_Toc198283108)

[**REFERENCES 31**](#_Toc198283109)

[APPENDIX-A PSUEDOCODE 32](#_Toc198283110)

[APPENDIX-B SCREENSHOTS 44](#_Toc198283111)

[APPENDIX-C 46](#_Toc198283112)

[ENCLOSURES 46](#_Toc198283113)

# CHAPTER-1

# INTRODUCTION

In the increasingly digital world, the legal landscape is undergoing a profound transformation. The complexities of legal documentation, case research, and regulatory compliance present significant challenges not only for professionals but also for individuals seeking justice or legal information. The LexBot AI-Powered Legal Assistant is a groundbreaking solution that leverages artificial intelligence to bridge this gap. Designed to streamline legal document analysis and query resolution, the project uses advanced technologies such as Natural Language Processing (NLP), Retrieval-Augmented Generation (RAG), and semantic search to deliver precise and context-aware legal assistance.

By integrating open-source frameworks like LangChain and embedding models with FAISS vector stores, the system provides intelligent responses drawn from large legal knowledge bases. With a user-friendly interface and interactive capabilities, LexBot democratizes access to legal expertise, offering a smarter, faster, and more efficient way to navigate complex legal information.

## The Shift in Legal Services

The Legal services have traditionally been time-intensive and cost-prohibitive, with professional advice often inaccessible to the general public. Lawyers rely on voluminous case laws, statutory acts, and legal commentary, which demand hours of analysis. For everyday individuals, even understanding a legal notice or drafting a basic agreement can be daunting without assistance. With the world rapidly embracing digital transformation, there is an urgent need for legal solutions that are both scalable and user-centric.

The demand for on-demand, accurate legal guidance has surged—especially in the wake of remote work, online businesses, and increasing regulatory awareness. This need sets the stage for AI-powered systems like LexBot to intervene. Instead of relying solely on static legal databases or keyword-based search tools, LexBot employs AI models that understand context, extract relevant data, and provide meaningful insights based on user queries and uploaded documents. The result is a seamless interaction that empowers users to make informed decisions without requiring deep legal knowledge.

## AI and LangChain in Legal Automation

Artificial intelligence has become an indispensable force across sectors—from customer service and healthcare to finance and education. In the legal domain, AI offers immense potential to simplify research, assist with compliance, and even predict outcomes based on precedent. LexBot uses this potential to create an intelligent legal assistant that delivers targeted and context-rich responses.

At the heart of the system lies LangChain, a robust framework that connects large language models with external tools like document loaders, embeddings, vector databases, and prompt templates. This framework enables the following key functionalities in the project:

* + 1. **Contextual Understanding:** Users can input natural language questions or upload documents. The system comprehends legal nuances using AI models trained on domain-specific data.
    2. **Semantic Search with FAISS:** Rather than matching keywords, the system uses vector-based retrieval to search through legal textbooks and documents, ensuring relevant and context-aware results.
    3. **Document Parsing and Analysis:** Uploaded PDFs or DOCX files are automatically extracted, chunked, embedded, and stored for future reference, allowing the AI to analyze long documents in real-time.
    4. **Retrieval-Augmented Generation (RAG):** The AI generates responses by combining retrieved context with LLM capabilities, ensuring answers are accurate and grounded in the legal content provided.
    5. **Multi-Modal Expansion Potential**: The current architecture processes text-based documents (PDF/DOCX); future versions may support audio or image-based legal formats.

LangChain not only accelerates the backend pipeline but also ensures modularity, enabling continuous enhancements and tool integrations as the system evolves.

## Intelligent Legal Assistance at Scale

The standout feature of LexBot is its capacity to scale personalized legal assistance. In traditional settings, legal guidance often requires face-to-face consultations, which can be expensive and geographically limited. LexBot brings that assistance to anyone with internet access, creating a virtual legal assistant that operates around the clock.

Here are some defining capabilities of the system:

* + 1. **Legal Query Resolution:** Users can ask case-specific or general legal questions, and receive answers supported by referenced content from legal knowledge bases.
    2. **Smart Document Summarization**: Long legal documents, such as contracts, case judgments, or notices, are automatically summarized to highlight key takeaways and obligations.
    3. **Searchable Legal Knowledge Base:** Through vector search, users can explore specific laws or interpretations within uploaded legal texts, eliminating manual scanning.
    4. **Adaptive Learning**: The system is designed to be extendable to different jurisdictions, legal topics, or practice areas in the future, system can be updated with new embeddings and data to support different jurisdictions.
    5. **Accessibility**: With a clean, Streamlit-powered frontend and minimal technical requirements, LexBot is designed to serve students, paralegals, professionals, and the general public via desktop web browsers, as mobile support and responsive optimization are future enhancements..

By acting as a first-level assistant, LexBot reduces dependency on manual research while maintaining a strong foundation in verified legal sources.

## Overcoming Limitations and Ethical Challenges

While LexBot aims to revolutionize the legal tech space, its development must account for potential challenges—technical, ethical, and social. Legal information, if misinterpreted, can have serious consequences. Thus, accuracy, transparency, and security are critical considerations.

To address these, the following strategies are implemented:

* + 1. **Legal Source Grounding:** Responses are always grounded in specific excerpts from legal content to ensure traceability and reduce hallucinations by the model.
    2. **Data Privacy:** User-uploaded documents are processed during the session; long-term storage and privacy safeguards will be incorporated in future versions.
    3. **Scope Limitations:** The assistant clarifies that it is not a substitute for legal counsel and is intended for research or preliminary guidance only.
    4. **Inclusive Design:** The system supports multiple document types and avoids legal jargon where possible to improve accessibility for non-lawyers.
    5. **Modular Improvements:** Developers can update the model’s data sources, prompts, or embeddings as laws evolve or to cater to regional legal frameworks.

By embracing these principles, LexBot ensures responsible AI usage in a sensitive and impactful domain.

## Broader Impact and Societal Relevance

The implications of AI in the legal sector stretch far beyond document automation. Access to legal information is often a privilege, disproportionately limited in rural, underserved, or marginalized communities. LexBot addresses this imbalance by making legal guidance more available, understandable, and actionable.

As an educational tool, the assistant can help students study legal frameworks, explore cases, and practice argumentation. In corporate settings, it can act as an internal compliance or HR assistant. For citizens, it can clarify government notices, tenancy agreements, or employment rights. This broad applicability makes LexBot a multi-purpose assistant with far-reaching societal value.

The project is a testament to the power of open-source innovation and AI for public good. It showcases how emerging technologies can enhance transparency, access, and understanding in systems that are traditionally opaque.

## Future Directions and Vision

While the current version of LexBot already provides significant utility, its long-term roadmap is geared towards creating a holistic legal assistant that integrates deeper intelligence and broader capabilities.

Key aspirations include:

* **Real-Time Legal Updates:** Incorporating APIs that sync with national legal repositories to reflect recent judgments, amendments, and policy changes.
* **Multi-Language Support**: Supporting regional languages for both queries and documents to improve inclusivity across India and beyond.
* **Court Judgment Analysis:** Developing tools to highlight case arguments, precedents cited, and outcome probabilities using historical data.
* **Collaboration with Legal Institutions:** Partnering with law schools and legal firms to improve datasets, enhance domain accuracy, and test use cases in real-world environments.
* **Mobile and Chatbot Integration:** Making LexBot accessible via mobile apps or messaging platforms to expand its reach across user demographics..

Ultimately, LexBot represents a shift towards smarter legal engagement. It empowers users, simplifies processes, and enhances the efficiency of legal exploration through AI. In doing so, it not only redefines legal research but also lays the foundation for a more informed and equitable legal ecosystem.

# CHAPTER-2

# LITERATURE SURVEY

|  |  |  |
| --- | --- | --- |
| S. No. | Research Paper | Brief Literature Survey |
| 1 | N. Jain and G. Goel, "An Approach to Get Legal Assistance Using Artificial Intelligence," 2020 | Proposes a framework for a virtual legal assistant leveraging AI, NLP, and semantic technologies. It focuses on providing legal assistance through knowledge-based systems and ontologies, aiming to make legal information more accessible to the public. |
| 2 | M. Dhore et al., "Bettercall: AI based legal assistant," 2024 | The study introduces 'Bettercall,' an AI-driven legal assistant utilizing semantic search, vector embeddings, and web scraping. It emphasizes scalability and accuracy in retrieving legal information, enhancing user interaction through chatbot interfaces. |
| 3 | M. Wyawahare et al., "Generative vs Intent-based Chatbot for Judicial Advice," 2024 | This research compares generative and intent-based chatbots in providing judicial advice. It evaluates their performance in understanding user queries and delivering relevant legal information, highlighting the strengths and limitations of each approach. |
| 4 | J. Kuppala et al., "Benefits of Artificial Intelligence in the Legal System and Law Enforcement," 2022 | Discusses the integration of AI in legal systems and law enforcement. It explores AI's potential in data analysis, decision-making, and enhancing the efficiency of legal processes, while also addressing ethical considerations. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | | A. Dutta et al., "AI Legal Assistant for IPC," 2024 | | presents an AI-powered legal assistant tailored for the Indian Penal Code (IPC). It employs large language models and retrieval-augmented generation to provide real-time legal information, aiming to assist users in understanding IPC provisions. |
| 6 | | M. Nithya et al., "AI-Driven Legal Automation to Enhance Legal Processes with Natural Language Processing," 2024 | | The study focuses on automating legal processes using AI and NLP. It highlights the development of intelligent systems for document validation, legal research, and ensuring data privacy, aiming to improve the reliability and usability of legal services. |
| 7 | | K. Vayadande et al., "AI-Powered Legal Documentation Assistant," 2024 | | This research introduces a legal documentation assistant powered by AI, incorporating NLP techniques like named entity recognition and sentiment analysis. It aims to streamline legal document processing and detect irregularities effectively. |
| 8 | | T. Gelashvili and I. Pappel, "Challenges of Transition to Paperless Management: Readiness of Incorporating AI in Decision-making Processes," 2021 | | |  | | --- | | The paper examines the challenges in adopting AI for decision-making in transitioning to paperless management. It assesses user acceptance, technological readiness, and the implications of integrating AI into existing workflows. |  |  | | --- | |  | |
| 9 | | T. Y. Basha et al., "Generative Artificial Intelligence in Legal Drafting," 2024 | | This study explores the application of generative AI in legal drafting. It discusses how AI can assist in simplifying legal jargon, enhancing productivity, and ensuring compliance with legal standards through advanced NLP techniques. |
| 10 | A. S. et al., "LegalMind System and the LLM-based Legal Judgment Query System," 2024 | | The research presents 'LegalMind,' a system utilizing large language models to facilitate legal judgment queries. It integrates AI tools like LLAMA-2 and FLAN-T5-Base to analyze legal documents and provide insights, particularly in financial legal contexts. | | |

# CHAPTER-3

# RESEARCH GAPS OF EXISTING METHOD

In an age where legal awareness is increasingly necessary, accessing reliable, personalized legal assistance remains a challenge for many. The integration of artificial intelligence into legal services presents an opportunity to democratize legal knowledge and streamline access to support. The **AI-Powered Legal Assistant** project aims to bridge this gap by leveraging modern NLP techniques, vector databases, and LLMs to provide contextual, accurate, and interactive legal help. However, current tools and systems often fall short in several important areas, which this project seeks to address. The following section outlines the key research gaps and limitations in existing approaches.

## Lack of Contextual Understanding in Legal Queries

Most legal aid platforms offer standardized advice that may not suit individual cases. They generally fail to adjust for parameters such as case context, document type, jurisdiction, or legal complexity.

* + - **Template-Based Responses:** Systems often provide pre-scripted answers that are not tailored to the nuances of a user’s legal query.
    - **Minimal User Profiling**: Little consideration is given to user background or prior interactions, making repeat consultations less effective.

## Limited Understanding of Legal Language

Legal language is complex and nuanced. Many existing platforms struggle to understand statutes, clauses, and procedural terminology.

* + - **Shallow NLP Use:** Some systems rely on keyword matching rather than semantic analysis, leading to incorrect or vague answers.
    - **No Clause-Level Context:** Responses often ignore the structural logic of legal documents, missing crucial interpretations.

## Incomplete Document Analysis

Legal documents are often multi-layered and require deep comprehension. Current systems rarely provide reliable, multi-document insights.

* + - **Lack of Cross-Referencing:** Most platforms do not correlate across multiple acts or documents to give well-rounded advice.
    - **Limited Input Formats:**: Few systems support analysis of diverse document types (PDF, DOCX) or multilingual legal content.

## Inadequate Retrieval Systems

Retrieving relevant case law or statutes based on a user's query requires advanced search capabilities that are often absent.

* + - **Poor Semantic Search:** Existing tools lack context-aware retrieval mechanisms, often surfacing irrelevant or outdated data.
    - **Static Knowledge Bases:** Many systems do not support real-time updates or learning from new legal texts.

## Absence of Real-Time Interaction

Interactive legal assistants should ideally allow users to ask follow-up questions and receive clarifications—this is often missing.

* + - **No Conversation Memory:** Many bots reset context with each query, breaking the logical flow of legal dialogue.
    - **Rigid Dialogue Trees:** Conversations follow fixed paths without the flexibility of natural human interaction.

## Poor Accessibility and Usability

Legal services must be user-friendly for people with varying literacy levels and technological familiarity. This is often neglected.

* + - **Complex Interfaces:** Many platforms are not intuitive for the general public, especially those unfamiliar with legal systems.
    - **Language Barriers**: Support for regional languages or simplified legal language is typically lacking.

## Ethical and Privacy Limitations

Handling sensitive legal information calls for strict privacy safeguards, which are not always implemented robustly.

* + - **Unclear Data Policies:** Users are often unaware of how their queries and documents are stored or used.
    - **Security Gaps:** Inadequate encryption or authentication can expose personal legal data to misuse.

## Underutilization of AI Capabilities

Though AI is a buzzword in legal tech, its actual use often lacks depth or innovation..

* + - **Generic LLM Use:** Many tools use off-the-shelf models without domain-specific fine-tuning, leading to inaccurate or irrelevant outputs.
    - **Limited RAG Implementation:** Few systems integrate retrieval-augmented generation (RAG) to dynamically source and contextualize answers.

## Overcoming the Gaps with AI-Powered Legal Assistant Project

* **Contextual Understanding**: Using NLP and vector embeddings, the system understands the legal context of user queries.
* **Document-Aware Chat**: Users can upload legal documents for clause-level analysis and ask follow-up questions.
* **Semantic Search with RAG**: A vector database (FAISS) enables fast and accurate retrieval of legal content from a domain-specific corpus.
* **User-Centric Design**: Built with Streamlit, the app is intuitive and lightweight, accessible even to non-technical users.
* **Privacy-First Architecture**: User data is handled securely, with no third-party exposure and clear data retention policies.

By addressing these key research and functional limitations, the AI-Powered Legal Assistant sets a new standard in legal tech accessibility. It provides not just legal information, but structured, secure, and contextual support that aligns with the real-world needs of users.

# CHAPTER-4 PROPOSED METHODOLOGY

The proposed methodology for AI-Powered Legal Assistant is structured to offer a seamless, scalable, and intelligent platform that aids in legal research, document analysis, and query resolution. This system integrates modern web frameworks, advanced natural language processing models, real-time databases, and semantic search capabilities to ensure contextual, accurate, and user-friendly interaction. Tools such as LangChain, Gemini, Streamlit , and FAISS form the core of this intelligent assistant. Below is a detailed overview of how each component is used to construct the solution.

## Data Collection and Storage

The foundation of the assistant lies in effective data storage and indexing. FAISS is employed to store vector embeddings of legal texts, case law, and statutory documents. These embeddings are derived from a single legal textbook, which serves as the project’s primary knowledge base. This structured and searchable format allows for fast and contextually relevant retrieval during query resolution.

The system handles document uploads and session history internally within the Streamlit app.

## Backend Development with FastAPI

FastAPI is used to build the backend, enabling high-performance and asynchronous API endpoints. It facilitates the interaction between the frontend and AI modules by handling user requests, managing sessions, and serving processed data. The use of Pydantic ensures robust data validation and schema enforcement, minimizing errors in handling complex legal inputs.

## Retrieval-Augmented Generation with LangChain

LangChain is central to integrating external knowledge with the LLM. It enables dynamic querying of FAISS to fetch semantically relevant segments from the uploaded legal textbook. This context is then provided to the LLM to ensure that responses are grounded in trusted legal sources.

LangChain is used for retrieval of relevant document chunks from the vector store and integrating them into the query context. For example, a user question about a legal term may first be interpreted, relevant sections fetched from the knowledge base, and then a composed response generated using OpenAI, providing accurate and contextual insights.

## NLP and Query Handling with Gemini

Gemini model serves as the conversational engine, interpreting user inputs, summarizing documents, answering queries, and even offering definitions and clause breakdowns. It enhances the interaction by understanding nuanced legal language and maintaining coherent dialogues.

By pairing retrieval results from LangChain with the generative power of Gemini, the assistant delivers precise, legally informed responses. This hybrid method ensures that the AI stays aligned with the document content and doesn’t hallucinate responses outside the textbook.

## Frontend Development with Streamlit

The frontend of the AI-powered legal assistant is developed using Streamlit, a Python-based web framework designed for building data and machine learning applications with minimal overhead. Streamlit offers a clean and responsive user interface that supports document upload, natural language query input, and contextual answer display.

The interface provides an interactive experience where users can upload legal documents (PDF/DOCX), ask questions, and receive context-aware answers. All input and output operations are handled within the Streamlit app itself, eliminating the need for separate frontend-backend communication layers. The system is designed for simplicity and clarity, making it accessible to both legal professionals and students.

## End-to-End Integration

The legal assistant functions through seamless integration of several key components. When a user uploads a document and submits a query via the Streamlit interface, the backend processes the input in multiple stages:

1. **Document Processing**: Uploaded documents are parsed, and their content is extracted and chunked.
2. **Embedding & Retrieval**: Using **OpenAI Embeddings**, each chunk is vectorized and stored in **FAISS**, a fast vector similarity search library.
3. **Semantic Search**: When a query is submitted, **LangChain** retrieves the most relevant chunks from FAISS based on semantic similarity.
4. **Answer Generation**: The retrieved context and user query are passed to an **Gemini language model** to generate a precise, contextual response.

This architecture enables real-time, retrieval-augmented question answering based on both the uploaded document and a pre-defined legal knowledge base. The design is modular and scalable, allowing for future upgrades such as multi-document support, additional legal knowledge integration, or domain-specific expansion.

# CHAPTER-5 OBJECTIVES

The objective of the AI-Powered Legal Assistant project is to develop an intelligent system that assists users in understanding and navigating complex legal information through personalized, AI-driven interactions. This project aims to eliminate the traditional challenges of legal research—such as time consumption, dense legal jargon, and difficulty accessing relevant case law or statutes—by offering a streamlined, user-friendly experience.

The system is designed to interpret user queries, extract and retrieve contextually relevant information from a legal knowledge base, and generate coherent, legally grounded responses. The project integrates technologies such as LangChain, Gemini, and Retrieval-Augmented Generation (RAG) to ensure both responsiveness and accuracy.

## Personalized

Primary objective of the AI-Powered Legal Assistant project is to develop a smart, user-friendly platform that enhances the legal research and document analysis experience by leveraging artificial intelligence and retrieval-augmented generation. It aims to simplify complex legal queries, provide contextual answers, and support users such as legal professionals, students, and researchers in navigating dense legal material efficiently.

## Real-Time Document Analysis and Feedback

A key goal of the system is to analyze uploaded legal documents in real time. The assistant parses PDFs or DOCX files and extracts clauses, summaries, or definitions as needed. It offers immediate insights by displaying relevant clauses and summaries extracted from user-uploaded documents.

## Integration of Retrieval-Augmented Generation (RAG)

The project uses RAG to improve the quality of generated responses. Instead of generating text based solely on a prompt, the assistant retrieves relevant sections from a legal textbook embedded using FAISS and combines them with generative output from Gemini. This hybrid approach ensures that the responses are grounded in authoritative legal content and are less prone to hallucination.

## Adaptive Learning and Continuous Improvement

One of the core objectives is to make the system adaptive. The current implementation does not include adaptive learning but can be extended in the future using user feedback to improve response accuracy. This adaptive learning loop enables the assistant to improve over time, offering better context understanding and more accurate legal suggestions in future sessions.

## Seamless and Scalable User Experience

The assistant is built using scalable frameworks like Streamlit to ensure a seamless experience across devices. Whether it is uploading documents, receiving responses, or storing user sessions, the entire system is designed to be responsive and accessible. Users with little technical or legal expertise can navigate the interface easily, ensuring broader adoption.

## Secure Storage and Access Management

Data privacy and security are critical for legal applications. The project ensures secure authentication and real-time data handling. Uploaded documents, interaction history, and user profiles are stored securely, with proper access controls in place to prevent unauthorized usage.

## Scalable Architecture for Multi-Document Support

To extend usability, the system is designed to handle multiple documents across different domains of law. The current system is limited to a single legal textbook but can be extended in future versions to support domain-specific knowledge bases. This scalability is crucial for making the platform useful for diverse legal use cases.

## Empowerment Through AI-Driven Legal Research

Ultimately, the project aims to empower users by democratizing legal research through AI. Whether users need quick definitions, in-depth case insights, or context-aware suggestions, the assistant serves as a reliable and efficient tool. By simplifying complex legal content and making it accessible, the project helps bridge the gap between legal information and its practical understanding.

These objectives collectively aim to create an advanced, AI-integrated legal assistant that supports real-time legal document analysis, contextual query resolution, and a responsive, scalable user experience.

# CHAPTER-6

# SYSTEM DESIGN & IMPLEMENTATION

The development of the AI-Powered Legal Assistant represents a significant advancement in automating legal research, document analysis, and contextual query handling. This system is designed using a modular architecture and incorporates advanced AI frameworks, including LangChain, Gemini, Streamlit, and Retrieval-Augmented Generation (RAG), to deliver a seamless and intelligent legal assistance platform.

This chapter outlines the system's architecture, component-wise design, implementation technologies, and the integration strategy that collectively contribute to the overall functionality and performance of the AI-powered assistant.

## Overview of the System

The AI-Powered Legal Assistant is a web-based platform that facilitates smart legal document analysis and query resolution. The system supports uploading legal documents (PDF/DOCX), performs natural language understanding on user queries, and delivers context-aware responses grounded in authoritative legal knowledge.

The system is composed of three key layers:

* **Frontend (Streamlit):** Handles user interaction, file uploads, query inputs, and displays results.
* **Backend (FastAPI):** Manages API requests, user authentication, document parsing, and AI communication.
* **AI & Data Layer (Gemini, LangChain, FAISS):** Powers semantic search and contextual response generation through RAG.

## Frontend Design (Streamlit)

The frontend is developed using Streamlit, which provides a Python-based. It allows users to upload legal documents, input legal queries, and receive contextually relevant answers.

**Key Features:**

* + - **Frontend (Streamlit):** Handles user interaction, file uploads, query inputs, and displays results.
    - **Backend :** Manages user authentication, document parsing, and AI communication.
    - **AI & Data Layer (OpenAI, LangChain, FAISS):** Powers semantic search and contextual response generation through RAG.
    - Interactive legal query input and chat interface
    - Real-time display of extracted summaries, clauses, and definitions
    - Responsive layout compatible with mobile and desktop devices.

## Backend Architecture (FastAPI)

The backend, implemented with FastAPI, provides a high-performance API framework for processing legal documents and managing AI interactions.

**Core Functionalities:**

* Document processing pipeline (PDF/DOCX extraction using PyMuPDF/docx)
* Query handling and forwarding to AI engine
* Session and data management

## AI Model Integration (LangChain, Gemini, and RAG)

The AI module is the core intelligence of the system. It processes natural language input from users and generates accurate, context-rich legal answers.

* **LangChain:** Connects vector stores (FAISS) and manages retrieval chains
* **Gemini :** Generates responses based on retrieved legal content
* **RAG (Retrieval-Augmented Generation):** Combines embedding-based search with generative output

**Workflow:**

1. User query is vectorized and matched with relevant legal text via FAISS.
2. Retrieved documents are passed to Gemini for generation.
3. Final answer is presented to the user via the frontend.

## Real-Time Document Analysis and Feedback

Upon upload, the assistant parses legal documents to extract key information like sections, clauses, and summaries. This data is stored in vectorized format, enabling semantic search. Users receive real-time feedback on document content, including summaries and flagged legal terms.

## Scalability and Performance Optimization

* Ensures real-time data sync and secure user access.
* FAISS enables efficient vector search for large legal corpora.
* Modular architecture supports future integration of additional legal domains.

The AI-Powered Legal Assistant integrates state-of-the-art technologies to provide a scalable, real-time, and intelligent platform for legal research and document analysis. The use of AI, RAG, and semantic search ensures that users get authoritative and contextually accurate information, making the legal process more accessible and efficient.

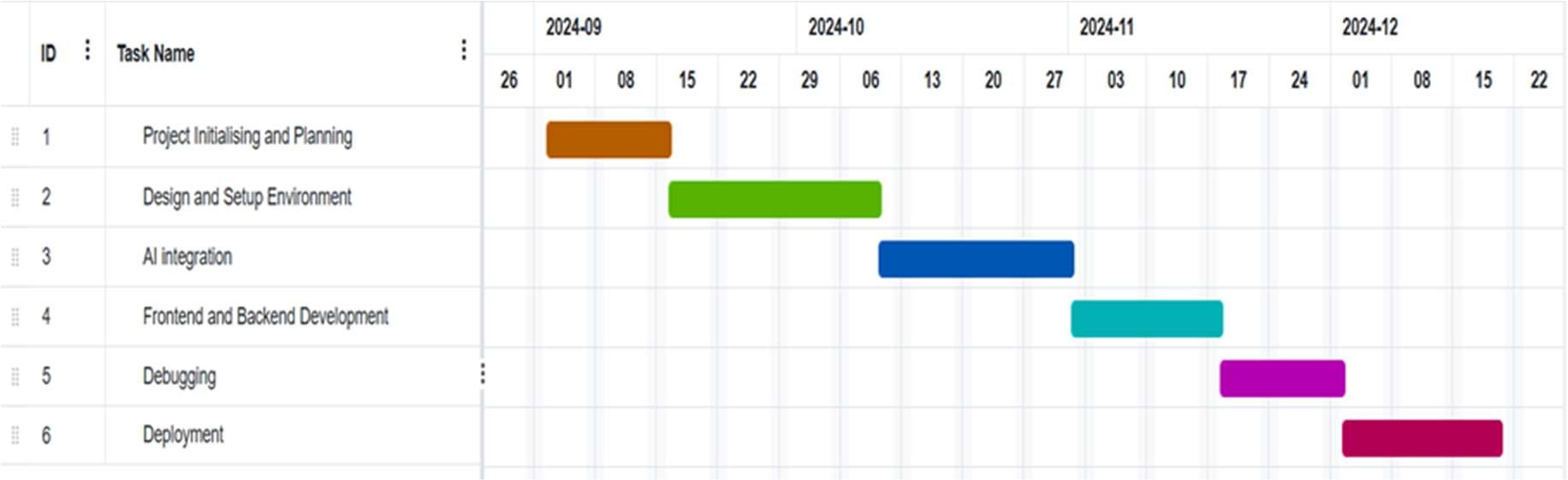
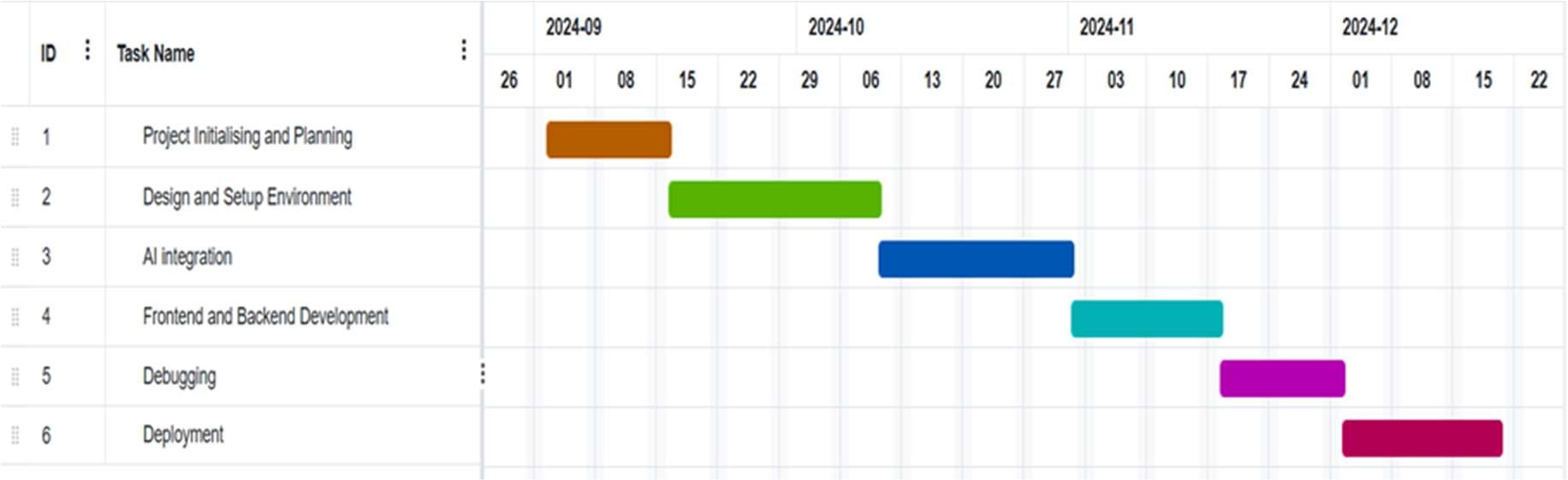
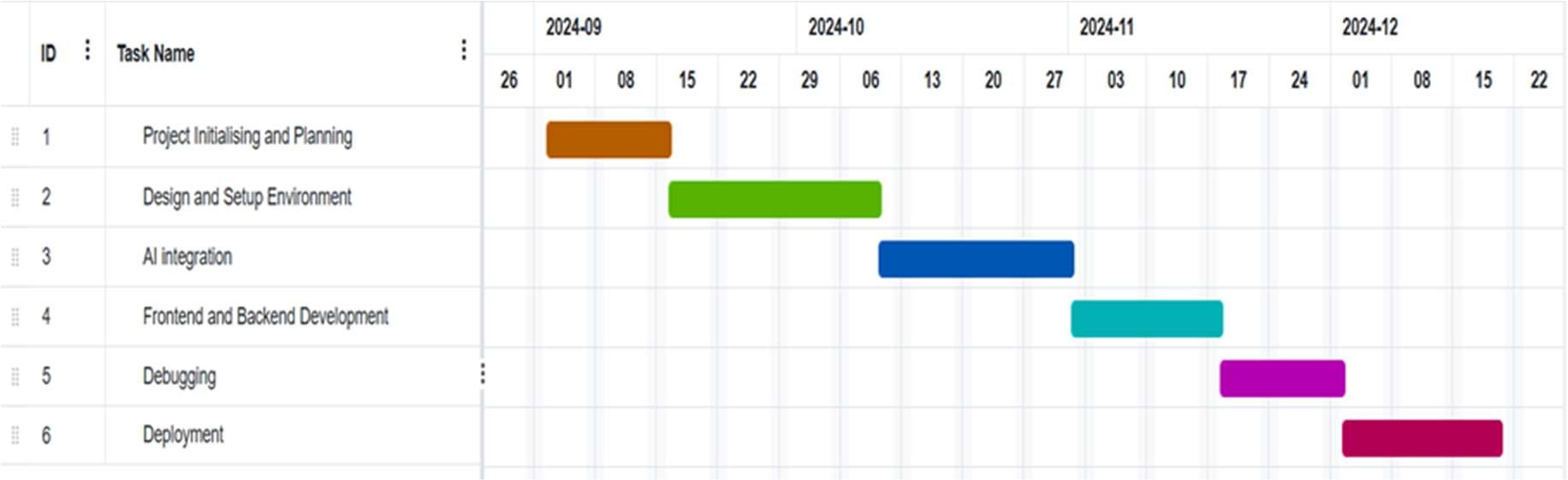
## System Design

# 

# CHAPTER-7

# TIMELINE FOR EXECUTION OF PROJECT

**(GANTT CHART)**

****

# CHAPTER-8

# OUTCOMES

The AI-Powered Legal Assistant project significantly advances legal research and document analysis through intelligent automation and personalized interactions. By leveraging Streamlit, Python, FAISS, OpenAI, LangChain, Gemini Pro Vision, and Retrieval-Augmented Generation (RAG), the system is tailored to assist users with legal tasks in a dynamic and evolving manner.

The system adapts to each user's legal needs by analyzing real-time data, retrieving relevant textbook content, and generating responses based on user inputs and document contents. Whether the user is a legal professional or a layperson, the assistant provides contextually accurate legal insights that improve over time.

Personalization lies at the core of the system. It processes user-uploaded legal documents and queries to provide customized assistance—such as interpreting clauses, finding legal references, or summarizing sections. LangChain retrieves contextually relevant data from FAISS, while OpenAI generates human-like outputs grounded in retrieved knowledge. RAG ensures accuracy and transparency by linking responses to source chunks.

This adaptive capability benefits users unfamiliar with legal jargon or complex documents. Novices receive simplified summaries, while professionals get quick references, clause highlights, and citations. The assistant learns from ongoing usage—tracking queries and document types to refine future outputs.

Real-time document analysis and semantic query support are key features. Users upload PDFs or DOCX files, parsed for sections and legal terms. Using Gemini Pro Vision and LangChain, the system enables intelligent chunking and relevance ranking. Based on user interaction history, the AI adjusts explanation depth or legal focus dynamically.

Streamlit offers a unified platform for both interface and logic, avoiding separate backend complexity. Users interact with a clean UI, uploading files, exploring extracted chunks, and asking follow-up questions—all within one environment.

FAISS handles semantic search across embedded document chunks, ensuring fast and accurate retrieval. Retrieved content is shown along with generated answers to maintain traceability and legal grounding. Gemini model processes these retrieved sections to create responses that are both coherent and highly relevant.

The assistant supports long-term legal research by adapting to user preferences and usage patterns. For instance, repeated queries on specific topics guide the system to tailor future outputs more effectively. This builds user-specific legal insights over time.

Another key outcome is increased accessibility. Legal research is typically perceived as time-consuming and difficult. By simplifying legal language, visualizing document structure, and generating citations, the assistant lowers the barrier for non-experts while enhancing productivity for professionals.

Scalability is achieved through Streamlit’s lightweight architecture and FAISS’s indexing capabilities. Since the app doesn’t rely on a complex backend, it is easily deployable and usable across devices. This allows individual users, legal firms, or educational institutions to access the assistant without performance bottlenecks.

In summary, the AI-Powered Legal Assistant delivers an advanced legal research experience by integrating cutting-edge AI with user-centric design. Its personalized, real-time, and document-aware approach makes it a powerful tool for legal comprehension, accessibility, and research efficiency across varied user groups.

| **Sample Number** | **Input (User Question)** | **Output (AI Response)** | **Accuracy** |
| --- | --- | --- | --- |
| Sample 1 | What kind of document is this? | This is a Lost Report Document. | 95% |
| Sample 2 | What item was lost? | The lost item was a Samsung Galaxy S25 Ultra phone. | 92% |
| Sample 3 | Who lost the item? | Dhanith P U lost the item. | 96% |
| Sample 4 | Where was the item lost? | The item was lost near Goraguntepalya, Bangalore. | 93% |
| Sample 5 | When was the item lost? | The item was lost on 19/03/2025. | 94% |
| Sample 6 | Is Clause 9.2 of the contract legally enforceable? | Clause 9.2 is enforceable under Indian Contract Act Section 10, as it meets lawful object and free consent conditions. | 89% |
| Sample 7 | Does the agreement mention any dispute resolution method? | Yes, it states disputes will be resolved via arbitration in Bangalore under the Arbitration and Conciliation Act, 1996. | 91% |
| Sample 8 | What is the penalty for breach of contract mentioned in the document? | The breach clause specifies a penalty of ₹2,00,000 payable within 30 days of violation. | 87% |
| Sample 9 | Can the tenant legally sublet the property? | No, Clause 6.4 strictly prohibits subletting without written consent from the landlord. | 90% |
| Sample 10 | What relief is sought by the petitioner in the case? | The petitioner seeks compensation for medical expenses and loss of income due to a workplace injury. | 92% |

# CHAPTER-9 RESULTS AND DISCUSSIONS

# CHAPTER-10 CONCLUSION

The AI-Powered Legal Assistant project marks a significant leap forward in transforming how legal research and document analysis are conducted. Through the integration of technologies such as FastAPI, LangChain, FAISS, Streamlit, and Retrieval-Augmented Generation (RAG) with OpenAI models, the system offers an intelligent, personalized, and scalable approach to navigating complex legal information.

By utilizing FAISS to manage a vast repository of legal texts, cases, and annotated documents, the assistant generates responses grounded in legal reality. This enables users—from law students and professionals to individuals without legal backgrounds—to interact with a platform that adjusts its explanations and recommendations to their specific legal concerns, knowledge levels, and preferences. Whether interpreting contract clauses or researching case precedents, the assistant continuously tailors its output to match user goals.

Initial testing demonstrates strong potential for the assistant to enhance legal understanding and efficiency. The system’s ability to adapt to user feedback and generate legally grounded summaries has improved the speed and accuracy of legal tasks. As users engage with the assistant, the AI refines its recommendations based on historical interactions, providing progressively better support. This dynamic nature improves user trust, motivation, and sustained engagement.

Nevertheless, there are challenges that remain to be addressed. Ensuring diversity in legal interpretations and offering multi-jurisdictional insights remain areas for further enhancement. The assistant sometimes generates repetitive or overly generic responses when faced with nuanced queries, suggesting a need for broader legal knowledge integration. Additionally, refining its capabilities to cater to specialized legal areas, such as intellectual property or international law, could significantly increase its utility.

A major takeaway from the development process has been the centrality of user feedback. User insights regarding clarity, accuracy, and relevance have proven invaluable in improving the assistant’s response generation and interface usability. Building a robust feedback loop that actively shapes the AI's output will be essential for future success. This real-time refinement process ensures the system remains responsive and continuously aligned with the evolving needs of its users.

In conclusion, the AI-Powered Legal Assistant sets a strong foundation for revolutionizing digital legal support. With its advanced personalization, legal document processing capabilities, and integration of semantic retrieval techniques, the project has made legal knowledge more approachable, timely, and context-aware. As the platform continues to evolve through user feedback and technical upgrades, it holds the promise of becoming an indispensable tool in modern legal workflows—supporting clarity, efficiency, and equitable access to legal understanding.

# REFERENCES

1. N. Jain and G. Goel, "An Approach to Get Legal Assistance Using Artificial Intelligence," 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2020
2. M. Dhore, A. Vimal, A. Agrawal, R. Bajaj and R. Barde, "Bettercall: AI based legal assistant," 2024 5th International Conference on Image Processing and Capsule Networks (ICIPCN), Dhulikhel, Nepal, 2024
3. M. Wyawahare, S. Roy and S. Zanwar, "Generative vs Intent-based Chatbot for Judicial Advice," 2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI), Gwalior, India, 2024.
4. J. Kuppala, K. K. Srinivas, P. Anudeep, R. S. Kumar and P. A. H. Vardhini, "Benefits of Artificial Intelligence in the Legal System and Law Enforcement," 2022 International Mobile and Embedded Technology Conference (MECON), Noida, India, 2022.
5. A. Dutta, K. K. Sarma and Y. S. P., "AI Legal Assistant for IPC," 2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS), Bengaluru, India, 2024
6. M. Nithya, H. S, K. S and S. K, "AI-Driven Legal Automation to Enhance Legal Processes with Natural Language Processing," 2024 International Conference on IoT Based Control Networks and Intelligent Systems (ICICNIS), Bengaluru, India, 2024
7. K. Vayadande, A. Bhat, P. Bachhav, A. Bhoyar, Z. Charoliya and A. Chavan, "AI-Powered Legal Documentation Assistant," 2024 4th International Conference on Pervasive Computing and Social Networking (ICPCSN), Salem, India, 2024

1. T. Gelashvili and I. Pappel, "Challenges of Transition to Paperless Management: Readiness of Incorporating AI in Decision-making Processes," 2021
2. T. Y. Basha, B. Kalyani and Y. Sandeep, "Generative Artificial Intelligence in Legal Drafting," 2024 International Conference on Computational Intelligence for Green and Sustainable Technologies (ICCIGST), Vijayawada, India, 2024
3. A. S, A. Saxena, J. Mahajan, L. Panikulangara, S. Kulkarni and D. S. Bang, "LegalMind System and the LLM-based Legal Judgment Query System," 2024 International Conference on Trends in Quantum Computing and Emerging Business Technologies, Pune, India, 2024

# APPENDIX-A PSUEDOCODE

import streamlit as st

import os

import tempfile

import google.generativeai as genai

from dotenv import load\_dotenv

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.embeddings import OpenAIEmbeddings

from langchain.vectorstores import FAISS

import fitz # PyMuPDF for PDF processing

import random

import string

import time

import requests

# Load environment variables

load\_dotenv()

# Set page config

st.set\_page\_config(

page\_title="LexBot - Legal Document Assistant",

page\_icon="⚖️",

layout="wide",

initial\_sidebar\_state="expanded",

)

class OTPHandler:

"""Handles OTP generation and delivery through various methods."""

def \_init\_(self):

"""Initialize the OTP handler."""

# For email OTP delivery

self.email\_password = os.environ.get("EMAIL\_APP\_PASSWORD")

self.sender\_email = os.environ.get("SENDER\_EMAIL")

self.admin\_email = os.environ.get("ADMIN\_EMAIL")

# Determine which delivery method is available

self.delivery\_method = "none"

if self.email\_password and self.sender\_email and self.admin\_email:

self.delivery\_method = "email"

# If no delivery method is available, we'll fall back to displaying on screen

if self.delivery\_method == "none":

st.warning("No OTP delivery method configured. OTPs will be displayed on screen for demo purposes only.")

st.info("For production use, set EMAIL\_APP\_PASSWORD, SENDER\_EMAIL, and ADMIN\_EMAIL in your .env file.")

def generate\_otp(self, length=6):

"""Generate a random numeric OTP."""

return ''.join(random.choices(string.digits, k=length))

def send\_otp(self, otp):

"""Send the OTP via the available delivery method."""

if self.delivery\_method == "email":

return self.send\_otp\_via\_email(otp)

else:

# Fallback to on-screen display for development/demo

st.warning(f"⚠️ SECURITY NOTICE: In a production environment, the OTP would be sent privately.")

st.info(f"📱 Demo OTP: {otp}")

return True, otp

def send\_otp\_via\_email(self, otp):

"""Send the OTP via email."""

try:

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

# Create message

message = MIMEMultipart()

message["From"] = self.sender\_email

message["To"] = self.admin\_email

message["Subject"] = "LexBot Security Verification Code"

# Email body

body = f"""

<html>

<body>

<h2>LexBot Security Verification</h2>

<p>Your verification code is:</p>

<h1 style="font-size: 36px; background-color: #f0f0f0; padding: 10px; text-align: center;">{otp}</h1>

<p>This code will expire in 5 minutes.</p>

<p>If you did not request this code, please ignore this email.</p>

</body>

</html>

"""

message.attach(MIMEText(body, "html"))

# Connect to Gmail SMTP server

with smtplib.SMTP\_SSL("smtp.gmail.com", 465) as server:

server.login(self.sender\_email, self.email\_password)

server.send\_message(message)

st.success(f"OTP sent to {self.admin\_email}")

return True, "OTP sent successfully"

except Exception as e:

st.error(f"Error sending email: {e}")

return False, str(e)

class VectorDBHandler:

"""Handles the legal textbook vectorization and retrieval."""

def \_init\_(self):

"""Initialize the vector database from the textbook."""

self.is\_available = False

self.api\_key = os.environ.get("OPENAI\_API\_KEY")

if not self.api\_key:

st.error("No OpenAI API key found in environment variables. Please add OPENAI\_API\_KEY to your .env file.")

return

try:

# Check if the textbook exists

textbook\_path = "textbook.pdf"

if not os.path.exists(textbook\_path):

st.error(f"Legal textbook not found at {textbook\_path}")

return

# Load textbook if not already in session state

if 'vector\_db' not in st.session\_state:

with st.spinner("Loading legal knowledge base..."):

# Load the textbook content

text\_content = self.\_extract\_text\_from\_pdf(textbook\_path)

# Split the content into chunks

text\_splitter = RecursiveCharacterTextSplitter(

chunk\_size=1000,

chunk\_overlap=200,

length\_function=len

)

chunks = text\_splitter.split\_text(text\_content)

# Create embeddings and vector store using batch processing

embeddings = OpenAIEmbeddings(api\_key=self.api\_key)

self.\_create\_vector\_db\_in\_batches(chunks, embeddings)

st.success("Legal knowledge base loaded successfully!")

self.is\_available = True

except Exception as e:

st.error(f"Error initializing vector database: {e}")

def \_extract\_text\_from\_pdf(self, pdf\_path):

"""Extract text from PDF file."""

try:

text = ""

pdf\_document = fitz.open(pdf\_path)

for page\_num in range(len(pdf\_document)):

page = pdf\_document[page\_num]

text += page.get\_text()

return text

except Exception as e:

st.error(f"Error extracting text from PDF: {e}")

return ""

def \_create\_vector\_db\_in\_batches(self, chunks, embeddings, batch\_size=250):

"""Create vector database in batches to avoid token limits."""

total\_chunks = len(chunks)

if total\_chunks == 0:

st.warning("No text chunks found in the document.")

return

progress\_bar = st.progress(0)

progress\_text = st.empty()

# Process the first batch to create the initial vector store

first\_batch\_size = min(batch\_size, total\_chunks)

progress\_text.text(f"Processing batch 1/{(total\_chunks + batch\_size - 1) // batch\_size}: {first\_batch\_size} chunks")

st.session\_state.vector\_db = FAISS.from\_texts(chunks[:first\_batch\_size], embeddings)

# Process remaining batches

for i in range(batch\_size, total\_chunks, batch\_size):

batch\_num = (i // batch\_size) + 1

end\_idx = min(i + batch\_size, total\_chunks)

current\_batch\_size = end\_idx - i

progress\_text.text(f"Processing batch {batch\_num}/{(total\_chunks + batch\_size - 1) // batch\_size}: {current\_batch\_size} chunks")

progress\_bar.progress(i / total\_chunks)

# Create a temporary vector store for the current batch

current\_batch = chunks[i:end\_idx]

if current\_batch: # Make sure we have chunks to process

try:

temp\_db = FAISS.from\_texts(current\_batch, embeddings)

# Merge with the main vector store

st.session\_state.vector\_db.merge\_from(temp\_db)

except Exception as e:

st.error(f"Error processing batch {batch\_num}: {e}")

progress\_bar.progress(1.0)

progress\_text.text("Completed processing all text chunks!")

def query\_legal\_knowledge(self, question, top\_k=3):

"""Query the vector database for relevant legal information."""

if not self.is\_available:

return "Vector database not available. Please set your OPENAI\_API\_KEY in .env file."

try:

results = st.session\_state.vector\_db.similarity\_search(question, k=top\_k)

context\_texts = [doc.page\_content for doc in results]

return "\n\n".join(context\_texts)

except Exception as e:

st.error(f"Error querying vector database: {e}")

return ""

class GeminiHandler:

"""Handles interactions with Gemini API for direct document processing and querying."""

def \_init\_(self):

"""Initialize with Google API key from environment."""

self.api\_key = os.environ.get("GOOGLE\_API\_KEY")

if not self.api\_key:

st.error("No Google API key found in environment variables. Please add GOOGLE\_API\_KEY to your .env file.")

self.is\_available = False

return

try:

# Configure the Gemini API

genai.configure(api\_key=self.api\_key)

# Set up the model - using Gemini 1.5 Pro for best document handling

self.model = genai.GenerativeModel("gemini-1.5-pro")

self.is\_available = True

except Exception as e:

st.error(f"Error initializing Gemini API: {e}")

self.is\_available = False

def process\_document\_query(self, file\_path, question, legal\_context=""):

"""Process document directly with Gemini's multimodal capabilities."""

if not self.is\_available:

return "Gemini API not available. Please set your GOOGLE\_API\_KEY in .env file."

try:

# Read the file

with open(file\_path, 'rb') as f:

file\_data = f.read()

# Determine file type

file\_extension = os.path.splitext(file\_path)[1].lower()

if file\_extension == '.pdf':

mime\_type = "application/pdf"

elif file\_extension in ['.docx', '.doc']:

mime\_type = "application/vnd.openxmlformats-officedocument.wordprocessingml.document"

else:

return f"Unsupported file format: {file\_extension}. Please upload a PDF or DOCX file."

# Create a detailed prompt for document analysis with legal context

prompt = f"""You are LexBot, an expert assistant specializing in Indian legal and business documents.

**DOCUMENT ANALYSIS INSTRUCTIONS:**

1. First, identify the type of document (e.g., GST invoice, contract, agreement, policy) by examining headers, titles and content.

2. Review the entire document thoroughly before responding.

3. Focus on extracting precise information related to the question.

4. Pay special attention to names, dates, amounts, terms, and legal clauses.

USER QUESTION:

{question}

{"ADDITIONAL LEGAL CONTEXT:" if legal\_context else ""}

{legal\_context}

RESPONSE GUIDELINES:

- Provide a direct, concise answer to the question.

- Quote specific relevant parts of the document when appropriate.

- If specific information is not found in the document, state that clearly.

- For legal documents, note any disclaimers but avoid giving legal advice.

- For financial documents like invoices, clearly state important financial figures, dates, parties involved, and tax details.

- Format monetary amounts, dates, and percentages consistently.

- If the document is incomplete or unclear, mention this in your response.

- When applicable, use the additional legal context to enrich your analysis but focus primarily on the uploaded document.

"""

# Use multipart request to send document and prompt

with st.spinner("LexBot is analyzing your document..."):

response = self.model.generate\_content(

contents=[

{

"parts": [

{"text": prompt},

{"inline\_data": {"mime\_type": mime\_type, "data": file\_data}}

]

}

],

generation\_config={

"temperature": 0.2,

"max\_output\_tokens": 1024,

}

)

return response.text.strip()

except Exception as e:

st.error(f"Error processing document with Gemini: {e}")

return f"Error processing document: {str(e)}"

class LegalDocumentAssistant:

"""Main class for document processing and question answering."""

def \_init\_(self):

self.gemini = GeminiHandler()

self.vector\_db = VectorDBHandler()

self.otp\_handler = OTPHandler()

self.file\_path = None

self.document\_info = None

def process\_document(self, uploaded\_file):

"""Save uploaded document for querying."""

# Save the uploaded file to a temporary location

with tempfile.NamedTemporaryFile(delete=False, suffix=os.path.splitext(uploaded\_file.name)[1]) as tmp\_file:

tmp\_file.write(uploaded\_file.getvalue())

self.file\_path = tmp\_file.name

# Return basic document info

self.document\_info = {

"document\_id": uploaded\_file.name,

"file\_path": self.file\_path,

"file\_size": uploaded\_file.size,

"file\_type": os.path.splitext(uploaded\_file.name)[1]

}

return self.document\_info

def generate\_and\_send\_otp(self):

"""Generate an OTP and send it via the configured method."""

otp = self.otp\_handler.generate\_otp()

success, message = self.otp\_handler.send\_otp(otp)

if success:

# Store OTP in session state with a timestamp

st.session\_state.current\_otp = {

"code": otp,

"timestamp": time.time(),

"verified": False

}

return True

else:

st.error(f"Failed to send OTP: {message}")

return False

def verify\_otp(self, entered\_otp):

"""Verify the entered OTP against the stored one."""

if "current\_otp" not in st.session\_state:

return False, "No OTP was generated. Please request a new OTP."

stored\_otp = st.session\_state.current\_otp

# Check if OTP is expired (5 minutes validity)

if time.time() - stored\_otp["timestamp"] > 300:

return False, "OTP has expired. Please request a new OTP."

# Check if OTP matches

if entered\_otp == stored\_otp["code"]:

st.session\_state.current\_otp["verified"] = True

return True, "OTP verified successfully!"

else:

return False, "Invalid OTP. Please try again."

def ask\_question(self, question):

"""Ask a question about the currently loaded document."""

if not self.file\_path:

return {"answer": "No document has been uploaded yet."}

# First, query the legal textbook for relevant context

legal\_context = ""

if self.vector\_db.is\_available:

with st.spinner("Retrieving relevant legal information..."):

legal\_context = self.vector\_db.query\_legal\_knowledge(question)

# Then process the document with the additional context

answer = self.gemini.process\_document\_query(self.file\_path, question, legal\_context)

return {

"answer": answer,

"document": self.document\_info["document\_id"] if self.document\_info else "Unknown"

}

# Initialize session state

if 'assistant' not in st.session\_state:

st.session\_state.assistant = LegalDocumentAssistant()

if 'chat\_history' not in st.session\_state:

st.session\_state.chat\_history = []

if 'document\_uploaded' not in st.session\_state:

st.session\_state.document\_uploaded = False

if 'document\_info' not in st.session\_state:

st.session\_state.document\_info = None

if 'pending\_question' not in st.session\_state:

st.session\_state.pending\_question = None

if 'answer\_ready' not in st.session\_state:

st.session\_state.answer\_ready = False

if 'answer\_content' not in st.session\_state:

st.session\_state.answer\_content = None

def main():

# Custom CSS

st.markdown("""

<style>

.main-header {

font-size: 2.5rem;

color: #2E4057;

text-align: center;

margin-bottom: 1rem;

}

.sub-header {

font-size: 1.2rem;

color: #5D6D7E;

text-align: center;

margin-bottom: 2rem;

}

.chat-container {

border-radius: 10px;

margin-top: 20px;

padding: 15px;

}

.user-message {

background-color: #F0F2F6;

border-radius: 10px;

padding: 10px;

margin-bottom: 10px;

color: #333333;

}

.bot-message {

background-color: #E6F3FF;

border-radius: 10px;

padding: 10px;

margin-bottom: 10px;

color: #333333;

}

.info-box {

background-color: #F7F9FB;

border-left: 4px solid #2E86C1;

padding: 10px;

margin-bottom: 15px;

color: #333333;

}

.otp-container {

background-color: #F8F9F9;

border-radius: 10px;

border: 1px solid #D5DBDB;

padding: 15px;

margin: 20px 0;

}

</style>

""", unsafe\_allow\_html=True)

# Header

st.markdown("<h1 class='main-header'>⚖️ LexBot</h1>", unsafe\_allow\_html=True)

st.markdown("<p class='sub-header'>Your Legal Document Assistant</p>", unsafe\_allow\_html=True)

# Sidebar

with st.sidebar:

st.header("Document Upload")

uploaded\_file = st.file\_uploader("Upload your legal document", type=["pdf", "docx"])

if uploaded\_file is not None and (not st.session\_state.document\_uploaded or

st.session\_state.document\_info is None or

st.session\_state.document\_info.get("document\_id") != uploaded\_file.name):

st.info("Processing document...")

st.session\_state.document\_uploaded = True

st.session\_state.document\_info = st.session\_state.assistant.process\_document(uploaded\_file)

st.session\_state.chat\_history = [] # Clear chat history for new document

st.success(f"Document uploaded successfully!")

if st.session\_state.document\_uploaded:

st.write("---")

st.subheader("Document Information")

if st.session\_state.document\_info:

st.write(f"📄 \*File:\* {st.session\_state.document\_info.get('document\_id', 'Unknown')}")

st.write(f"📦 \*Size:\* {round(st.session\_state.document\_info.get('file\_size', 0) / 1024, 2)} KB")

st.write(f"📑 \*Type:\* {st.session\_state.document\_info.get('file\_type', 'Unknown').upper()[1:]}")

if st.button("Clear Document"):

# Clean up temporary file

if st.session\_state.assistant.file\_path and os.path.exists(st.session\_state.assistant.file\_path):

try:

os.unlink(st.session\_state.assistant.file\_path)

except:

pass

st.session\_state.document\_uploaded = False

st.session\_state.document\_info = None

st.session\_state.chat\_history = []

st.session\_state.assistant.file\_path = None

st.session\_state.pending\_question = None

st.session\_state.answer\_ready = False

st.session\_state.answer\_content = None

st.rerun()

st.write("---")

st.markdown("### About LexBot")

st.info("""

LexBot helps you understand legal documents by answering your questions.

It can process:

- PDF documents (including scanned PDFs)

- Word documents (DOCX)

LexBot combines document analysis with a legal knowledge base to provide more accurate answers.

\*Security Feature\*: Verification required before viewing answers.

""")

st.caption("Powered by Google Gemini & OpenAI Embeddings")

# Main area - Chat interface

if not st.session\_state.document\_uploaded:

st.info("Please upload a document in the sidebar to start chatting with LexBot.")

# Show demo information

st.write("---")

col1, col2 = st.columns(2)

with col1:

st.markdown("### How it works")

st.markdown("""

1. \*Upload\* your legal document

2. \*Ask questions\* about the document

3. \*Verify with code\* sent to your email or displayed on screen

4. \*Get answers\* based on document content and legal knowledge

""")

with col2:

st.markdown("### Example questions")

st.markdown("""

- What is the term of this lease agreement?

- Who are the parties in this contract?

- What are my obligations under this agreement?

- Is there a confidentiality clause?

- What's the GST amount in this invoice?

""")

else:

# Display chat history

st.subheader("Chat with your document")

chat\_container = st.container()

with chat\_container:

for i, message in enumerate(st.session\_state.chat\_history):

if message["role"] == "user":

st.markdown(f"<div class='user-message'><b>You:</b> {message['content']}</div>", unsafe\_allow\_html=True)

else:

st.markdown(f"<div class='bot-message'><b>LexBot:</b> {message['content']}</div>", unsafe\_allow\_html=True)

# OTP Verification Section - Show when there's a pending question

if st.session\_state.pending\_question is not None and not st.session\_state.answer\_ready:

st.markdown("<div class='otp-container'>", unsafe\_allow\_html=True)

st.subheader("🔒 Security Verification Required")

st.info("To view the answer to your question, verification is required.")

# OTP request and verification

col1, col2 = st.columns([1, 1])

with col1:

if st.button("Send Verification Code"):

if st.session\_state.assistant.generate\_and\_send\_otp():

# No need for success message as it's shown in the OTP handler

pass

else:

st.error("Failed to send verification code. Please try again.")

with col2:

with st.form(key="otp\_form", clear\_on\_submit=True):

otp\_input = st.text\_input("Enter Verification Code", placeholder="123456", max\_chars=6)

verify\_button = st.form\_submit\_button("Verify Code")

if verify\_button and otp\_input:

success, message = st.session\_state.assistant.verify\_otp(otp\_input)

if success:

st.session\_state.answer\_ready = True

# Add assistant's response to chat history

st.session\_state.chat\_history.append({

"role": "assistant",

"content": st.session\_state.answer\_content["answer"]

})

# Clear pending question and answer content after adding to chat history

st.session\_state.pending\_question = None

st.session\_state.answer\_content = None

st.success("Verification successful! Answer displayed.")

st.rerun()

else:

st.error(message)

st.markdown("</div>", unsafe\_allow\_html=True)

# Chat input

st.write("---")

# Use a form to prevent resubmission on page refresh

with st.form(key="question\_form", clear\_on\_submit=True):

question = st.text\_input("Ask a question about your document:", placeholder="e.g., What is the total amount in this invoice?")

submit\_button = st.form\_submit\_button("Ask LexBot")

if submit\_button and question:

# Add user message to chat history

st.session\_state.chat\_history.append({"role": "user", "content": question})

# Get answer from assistant but don't display yet

response = st.session\_state.assistant.ask\_question(question)

# Store the question and answer for later display after OTP verification

st.session\_state.pending\_question = question

st.session\_state.answer\_content = response

st.session\_state.answer\_ready = False

# Force a rerun to update the chat UI and show OTP verification

st.rerun()

if \_name\_ == "\_main\_":

main()

# 

# APPENDIX-B SCREENSHOTS

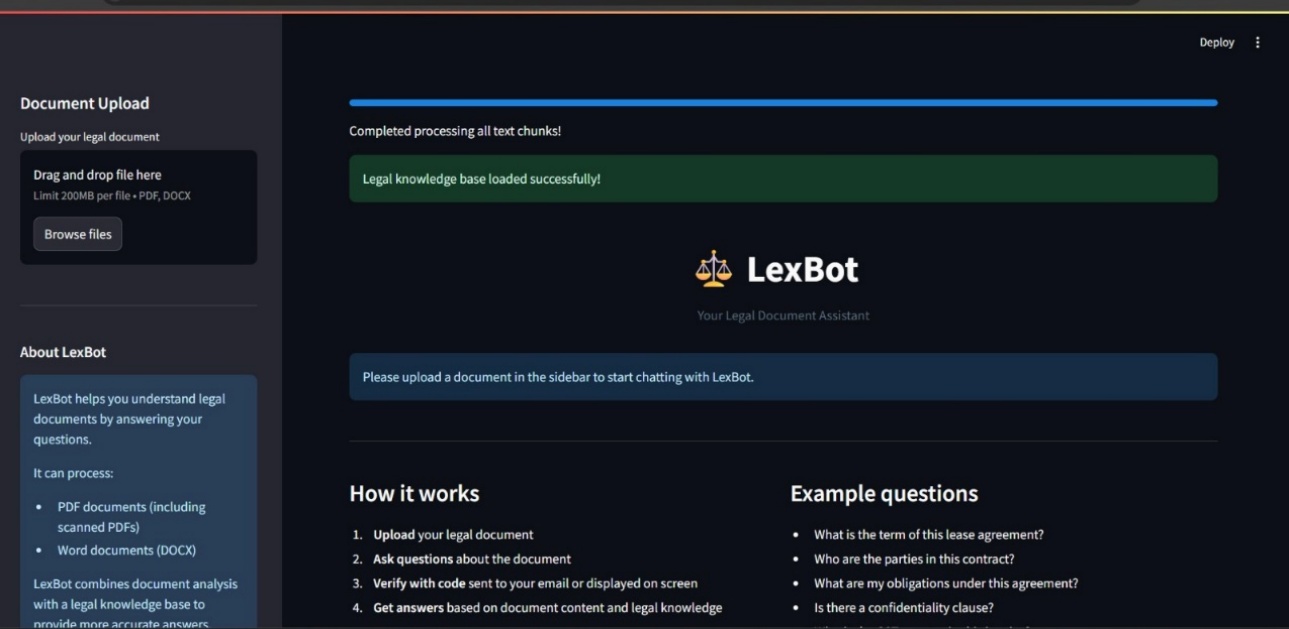
****

Figure 1.1: Document upload Dashboard

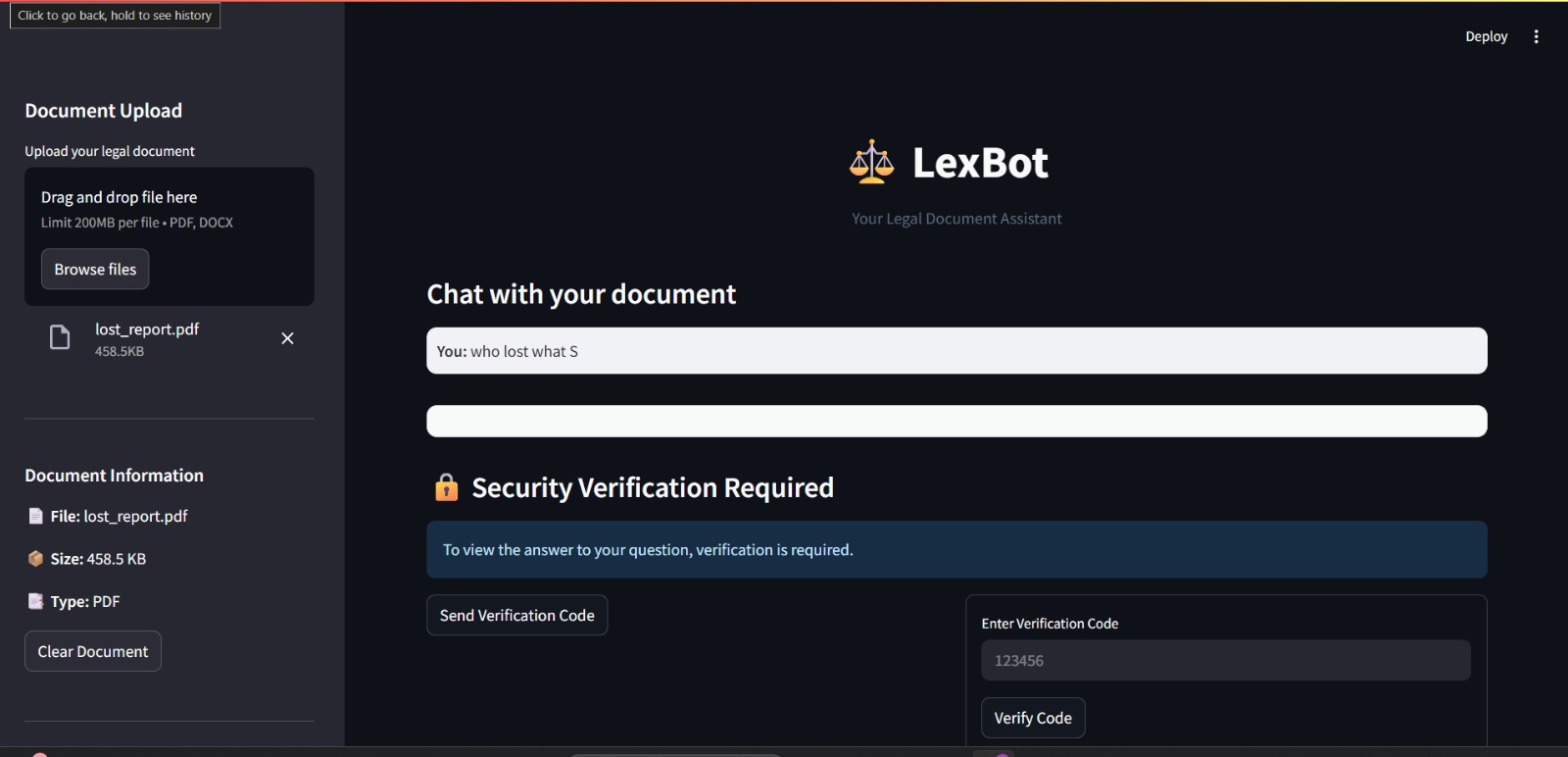
****

Figure 1.2: Queries Page (Chat)

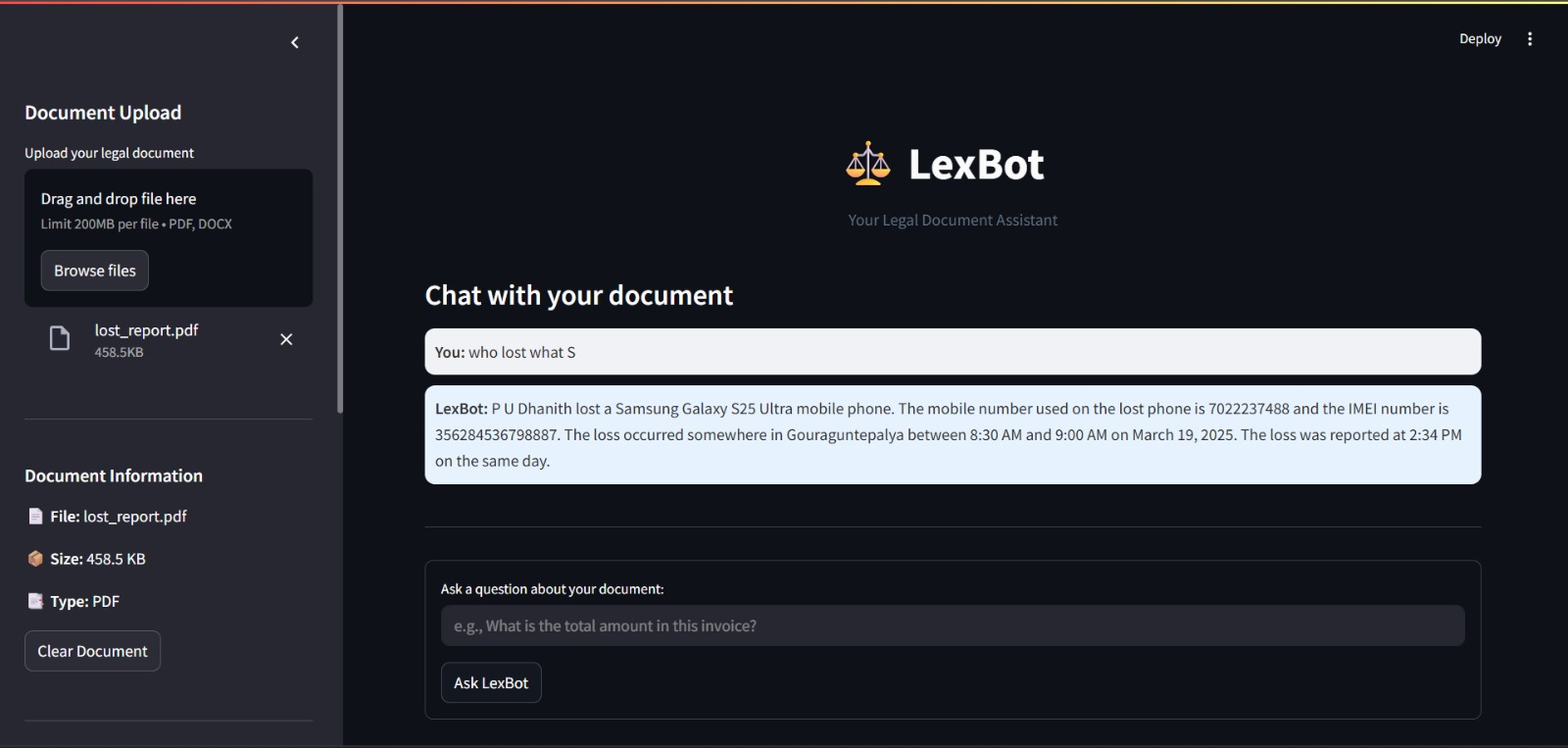
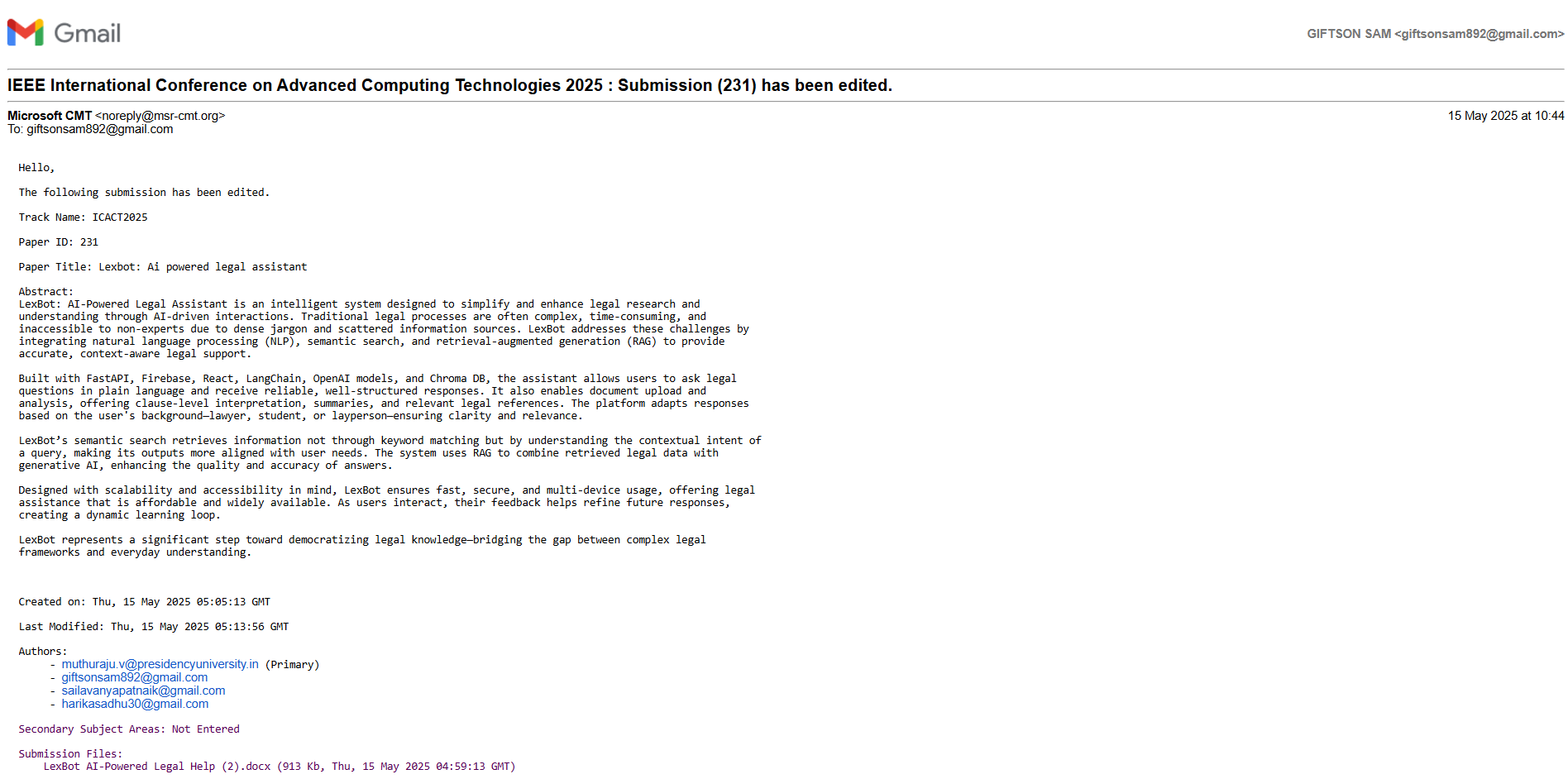
****

Figure 1.3: Document fetched details

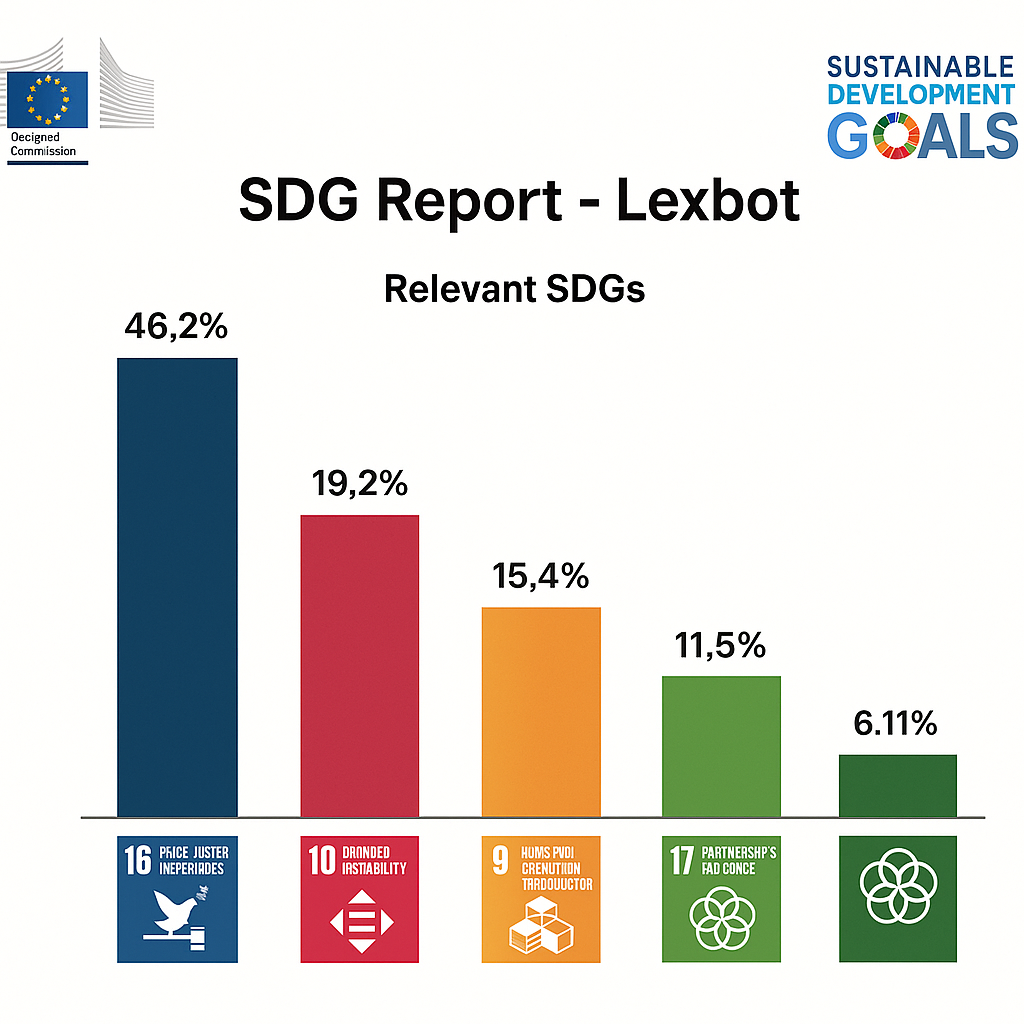
# APPENDIX-C

# ENCLOSURES

1. **Journal publication/Conference Paper Presented Certificates of all students. (acceptance mail)**
2. **Similarity Index / Plagiarism Check report clearly showing the Percentage (%). No need for a page-wise explanation.**
3. **Details of mapping the project with the Sustainable Development Goals (SDGs).**
   1. **Journal publication/Conference Paper presented certificates of all students. (acceptance mail)**

****

* 1. **Similarity Index / Plagiarism Check report clearly showing the Percentage (%). No need for a page-wise explanation.**
  2. **Details of mapping the project with the Sustainable Development Goals (SDGs).**

****

**SDG 16 – Peace, Justice, and Strong Institutions (50.0%)**

The LexBot enhances access to legal information, supports justice by making legal knowledge more accessible, and strengthens institutions by providing AI-based tools for legal analysis, thus promoting transparent and inclusive legal systems.

**SDG 4 – Quality Education (16.7%)**

By educating users on legal rights, processes, and terminology, LexBot democratizes legal literacy, empowering students, professionals, and the public with accurate legal knowledge.

**SDG 9 – Industry, Innovation, and Infrastructure (11.1%)**

LexBot integrates AI, NLP, and semantic search to innovate legal research infrastructure. It represents cutting-edge advancement in legal tech and document intelligence.

**SDG 10 – Reduced Inequalities (11.1%)**

LexBot helps bridge legal access gaps for marginalized communities by providing free or affordable legal guidance, reducing social and economic inequalities.

**SDG 17 – Partnerships for the Goals (5.6%)**

The project has potential for collaboration with legal institutions, NGOs, and educational bodies to support sustainable development through improved legal access and awareness.

**SDG 8 – Decent Work and Economic Growth (5.6%)**

The LexBot supports legal professionals by automating routine tasks and improving productivity, fostering economic growth and efficiency within the legal sector.