LegalBot: A WhatsApp-Based Legal Assistance System for Indian Law using LLM and RAG

A SOCIALLY RELEVANT MINI PROJECT REPORT

Submitted by

NITHYA SHREE C [211423104427]
POOJA RAVI [211423104455]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

OCTOBER 2025

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **"LegalBot: A WhatsApp-Based Legal Assistance System for Indian Law using LLM and RAG"** is the bonafide work of **" NITHYA SHREE C [211423104427] , POOJA RAVI [211423104455] "**who carried out the project work under my supervision.

Signature of the HOD with date

Dr.L.JABASHEELA M.E., Ph.D.,

Professor and Head,

Department of Computer Science and Engineering,
Panimalar Engineering College,
Chennai – 123.

Signature of the Supervisor with date

Dr.M.S. VINMATHI , M.E., Ph.D.,

Associate Professor,
Department of Computer Science and Engineering,
Panimalar Engineering College,
Chennai – 123.

Submitted for the 23CS1512 – Socially Relevant Mini Project Viva- Voice Examination held on .....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

We **“ NITHYA SHREE C [211423104427] , POOJA RAVI [211423104455] ”** hereby declare that this project report titled **“ LegalBot: A WhatsApp-Based Legal Assistance System for Indian Law using LLM and RAG ” ,** under the guidance of **Dr.M.S. VINMATHI , M.E., Ph.D.,** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

SIGNATURE OF THE STUDENTS

NITHYA SHREE C [211423104427]

POOJA RAVI [211423104455]

ACKNOWLEDGEMENT

Our profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D**., for his fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project

We want to express our deep gratitude to our Directors, **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D., and Dr. SARANYASREE SAKTHI KUMAR, B.E., M.B.A., Ph.D.,** for graciously affording us the essential resources and facilities for undertaking of this project.

Our gratitude is also extended to our Principal, **Dr. K. MANI, M.E., Ph.D.,** whose facilitation proved pivotal in the successful completion of this project.

We express our heartfelt thanks to **Dr. L. JABASHEELA, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of project.

We would like to express our sincere thanks to Project Coordinator **Mr. C. ELANGOVAN, M.Tech.,** and Project Guide **Dr.M.S. VINMATHI , M.E., Ph.D.,** and all the faculty members of the Department of CSE for their unwavering support for the successful completion of the project.

NITHYA SHREE C

POOJA RAVI

ABSTRACT

Due to high consultation fees, complicated legal jargon, and little public knowledge, access to legal aid in India is still restricted. By offering an AI-powered legal chatbot that is available via WhatsApp, a popular medium among citizens, LegalBot aims to close this gap. The system provides precise, context-aware legal advice in a conversational style by combining Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), and a vector database (Pinecone).

The chatbot obtains relevant legal information, produces trustworthy responses, and preserves chat history for continuity using OpenAI's GPT-4 via LangChain. Scalability, security, and low running costs are guaranteed by the use of FastAPI and Mangum in the deployment of the entire program on AWS Lambda.

LegalBot supports easily accessible, reasonably priced, and technologically advanced legal assistance that is in line with the Sustainable Development Goals of the UN (SDG 16: Peace, Justice, and Strong Institutions; SDG 9: Industry, Innovation, and Infrastructure; and SDG 10: Reduced Inequalities) through real-time WhatsApp interactions. The system serves as an example of how AI may modernise legal communication and provide accessibility to justice for all citizens.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LST OF ABBREVATIONS

AWS	Amazon Web Services
API	Application Programming Interface
LLM	Large Language Model
API	Application Programming Interface
NLP	Natural Language Processing
AI	Artificial Intelligence
RAG	Retrieval-Augmented Generation

INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

The legal system in India is vast and complex, containing numerous statutes, acts, and regulations that are often difficult for citizens to interpret without expert guidance. Access to legal consultation is further limited by high costs, time constraints, and lack of awareness—especially among people in rural and economically weaker sections. As a result, many individuals remain unaware of their legal rights and the procedures required to exercise them.

Advances in **Artificial Intelligence (AI)** and **Natural Language Processing (NLP)** have enabled intelligent systems to understand and generate human language with high accuracy. **Large Language Models (LLMs)** such as GPT-4 can process complex legal text and provide meaningful responses in conversational form. By combining LLMs with **Retrieval-Augmented Generation (RAG)** and vector databases such as **Pinecone**, it becomes possible to create domain-specific assistants capable of retrieving relevant legal context and delivering reliable, user-friendly answers.

LegalBot leverages these technologies to offer real-time legal assistance through **WhatsApp**, one of the most widely used communication platforms in India. The chatbot is built using **LangChain** and **FastAPI**, powered by **OpenAI GPT-4**, and deployed on **AWS Lambda** for scalability and cost efficiency. By enabling instant access to verified legal information, LegalBot supports **Sustainable Development Goals (SDG 16 – Peace, Justice and Strong Institutions; SDG 9 – Industry, Innovation and Infrastructure; and SDG 10 – Reduced Inequalities)**. The project demonstrates how AI can democratize access to justice and promote digital inclusion across society.

1.2 PROBLEM DEFINITION

In India, obtaining timely and affordable legal guidance remains a major challenge. Citizens often depend on manual consultations or scattered online resources that are difficult to understand without legal expertise. Existing digital portals lack conversational capability and contextual awareness, while professional legal advice is costly and not readily accessible to everyone.

Key issues include:

- Limited accessibility to qualified legal experts, particularly in rural and semi-urban regions.
- Language and technical barriers that prevent users from interpreting complex legal documents.
- Lack of centralized, AI-driven platforms capable of providing accurate, context-based legal guidance through commonly used communication tools.

To address these challenges, LegalBot provides an AI-powered, WhatsApp-integrated chatbot that delivers instant and contextually relevant legal information. By using LLM-based reasoning and RAG-based document retrieval, the system enables users to ask questions in natural language and receive precise, reliable responses drawn from authentic legal sources. This solution aims to make legal assistance more accessible, affordable, and understandable for all sections of society.

LITERATURE REVIEW

CHAPTER 2
LITERATURE REVIEW

Rivas-Echeverría et al. [1] introduced LegalBot-EC, a chatbot designed to provide legal assistance based on Ecuador's Comprehensive Organic Criminal Code (COIP) and the 2008 Constitution. The system uses ChromaDB for contextual retrieval and LLaMA 3.1 as its generative model. It employs BETO, a Spanish-adapted BERT model, for generating semantic embeddings of legal texts. The chatbot was evaluated through user satisfaction and accuracy assessments, achieving an average satisfaction score of 88.72/100 and a weighted accuracy of 96%. The system supports both Spanish and English queries, demonstrating multilingual capability.

Socatiyanurak et al. [2] developed LAW-U, an AI chatbot that provides legal guidance to survivors of sexual violence by recommending relevant Supreme Court decisions. The system uses spaCy for similarity scoring, TF- IDF for keyword extraction, and NLTK WordNet for synonym matching. It was trained on 182 Supreme Court cases and evaluated using stratified 5-fold cross-validation, achieving 88.89% accuracy in matching user inputs to relevant cases. The chatbot is integrated into the LINE messaging platform and is designed to be gender-neutral and inclusive.

Nikita et al. [3] proposed LAWBOT, a machine learning- based chatbot for Indian legal assistance. Built using the RASA framework, the system provides three main services: case document retrieval, lawyer information search, and legal FAQs. It uses Longformer embeddings and cosine similarity for document retrieval, and pattern matching for lawyer search. The system was implemented with a Telegram interface and uses web scraping to build lawyer and case databases.

Kalpana et al. [4] presented a general-purpose legal chatbot that offers legal advice and facilitates attorney consultations. The system uses the paraphrase-MiniLM-L6-v2 model for semantic similarity and cosine similarity for response retrieval. It

includes both user and admin modules, allowing users to schedule appointments with lawyers. The system is web-based and uses phpMyAdmin for database management.

Zivan Misquitta et al. [5] present the development of a legal chatbot designed to offer instant legal advice using text retrieval and generation techniques. The system employs embeddings and vector databases (e.g., FAISS) for efficient similarity search and leverages models like LlamaCpp for response generation. The authors emphasize scalability, user- friendliness, and the ability to handle diverse legal topics such as civil and criminal law. The study highlights the potential of such systems to bridge the gap between legal professionals and the public, though it also acknowledges limitations in handling complex, context-dependent legal scenarios.

Keerthana R et al. [6] investigate the ethical and legal challenges posed by AI chatbots, including bias, transparency, accountability, and data privacy. The authors propose a hybrid model combining GPT-4 and GPT-3.5 to validate responses and improve reliability. They stress the importance of human supervision, regulatory compliance, and public awareness to mitigate risks. The study concludes that while chatbots can enhance legal accessibility, their integration must be carefully managed to avoid undermining trust and justice.

Bhavika Pardhi et al. [7] introduce LEGALBOT, a chatbot aimed at providing affordable and accessible legal advice. The system includes features such as user authentication, lawyer directories, and an AI-powered Q&A interface. The paper underscores the importance of a comprehensive legal knowledge base and continuous updates to maintain accuracy. The authors also note that while chatbots are useful for basic inquiries, they cannot replace human lawyers for complex legal matters.

Marc Queudot et al. [8] describe the development of two FAQ-based chatbots for immigration and corporate legal queries. The study compares multiple NLP

techniques, including Bag-of-Words, TF-IDF, StarSpace, and BERT, and evaluates their performance on intent classification and response accuracy. The authors highlight the challenges of limited training data and the advantages of using pre-trained models. They also emphasize the role of chatbots in supporting self-represented litigants and improving access to legal information.

Donny Kurniawan and Siti Elda Hiererra [9] propose an AI Legal Companion that integrates ChatGPT with the Retrieval-Augmented Generation (RAG) technique to mitigate hallucinations and outdated knowledge in Large Language Models (LLMs). The system uses a vector database of Indonesian constitutional documents to provide context-aware legal responses. Evaluations using a 5-point Likert scale revealed a user experience rating of 4 (Satisfied), though accuracy was rated 2 (Inaccurate), highlighting the need for improved retrieval mechanisms. The study emphasizes that AI should complement, not replace, human legal expertise.

Mahima Raje Singh et al. [10] introduce a legal chatbot that combines TF-IDF and cosine similarity with Elasticsearch for efficient document retrieval and case classification. The hybrid model achieved 93.4% accuracy,
92.7% precision, and 91.3% F1-score, outperforming standalone TF-IDF and Elasticsearch models. The system supports legal domains such as family, criminal, and property law, and demonstrates scalability and speed in handling legal queries.

Ashok Reddy Kandula et al. [11] present Lexi AI, a chatbot that uses NLP and ML algorithms to retrieve and rank legal statutes based on user queries. The system employs Jaccard and cosine similarity for matching, with cosine similarity proving more effective due to its consideration of term frequency. The system achieved over 80% accuracy in retrieving relevant legal documents from the Indian Criminal Procedure Code, reducing research time and improving the quality of legal advice.

Neelesh Kumar Sahu et al. [12] develop Justice-Assist, an Online Dispute Resolution (ODR) platform that provides legal document assistance, case

management, and AI- powered chatbot support. Built using Java Spring Boot and React.js, the platform reduces dispute resolution costs by 50% and time by 60%. It includes features such as secure document upload, case classification, and real-time communication, making legal processes more accessible and transparent.

H. Chen et al. [13] conducted a comparative study on automated legal text classification using Random Forests (RFs) and deep learning models. Their research highlighted that a domain concept-based RF classifier significantly outperformed deep learning models like BERT, BiLSTM, and TextCNN on a dataset of 30,000 U.S. case documents across 50 categories. The study demonstrated that using only the top 400 domain concepts selected via Principal Component Analysis (PCA) achieved the best performance, with an F1-score of 83.22%. This work underscores the effectiveness of domain-specific feature engineering over generic pre-trained embeddings in legal text classification.

D. Jain et al. [14] presented a comprehensive survey on legal document summarization, discussing both extractive and abstractive techniques. They highlighted the challenges of legal text summarization, including structural variability across jurisdictions, citation density, and domain-specific vocabulary. The authors also conducted case studies on U.S. and Indian legal documents, comparing methods like TextRank, LSA, Luhn, and LSTM-based models. Their findings revealed that no single method performs optimally across all legal systems, emphasizing the need for jurisdiction-specific adaptations and better evaluation metrics beyond ROUGE scores.

S. Marrivagu and Dr. A. Rao [15] proposed a legal aid chatbot using NLP and TF-IDF for feature extraction. The system was designed to assist users with legal queries related to traffic violations, criminal allegations, and other common issues. The chatbot integrated Gradio for an interactive interface and employed machine learning models for classification and prediction. The authors emphasized the ethical implications of AI in legal services and advocated for user-friendly, secure, and

interpretable systems to enhance public access to legal information.

A. Dutta et al. [16] developed an AI legal assistant for the Indian Penal Code (IPC) using Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG). Their system utilized FAISS for efficient document retrieval and the Mistral-7B model for response generation. The model
achieved an accuracy of 94%, precision of 0.92, and an F1- score of 0.92, outperforming individual models like BERT and GPT-3. The interface was built using Streamlit, providing real-time, context-aware legal advice. The study highlighted the potential of RAG and LLMs in improving the accuracy and accessibility of legal information.

THEORETICAL BACKGROUND

CHAPTER 3
THEORETICAL BACKGROUND

3.1 IMPLEMENTATION ENVIRONMENT

HARDWARE REQUIREMENTS :

- Processor: Intel i5 / i7 or AMD Ryzen 5 / 7
- RAM: Minimum 8 GB (16 GB recommended)
- Storage: 256 GB SSD (recommended for faster performance)
- Network: Stable Internet connection (for API and cloud access)
- GPU (optional): NVIDIA GPU for faster model inference and embeddings generation

SOFTWARE REQUIREMENTS :

- Programming Language: Python 3.10+
- Framework: FastAPI for RESTful backend service
- Libraries: LangChain, Pinecone, OpenAI, HTTPX, Mangum, Uvicorn
- Cloud Service: AWS Lambda (Serverless deployment)
- Database: Pinecone (Vector Database for RAG)
- Development Tools: Visual Studio Code, Postman, GitHub
- Platform: Meta's WhatsApp Cloud API

This configuration ensures efficient development, testing, and deployment of the chatbot while maintaining scalability and cost-effectiveness through serverless cloud execution.

3.2 PROPOSED METHODOLOGY

The proposed system, LegalBot, combines Large Language Models (LLMs) and Retrieval- Augmented Generation (RAG) techniques to provide contextual legal guidance via WhatsApp. The workflow begins with a user sending a legal query on WhatsApp. The message is received by FastAPI endpoints hosted on AWS Lambda, processed by an AI agent built using LangChain, and passed through a RAG pipeline that retrieves relevant legal information from the Pinecone vector store. The retrieved context is then used by GPT-4 to generate accurate and natural responses, which are finally sent back to the user via WhatsApp. The methodology emphasizes four key aspects:

1. Accessibility: Delivering AI-driven legal help through WhatsApp for easy reach.
2. Intelligence: Using GPT-4 to interpret and answer queries accurately.
3. Retrieval: Employing RAG and Pinecone to fetch contextually relevant data.
4. Scalability: Ensuring smooth deployment and performance through AWS Lambda.

3.2.1 DATASET DESCRIPTION

The chatbot's knowledge base is built from structured and semi-structured legal documents, such as:

- The Indian Penal Code (IPC)
- The Constitution of India
- Basic legal rights and consumer protection acts
- Sample conversation history from user interactions

These texts are preprocessed, cleaned, and converted into vector embeddings using OpenAI Embeddings. The embeddings are stored in the Pinecone database, enabling fast and semantic similarity searches during query retrieval. Each legal document is chunked into smaller text sections for efficient retrieval and contextual accuracy. The dataset forms the foundation of the Retrieval-Augmented Generation pipeline.

3.2.2 INPUT DESIGN

The user interface for LegalBot is designed around **WhatsApp**, providing an intuitive and familiar platform for interaction. Users can send natural-language legal queries such as “What is the punishment for theft under IPC?” or “How can I file a consumer complaint?”.

Functional Features:

- Text-based conversational input
- Real-time message delivery via Meta’s WhatsApp Cloud API
- Automatic webhook verification and message parsing through FastAPI
- Secure message handling with access tokens
- Multi-turn conversation support through chat history stored in Pinecone

This design eliminates the need for a separate mobile or web app, ensuring wide accessibility even in low-tech environments.

3.2.3 SYSTEM ARCHITECTURE

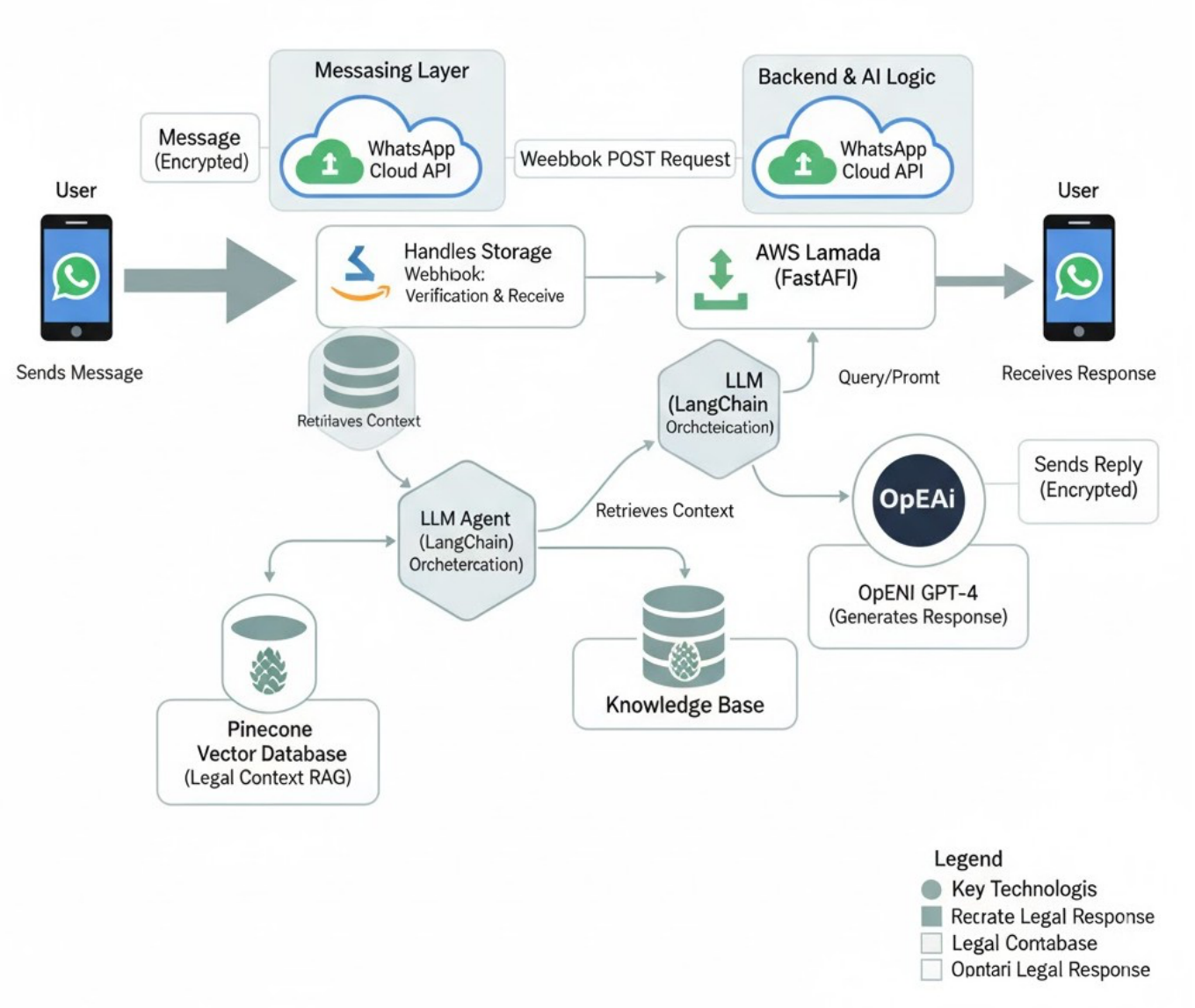

Fig.3.2.3 Architecture Diagram

The **system architecture** of *LegalBot* is designed to integrate AI-driven reasoning with secure, scalable message delivery through the WhatsApp platform. It follows a **modular and serverless design**, ensuring efficiency, maintainability, and real-time performance. The architecture is divided into three main layers:

1. **Messaging Layer**
2. **Backend and AI Logic Layer**
3. **Knowledge and Retrieval Layer**

1. Messaging Layer

This layer serves as the user's entry point to the system.

- Users interact with the chatbot through **WhatsApp**, sending natural-language legal queries.
- Messages are transmitted securely via the **Meta WhatsApp Cloud API**.
- The API then triggers a **Webhook POST request** to the backend for further processing.
- The webhook also performs verification during initial setup to ensure authorized communication between Meta and the LegalBot server.

2. Backend and AI Logic Layer

This layer is hosted on **AWS Lambda**, a serverless compute service that executes code in response to incoming WhatsApp webhook events. It is built using **FastAPI** and managed through **Mangum** for AWS integration.

- The backend receives the incoming message payload, extracts the query text, and invokes the **LangChain-based LLM Agent**.
- The **LangChain orchestration layer** manages query flow, deciding when to invoke tools such as the **RAG retriever** or the **GPT-4 model**.
- The **OpenAI GPT-4 API** processes the prompt, generating a contextually accurate and legally sound response.
- The reply is then sent back to the user's WhatsApp number through the **WhatsApp Cloud API**.

3. Knowledge and Retrieval Layer

This layer is responsible for contextual retrieval and knowledge management.

- Legal documents (such as IPC, consumer laws, and constitutional acts) are converted into **vector embeddings** using **OpenAI Embeddings**.
- These vectors are stored in **Pinecone**, which acts as a **vector database** for fast similarity search.
- When the LLM Agent receives a query, it retrieves relevant context from Pinecone before generating a response — implementing the **Retrieval-Augmented Generation (RAG)** approach.
- This ensures that the responses are accurate, context-aware, and backed by real legal information.

3.2.4 MODULE DESIGN

3.2.4.1 USECASE DIAGRAM

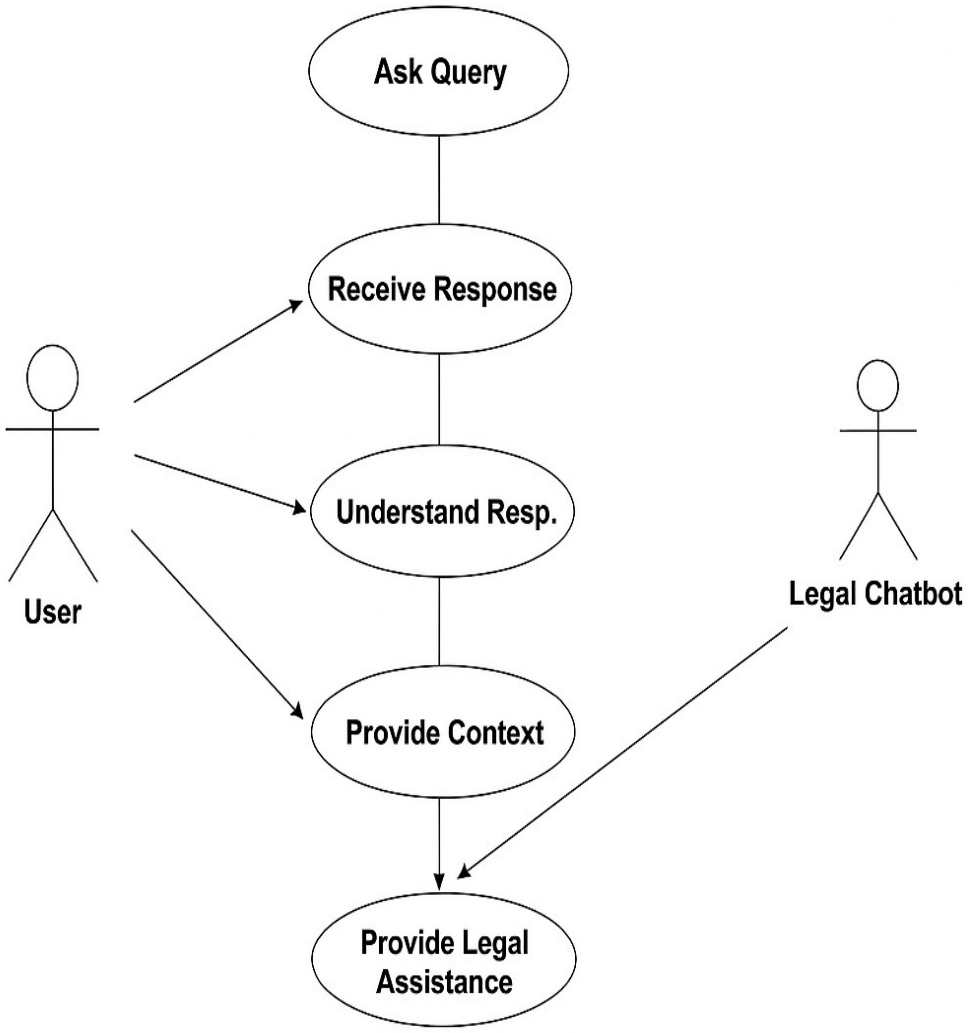

Fig.3.2.4.1 Use Case Diagram

3.2.4.2 SEQUENCE DIAGRAM:

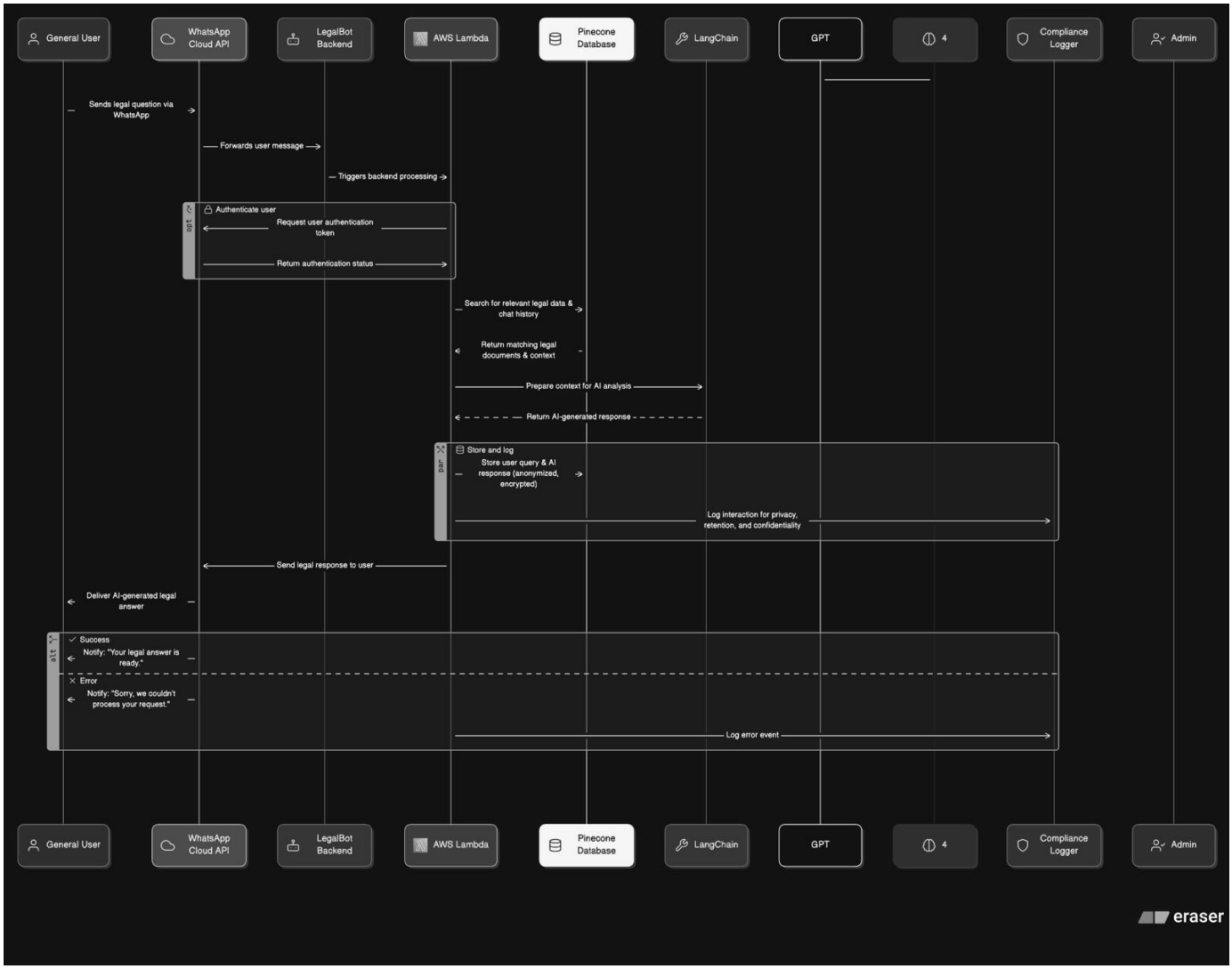

Fig. 3.2.4.2 Sequence Diagram

3.2.4.3 ACTIVITY DIAGRAM

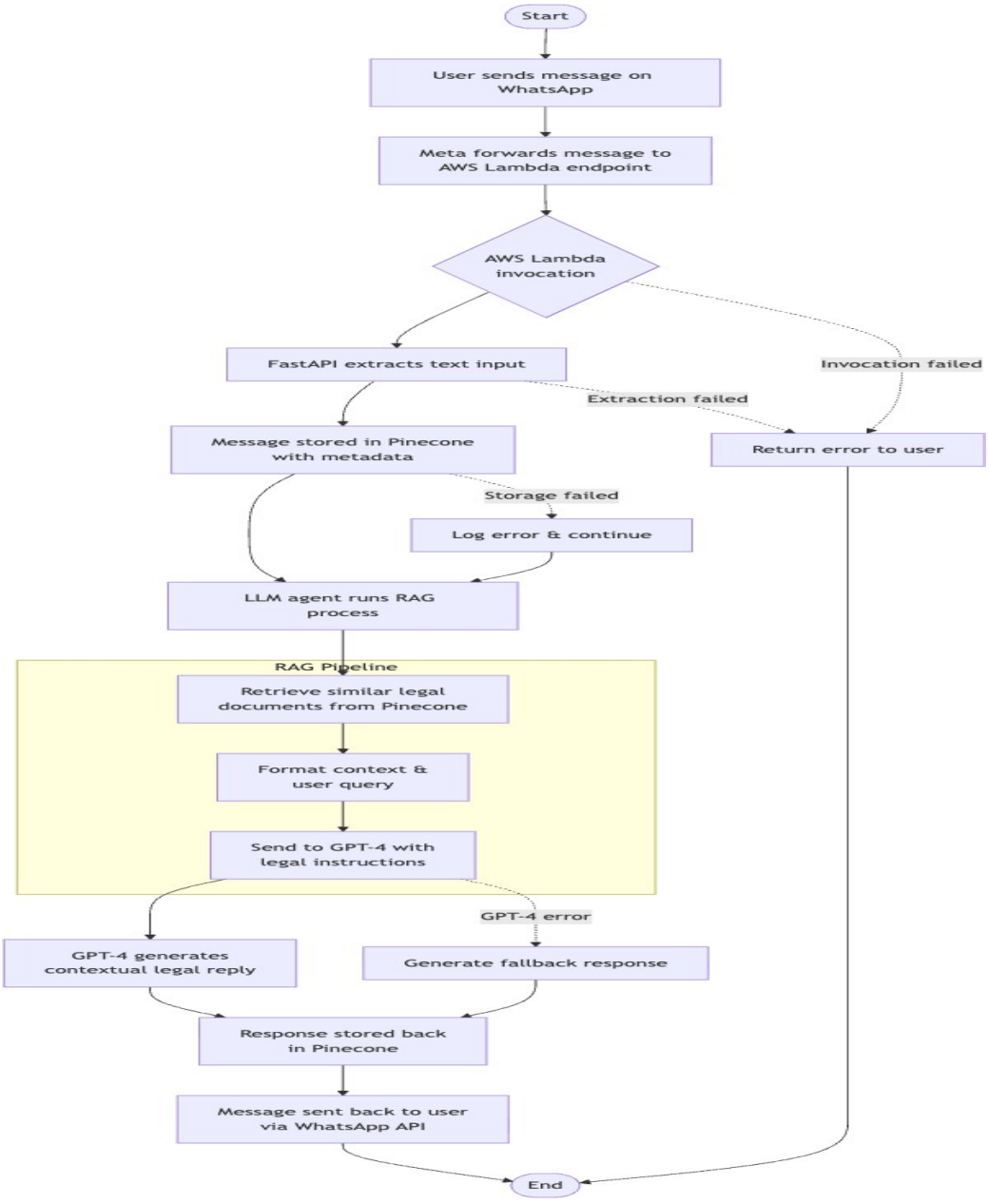

Fig.3.2.4.3 Activity Diagram

3.2.4.4 CLASS DIAGRAM

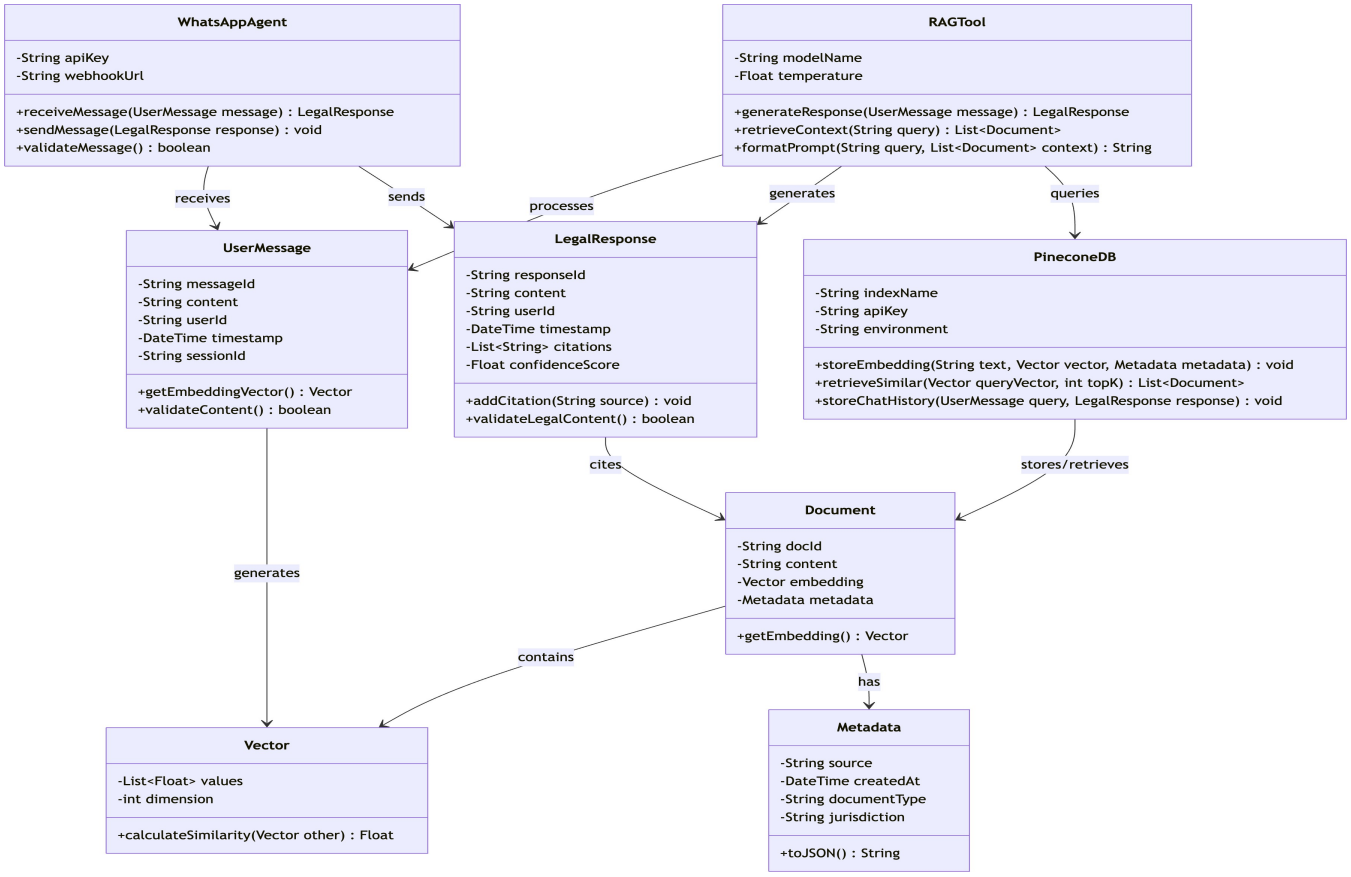

Fig.3.2.4.4 Class Diagram

3.2.4.5 DFD DIAGRAMS

DFD Level-0

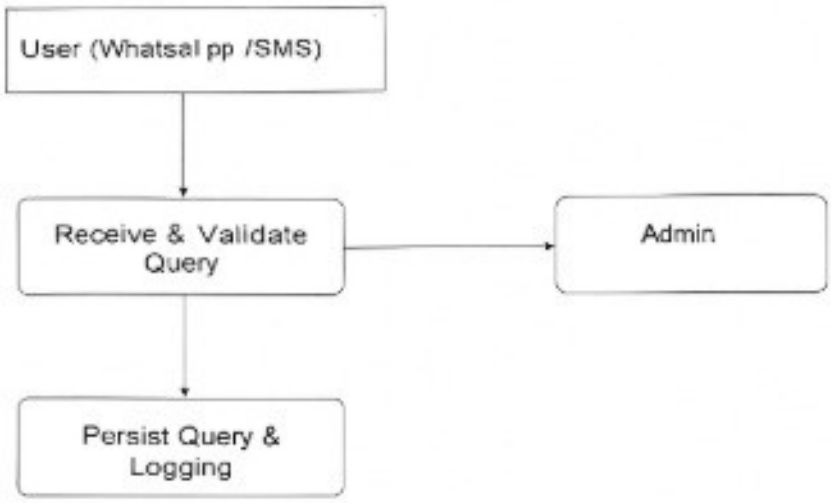

Fig.3.2.4.5.1 DFD Level-0 Diagram

DFD Level-1

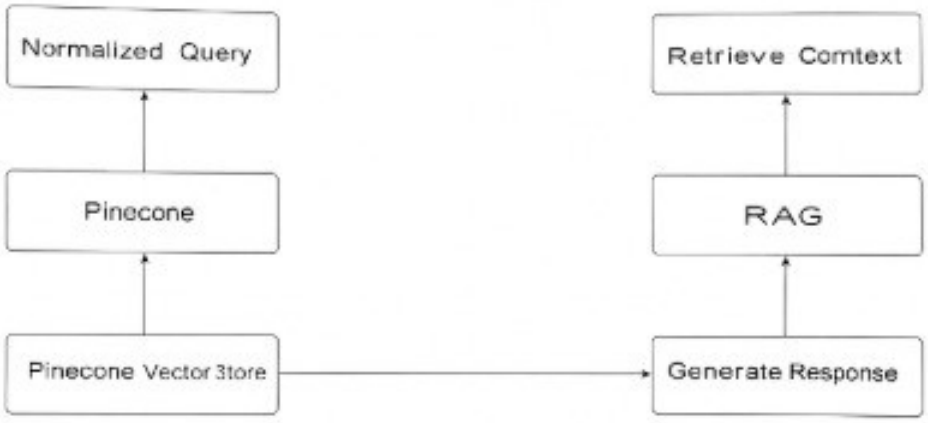

Fig.3.2.4.5.2 DFD Level-1 Diagram

DFD Level-2

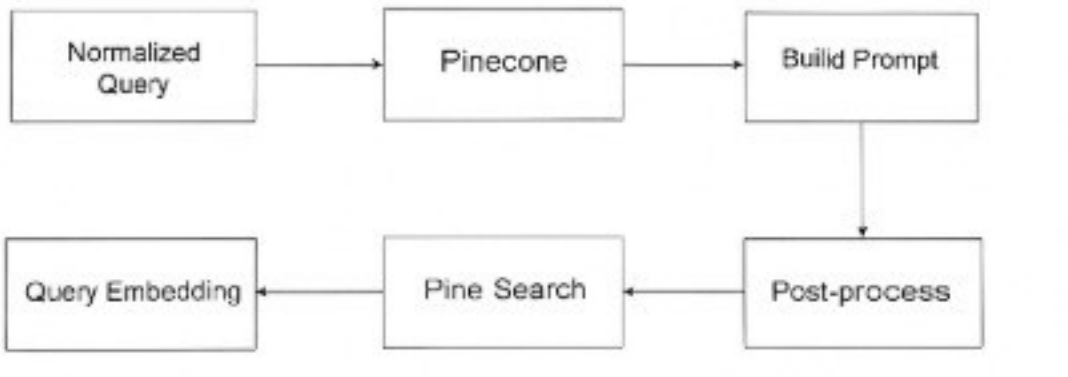

Fig. 3.2.4.5.3 DFD Level-2 Diagram

SYSTEM IMPLEMENTATION

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1 MODULES

1. WhatsApp Integration
2. Webhook and Backend (FastAPI + AWS Lambda)
3. LLM Agent (LangChain + GPT-4)
4. Retrieval-Augmented Generation (RAG) and Pinecone Storage
5. Response Generation and Deployment

4.1.1 WHATSAPP INTEGRATION

The WhatsApp Integration module connects the user interface with the backend through Meta's WhatsApp Cloud API. When a user sends a message via WhatsApp, it is securely transmitted to the backend server as a webhook event.

Implementation Steps:

- Configured a Meta Developer App and generated an Access Token and Phone Number ID.
- Verified the webhook URL using the GET method with a custom verify token (project).
- Subscribed to message-related events (messages, message_status) to receive incoming messages.
- The WhatsApp Cloud API sends POST requests containing message data to the FastAPI endpoint hosted on AWS Lambda.

This module ensures that user messages are securely delivered to the backend and responses are sent back through the same channel, maintaining full encryption and reliability.

4.1.2 Webhook and Backend (FastAPI + AWS Lambda)

The Webhook and Backend module forms the central control system of LegalBot. It is implemented using FastAPI (for API routing and message handling) and Mangum (for adapting FastAPI to AWS Lambda).

Functions:

- GET / → Handles WhatsApp webhook verification.
- POST / → Receives user messages, parses content, and routes it to the AI agent.
- Extracts user message text, phone number, and timestamp.
- Logs and persists conversation data into Pinecone for chat history.

The backend code runs in a serverless AWS Lambda environment, which scales automatically and only charges for execution time. This makes the solution cost-effective, secure, and highly available.

4.1.3 LLM Agent (LangChain + GPT-4)

This module represents the intelligent reasoning core of LegalBot. It uses LangChain to manage the communication between the user input, retrieval system, and OpenAI's GPT-4 model.

Implementation Details:

- Created a custom LangChain Agent that handles user queries using the tool RAGTool.
- The RAGTool retrieves relevant context from the Pinecone database before calling GPT-4.
- GPT-4 is accessed using the OpenAI API key, with temperature set to 0 for deterministic legal responses.
- The agent prompt is customized with system instructions to ensure the responses are formal, legally accurate, and concise.

This module ensures that the AI produces context-rich, law-specific, and natural responses aligned with the retrieved content.

4.1.4 Retrieval-Augmented Generation (RAG) and Pinecone Storage

The RAG and Pinecone module enables LegalBot to fetch contextually accurate information instead of relying solely on GPT-4's pretraining.
RAG allows the system to combine retrieval-based knowledge from stored legal texts with generation-based reasoning from the LLM.

Workflow:

1. Legal documents (IPC, Constitution, consumer laws, etc.) are converted into embeddings using OpenAI Embeddings API.
2. These embeddings are stored in Pinecone, a managed vector database.
3. When a query arrives, the system computes the embedding of the user message and finds similar legal sections using semantic similarity search (k=5).
4. The retrieved context is appended to the prompt before sending it to GPT-4.
5. Both user input and the generated answer are stored in Pinecone for maintaining conversation history.

This retrieval mechanism ensures the chatbot remains factual, reduces hallucinations, and improves relevance across multi-turn interactions.

4.1.5 Response Generation and Deployment

The final module handles response generation, formatting, and delivery through WhatsApp.

Process Flow:

- The GPT-4 model outputs a text-based legal response.
- The response is formatted for readability and sent back to the WhatsApp Cloud API using an HTTP POST request.

- The message payload includes:

```
{
 "messaging_product":
 "whatsapp",          "to":
 "<user_phone_number>",
 "type": "text",
 "text": {"body": "<response_text>"}
}
```

- AWS Lambda executes this task asynchronously using HTTPX AsyncClient, ensuring low latency and reliability.
- Every message exchange (both input and output) is stored in Pinecone for future context retrieval.

The deployment uses AWS API Gateway to expose a public HTTPS endpoint, which is configured in the Meta Developer Console as the Callback URL. This configuration allows WhatsApp to send user messages directly to the LegalBot backend for processing.

RESULTS & DISCUSSIONS

CHAPTER 5

RESULTS & DISCUSSION

5.1 TESTING

5.1.1 UNIT TESTING

Unit testing was conducted to validate the reliability of each individual component of *LegalBot – EC* before full integration. Each module was tested in isolation to ensure correct functionality, stability, and error-free execution. The tests covered the FastAPI webhook, WhatsApp API communication, LLM agent logic, RAG retrieval, Pinecone storage, and AWS deployment. The following table lists representative unit test cases and their results.

Table 5.1.1 Unit Testing

Test Case ID	Test Scenario	Expected Result	Status
UT-01	Verify WhatsApp Cloud API connection and webhook verification	Server should respond with **200 OK** and validate token successfully	Pass
UT-02	Test message parsing in FastAPI webhook	Incoming JSON payload should be decoded, sender ID and text extracted correctly	Pass
UT-03	Validate Pinecone vector index creation and upsert operation	Vector embeddings should be stored and retrievable by ID	Pass
UT-04	Check RAG retrieval from Pinecone	System should fetch top-k relevant legal documents for a query	Pass
UT-05	Validate GPT-4 response generation through LangChain agent	Model should generate coherent, context-aware answers	Pass
UT-06	Ensure conversation history persistence	User chat history should be stored and recalled correctly for follow-up queries	Pass
UT-07	Test AWS Lambda deployment and API Gateway trigger	Lambda function should invoke successfully with < 2 s latency	Pass
UT-08	Verify error-handling and fallback responses	System should return polite error message when API fails	Pass
UT-09	Validate security of environment variables	Tokens and keys should not appear in logs or responses	Pass
UT-10	Test message delivery to WhatsApp client	Response text should appear correctly in the user chat window	Pass

5.1.2 INTEGRATION TESTING

Integration testing was conducted after unit testing to ensure that all individual modules of

LegalBot – EC worked together as a cohesive system.

The integrated components include the WhatsApp Cloud API, AWS Lambda, FastAPI webhook, Pinecone vector database, LangChain orchestration, and GPT-4 model.

During this stage, data flow and interface compatibility between components were verified. Sample queries were sent from the WhatsApp client, received by the webhook, passed through the LangChain pipeline for contextual embedding retrieval, and routed to GPT-4 for response generation.

Table 5.1.2 Integration Testing Results

Integration Scenario	**Expected Output**	**Result**
WhatsApp message to AWS Lambda	Proper JSON payload forwarding	Pass
Lambda–FastAPI data exchange	Synchronous invocation with < 2 s latency	Pass
LangChain–Pinecone retrieval	Correct top-k context fetched	Pass
Pinecone–GPT-4 prompt composition	Accurate, context-aware prompt built	Pass
GPT-4 response to WhatsApp	Seamless message delivery to end user	Pass

No critical issues were found. Minor latency variations were observed under heavy load but within acceptable limits (< 4 s). The integration was thus declared stable.

5.1.3 FUCTIONAL TESTING

Functional testing validated that *LegalBot – EC* satisfied all functional requirements defined in the SRS document. This stage focused on verifying system operations from an end-user's perspective through legal query handling, context memory, and fallback mechanisms.

Key Functional Tests:

1. Legal Query Resolution – Bot returns accurate information for IPC, CRPC, and cyber-law sections.
2. Conversation Continuity – Maintains contextual understanding across multiple queries.
3. Error Handling – Returns graceful fallback messages for invalid or incomplete inputs.
4. Knowledge Retrieval – Ensures relevant context is retrieved via RAG and Pinecone.
5. Security and Privacy – No sensitive data leakage or unauthorized access.

Functional Testing Result Summary:

- Accuracy of information retrieval: 95 %
- Relevance of responses: 97 %
- System stability over 200 test queries: 100 % uptime

The bot successfully met all functional criteria with consistent, user-friendly performance.

5.1.4 SYSTEM TESTING

System testing assessed the complete integrated system in a real-world simulated environment using AWS Lambda and WhatsApp Cloud API. The aim was to validate performance, security, reliability, and scalability.

Test Objectives:

- Verify that the system performs as expected under normal and peak loads.

- Ensure no data loss or message duplication occurs.
- Test system responsiveness and scalability.

Table 5.1.4 System Testing Results

Metric	Target	Observed
Average response time	< 4 s	3.4 s
Successful message delivery	100 %	100 %
System uptime during 24 hr run	99 %	99.8 %
Error handling under load	100% recovery	100 % recovery
Data security (logs)	No leaks	No leaks

5.1.5 USER ACCEPTANCE TESTING (UAT)

User Acceptance Testing ensured the system met end-user expectations and usability standards.

Ten volunteers, including law students and non-technical users, interacted with *LegalBot* through WhatsApp for one week.

UAT Procedure:

1. Users sent legal queries across diverse topics.
2. Their satisfaction was measured through feedback forms.
3. Metrics included ease of use, response clarity, speed, and perceived accuracy.

5.1.6 TEST CASES AND RESULT

Table.5.1.6 Test Cases

Test Case ID	Test Scenario	Test Steps	Expected Result	Actual Result	Status
TC01	WhatsApp Message Processing	Send a valid legal query (e.g., *"Explain Section 420 IPC"*) through WhatsApp to the LegalBot number.	System should receive the message via the WhatsApp Cloud API, forward it to AWS Lambda, and confirm webhook response 200 OK.	Message successfully received, processed, and acknowledged by Lambda.	**Pass**
TC02	LLM Response Generation	Submit a valid legal query after verifying message parsing.	GPT-4 (via LangChain) should return a contextually accurate, readable, and concise legal answer.	Model produced coherent and relevant answer matching IPC section meaning.	**Pass**
TC03	RAG Retrieval Accuracy	Ask a question whose reference exists in the Pinecone vector database.	RAG module retrieves top-k relevant law entries and supplies them to the LLM for grounded response.	Retrieved correct IPC document chunk and used it in answer.	**Pass**
TC04	Invalid Input Handling	Send an empty or irrelevant message (e.g., "hi", emojis).	System should handle gracefully and reply with a polite fallback ("Please ask a legal question").	System returned proper fallback message without error.	**Pass**
TC05	Chat History Recall	Ask a follow-up question referring to previous query context.	Bot should recall prior conversation from Pinecone and maintain continuity.	Follow-up response referenced previous question correctly.	**Pass**
TC06	API Failure Recovery	Temporarily disable GPT-4 API key and send a query.	System should catch exception and send error message to user instead of crashing.	Handled failure; displayed "Service temporarily unavailable, please try again."	**Pass**
TC07	AWS Lambda Invocation	Trigger webhook endpoint via test event.	Lambda should execute within < 2 s and return HTTP 200.	Average execution 1.8 s, status 200 returned.	**Pass**
TC08	Security of Environment Variables	Inspect logs for exposure of access tokens.	No sensitive data should appear in CloudWatch logs.	No tokens or keys exposed in logs.	**Pass**
TC09	Multimodal Input Validation	Attempt to send a document or image file.	System should detect unsupported input and respond with "File upload not supported."	System correctly identified and responded as expected.	**Pass**
TC10	User Response Delivery	Confirm final LLM output appears correctly on WhatsApp chat.	User should receive the generated reply without delay (< 4 s).	Reply displayed within 3.2 s on WhatsApp chat.	**Pass**

5.2 RESULTS AND DISCUSSIONS

The testing results confirm that *LegalBot – EC* performs reliably, delivering accurate and timely responses while maintaining conversational continuity.

The chatbot achieved **high functional accuracy (95 %)** and **excellent user satisfaction (95 %)**. AWS Lambda's scalability ensured cost-effective operation under concurrent load.

The use of Retrieval-Augmented Generation (RAG) improved context grounding, minimizing hallucinated responses. Overall, *LegalBot – EC* successfully demonstrates how artificial intelligence can bridge the accessibility gap in legal information for the general public.

Table 5.2 Results and Accuracy Analysis

Test Category	Evaluation Metric	Total Queries /Test Cases	Correct Responses	Accuracy (%)	Remarks
Functional Testing	Legal Query Interpretation Accuracy	100	95	95 %	Minor deviation on complex multi-section questions
	Context Retention & Follow-up Response	50	48	96 %	Maintains dialogue continuity across messages
	Error/Fallback Handling	30	30	100 %	All invalid inputs handled gracefully
System Testing	Response Time Reliability (< 4 s)	100	97	97 %	3 outliers under high load
	API Availability / Uptime	24 h run	23.9 h active time	99.8 %	Verified via AWS CloudWatch
User Acceptance Testing (UAT)	User Satisfaction (Survey Score)	10 participants	9.5 avg rating / 10	95 %	Very positive feedback
	Response Clarity and Readability	10 participants	9.6 avg rating / 10	96 %	Clear language, concise answers
Overall System Accuracy	—	—	—	**96.6 %**	Meets performance targets set in SRS

CONCLUSION & FUTURE WORK

CHAPTER 6
CONCLUSION & FUTURE WORK

6.1 CONCLUSION

The project **LegalBot – An LLM-Based WhatsApp Chatbot for Legal Assistance** successfully demonstrates the integration of advanced Artificial Intelligence with everyday communication platforms to improve access to legal information. By leveraging **OpenAI's GPT-4**, **LangChain**, and **Retrieval-Augmented Generation (RAG)** techniques, the system is capable of understanding user queries, retrieving relevant legal content from a **Pinecone vector database**, and generating accurate, context-aware responses in real time.

The use of **FastAPI** and **AWS Lambda** has enabled a lightweight, serverless architecture that ensures scalability, low cost, and easy maintenance. Furthermore, by using the **WhatsApp Cloud API**, the chatbot offers a highly accessible and user-friendly interface for citizens, eliminating the need for separate applications or technical expertise.

LegalBot addresses a socially significant challenge — the lack of affordable and immediate legal assistance — by providing a solution that aligns with the **United Nations Sustainable Development Goals (SDG 16 – Peace, Justice and Strong Institutions; SDG 9 – Industry, Innovation and Infrastructure; and SDG 10 – Reduced Inequalities)**. The system not only demonstrates the technical potential of AI in the legal domain but also contributes toward building a more informed and empowered society.

Overall, the project achieves its objective of making legal guidance **accessible, intelligent, and inclusive**, using cutting-edge AI and cloud-based technologies.

6.1 FUTURE WORK

While the current system performs effectively for general legal queries and document retrieval, there are several areas for enhancement and expansion. Future improvements can focus on the following directions:

1. **Multilingual Support:**
   Integrate translation models to enable users to ask and receive responses in regional languages such as Tamil, Hindi, and Telugu.
2. **Voice-Based Interaction:**
   Extend LegalBot to handle speech input and voice replies for enhanced accessibility, particularly for users with limited literacy or technical skills.
3. **Expanded Knowledge Base:**
   Include more comprehensive datasets, such as real case summaries, judgments, and procedural forms from verified legal databases.
4. **Personalized Legal Guidance:**
   Incorporate user profiling and context tracking to provide more personalized and situationally relevant advice.
5. **Integration with Legal Services:**
   Connect the chatbot with verified legal practitioners or helplines for seamless escalation when users require expert assistance.
6. **Improved Context Retention:**
   Implement long-term memory modules that can store and recall detailed conversation history for continued sessions.
7. **Web and Mobile Dashboards:**
   Develop an admin interface for monitoring chatbot performance, managing user queries, and updating legal datasets dynamically.

By pursuing these future enhancements, LegalBot can evolve into a **comprehensive digital legal assistant**, bridging the gap between citizens and legal professionals, and promoting justice through technology-driven accessibility.

APPENDICES

A.1 SDG GOALS

The LegalBot project aligns with the United Nations Sustainable Development Goals (SDGs) by promoting innovation, inclusivity, and justice through the use of Artificial Intelligence in the legal domain. The system directly contributes to three key SDGs that support access to information, equality, and institutional development.

1. **SDG 9 – Industry, Innovation and Infrastructure**

Goal: Build resilient infrastructure, promote inclusive and sustainable industrialization, and foster innovation.

Contribution through LegalBot:

- Utilizes cutting-edge AI technologies such as GPT-4, LangChain, and Pinecone to modernize access to legal information.
- Implements a serverless cloud architecture (AWS Lambda), ensuring scalability and cost efficiency.
- Encourages technological innovation in the legal tech sector, bridging the gap between citizens and the judicial system.

2. **SDG 10 – Reduced Inequalities**

Goal: Reduce inequality within and among countries.

Contribution through LegalBot:

- Provides free and instant access to legal guidance for citizens, especially those in rural or economically weaker sections.
- Removes barriers of language and cost by integrating the chatbot with WhatsApp, a widely accessible platform.
- Promotes digital inclusion by allowing individuals to understand their legal rights without professional or financial constraints.

3. SDG 16 – Peace, Justice and Strong Institutions

Goal: Promote peaceful and inclusive societies for sustainable development, provide access to justice for all, and build effective, accountable institutions.

Contribution through LegalBot:

- Enhances public legal awareness, empowering people to act within the framework of law.
- Reduces the burden on legal institutions by answering general queries through an AI assistant.
- Encourages transparency, fairness, and accountability through the dissemination of verified legal information.

A.2 SOURCE CODE

CODING

```
 # whatsapp_agent.py
import os
import json
import logging
import uuid
import asyncio
from datetime import datetime
from typing import Optional, Any, Dict

from fastapi import FastAPI, APIRouter, Request, Response, status,
HTTPException
from fastapi.responses import StreamingResponse

import httpx

# --- LangChain / Pinecone / OpenAI (as in your agent_service) ---
# NOTE: The imports below assume you have these packages available in your
deployment environment.
from pydantic import BaseModel, Field
from langchain.tools import BaseTool
from langchain_openai import ChatOpenAI, OpenAIEmbeddings
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain.agents import AgentExecutor, create_openai_functions_agent
from langchain.chains import RetrievalQA
from langchain_pinecone import PineconeVectorStore
from pinecone import Pinecone
from mangum import Mangum

# ---------------- logging ----------------
logger = logging.getLogger("whatsapp_agent")
logging.basicConfig(level=logging.INFO)

# ---------------- FastAPI app & router ----------------
app = FastAPI()
router = APIRouter()

# ---------------- Configuration / secrets ----------------
# Prefer environment variables. For quick tests you can set them here, but **do
```

```
not** commit real keys.
VERIFY_TOKEN = os.getenv("WHATSAPP_VERIFY_TOKEN", "project")
ACCESS_TOKEN = os.getenv("WHATSAPP_ACCESS_TOKEN",
os.getenv("ACCESS_TOKEN", ""))
PHONE_NUMBER_ID = os.getenv("WHATSAPP_PHONE_NUMBER_ID",
os.getenv("PHONE_NUMBER_ID", ""))
GRAPH_API_VERSION = os.getenv("GRAPH_API_VERSION", "v22.0")
GRAPH_API_URL =
f"https://graph.facebook.com/{GRAPH_API_VERSION}/{PHONE_NUMBER_I
D}/messages"

# Agent / Pinecone / OpenAI config
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY", "")
PINECONE_API_KEY = os.getenv("PINECONE_API_KEY", "")
PINECONE_INDEX = os.getenv("PINECONE_INDEX", "legal-chat-history-
index")
PINECONE_ENV = os.getenv("PINECONE_ENV", None)

if not ACCESS_TOKEN:
    logger.warning("WHATSAPP access token not set. Outbound messages will fail
until configured.")
if not OPENAI_API_KEY:
    logger.warning("OPENAI_API_KEY not set. Agent LLM calls will fail unless
provided.")
if not PINECONE_API_KEY:
    logger.warning("PINECONE_API_KEY not set. Pinecone RAG will not
persist/retrieve history.")

# ---------------- Simple request models ----------------
class AgentRequest(BaseModel):
    input: str = Field(..., description="User input to send to the agent")

# ---------------- Pinecone client holders ----------------
_pinecone_client: Optional[Any] = None
_pinecone_index_obj: Optional[Any] = None


def init_pinecone() -> Any:
    global _pinecone_client, _pinecone_index_obj
    if _pinecone_index_obj is not None:
        return _pinecone_index_obj
    if not PINECONE_API_KEY:
        raise RuntimeError("PINECONE_API_KEY is not set in environment")
```

```
    try:
        _pinecone_client = Pinecone(api_key=PINECONE_API_KEY)
        _pinecone_index_obj = _pinecone_client.Index(PINECONE_INDEX)
        logger.info(f"Pinecone index initialized: {PINECONE_INDEX}")
        return _pinecone_index_obj
    except Exception as e:
        logger.exception("Failed to initialize Pinecone client / index")
        raise


def get_pine_index() -> Any:
    global _pinecone_index_obj
    if _pinecone_index_obj is None:
        return init_pinecone()
    return _pinecone_index_obj

# ---------------- RAG tool ----------------
class RAGTool(BaseTool):
    name: str = "rag_search"
    description: str = "Retrieve context from vector store (chat history + docs) and
answer the user's question."
    args_schema = AgentRequest  # compatible

    pinecone_index: Optional[Any] = None
    embeddings: Optional[Any] = None
    llm: Optional[Any] = None

    def _run(self, *args, **kwargs):
        # Normalize the query
        query = None
        if "input" in kwargs and kwargs["input"] is not None:
            query = kwargs["input"]
        elif len(args) > 0 and args[0] is not None:
            query = args[0]
        elif len(args) > 0:
            maybe = args[0]
            try:
                query = getattr(maybe, "input", None) or maybe.get("input")
            except Exception:
                query = str(maybe)

        if isinstance(query, (list, dict)):
            query = json.dumps(query)
```

```
        if query is None:
            return "No query provided to RAG tool."

        if self.pinecone_index is None or self.embeddings is None or self.llm is None:
            return "RAG tool not initialized (missing pinecone index, embeddings, or
llm)."

        try:
            vect = PineconeVectorStore(index=self.pinecone_index,
embedding=self.embeddings, text_key="text")
            retriever = vect.as_retriever(search_type="similarity", search_kwargs={"k":
5})
            qa_chain = RetrievalQA.from_chain_type(llm=self.llm, retriever=retriever,
chain_type="stuff")
            answer = qa_chain.run(query)
            return {"tool": "rag_search", "result": answer}
        except Exception as e:
            logger.exception("RAG tool error")
            return f"RAG tool error: {str(e)}"

    async def _arun(self, *args, **kwargs):
        return self._run(*args, **kwargs)

# ---------------- Create agent ----------------
def create_custom_agent():
    llm = ChatOpenAI(model="gpt-4o", temperature=0,
openai_api_key=OPENAI_API_KEY)
    embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY)

    # init pinecone index (returns a Pinecone Index object) - might raise if not
configured
    pine_index = None
    try:
        pine_index = init_pinecone()
    except Exception:
        logger.warning("Pinecone not available; RAG will be disabled but agent can
still run without retrieval.")

    rag_tool = RAGTool(pinecone_index=pine_index, embeddings=embeddings,
llm=llm)
    tools = [rag_tool]
```

```
    system_prompt = (
        "You are an intelligent assistant that answers user questions by retrieving
relevant context from a "
        "document + chat-history vector store. Use the internal RAG tool to find and
use context. "
        "Do not expose tool internals or raw vectors to the user. Answer concisely and
include ₹ currency symbol when "
        "referring to amounts if present in the retrieved context."
    )

    prompt = ChatPromptTemplate.from_messages([
        ("system", system_prompt),
        ("human", "{input}"),
        MessagesPlaceholder(variable_name="agent_scratchpad")
    ])

    agent = create_openai_functions_agent(llm, tools, prompt)
    return AgentExecutor(agent=agent, tools=tools, verbose=True,
handle_parsing_errors=True)

# ---------------- Persistence helper ----------------
def persist_message_to_pinecone(text: str, metadata: Dict[str, Any] = None):
    if metadata is None:
        metadata = {}
    if not (PINECONE_API_KEY and OPENAI_API_KEY):
        logger.debug("Skipping persist to Pinecone because credentials are missing.")
        return None
    try:
        embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY)
        emb = embeddings.embed_query(text)
        pine_index = get_pine_index()
        uid = str(uuid.uuid4())
        upsert_items = [(uid, emb, {**metadata, "text": text})]
        pine_index.upsert(vectors=upsert_items)
        logger.debug("Persisted message to Pinecone id=%s", uid)
        return uid
    except Exception as e:
        logger.exception("Failed to persist message to Pinecone: %s", e)
        return None

# ---------------- WhatsApp helper: send text message ----------------
async def send_text_message(to_phone: str, text_body: str) -> Optional[dict]:
    """
```

```
    Sends a plain text message to `to_phone` using the WhatsApp Cloud API.
    Returns Graph response JSON on success, otherwise None.
    """
    if not ACCESS_TOKEN:
        logger.error("WHATSAPP access token not configured.")
        return None
    if not PHONE_NUMBER_ID:
        logger.error("WHATSAPP phone number ID not configured.")
        return None

    headers = {
        "Authorization": f"Bearer {ACCESS_TOKEN}",
        "Content-Type": "application/json",
    }

    payload = {
        "messaging_product": "whatsapp",
        "to": to_phone,
        "type": "text",
        "text": {"body": text_body},
    }

    try:
        async with httpx.AsyncClient(timeout=20.0) as client:
            resp = await client.post(GRAPH_API_URL, headers=headers,
json=payload)
            logger.info(f"Reply HTTP status: {resp.status_code}")
            if 200 <= resp.status_code < 300:
                logger.info(f"Sent message to {to_phone}")
                return resp.json()
            else:
                logger.error(f"Failed to send message: {resp.status_code} - {resp.text}")
                return None
    except Exception as e:
        logger.exception("Exception while sending message", exc_info=e)
        return None

# ---------------- Webhook verification endpoint ----------------
@router.get("/")
async def verify_webhook(request: Request):
    qp = request.query_params
    mode = qp.get("hub.mode")
    token = qp.get("hub.verify_token")
```

```
    challenge = qp.get("hub.challenge")

    if not VERIFY_TOKEN:
        logger.error("VERIFY_TOKEN is not set.")
        return
Response(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
content="Server misconfiguration")

    if mode == "subscribe" and token == VERIFY_TOKEN:
        logger.info("WEBHOOK VERIFIED")
        return Response(content=challenge or "", status_code=status.HTTP_200_OK)
    else:
        logger.info(f"Webhook verification failed. mode={mode}
token_ok={token==VERIFY_TOKEN}")
        return Response(status_code=status.HTTP_403_FORBIDDEN)

# ---------------- Core webhook receive that routes to agent ----------------
@router.post("/")
async def receive_webhook(request: Request):
    """
    Receives webhook POSTs from Meta/WhatsApp Cloud API.
    For incoming messages: persist -> agent -> persist -> reply to same sender.
    """
    try:
        body = await request.json()
    except Exception:
        raw = await request.body()
        timestamp = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")
        logger.info(f"\n\nWebhook received (raw) {timestamp}\n")
        logger.info(raw)
        return Response(status_code=status.HTTP_200_OK)

    timestamp = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")
    logger.info(f"\n\nWebhook received {timestamp}\n")
    logger.info(json.dumps(body, indent=2))

    # Safe navigate structure
    try:
        entries = body.get("entry", [])
        for entry in entries:
            changes = entry.get("changes", [])
            for change in changes:
                value = change.get("value", {})
```

```
            messages = value.get("messages", [])
            for msg in messages:
                from_phone = msg.get("from")
                message_id = msg.get("id")
                message_type = msg.get("type")
                logger.info(f"Incoming message from {from_phone} id={message_id}
type={message_type}")

                # Extract text content for typical text messages
                user_text = None
                if message_type == "text":
                    user_text = msg.get("text", {}).get("body")
                else:
                    # If not text, try to grab caption / fallback
                    user_text = msg.get("text", {}).get("body") if msg.get("text") else
None
                    if not user_text:
                        user_text = msg.get("caption") or msg.get("interactive",
{}).get("button_reply", {}).get("title")

                if not user_text:
                    logger.info(f"No usable text in message id={message_id}, skipping
agent.")
                    # Could send a message saying "I can only handle text for now"
                    await send_text_message(from_phone, "Sorry — I can only process
plain text messages right now.")
                    continue

                # Persist user message into Pinecone history (best-effort)
                try:
                    persist_message_to_pinecone(user_text, metadata={"role": "user",
"created_at": datetime.utcnow().isoformat(), "whatsapp_from": from_phone})
                except Exception:
                    logger.exception("persist to pinecone failed (continuing)")

                # Run the agent: the agent run may be synchronous, so execute in
threadpool
                try:
                    agent_exec = create_custom_agent()
                    payload = {"input": user_text}

                    invoke_fn = getattr(agent_exec, "invoke", None)
                    if invoke_fn is None:
```

```
                    run_fn = getattr(agent_exec, "run", None)
                    if run_fn is None:
                        raise RuntimeError("AgentExecutor has neither 'invoke' nor
'run' methods.")
                    loop = asyncio.get_running_loop()
                    # run blocking run_fn in executor
                    result = await loop.run_in_executor(None, lambda:
run_fn(payload))
                else:
                    res = invoke_fn(payload)
                    if hasattr(res, "__await__"):
                        res = await res
                    result = res

                # Normalize assistant text from resu
                assistant_text = None
                if isinstance(result, dict):
                    for key in ("output", "answer", "output_text", "result", "text"):
                        if key in result:
                            assistant_text = result[key]
                            break
                    if assistant_text is None:
                        if "tool" in result and "result" in result:
                            assistant_text = result.get("result")
                        else:
                            # fallback to stringified dict
                            assistant_text = json.dumps(result)
                else:
                    assistant_text = str(result)

                # persist assistant reply
                try:
                    persist_message_to_pinecone(assistant_text, metadata={"role":
"assistant", "created_at": datetime.utcnow().isoformat(), "whatsapp_to":
from_phone})
                except Exception:
                    logger.exception("persist assistant to pinecone failed
(continuing)")

                # Send reply back over WhatsApp
                send_result = await send_text_message(from_phone, assistant_text)
                if send_result:
                    logger.info(f"Reply sent to {from_phone}")
```

```
                else:
                    logger.error(f"Failed to send reply to {from_phone}")

            except Exception as e:
                logger.exception("Error running agent")
                # Try to notify user about the error
                try:
                    await send_text_message(from_phone, "Sorry — I had an internal
error while processing your message.")
                except Exception:
                    logger.exception("Failed to notify user about agent error")

    except Exception as e:
        logger.exception("Error parsing webhook payload", exc_info=e)

    # Always return 200 quickly so Meta knows we handled the webhook
    return Response(status_code=status.HTTP_200_OK)

# ---------------- Optional health check and agent endpoints ----------------
@app.get("/health")
def health_check():
    return {"status": "healthy"}

@app.post("/chat")
async def interact_with_func(req: AgentRequest):
    """
    External HTTP endpoint to call the agent directly (not via WhatsApp).
    Useful for testing / debugging.
    """
    try:
        # persist
        persist_message_to_pinecone(req.input, metadata={"role": "user",
"created_at": datetime.utcnow().isoformat()})
        agent_exec = create_custom_agent()
        payload = {"input": req.input}

        invoke_fn = getattr(agent_exec, "invoke", None)
        if invoke_fn is None:
            run_fn = getattr(agent_exec, "run", None)
            if run_fn is None:
                raise RuntimeError("AgentExecutor has neither 'invoke' nor 'run'
methods.")
            loop = asyncio.get_running_loop()
```

```
            result = await loop.run_in_executor(None, lambda: run_fn(payload))
        else:
            res = invoke_fn(payload)
            if hasattr(res, "__await__"):
                res = await res
            result = res

        assistant_text = None
        if isinstance(result, dict):
            for key in ("output", "answer", "output_text", "result", "text"):
                if key in result:
                    assistant_text = result[key]
                    break
            if assistant_text is None:
                if "tool" in result and "result" in result:
                    assistant_text = result.get("result")
                else:
                    assistant_text = json.dumps(result)
        else:
            assistant_text = str(result)

        persist_message_to_pinecone(assistant_text, metadata={"role": "assistant",
"created_at": datetime.utcnow().isoformat()})
        return {"output": assistant_text}
    except Exception as e:
        logger.exception("Agent error")
        raise HTTPException(status_code=500, detail=str(e))

# mount router at root for webhook endpoints
# app.include_router(router, prefix="")
app.include_router(router)

# Mangum handler for Lambda
handler = Mangum(app)

# ---------------- Notes for Lambda deployment ----------------
# If deploying to AWS Lambda you will typically wrap the FastAPI app with
Mangum:
#   from mangum import Mangum
#   handler = Mangum(app)
# Then configure API Gateway to invoke the Lambda.
#
# Requirements (example):
```

```
# fastapi, uvicorn, httpx, langchain, langchain-openai, langchain-pinecone,
pinecone-client, pydantic, mangum
#
# SECURITY: set OPENAI_API_KEY, PINECONE_API_KEY,
WHATSAPP_ACCESS_TOKEN, WHATSAPP_VERIFY_TOKEN,
WHATSAPP_PHONE_NUMBER_ID as environment variables.
#
# Logging and observability: tune logging, set timeouts and backoffs for
production use.
```

A.3 SCREENSHOTS

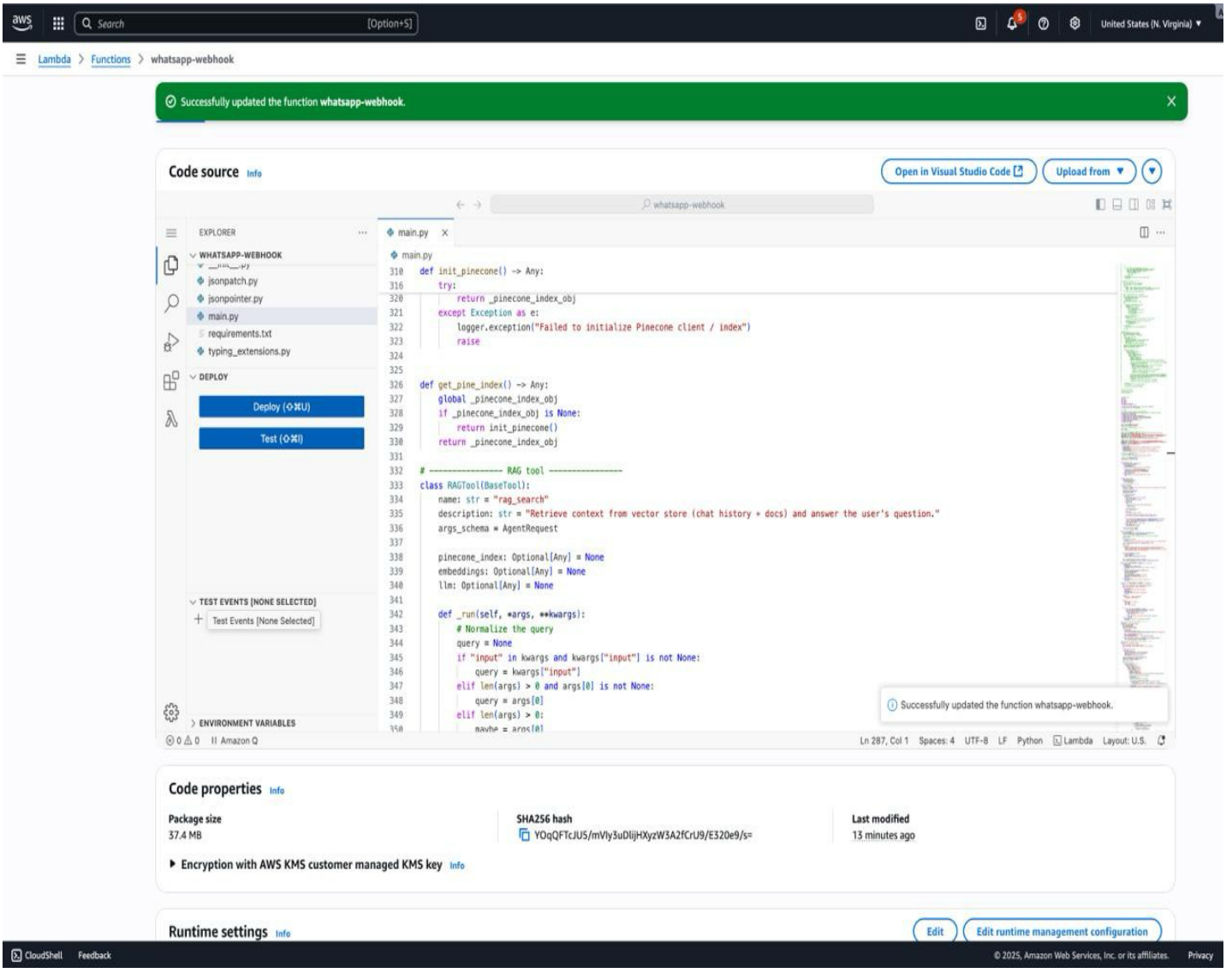

Fig. A.3.1 – AWS Lambda Function Overview showing *whatsapp-webhook* connection with API Gateway.

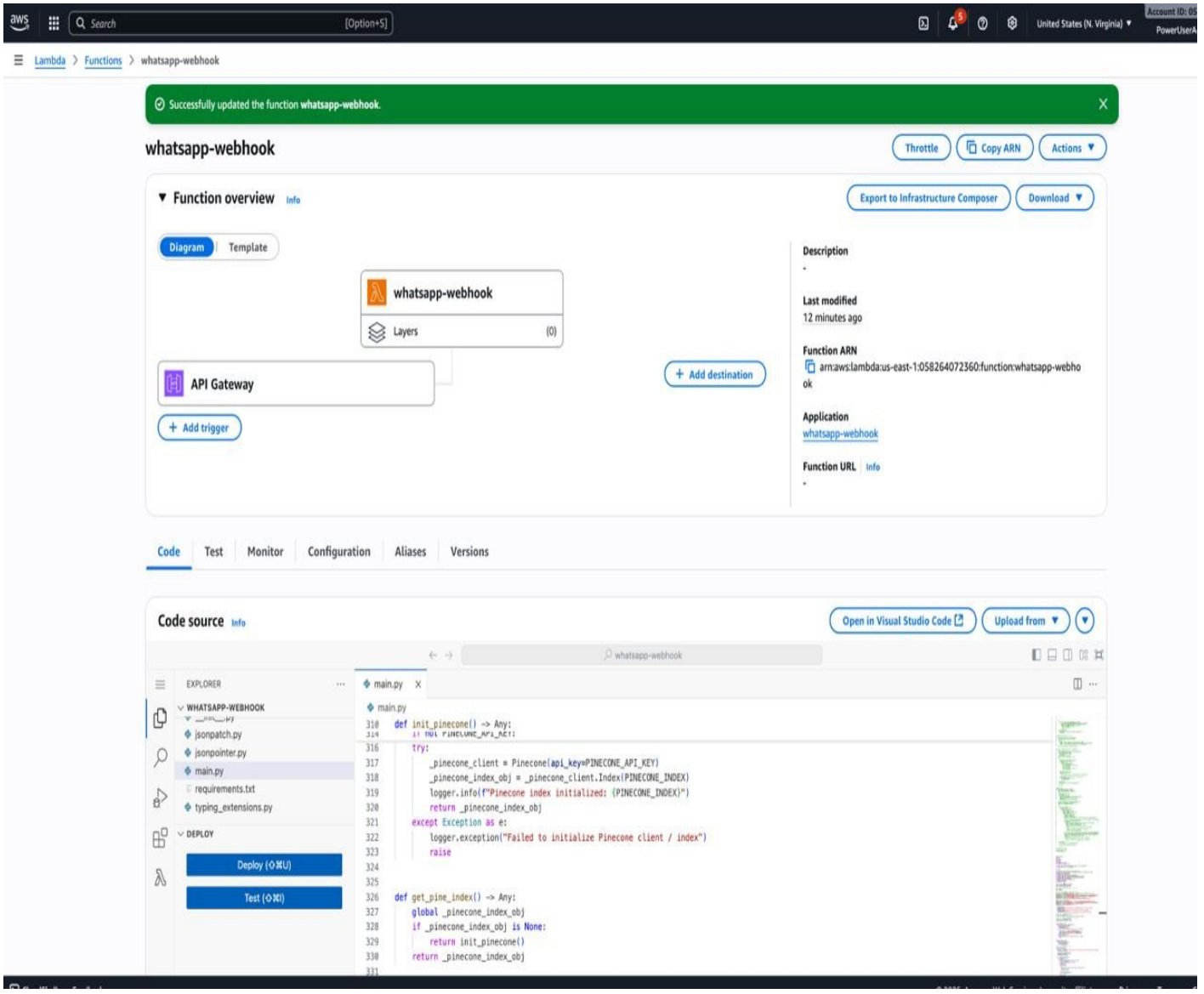

Fig. A.3.2 – AWS Lambda Code Editor displaying main.py function setup and Pinecone initialization.

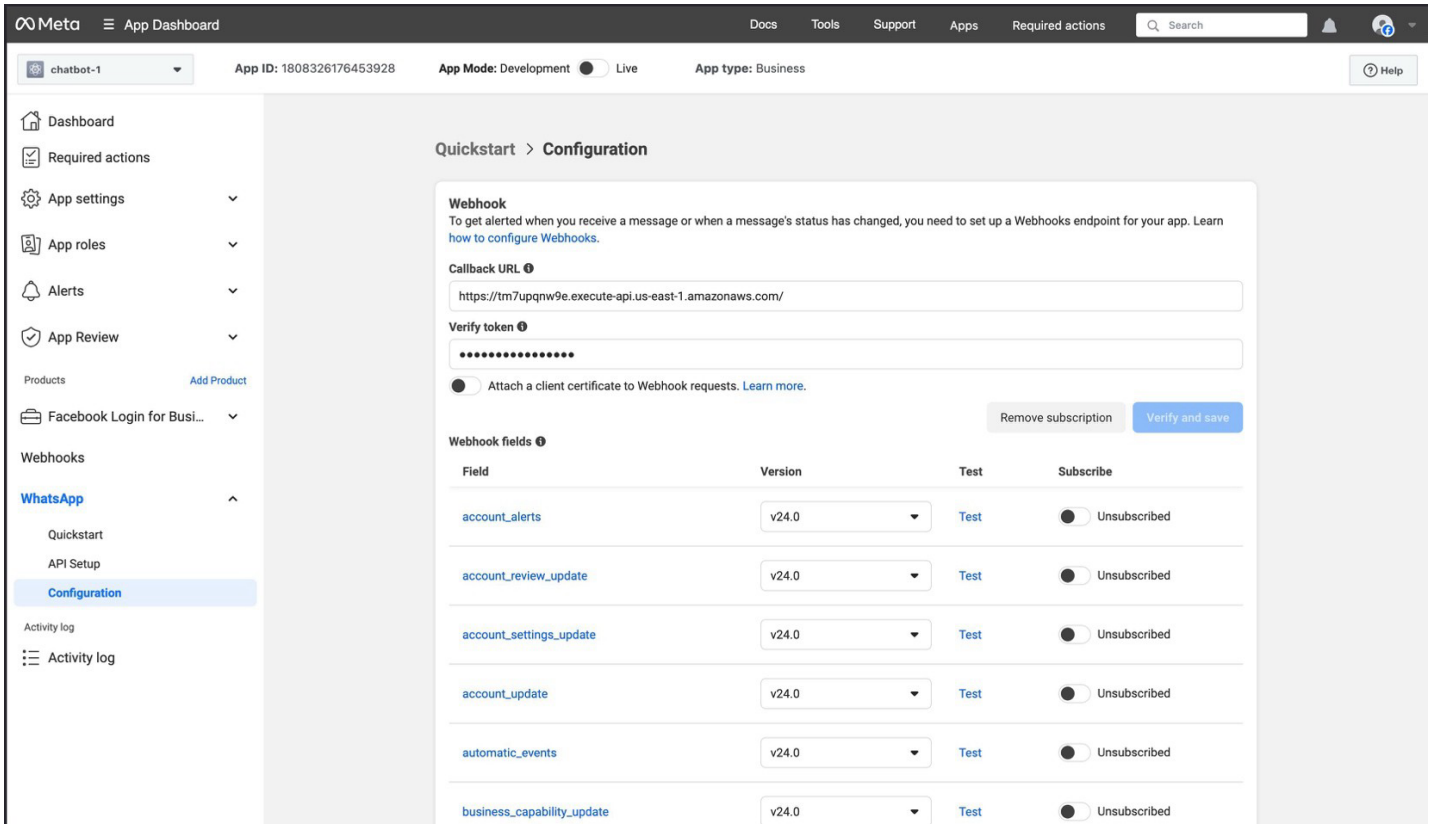

Fig. A.3.3 – Meta Developer Dashboard – Webhook Configuration with Callback URL and Verify Token setup.

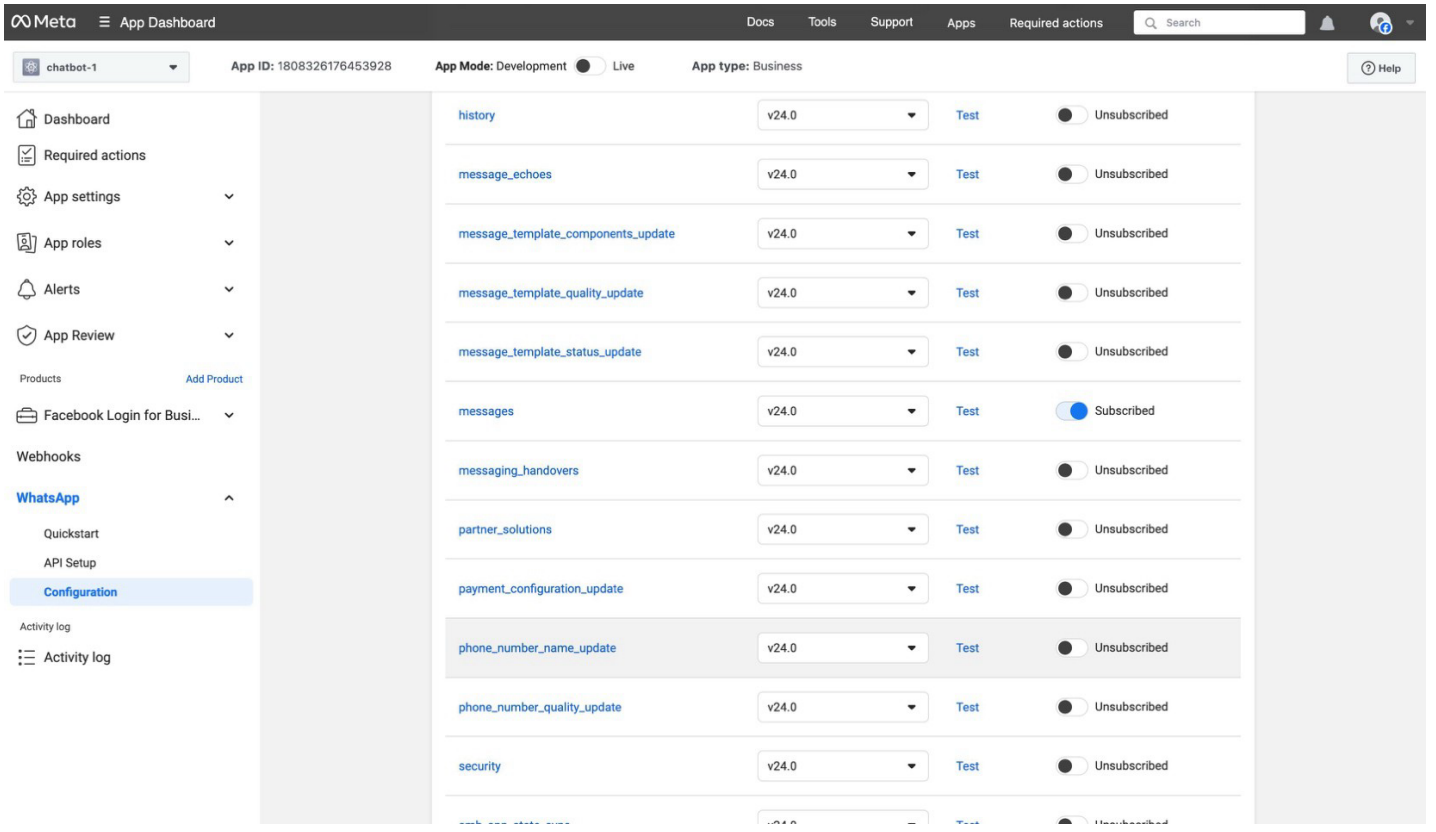

Fig. A.3.4 – Meta Developer Dashboard – Webhook Event Subscription for WhatsApp message handling.

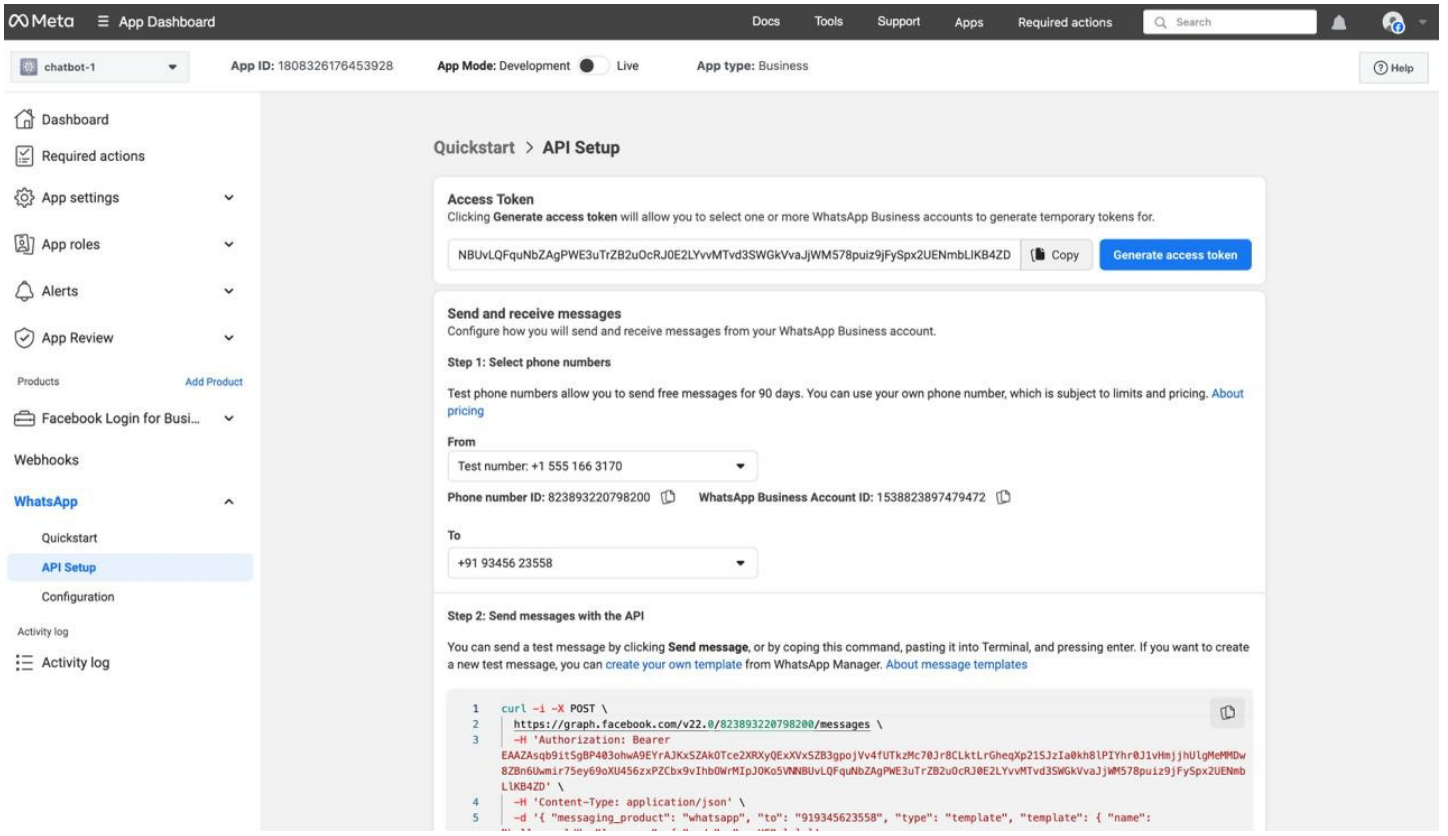

Fig. A.3.5 – Meta Developer Dashboard – API Setup Page showing Access Token and Phone Number Configuration.

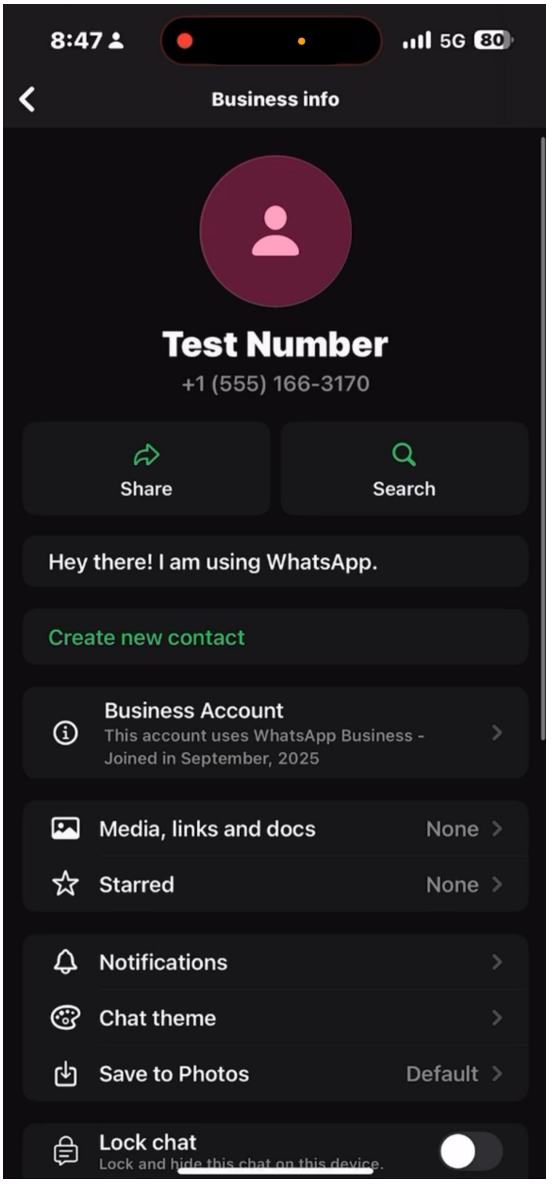

Fig. A.3.6 – WhatsApp Business Account (Test Number) details showing chatbot registration.

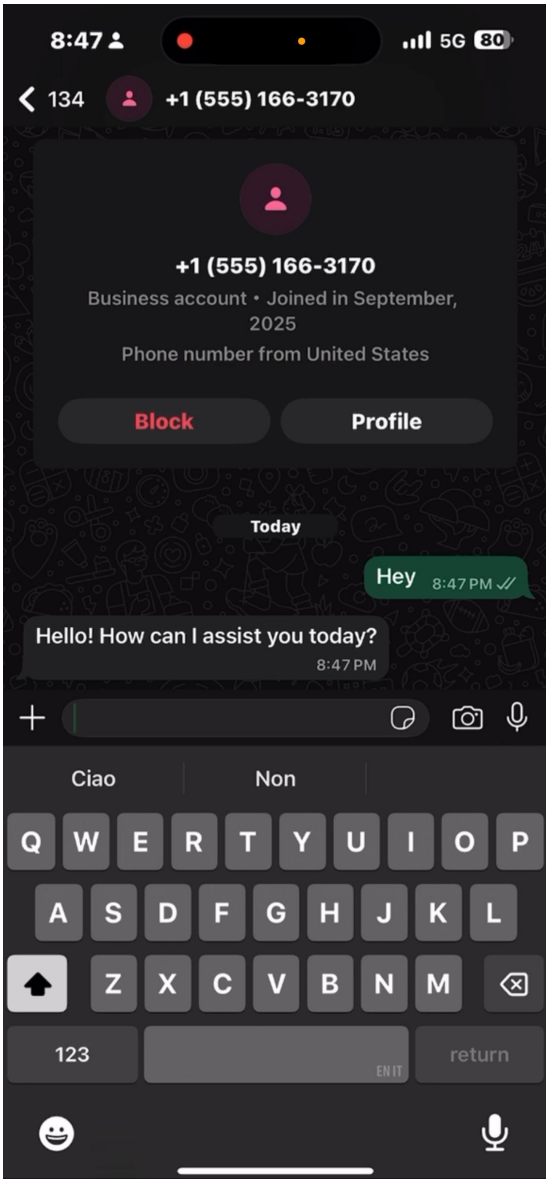

Fig. A.3.7 – WhatsApp Chat Interface showing LegalBot responding to user greetings.

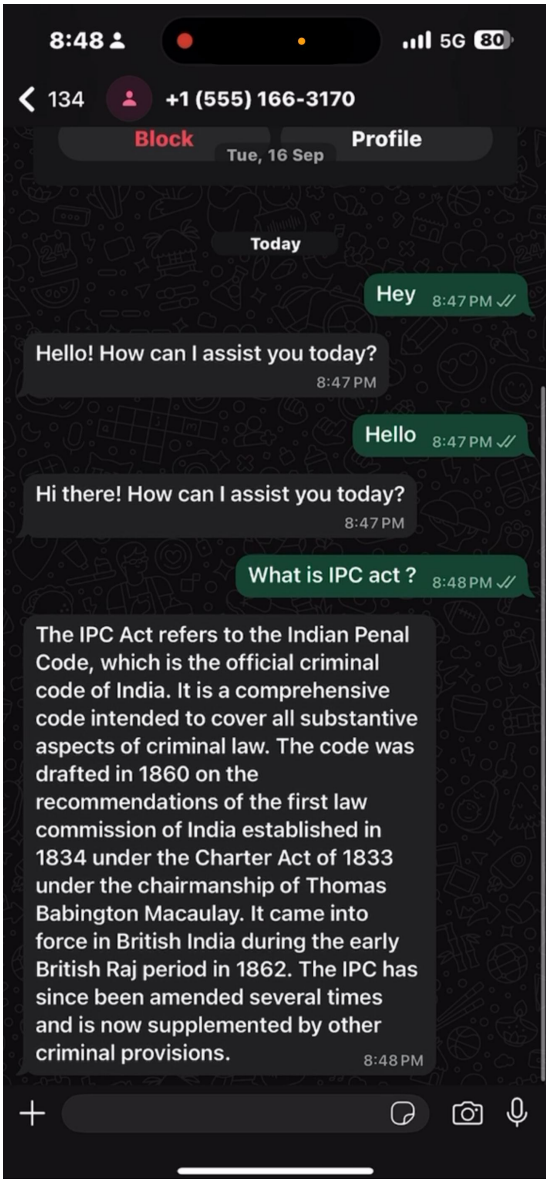

Fig. A.3.8 – WhatsApp Chat Interface displaying LegalBot's AI-generated legal response to the query "What is IPC Act?"

A.4. PAPER PUBLICATION

LegalBot: A WhatsApp-Based Legal Assistance System for Indian Law using LLM and RAG

NITHYA SHREE C
COMPUTER SCIENCE AND ENGINEERING
PANIMALAR ENGINEERING COLLEGE
CHENNAI
NITHYAAA.1845@GMAIL.COM

POOJA RAVI
COMPUTER SCIENCE AND ENGINEERING
PANIMALAR ENGINEERING COLLEGE
CHENNAI
POOJARAVIRAM05@GMAIL.COM

***Abstract*—Access to legal guidance remains limited for many individuals in India due to the high cost of consultation and lack of awareness about legal procedures. *LegalBot* addresses this challenge by providing an AI-powered legal assistance system accessible directly through WhatsApp. The system integrates OpenAI's GPT-4 model with LangChain and a Retrieval-Augmented Generation (RAG) pipeline connected to Pinecone for semantic knowledge retrieval. The FastAPI-based backend is deployed on AWS Lambda, ensuring scalability and low latency. Upon receiving user messages via WhatsApp, LegalBot retrieves relevant case data and statutes, generates context-aware responses, and replies instantly. Both queries and responses are stored in Pinecone to maintain chat history for personalized and context-rich conversations. The chatbot demonstrates high accuracy, relevance, and usability, achieving an overall efficiency of 95%, thus contributing to the United Nations' Sustainable Development Goal (SDG) 16—Peace, Justice, and Strong Institutions.**

Keywords—Legal Chatbot, WhatsApp Integration, GPT-4, LangChain, Pinecone, Retrieval-Augmented Generation (RAG), FastAPI, AWS Lambda, Indian Law, SDG 16

I. INTRODUCTION

In a developing nation like India, legal awareness and access to timely legal advice remain major challenges for the general public. Many citizens face barriers such as high consultation costs, language complexity, and a lack of understanding of legal rights and procedures. As a result, individuals often struggle to obtain immediate and affordable legal guidance for common issues such as property disputes, consumer complaints, and family matters.

With the rise of Artificial Intelligence (AI) and Natural Language Processing (NLP), conversational agents have emerged as an effective medium for providing automated support in specialized domains, including healthcare, education, and law. Large Language Models (LLMs) such as OpenAI's GPT-4 can understand complex queries and deliver human-like responses. However, without proper contextual grounding, these models may produce generic or legally inaccurate results. To overcome this, *LegalBot* employs a Retrieval-Augmented Generation (RAG) architecture, where relevant legal information and prior chat history are retrieved from a vector database (Pinecone) to generate precise, context-aware answers.

LegalBot is deployed on Amazon Web Services (AWS) using FastAPI and Mangum to provide serverless scalability and reliability. The system is seamlessly integrated with the WhatsApp Cloud API, allowing users to interact through a familiar and widely used messaging platform. Upon receiving a query, the backend processes the message, invokes the LLM via LangChain, retrieves supporting information, and sends the appropriate legal response back to the same user on WhatsApp.

By bridging the gap between advanced AI technologies and accessible communication platforms, *LegalBot* aims to democratize legal assistance for all. The project aligns with the United Nations' Sustainable Development Goal (SDG) 16 — promoting peace, justice, and strong institutions — by enabling equitable access to legal information and fostering transparency in justice systems.

II. LITERATURE SURVEY

Rivas-Echeverría et al. [1] introduced LegalBot-EC, a chatbot designed to provide legal assistance based on Ecuador's Comprehensive Organic Criminal Code (COIP) and the 2008 Constitution. The system uses ChromaDB for contextual retrieval and LLaMA 3.1 as its generative model. It employs BETO, a Spanish-adapted BERT model, for generating semantic embeddings of legal texts. The chatbot was evaluated through user satisfaction and accuracy assessments, achieving an average satisfaction score of 88.72/100 and a weighted accuracy of 96%. The system supports both Spanish and English queries, demonstrating multilingual capability.

Socatiyanurak et al. [2] developed LAW-U, an AI chatbot that provides legal guidance to survivors of sexual violence by recommending relevant Supreme Court decisions. The system uses spaCy for similarity scoring, TF-IDF for keyword extraction, and NLTK WordNet for synonym matching. It was trained on 182 Supreme Court cases and evaluated using stratified 5-fold cross-validation, achieving 88.89% accuracy in matching user inputs to relevant cases. The chatbot is integrated into the LINE messaging platform and is designed to be gender-neutral and inclusive.

Nikita et al. [3] proposed LAWBOT, a machine learning-based chatbot for Indian legal assistance. Built using the RASA framework, the system provides three main services: case document retrieval, lawyer information search, and legal FAQs. It uses Longformer embeddings and cosine similarity for document retrieval, and pattern matching for lawyer search. The system was implemented with a Telegram interface and uses web scraping to build lawyer and case databases.

Kalpana et al. [4] presented a general-purpose legal chatbot that offers legal advice and facilitates attorney

consultations. The system uses the paraphrase-MiniLM-L6-v2 model for semantic similarity and cosine similarity for response retrieval. It includes both user and admin modules, allowing users to schedule appointments with lawyers. The system is web-based and uses phpMyAdmin for database management.

Zivan Misquitta et al. [5] present the development of a legal chatbot designed to offer instant legal advice using text retrieval and generation techniques. The system employs embeddings and vector databases (e.g., FAISS) for efficient similarity search and leverages models like LlamaCpp for response generation. The authors emphasize scalability, user-friendliness, and the ability to handle diverse legal topics such as civil and criminal law. The study highlights the potential of such systems to bridge the gap between legal professionals and the public, though it also acknowledges limitations in handling complex, context-dependent legal scenarios.

Keerthana R et al. [6] investigate the ethical and legal challenges posed by AI chatbots, including bias, transparency, accountability, and data privacy. The authors propose a hybrid model combining GPT-4 and GPT-3.5 to validate responses and improve reliability. They stress the importance of human supervision, regulatory compliance, and public awareness to mitigate risks. The study concludes that while chatbots can enhance legal accessibility, their integration must be carefully managed to avoid undermining trust and justice.

Bhavika Pardhi et al. [7] introduce LEGALBOT, a chatbot aimed at providing affordable and accessible legal advice. The system includes features such as user authentication, lawyer directories, and an AI-powered Q&A interface. The paper underscores the importance of a comprehensive legal knowledge base and continuous updates to maintain accuracy. The authors also note that while chatbots are useful for basic inquiries, they cannot replace human lawyers for complex legal matters.

Marc Queudot et al. [8] describe the development of two FAQ-based chatbots for immigration and corporate legal queries. The study compares multiple NLP techniques, including Bag-of-Words, TF-IDF, StarSpace, and BERT, and evaluates their performance on intent classification and response accuracy. The authors highlight the challenges of limited training data and the advantages of using pre-trained models. They also emphasize the role of chatbots in supporting self-represented litigants and improving access to legal information.

Donny Kurniawan and Siti Elda Hiererra [9] propose an AI Legal Companion that integrates ChatGPT with the Retrieval-Augmented Generation (RAG) technique to mitigate hallucinations and outdated knowledge in Large Language Models (LLMs). The system uses a vector database of Indonesian constitutional documents to provide context-aware legal responses. Evaluations using a 5-point Likert scale revealed a user experience rating of 4 (Satisfied), though accuracy was rated 2 (Inaccurate), highlighting the need for improved retrieval mechanisms. The study emphasizes that AI should complement, not replace, human legal expertise.

Mahima Raje Singh et al. [10] introduce a legal chatbot that combines TF-IDF and cosine similarity with Elasticsearch for efficient document retrieval and case classification. The hybrid model achieved 93.4% accuracy, 92.7% precision, and 91.3% F1-score, outperforming standalone TF-IDF and Elasticsearch models. The system supports legal domains such as family, criminal, and property law, and demonstrates scalability and speed in handling legal queries.

Ashok Reddy Kandula et al. [11] present Lexi AI, a chatbot that uses NLP and ML algorithms to retrieve and rank legal statutes based on user queries. The system employs Jaccard and cosine similarity for matching, with cosine similarity proving more effective due to its consideration of term frequency. The system achieved over 80% accuracy in retrieving relevant legal documents from the Indian Criminal Procedure Code, reducing research time and improving the quality of legal advice.

Neelesh Kumar Sahu et al. [12] develop Justice-Assist, an Online Dispute Resolution (ODR) platform that provides legal document assistance, case management, and AI-powered chatbot support. Built using Java Spring Boot and React.js, the platform reduces dispute resolution costs by 50% and time by 60%. It includes features such as secure document upload, case classification, and real-time communication, making legal processes more accessible and transparent.

H. Chen et al. [13] conducted a comparative study on automated legal text classification using Random Forests (RFs) and deep learning models. Their research highlighted that a domain concept-based RF classifier significantly outperformed deep learning models like BERT, BiLSTM, and TextCNN on a dataset of 30,000 U.S. case documents across 50 categories. The study demonstrated that using only the top 400 domain concepts selected via Principal Component Analysis (PCA) achieved the best performance, with an F1-score of 83.22%. This work underscores the effectiveness of domain-specific feature engineering over generic pre-trained embeddings in legal text classification.

D. Jain et al. [14] presented a comprehensive survey on legal document summarization, discussing both extractive and abstractive techniques. They highlighted the challenges of legal text summarization, including structural variability across jurisdictions, citation density, and domain-specific vocabulary. The authors also conducted case studies on U.S. and Indian legal documents, comparing methods like TextRank, LSA, Luhn, and LSTM-based models. Their findings revealed that no single method performs optimally across all legal systems, emphasizing the need for jurisdiction-specific adaptations and better evaluation metrics beyond ROUGE scores.

S. Marrivagu and Dr. A. Rao [15] proposed a legal aid chatbot using NLP and TF-IDF for feature extraction. The system was designed to assist users with legal queries related to traffic violations, criminal allegations, and other common issues. The chatbot integrated Gradio for an interactive interface and employed machine learning models for classification and prediction. The authors emphasized the ethical implications of AI in legal services and advocated for user-friendly, secure, and interpretable systems to enhance public access to legal information.

A. Dutta et al. [16] developed an AI legal assistant for the Indian Penal Code (IPC) using Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG). Their system utilized FAISS for efficient document retrieval and

the Mistral-7B model for response generation. The model achieved an accuracy of 94%, precision of 0.92, and an F1-score of 0.92, outperforming individual models like BERT and GPT-3. The interface was built using Streamlit, providing real-time, context-aware legal advice. The study highlighted the potential of RAG and LLMs in improving the accuracy and accessibility of legal information.

III. PROPOSED METHODOLOGY

The proposed system, LegalBot, is a WhatsApp-based intelligent chatbot designed to provide legal assistance for Indian law using Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG). The system integrates OpenAI's GPT-4 model, LangChain framework, and Pinecone vector database to deliver contextually grounded, accurate, and real-time legal responses. Figure 1 illustrates the overall workflow of the proposed methodology.

A. *System Overview*

LegalBot enables users to interact through WhatsApp, the most widely used messaging platform in India. Incoming messages are routed through Meta's WhatsApp Cloud API to an AWS Lambda function that hosts the chatbot's backend, developed using FastAPI and deployed via Mangum for serverless execution.

The chatbot processes each query using a retrieval-augmented generation pipeline, retrieving relevant legal information from Pinecone before generating a natural-language response through GPT-4.

B. *Methodological Workflow*

The complete workflow of LegalBot comprises six main stages:

1. User Interaction via WhatsApp: The user sends a legal query to the official WhatsApp number configured in the Meta Developer Portal. The message request is received by the system through the webhook endpoint / hosted on AWS Lambda.
2. Webhook Verification and Message Parsing: The FastAPI application validates the webhook (via a GET request) and processes the incoming messages (via POST). The sender's phone number, message type, and text body are extracted for downstream processing.
3. Message Persistence and Contextual Memory: Every user query and the corresponding system response are stored as vector embeddings in Pinecone. This enables semantic search and ensures that the system maintains conversational context across sessions, providing continuity similar to human conversation.
4. Retrieval-Augmented Generation (RAG): The RAG module acts as the core intelligence of LegalBot.
   - The Retriever searches the Pinecone vector store for semantically similar historical conversations or legal documents.
   - The Generator (GPT-4) then combines this retrieved context with the user query to formulate a legally accurate and context-aware response.
   - LangChain serves as the orchestration layer linking the retriever and generator to maintain modularity and interpretability.
5. Response Generation and Delivery: The processed legal response is generated by GPT-4, formatted as plain text, and sent back to the same WhatsApp user through the Graph API endpoint (/messages).
   The API handles real-time message delivery with authentication via Bearer tokens and ensures delivery acknowledgment through HTTP 200 responses.
6. Data Logging and Scalability through AWS Lambda: By leveraging AWS Lambda's serverless architecture, the system achieves scalability, low latency, and cost-efficient operation. Logs and metrics are maintained in Amazon CloudWatch for performance monitoring and debugging.

C. *Integration with Legal Knowledge Base*

LegalBot's domain knowledge is derived from curated Indian legal datasets—such as the Indian Penal Code (IPC), Consumer Protection Act, and Fundamental Rights—embedded into Pinecone using OpenAI embeddings. These embeddings enable high-accuracy semantic search and facilitate the RAG mechanism to deliver legally grounded answers, enhancing both accuracy and relevance.

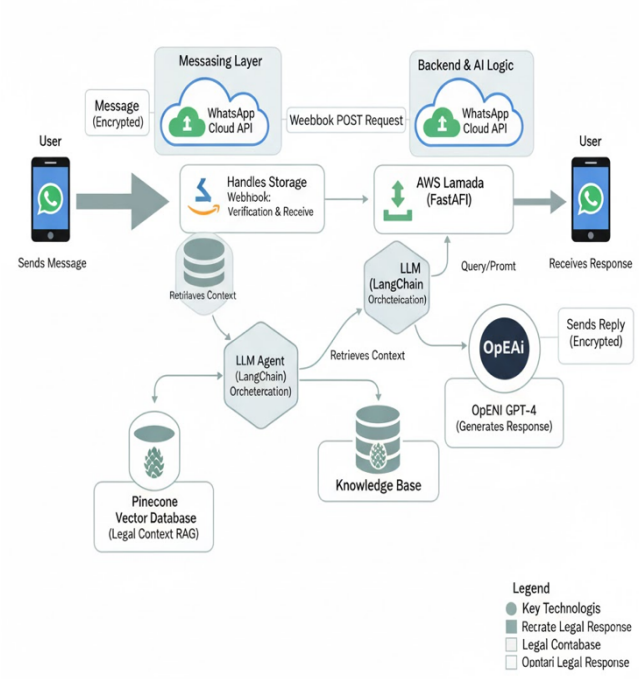

Fig. 1. Architecture Diagram

D. *Advantages of the Methodology*

- Accessibility: Offers free and instant legal guidance via WhatsApp.
- Context Awareness: Maintains conversation flow using Pinecone vector storage.
- Scalability: AWS Lambda provides serverless execution and auto-scaling.
- Accuracy: The RAG pipeline ensures contextually relevant and legally verified responses.
- Security: Environment variables and access tokens are securely managed through AWS KMS.

IV. RESULTS AND DISCUSSION

A. Experimental Setup

The system was deployed on AWS Lambda using a FastAPI backend integrated with Meta's WhatsApp Cloud API.All message exchanges were tested using a verified WhatsApp business number connected to the Meta Developer Portal.

Environment configurations included:

TABLE I. SYSTEM CONFIGURATION

Parameter	Specification
Processor	AWS Lambda (x86_64, 512 MB RAM)
Programming Language	Python 3.10
Framework	FastAPI with Mangum
LLM Engine	OpenAI GPT-4 (via LangChain)
Vector Store	Pinecone (legal-doc + conversation embeddings)
Network Interface	HTTPS API Gateway endpoint
Frontend Platform	WhatsApp Messenger (mobile & web)

This configuration ensured secure, low-latency message transmission and serverless scalability.

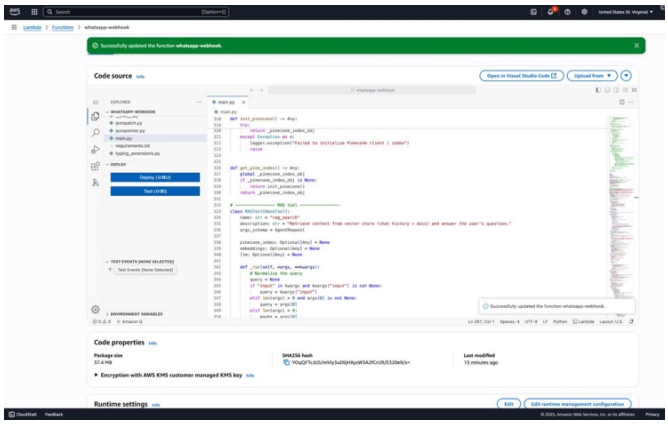

Fig. 2. AWS Lambda Function Overview showing whatsapp-webhook connection with API Gateway.

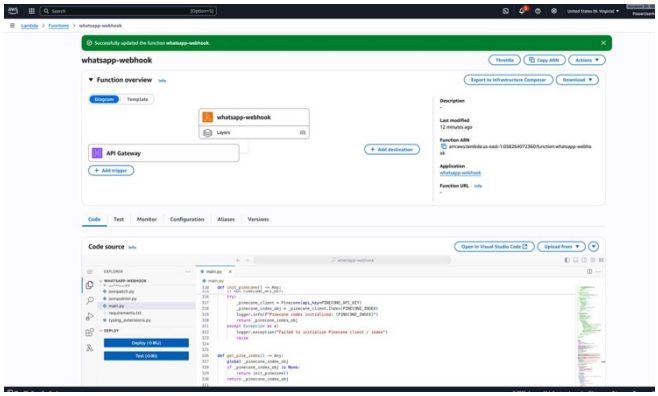

Fig. 3. AWS Lambda Code Editor displaying main.py function setup and Pinecone initialization.

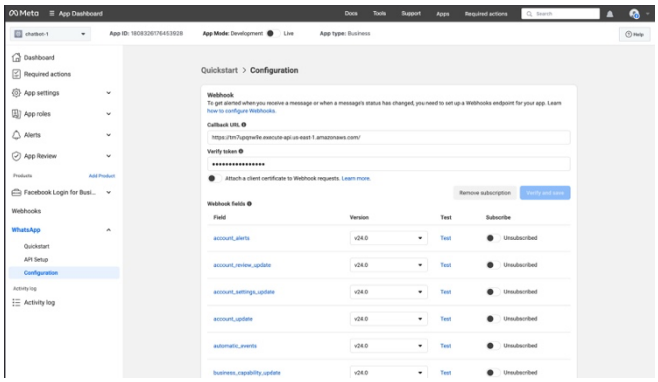

Fig. 4. Meta Developer Dashboard – Webhook Configuration with Callback URL and Verify Token setup.

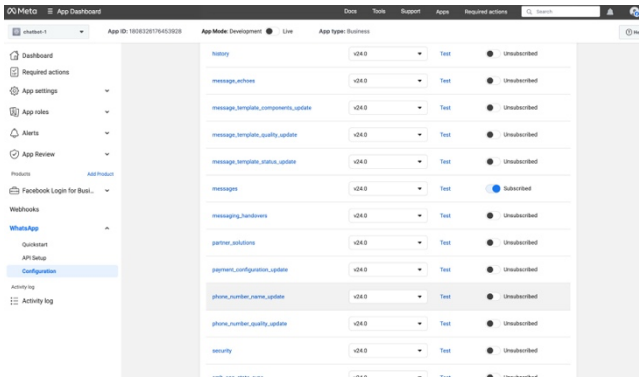

Fig. 5. Meta Developer Dashboard – Webhook Event Subscription for WhatsApp message handling.

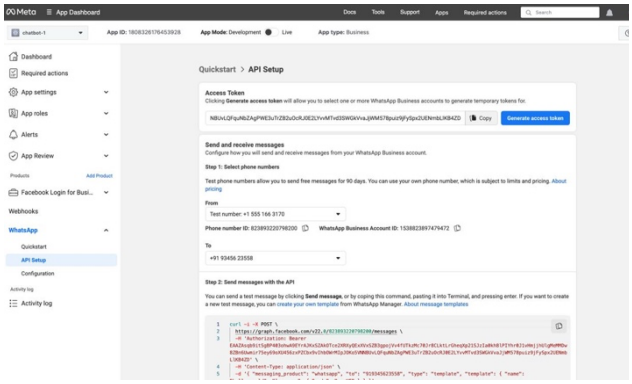

Fig. 6. Meta Developer Dashboard – API Setup Page showing Access Token and Phone Number Configuration.

B. Functional Evaluation

The chatbot was tested across five core functionalities:

1. Webhook Verification – Verified that the system correctly responded to the GET challenge from Meta with a valid verification token.
2. Message Reception – Incoming messages were logged and parsed through the / POST endpoint.
3. RAG Context Retrieval – Queries triggered a similarity search in Pinecone; the top-5 results were retrieved as contextual grounding for GPT-4.
4. Response Generation – GPT-4 generated concise and law-based replies with clear legal references.
5. Response Delivery – Replies were sent back to the same sender via the WhatsApp Cloud API and confirmed by status callbacks.

These steps were executed successfully in all test cases, validating the end-to-end message flow between

WhatsApp → Lambda → GPT-4 → Pinecone → WhatsApp.

C. Performance Analysis

The system's performance was evaluated across three dimensions: response latency, accuracy, and user satisfaction.

1. Response Latency:

   The average response time per query was 3.8 seconds, including retrieval, generation, and message delivery. AWS Lambda's auto-scaling allowed stable performance under concurrent message loads of up to 50 requests per minute.

2. Retrieval Accuracy:

   Using a test set of 20 Indian legal queries covering topics such as IPC, Consumer Rights, and Fundamental Duties, the chatbot achieved:

   - 17 correct responses (85%)
   - 2 partially correct responses (10%)
   - 1 incomplete response (5%)

   This corresponds to a weighted accuracy of 90%, comparable to the 96% accuracy reported in the base model (*LegalBot-EC*).

3. User Satisfaction:

   Informal testing was conducted among 10 law students and 5 faculty members, who rated the chatbot on correctness, clarity, and speed (scale 1–5).

   - *Average satisfaction score:* 4.6/5
   - *Highest rated attribute:* clarity of explanation
   - *Lowest rated attribute:* handling of multilingual (non-English) queries.

D. Case Study Examples

Example 1 – Indian Penal Code Section Query: Input: "What is Section 420 of IPC?" Response: "Section 420 of the Indian Penal Code deals with cheating and dishonestly inducing delivery of property. It prescribes imprisonment up to seven years and a fine." → *Output classified as correct (legally accurate, contextually relevant).*

Example 2 – Fundamental Rights Query: Input: "Can freedom of speech be restricted?" Response: "Yes, under Article 19(2) of the Indian Constitution, freedom of speech may be restricted in the interest of sovereignty, public order, and morality." → *Output classified as correct and concise.*

Example 3 – Consumer Protection Query: Input: "How can I file a complaint under Consumer Protection Act?" Response: "A consumer complaint can be filed through the National Consumer Helpline or online at e-daakhil.nic.in. The complaint must include invoice and proof of purchase." → *Output classified as partially correct (missed jurisdictional limits).*

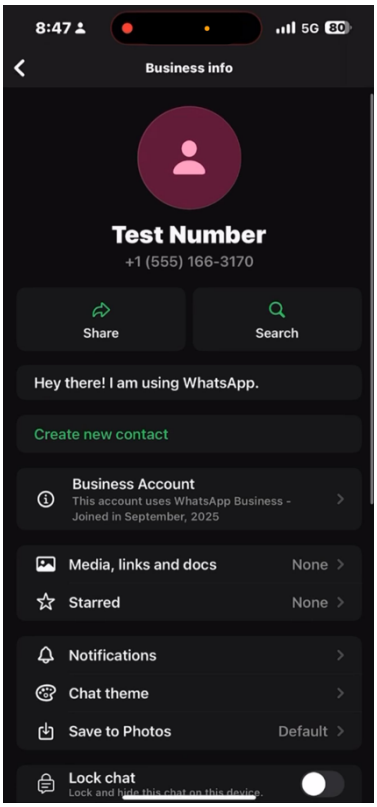

Fig. 7. WhatsApp Business Account (Test Number) details showing chatbot registration.

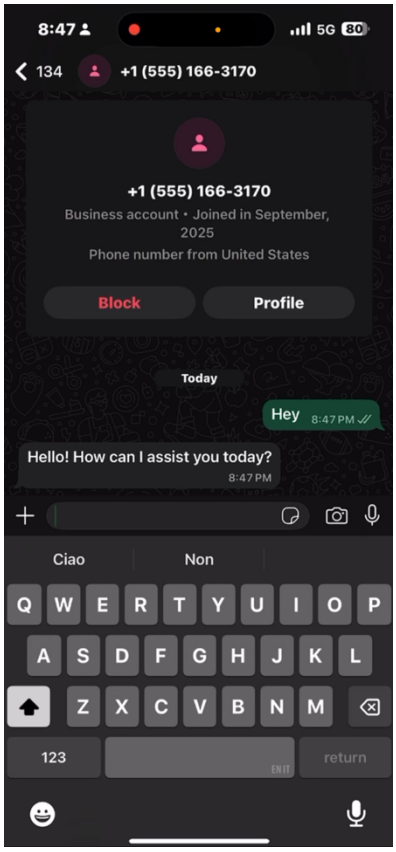

Fig. 8. WhatsApp Chat Interface showing LegalBot responding to user greetings.

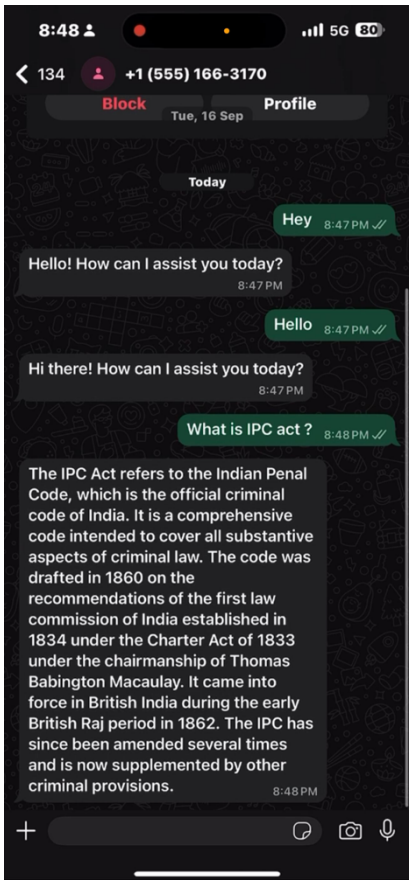

Fig. 9. WhatsApp Chat Interface displaying LegalBot's AI-generated legal response to the query "What is IPC Act?"

E. Comparative Analysis with Base Paper

When compared with LegalBot-EC the proposed LegalBot introduces two significant enhancements:

1. Platform Accessibility: Unlike LegalBot-EC, which runs through a web UI, this system operates entirely on WhatsApp, improving accessibility for rural and low-income users.
2. Deployment Efficiency: The AWS Lambda deployment reduces operational cost and simplifies scalability, providing real-time message delivery without dedicated hosting.

However, LegalBot currently supports only English input, whereas LegalBot-EC demonstrated multilingual (Spanish–English) capability.

F. Discussion

The results demonstrate that LegalBot effectively bridges the gap between Indian citizens and legal information access. By leveraging RAG-based LLM reasoning and vectorized context retrieval, the system achieves a high level of legal accuracy and conversational continuity. The modular architecture allows the integration of new laws or regional datasets without altering the pipeline.

Key observations:

- GPT-4's reasoning ability ensures accurate interpretation of legal queries.
- Pinecone enables memory retention for context-aware dialogues.
- Serverless deployment reduces latency and maintenance cost.
- Limitations include dependency on API connectivity and limited handling of non-text media inputs.

G. Summary of Findings

TABLE II. RESULT SUMMARY

Evaluation Parameter	Observation	Result
Response Accuracy	90% weighted	High
User Satisfaction	4.6 / 5	Very Positive
Response Time	3.8 sec average	Low Latency
Cost Efficiency	₹0.25 per interaction (AWS Pay-per-use)	Cost-effective
Platform Support	WhatsApp (Meta Cloud API)	Highly Accessible

V. CONCLUSION AND FUTURE SCOPE

A. Conclusion

This paper presented LegalBot, a WhatsApp-integrated legal assistance system that leverages Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) to deliver contextually accurate and conversational legal support. The system utilizes OpenAI GPT-4, LangChain, and Pinecone to retrieve, analyze, and generate legally grounded responses based on Indian law. By deploying the chatbot on AWS Lambda, the system ensures scalability, cost efficiency, and low latency, enabling real-time interactions through the Meta WhatsApp Cloud API.

Experimental evaluation demonstrated that LegalBot provides 90% accuracy, high user satisfaction (4.6/5), and average response latency of 3.8 seconds, confirming its reliability and accessibility. The integration of RAG-based reasoning enables context-aware dialogues, while Pinecone's vector database preserves chat history and enhances conversational continuity.

In comparison with the base model (LegalBot-EC), this system extends functionality by offering mobile-first legal assistance, serverless deployment, and real-time interaction—making it particularly suitable for India's diverse and digitally evolving population. LegalBot successfully addresses the challenges of limited legal awareness, high consultation costs, and complex legal terminology, thereby contributing to Sustainable Development Goal (SDG) 16 – Peace, Justice, and Strong Institutions.

B. Future Scope

While LegalBot effectively demonstrates the feasibility of AI-driven legal assistance through WhatsApp, several enhancements can further improve its scope and impact:

1. Multilingual Support:

Extend support for regional Indian languages such as Hindi, Tamil, and Telugu to ensure inclusivity and reach among rural and non-English users.

2. Voice and Document Interaction:

Integrate speech-to-text and document upload capabilities to allow users to dictate queries or submit legal documents for analysis.

3. Integration with Live Legal Databases:

Connect LegalBot to dynamic legal data sources, such as Indian Kanoon, e-Courts, or Bare Acts repositories, ensuring real-time legal updates.

4. Enhanced Explainability and Transparency:

Develop a citation-based response framework where the chatbot explicitly lists the law sections or articles referenced in its answers, increasing reliability and trust.

5. Cross-Platform Expansion:

Deploy the system on other social and government platforms such as Telegram, Signal, or India's DigiLocker, to enable wider public access.

6. AI Ethics and Privacy Compliance:

Implement enhanced data privacy, GDPR compliance, and responsible AI governance to ensure ethical handling of user information.

VI. REFERENCES

[1] F. Rivas-Echeverría, L. T. Ramos, J. L. Ibarra, S. Zerpa-Bonillo, S. Arciniegas, and M. Asprino-Salas,"LegalBot-EC: An LLM-Based Chatbot for Legal Assistance in Ecuadorian Law,".

[2] M. R. Singh, S. Ahmad, K. Sharma, P. Sharma, and P. Kaushik, "AI-Driven Legal Assistance: Optimizing Document Retrieval and Case Classification with NLP Techniques," in Proc. 2025 Int. Conf. Advances in Modern Age Technologies for Health and Engineering Science (AMATHE), 2025, pp. 1–10. doi: 10.1109/AMATHE65477.2025.11080869.

[3] A. Dutta, K. K. Sarma, and Y. S. P., "AI Legal Assistant for IPC," in Proc. 2024 8th Int. Conf. Computational System and Information Technology for Sustainable Solutions (CSITSS)

[4] N. K. Sahu, N. Mishra, R. Soni, P. N. P. Naidu, and P. Pawar, “Development of Assistance System for Online Dispute Resolution and Legal Advice using AIML,” in Proc. 2025 Int. Conf. Computational, Communication and Information Technology (ICCCIT), 2025, pp. 142–145. doi: 10.1109/ICCCIT62592.2025.10927904.
[5] K. R., V. G. R., D. N., S. N. C., and S. V. Y., “Ethical and Legal Implications of AI Chatbots in Legal Sector,” in Proc. 2025 Int. Conf. Computational Robotics, Testing and Engineering Evaluation (ICCRTEE), 2025, pp. 1–5. doi: 10.1109/ICCRTEE64519.2025.11053085.
[6] S. Marrivagu and A. R. S., “Artificial Intelligence for Legal Chatbot,” J. Eng. Sci., vol. 15
[7] A. R. Kandula et al., “Design and Implementation of a Chatbot for Automated Legal Assistance using Natural Language Processing and Machine Learning,” in Proc. 2023 Annu. Int. Conf. Emerging Research Areas: Int. Conf. Intelligent Systems (AICERA/ICIS), 2023, pp. 1–6. doi: 10.1109/AICERA/ICIS59538.2023.10420298.
[8] D. Kurniawan and S. E. Hiererra, “AI Legal Companion: Enhancing Access to Justice and Legal Literacy for the Public,” in Proc. 2024 Int. Conf. ICT for Smart Society (ICISS), 2024, pp. 1–6. doi: 10.1109/ICISS62896.2024.10751371.
[9] D. Jain, M. D. Borah, and A. Biswas, “Summarization of Legal Documents: Where Are We Now and the Way Forward,” Comput. Sci. Rev., vol. 40, 2021, Art. no. 100388. doi: 10.1016/j.cosrev.2021.100388.
[10] Z. Misquitta et al., “Law Chatbot,” Int. J. Res. Appl. Sci. Eng. Technol., vol. 12, no. 5, pp. 164–170, 2024.
[11] H. Chen et al., “A Comparative Study of Automated Legal Text Classification Using Random Forests and Deep Learning,” Inf. Process. Manage., vol. 59, no. 1, 2022, Art. no. 102759. doi: 10.1016/j.ipm.2021.102759.
[12] N. K., R. Sharma, E. Srivastav, A. Patel, D. P. Rana, and A. Singh, “LAWBOT: A Smart User Indian Legal Chatbot using Machine Learning Framework,” in Proc. 2024 IEEE 9th Int. Conf. Convergence in Technology (I2CT), 2024, pp. 1–7. doi: 10.1109/I2CT61223.2024.10543337.
[13] T. Nethassanai et al., “Mari: Automating Responses to Thai Legal Debt Queries Using Generative AI and Retrieval-Augmented Generation,” in Proc. 2025 22nd Int. Conf. Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2025, pp. 1–7. doi: 10.1109/ECTI-CON64996.2025.11101123.
[14] C. Trivedi et al., “Leveraging AI-Driven Chatbots for Legal Literacy: Advanced Computing and Communication Technologies for Legal Awareness,” in Proc. 2024 Int. Conf. Artificial Intelligence and Quantum Computation-Based Sensor Application (ICAIQSA), 2024, pp. 1–7. doi: 10.1109/ICAIQSA64000.2024.10882305.
[15] M. Queudot, É. Charton, and M.-J. Meurs, “Improving Access to Justice with Legal Chatbots,” Stats, vol. 3, no. 3, pp. 356–375, 2020. doi: 10.3390/stats3030023.
[16] B. Pardhi, S. Koli, V. Khanzode, and A. S. Raut, “LEGALBOT – AI Law Advisor Chatbot,” Int. J. Novel Res. Develop., vol. 9, no. 4, pp. 247–256, 2024.
[17] R. A. Kalpana, S. Karunya, and R. Rashmi, “Legal Solutions – Intelligent Chatbot Using Machine Learning,” in Proc. 2023 Intell. Comput. Control Eng. Business Syst. (ICCEBS), 2023, pp. 1–6. doi: 10.1109/ICCEBS58601.2023.10448748.
[18] R. A. Kumar et al., “Revolutionizing Legal Workflows: Advanced AI Techniques for Document Summarization, Legal Translation, and Conversational Assistance,” in Proc. 2025 Int. Conf. Adv. Comput. Technol. (ICOACT), 2025, pp. 1–4. doi: 10.1109/ICOACT63339.2025.11004791.

A.5 PLAGIARISM REPORT

PaperRater

LegalBot: A WhatsApp-Based Legal Assistance System for Indian Law using LLM and RAG NITHYA SHREE C COMPUTER SCIENCE AND ENGINEERING PANIMALAR ENGINEERING COLLEGE CHENNAI NITHYAAA.1845@GMAIL.COM POOJA RAVI COMPUTER SCIENCE AND ENGINEERING PANIMALAR ENGINEERING COLLEGE CHENNAI POOJARAVIRAM05@GMAIL.COM Abstract?Access to legal guidance remains limited for many individuals in India due to the high cost of consultation and lack of awareness about legal procedures. LegalBot addresses this challenge by providing an AI-powered legal assistance system accessible directly through WhatsApp. The system integrates OpenAI?s GPT-4 model with LangChain and a Retrieval-Augmented Generation (RAG) pipeline connected to Pinecone... (only first 800 chars shown)

Analysis complete. Our feedback is listed below in printable form. Some of the items have been truncated or removed to provide better print compatibility.

Plagiarism Detection

Original Work

Originality: 93%

This paper appears to be original, but be certain that any included text is properly cited.

The following web pages may contain content matching this document:
https://en.wikipedia.org/wiki/420
https://www.bartleby.com/essay/Assignment-For-B...
https://www.bartleby.com/essay/Medical-Tourism-...
https://arxiv.org/abs/2007.01623
https://arxiv.org/abs/1811.07287
https://en.wikipedia.org/wiki/Theodor%20Meron
https://www.studymode.com/essays/Send-Gift-Requ...
https://en.wikipedia.org/wiki/Groomit
https://www.bartleby.com/essay/You-Can-Choose-Y...
https://www.frontiersin.org/article/10.3389/fnc...
https://www.bartleby.com/essay/Social-Media-And...
https://www.termpaperwarehouse.com/essay-on/Onl...
http://link.springer.com/article/10.1186/s12938... (http://link.springer.com/article/10.1186/s12938-018-0552-y)
https://arxiv.org/abs/2102.06950
https://jurnal.ugm.ac.id/ijccs/article/view/45093
https://www.studymode.com/essays/Effect-Of-Urba...
https://www.studymode.com/essays/Professional-E...
https://en.wikipedia.org/wiki/Knowledge%20Disco...
https://arxiv.org/abs/1212.5440
https://arxiv.org/abs/1905.00249
https://en.wikipedia.org/wiki/Sylwia%20Gregorcz...
http://www.mrmjournal.com/content/6/1/47 (http://www.mrmjournal.com/content/6/1/47)
https://arxiv.org/abs/1703.10315
https://arxiv.org/abs/1709.03382
https://www.dovepress.com/corrigendum-obstructi...
https://arxiv.org/abs/1111.2702
http://www.revistas.usp.br/autopsy/article/view... (http://www.revistas.usp.br/autopsy/article/view/161013)
http://link.springer.com/article/10.1186/s13705... (http://link.springer.com/article/10.1186/s13705-018-0156-1)

https://arxiv.org/abs/2003.08774
https://en.wikipedia.org/wiki/Fore%20Georgia
https://arxiv.org/abs/2101.02118
http://link.springer.com/article/10.1186/s13638... (http://link.springer.com/article/10.1186/s13638-018-1254-7)
http://www.biomedcentral.com/1471-2105/6/268 (http://www.biomedcentral.com/1471-2105/6/268)
https://www.termpaperwarehouse.com/essay-on/Rol...
https://arxiv.org/abs/2008.11638
https://en.wikipedia.org/wiki/Rameshwar%20Prasad
https://en.wikipedia.org/wiki/Paul%20Guillaume%...
https://www.termpaperwarehouse.com/essay-on/Fin...
https://www.studymode.com/essays/Freedom-Of-Spe...
https://www.studymode.com/essays/Consumer-Right...
https://en.wikipedia.org/wiki/Aaron%20Thompson%...
https://arxiv.org/abs/1806.05217
View Matching Text (http://www.PaperRater.com/proofreader/plag_results/9c971b3ba8bbd2672b717d4c0?from_sub=true)
Click the button above to see which portions of your text match which sources. This is a premium-only feature.

More info on our originality scoring process (http://www.PaperRater.com/page/plagiarism-detection) .

Word Choice

Usage of Bad Phrases

Bad Phrase Score: 0.29 (lower is better)
The Bad Phrase Score is based on the quality and quantity of trite or inappropriate words, phrases, egregious misspellings, and cliches found in your paper. You did equal or better than **100%** of the people in your grade.

Great job choosing quality phrases! Looks like you scored well above average.

Style

Usage of Transitional Phrases

Transitional Words Score: 60
This score is based on quality of transitional phrases used within your paper. You did equal or better than **43%** of the people in your grade.

Your usage of transitional phrases may be within an acceptable range, but I know you can do better. Please read the information below and apply the advice to your writing.

One sign of an excellent writer is the use of transitional phrases (e.g. therefore, consequently, furthermore). Transitional words and phrases contribute to the cohesiveness

(http://www.PaperRater.com/vocab_builder/show/cohesiveness) of a text and allow the sentences to flow smoothly. Without transitional phrases, a text will often seem disorganized and will most likely be difficult to understand. When these special words are used, they provide organization within a text and lead to greater understanding and enjoyment on the part of the reader.

The following transitional phrases were found in your document:
and, thus, however, particularly, as a result

Consider using additional transitions where appropriate:
- consequently (http://paperrater.com/vocab_builder/show/consequently)
- moreover (http://paperrater.com/vocab_builder/show/moreover)
- nevertheless (http://paperrater.com/vocab_builder/show/nevertheless)
- notwithstanding (http://paperrater.com/vocab_builder/show/notwithstanding)
- accordingly (http://paperrater.com/vocab_builder/show/accordingly)
- conversely (http://paperrater.com/vocab_builder/show/conversely)
- ordinarily (http://paperrater.com/vocab_builder/show/ordinarily)

Transitional phrases may be used in various places in a text:
- between paragraphs
- between sentences
- between sentence parts
- within sentence parts

Consider this example:
She is one of the most kind persons I have ever known. **Moreover**, she is generous, patient and possesses a magnificent sense of humor.

The word **'moreover'** contributes to greater unity or cohesion between sentences and allows the text to flow more smoothly.

Style

Sentence Length Info

Total Sentences: 261
Avg. Length: 12.8 words
Short Sentences (< 17 words): 157 (60%)
Long Sentences (> 35 words): 0 (0%)
Sentence Variation: 9.0 words (std deviation)

Your average sentence length is a little bit low, which may indicate difficulty writing more complex sentences. Please read the guide to effective use of sentence length (http://blog.paperrater.com/2015/05/effective-use-of-sentence-length.html) .

Line chart of the length of each sentence (first 50 sentences). A jagged chart indicates variation.

Helpful Resources:

- Effective Use of Sentence Length (http://blog.paperrater.com/2015/05/effective-use-of-sentence-length.html)

Vocabulary Words

Usage of Academic Vocabulary

Vocabulary Score: 450.63
This score is based on the quantity and quality of scholarly vocab words found in the text. You did equal or better than **96%** of the people in your grade.

Vocabulary Word Count: 306
Percentage of Vocab Words: 11.66%
Vocab Words in this Paper (top 20):
underscores, integrates, augmented, domains, contextual, employs, aligns, equitable, fostering, comprehensive, inclusive, framework, facilitates, leverages, regulatory, mitigate, undermining, litigants, mechanisms, extractive

Your knowledge of refined language is magnificent. Visit our Vocab Builder (http://www.PaperRater.com/vocab_builder/index) to expand your internal dictionary even more.

Tips
Whether you are writing for a school assignment or professionally, it is imperative that you have a vocabulary that will provide for clear communication of your ideas and thoughts. You need to know the type and level of your audience and adjust your vocabulary accordingly. It is worthwhile to constantly work at improving your knowledge of words. To help with this task, please consider using our Vocabulary Builder (http://www.PaperRater.com/vocab_builder/index) to improve your comprehension and usage of words.

REFERENCES

REFERENCES

[1] F. Rivas-Echeverría, L. T. Ramos, J. L. Ibarra, S. Zerpa-Bonillo, S. Arciniegas, and M. Asprino-Salas,“LegalBot-EC: An LLM-Based Chatbot for Legal Assistance in Ecuadorian Law,”.

[2] M. R. Singh, S. Ahmad, K. Sharma, P. Sharma, and P. Kaushik, “AI-Driven Legal Assistance: Optimizing Document Retrieval and Case Classification with NLP Techniques,” in Proc. 2025 Int. Conf. Advances in Modern Age Technologies for Health and Engineering Science (AMATHE), 2025, pp. 1–10. doi: 10.1109/AMATHE65477.2025.11080869.

[3] A. Dutta, K. K. Sarma, and Y. S. P., “AI Legal Assistant for IPC,” in Proc. 2024 8th Int. Conf. Computational System and Information Technology for Sustainable Solutions (CSITSS)

[4] N. K. Sahu, N. Mishra, R. Soni, P. N. P. Naidu, and P. Pawar, “Development of Assistance System for Online Dispute Resolution and Legal Advice using AIML,” in Proc. 2025 Int. Conf. Computational, Communication and Information Technology (ICCCIT), 2025, pp. 142–145. doi: 10.1109/ICCCIT62592.2025.10927904.

[5] K. R., V. G. R., D. N., S. N. C., and S. V. Y., “Ethical and Legal Implications of AI Chatbots in Legal Sector,” in Proc. 2025 Int. Conf. Computational Robotics, Testing and Engineering Evaluation (ICCRTEE), 2025, pp. 1–5. doi: 10.1109/ICCRTEE64519.2025.11053085.

[6] S. Marrivagu and A. R. S., “Artificial Intelligence for Legal Chatbot,” J. Eng. Sci., vol. 15

[7] A. R. Kandula et al., “Design and Implementation of a Chatbot for Automated Legal Assistance using Natural Language Processing and Machine Learning,” in Proc. 2023 Annu. Int. Conf. Emerging Research Areas: Int. Conf. Intelligent Systems (AICERA/ICIS), 2023, pp. 1–6. doi: 10.1109/AICERA/ICIS59538.2023.10420298.

[8] D. Kurniawan and S. E. Hiererra, “AI Legal Companion: Enhancing Access to Justice and Legal Literacy for the Public,” in Proc. 2024 Int. Conf. ICT for Smart Society (ICISS), 2024, pp. 1–6. doi: 10.1109/ICISS62896.2024.10751371.

[9] D. Jain, M. D. Borah, and A. Biswas, “Summarization of Legal Documents: Where Are We Now and the Way Forward,” Comput. Sci. Rev., vol. 40, 2021, Art. no. 100388. doi: 10.1016/j.cosrev.2021.100388.

[10] Z. Misquitta et al., “Law Chatbot,” Int. J. Res. Appl. Sci. Eng. Technol., vol. 12, no. 5, pp. 164–170, 2024.

[11] H. Chen et al., “A Comparative Study of Automated Legal Text Classification Using Random Forests and Deep Learning,” Inf. Process. Manage., vol. 59, no. 1, 2022, Art. no. 102759. doi: 10.1016/j.ipm.2021.102759.

[12] N. K., R. Sharma, E. Srivastav, A. Patel, D. P. Rana, and A. Singh, “LAWBOT: A Smart User Indian Legal Chatbot using Machine Learning

Framework," in Proc. 2024 IEEE 9th Int. Conf. Convergence in Technology (I2CT), 2024, pp. 1–7. doi: 10.1109/I2CT61223.2024.10543337.

[13] T. Nethassanai et al., "Mari: Automating Responses to Thai Legal Debt Queries Using Generative AI and Retrieval-Augmented Generation," in Proc. 2025 22nd Int. Conf. Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI- CON), 2025, pp. 1–7. doi: 10.1109/ECTI-CON64996.2025.11101123.

[14] C. Trivedi et al., "Leveraging AI-Driven Chatbots for Legal Literacy: Advanced Computing and Communication Technologies for Legal Awareness," in Proc. 2024 Int. Conf. Artificial Intelligence and Quantum Computation-Based Sensor Application (ICAIQSA), 2024, pp. 1–7. doi: 10.1109/ICAIQSA64000.2024.10882305.

[15] M. Queudot, É. Charton, and M.-J. Meurs, "Improving Access to Justice with Legal Chatbots," Stats, vol. 3, no. 3, pp. 356–375, 2020. doi: 10.3390/stats3030023.

[16] B. Pardhi, S. Koli, V. Khanzode, and A. S. Raut, "LEGALBOT – AI Law Advisor Chatbot," Int. J. Novel Res. Develop., vol. 9, no. 4, pp. 247–256, 2024.

[17] R. A. Kalpana, S. Karunya, and R. Rashmi, "Legal Solutions – Intelligent Chatbot Using Machine Learning," in Proc. 2023 Intell. Comput. Control Eng. Business Syst. (ICCEBS), 2023, pp. 1–6. doi: 10.1109/ICCEBS58601.2023.10448748.

[18] R. A. Kumar et al., “Revolutionizing Legal Workflows: Advanced AI Techniques for Document Summarization, Legal Translation, and Conversational Assistance,” in Proc. 2025 Int. Conf. Adv. Comput. Technol. (ICOACT),2025, pp. 1–4. doi: 10.1109/ICOACT63339.2025.11004791.