Criminal Procedure Assistant: A Generative AI Legal Interface

Author: Tanish Kandivlikar
Course: DS552 – Generative AI
Instructor: Narahara Chari Dingari, Ph.D.
Date: April 20, 2025

1. Introduction and Objective

1.1 Introduction

The Criminal Procedure Assistant is a web-based legal interface that allows users—law students, legal practitioners, and laypersons—to query the Massachusetts Rules of Criminal Procedure. The application delivers accurate, citation-backed answers in a structured IRAC format, facilitating better comprehension and legal research.

1.2 Objective

• To provide a reliable, intuitive interface for legal queries.
• To leverage Retrieval-Augmented Generation (RAG) for grounded answers.
• To return results in IRAC format (Issue, Rule, Application, Conclusion) for legal clarity.

1.3 Suitability of Generative Model

OpenAI's GPT-4 model is employed through the `langchain_openai.ChatOpenAI` wrapper. It is ideal for:
• Understanding natural legal language queries.
• Generating coherent IRAC-structured answers.
• Seamlessly integrating retrieved legal context into responses.

2. Selection of Generative AI Model

2.1 Chosen Model

The model used is GPT-4, accessed via `ChatOpenAI` in the application backend (`app.py`).

2.2 Justification

GPT-4 provides state-of-the-art performance in legal reasoning, offering both low latency and high-quality natural language generation suited for real-time legal queries.

3. Project Definition and Use Case

3.1 Application Concept

The application is a Flask-based web platform that features:
• `/` route: renders a user-friendly chat interface (chat.html).
• `/get` route: processes user input and dispatches either a direct lookup or RAG-based response.

3.2 Integration of the Generative Model

• PDF ingestion via `store_index.py`, which loads Massachusetts legal PDFs, splits them using `text_split`, embeds them using HuggingFace transformers, and stores them in Pinecone.
• A RAG chain is created using `create_retrieval_chain` in `app.py`, with prompts provided in `src/prompt.py`.
• Direct lookup capability is also supported using regex patterns (e.g., 'Rule X.Y') and helper functions from `src/helper.py`.

4. Implementation Plan

4.1 Technology Stack

• Python 3.10, Flask, LangChain, Pinecone, OpenAI API
• Embeddings: `sentence-transformers/all-MiniLM-L6-v2`
• Frontend: HTML (Jinja2 templates), CSS

4.2 Web Framework

Flask is used for the backend, while Jinja2 templates power the frontend. Asynchronous interaction is enabled using AJAX requests.

4.3 Development Steps

1. Ingest PDFs and upsert embeddings to Pinecone using `store_index.py`.
2. Construct the core logic in `src/helper.py` and `src/prompt.py`.
3. Create Flask routes in `app.py` for chat and metrics endpoints.
4. Build UI with `chat.html` and `metrics.html`.
5. Containerize using Docker.
6. Deploy to AWS using GitHub Actions (CI/CD).

5. Model Evaluation and Performance Metrics

5.1 Inference Time

Measured using `MetricsCallbackHandler`. Performance goal:

• Direct lookups < 2 seconds

• RAG queries < 3 seconds

5.2 Resource Usage

Runs on CPU-based EC2 instances, minimal RAM (~300MB). OpenAI and Pinecone handle external compute-heavy tasks.

5.3 Accuracy Metrics

• RAG answers rated by humans on a 1–5 scale with ≥4.0 average target for clarity and correctness. (Human Feedback)

5.4 User Experience

Metrics dashboard at `/metrics` provides insights on request count, response time, and token usage.

6. Deployment Strategy

6.1 Hosting Options

• AWS EC2 + ECR with CI/CD pipeline via GitHub Actions.

• Alternative: Streamlit Cloud or Hugging Face Spaces.

6.2 Accessibility

Accessible via public URL (http://3.20.132.114:8080) with no authentication required.

6.3 User Flow

1. Navigate to the homepage.
2. Enter a question or rule citation.
3. Receive an IRAC-based answer.
4. Monitor usage on `/metrics`.

7. Expected Outcomes and Challenges

7.1 Expected Impact

• Streamlined access to legal procedures.

• Effective educational tool for students and professionals.

7.2 Challenges

• API rate limits for OpenAI/Pinecone
• Managing long and ambiguous queries
• Ensuring retrieved chunks are relevant

7.3 Mitigation

• Query chunking and summarization.
• Caching frequent questions.
• Fine-tuning prompts for better RAG relevance.

8. Resources Required

8.1 Software

Python dependencies (see `requirements.txt`), OpenAI and Pinecone accounts.

8.2 Hardware / Cloud

• EC2 t2.medium or larger instance
• Internet-accessible compute environment

9. Conclusion

The Criminal Procedure Assistant demonstrates how generative AI can transform legal research through real-time, structured reasoning. It combines document retrieval, IRAC-style generation, and performance metrics in a production-grade deployment pipeline.