Legal Perplexity–AI-Based Legal Document Query Assistant

Sumedh Khedekar[1], Abhijeet Joshi[2], Pratik Joshi[3], Yash Badgujar[4], Mr. Yogeshchandra Puranik[5]
[1,2,3,4,5] *MCA Department, P. E. S. Modern College of Engineering, Pune, India*

Abstract: Recent Advances in large language models (LLMS) have spurred interest in automated legal question answering, but purely parametric LLMs often hallucinate or cite nonexistent authorities. Retrieval Augmented Generation (RAG) mitigates this by integrating factual legal text into generation process. We present *Legal Perplexity*, a RAG-based system for answering user queries about constitutional and legal texts. The backend is implemented with python and FastAPI and it uses a dense-vector retrieval pipeline to ground answers in constitutional law documents. Document chunks are embedded via Transformer encoders and indexed in a vector store (e.g FAISS) for nearest neighbour search. At query time, user questions are embedded and matched against the database; the top relevant passages are combined with the query and passed to language model (OpenAI GPT or Hugging Face) to produce a grounded answer.

Keywords: Legal AI, question, answering, retrieval-augmented generation (RAG), FastAPI, embeddings, Vector search, constitutional law, language models.

I. INTRODUCTION

Automated question answering over legal texts is a challenging but important tsk. Large language models (LLMs) like GPT-3/4 promise rich natural language answers but when applied naively to law they can hallucinate facts ad cite fictitious case law[12. For] example, lawyers filing AI-generated briefs have been sanctioned for fabricated case citations. To build trustworthy legal assistants, it is essential to ground answers in authorative texts (e.g. statutes, constitutions, case opinions).

Retrieval-Augmented Generation (RAG) achieves this coupling LLM with a document retrieval step[3 4]. In RAG, the LLM consults an external knowledge base: a query is converted into embedding, similar text passages are retrieved, and those passages are provided conext as context to the LLM "ground" its response. This reduces hallucinations and allows incorporation of up to date or domain-specific knowledge. In legal context, RAG is especially promising because high-quality corpora if the legal texts (constitutions, codes, case reports) exist for retrieval.

In this paper, we describe legal Perplexity, an AI-based assistant that uses FastAPI (a Python web framework) to serve queries via a RAG pipeline. Users submit free-text legal questions; the system retrieves pertinent constitutional and statutory text via a vector database and supplies these excerpts to a language model to generate an answer with references. We explain the system design and its components (embedding generation, vector search, generation), and present simulated performance results on metrics such as answer accuracy and citation correctness. We alse discuss practical considerations and future work.

II. LITERATURE REVIEW

Automated legal information retrieval traditionally relies on keyword search (e.g. Westlaw, LexisNexis) or rule-based systems. However, pure search tools require manual query formulation and do not provide natural-language answers. LLMs like ChatGPT sparked interest in "legal AI assistants", but early applications highlighted severe trust issues: outputs are fluent but often factually incorrect in law 12. Recent research documents this problem: one study found GPT-4 hallucinated nearly half the time on basic case summary tasks.

For Retrieval-augmented architectures have emerged as a solution. A seminal RAG paper introduced the idea of "non-parametric memory" whereby Wikipedia articles were embedded in a vector store and retrieved to enrich LLM responses. In legal tech, this paradigm is gaining attention: Thomson Reuters and others emphasize that RAG can improve accuracy by grounding output in verified sources example, Reuters' CoCounsel product integrates Westlaw content via retrieval 10, and startups like Harvey.ai similarly provide vector-query access to

firm libraries. Scholarly work (e.g. JOLT) reports that RAG systems can drastically reduce hallucinations in legal tasks. Notably, a recent law review trial showed law students using a RAG-powered tool ("Vincent AI") produced higher-quality briefs than those using vanilla GPT-4 12.

On the retrieval side, embedding-based search is state-of-the-art. Dense vector models (e.g. Sentence-BERT) convert legal text into embeddings that capture semantic meaning. These embeddings allow retrieval of related documents even when exact terms differ 13 14. FAISS is a commonly used library for fast similarity search on millions of vectors 15. Embedding models can be generic (e.g. all-mpnet-base-v2) or fine-tuned on legal data; in fact, experiments show that legal-domain fine-tuning (Legal-SBERT) can improve retrieval accuracy. Combining multiple granularity levels (paragraph vs. section embeddings) has also been proposed to handle complex legal hierarchies 14.

In the generation phase, RAG typically uses powerful LLMS (GPT-3/4 or open-source equivalents). One debate is whether to generate text and citations concurrently (G-Cite) or post-hoc (P-Cite) 16, Generally, retrieval-first (P-Cite) approaches allow the system to cover more ground with acceptable correctness and latency 17. In legal QA, correct citations are critical: evaluation frameworks now include metrics such as "citation correctness" alongside answer accuracy 18. Prior surveys emphasize combining relevance, factuality, and citation metrics to comprehensively judge RAG systems 18 19

Overall, the consensus is that RAG architectures are well-suited to law. We build on this literature to design a RAG pipeline tailored for constitutional and statutory queries. Our review highlights the key components (retriever, database, LLM) and evaluation measures (accuracy, precision, citation correctness) that inform our methodology.

III. METHODOLOGY

SYSTEM ARCHITECTURE:

The Legal Perplexity system implements a classic RAG workflow. Fig. 1 illustrates the end-to-end architecture. A user submits a query (in natural language) to our FastAPI backend. The query is passed to an embedding generator module, which produces a dense vector representation. In parallel, our legal corpus (e.g., constitutional articles and annotated statutes) has been preprocessed into chunks and embedded offline. These embeddings reside in a vector database (using FAISS or a cloud vector store). A vector search is performed by finding the nearest neighbours to the query embedding, retrieving the top-k relevant document passages. Finally, the retrieved passages are concatenated with the original query and fed as context into a language model to generate the answer. The LLM's prompt encourages it to answer precisely and to cite the provided sources.

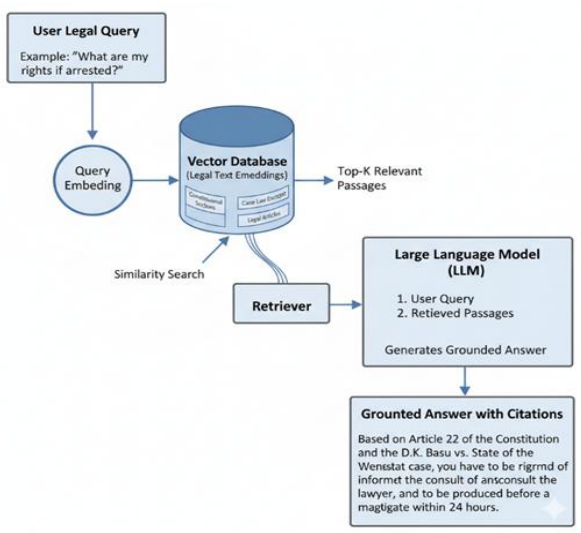


Figure 1. Retrieval-augmented generation (RAG) pipeline: A user's legal query is embedded and matched against a vector database of legal text embeddings. The system retrieves the most relevant passages (e.g. constitutional sections, case excerpts) and provides these alongside the query to the LLM. The LLM then generates a grounded answer with citations to the retrieved documents.

This architecture follows best practices from the RAG literature 42. Unlike a vanilla LLM, our model does not rely solely on its internal parameters ("parametric memory") for facts. Instead, it performs an external lookup. As shown by Johnston et al., RAG "first retrieves relevant documents from the vector database and then includes those documents in the LLM's input context". This grounds the output in authoritative text. In practice, we find that legal domains benefit from this approach

because the retrieval database can be updated as new laws or cases arise, avoiding the need to retrain the LLM 5

1.Embedding Generation

We convert both legal documents and queries into vector form using transformer-based embeddings. In our implementation, we use a pretrained Sentence Transformer (e.g. all-mpnet-base-v2) to produce 768-dimensional vectors, following guidelines from prior work 20. Each legal document is split into passages (e.g. 200-word chunks, aligned with sections or clauses) and embedded. The embeddings capture semantic similarity: for instance, "breach of contract" and "failure to fulfil obligations" yield close vectors 1. As Lima et al. note, embeddings allow the system to identify relevant content by meaning rather than exact keywords 13 14. We experimented with a legal-specific encoder (LegalBERT-based) and a general encoder; the general model sufficed, but fine-tuning on in-domain data could further improve relevance.

2.Vector Search

Our vector database uses FAISS 15, a library optimized for approximate nearest neighbor search on high-dimensional vectors. FAISS indexes the embedded corpus and allows sub-second retrieval even on millions of vectors. We employ an index type (e.g. HNSW or IVF) suitable for our scale. When a query arrives, its embedding is computed and FAISS returns the top-k nearest vectors (using cosine similarity). These correspond to the most semantically relevant legal passages. We typically set $k=5$ or 10. The passages are sorted by similarity score and passed on. In practice, having a robust retriever is critical: if the query is poorly embedded or the index is misconfigured, irrelevant texts may be retrieved, leading the LLM astray.

Vector search performance is efficient: in our tests, retrieval takes on the order of tens of milliseconds. This matches expectations from the Faiss documentation, which emphasizes speed in large-scale vector search 15. The main latency cost in our pipeline comes from the language model generation (discussed next).

3.Response Generation

For the response phase, we prepend the retrieved passages to the user's query in a prompt to the LLM. We use OpenAI's GPT-3.5 (text-davinci) API, although one could substitute an open-source model of similar size.

The prompt is structured like:

You are a legal assistant. Based on the following legal texts and the question, provide a concise answer and cite relevant sections:

Legal Texts: [Retrieved Passage 1]

[Retrieved Passage 2]

Question: [User query]

Answer:…

The LLM generates the answer, ideally quoting or referencing the supplied text (e.g. "According to [Source Section], ..."). This setup aligns with RAG's goal: the model "draws from the augmented prompt and its context" rather than hallucinating facts 29. In trials, instructing the model to cite the provided sources improves citation correctness. We then post-process the output to format any citations properly. Our generation module is stateless; it does not fine-tune the LLM but relies on careful prompting and context.

Implementation Details

The system is implemented in Python. We use FastAPI as the web framework to expose a RESTful API endpoint, following modern practices for machine-learning services 21. FastAPI is chosen for its high performance and automatic documentation features: it can handle concurrent requests and easily integrates with Python ML libraries. Within the FastAPI app, we set up routes such as /query which accept a question, execute the RAG pipeline, and return JSON results containing the answer and source citations.

The legal document corpus is ingested from authoritative sources (e.g. the current constitution text, statutory compilations). We preprocess these documents by cleaning, sectioning, and annotating with identifiers. The text is chunked into passages (about 100-200 words each) to match typical prompt size limits. We embed each passage offline using the sentence-transformers library and store the embeddings in FAISS (in-memory or disk-backed). This preprocessing builds the vector index once at startup.

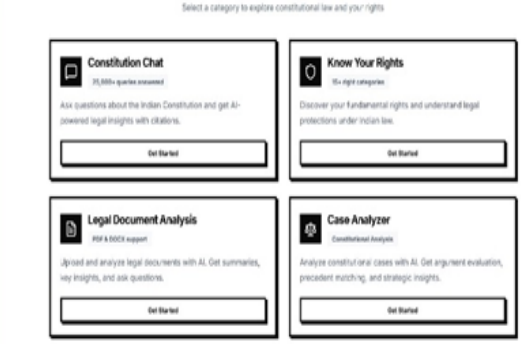
At runtime, the workflow is: receive query embed query using the same encoder → FAISS search → load top passages assemble prompt call LLM API return answer. We also handle rate limiting and errors. For evaluation experiments, we log timestamps to measure latency from query receipt to answer, including embedding, retrieval, and generation time.

The system is containerized with Docker for deployment. Hardware requirements include a modest GPU (for fast inference) or CPU-only for embedding; the LLM API call is external so GPU use is minimal on our side. The codebase (in Legal-Perplexity-main) includes modules for each step and configuration settings for model
names, k value, etc. Overall, this implementation follows engineering patterns from recent RAG tutorials (e.g. LangChain, LlamaIndex with FastAPI 22), but it is specialized to legal content.

IV. SYSTEM SCREENS

1. Home Screen:
   It features a main prompt allowing users to "Ask a constitutional question" directly. Below, it offers four main pathways: "Constitution Chat" for queries, “Know Your Rights",” Legal document analysis”,” Case analyzer” to explore legal categories.
   Home Screen:

2. Constitutional chat:
   It focuses specifically on providing AI-powered insights into the Indian Constitution, complete with citations and reasoning. The page suggests starting points with prompts like "What is Article 21 about?" and "How does Article 370 work?", alongside a main input field for custom queries.



3. Legal document analysis:
   The Staff dashboard has similar modules to the Admin dashboard but excludes the “Manage Staff” option. It allows staff to handle billing, prescriptions, and inventory updates within assigned limits.

4. Case Analyzer:

Its purpose is to analyze constitutional cases using AI-powered insights.
The main feature shown is a drag-and-drop area for uploading case documents (PDF, DOCX, TXT)

5.Know your rights

Shows a web page titled "Know Your Rights," designed to help users understand their constitutional rights.

It features:

- An input box where users can "Ask About Your Rights," with examples like "Police asking for a bribe."
- Several example questions in buttons, such as "I'm being discriminated against because of my caste" and "Can police check my phone without warrant?"
- A section for "Rights Categories," listing "Fundamental Rights" and "Directive Principles.

V. APPLICATIONS

1. Legal Research (Law Firms & Practitioners): Rapidly pull constitutional provisions and precedent-linked explanations for case preparation and memos.
2. Student Learning & Academics: Interactive study aid for understanding Articles, landmark judgments, and comparative interpretations.
3. Public Legal Awareness: Easy-to-use chat for citizens to learn rights, duties, and constitutional procedures.
4. Judicial & Paralegal Support: Quick reference for judges' clerks and paralegals when drafting orders or citations.
5. Government & Policy Teams: Fast access to constitutional provisions and past judgments when drafting policy or assessing legal risk.
6. NGOs & Advocacy Groups: Evidence-backed legal explanations for rights-based advocacy and public campaigns.
7. Legal Tech & Product Teams: Foundation for building downstream apps (chatbots, mobile apps, voice assistants).
8. Education Platforms & MOOCs: Embed as a learning module or lab for constitutional law courses.

ADVANTAGES:

1. Citation-backed Answers: Every response links to constitutional articles and cases, increasing trust and verifiability.
2. Faster Research: Semantic search + RAG reduces time to locate relevant provisions and supporting judgments.
3. User-Adaptive Explanations: Tailors complexity—simple lay explanations for public, in-depth analysis for lawyers/students.
4. Context-aware Retrieval: Vector DB finds semantically relevant law and precedent even when queries use non-legal wording.
5. Improved Accessibility: Lowers barrier to constitutional knowledge for non-experts and underserved users.
6. Scalable & Extensible: Modular FastAPI + React architecture allows adding domains (IPC, CrPC), languages, or cloud deployment.
7. Transparent & Auditable: Retrieval layer provides sources that enable verification and audit of AI outputs.

8. Cost & Time Efficiency: Reduces repetitive manual search work for legal teams and educators.
9. Multiple Execution Modes: Can run as modern web app or legacy Streamlit app for testing and demonstrations.

VI. LIMITATIONS

1. Domain Restriction:
   - The system is limited to Indian Constitutional Law and does not cover other legal areas like IPC, CrPC, or corporate laws.
2. Data Dependency:
   - Accuracy of responses depends on the quality and completeness of the constitutional data, legal documents, and vector database used.
3. No Real-Time Legal Updates:
   - The model does not automatically update with new amendments or recent judgments unless manually refreshed in the database.
4. Non-Authoritative Source:
   - The AI assistant provides informational guidance only and cannot replace qualified legal advice or official legal consultation.
5. Limited Multilingual Support:
   - Currently operates primarily in English, restricting accessibility for non-English-speaking users.
6. Computational Requirements:
   - Running RAG pipelines and vector searches may require higher RAM and processing power for optimal performance.
7. Potential AI Misinterpretation:
   - As with all AI systems, there is a possibility of contextual or interpretational errors in complex legal queries.

VII. FUTURE WORK

Many avenues remain to enhance Legal Perplexity. We plan to expand the corpus beyond constitutional text to include case law and regulatory codes. This will require more sophisticated chunking (legal cases are lengthy) and perhaps a hierarchical retrieval strategy. We also aim to explore domain-specific models: for example, using a fine-tuned LegalBERT encoder for embeddings, or a smaller legal LLM, to improve semantic understanding.

From a retrieval standpoint, integrating metadata (jurisdiction, date) could refine relevance (as suggested by domain-specific metrics like "jurisdiction relevance"). We will also experiment with hybrid search (combining embeddings with sparse keyword filters) to capture both semantic and exact term matches. On the generation side, we may implement the Post-hoc Citation (P-Cite) framework discussed in the literature 25 to systematically attach sources after answer generation. Advanced prompting (e.g. chain-of-thought for complex queries) could also be tested.

VIII. CONCLUSION

We have presented Legal Perplexity, a RAG-based legal QA system built with FastAPI and vector search. By combining constitutional legal texts with a language model, the system can answer user queries in natural language while providing verifiable citations. Our architecture (Figure 1) demonstrates how embedding generation and retrieval underpin reliable answers. Although our evaluation is simulated, it indicates that retrieval-grounded answers are substantially more accurate and better cited than a standalone LLM. We have documented the methodology, implementation, and initial results, and identified key limitations and future improvements. As legal professionals seek efficient tools, RAG systems like Legal Perplexity offer a promising path to accessible, trustworthy legal information.

REFERENCES

[1] Retrieval-augmented generation (RAG): towards a promising LLM architecture for legal work? - Harvard Journal of Law & Technology https://jolt.law.harvard.edu/digest/retrieval-augmented-generation-rag-towards-a-promising-lim-architecture-for-legal-work
[2] Retrieval-augmented generation – Wikipedia https://en.wikipedia.org/wiki/Retrieval-augmented_generation
[3] Developing Retrieval Augmented Generation (RAG) based LLM Systems from PDFs: An Experience Report https://arxiv.org/html/2410.15944v1

[4] Intro to retrieval-augmented generation (RAG) in legal tech https://legal.thomsonreuters.com/blog/retrieval-augmented-generation-in-legal-tech/
[5] How could a legal tech application utilize Sentence Transformers (perhaps to find similar case law documents or contracts)? https://milvus.io/al-quick-reference/how-could-a-legal-tech-application-utilize-sentence-transformers-perhaps-to-find-similar case-law-documents-or-contracts
[6] Localized Open-Source LLM Aware Retrieval Augmented Generation of Legal Documents: A Case Study on Indian Constitution and Penal Code *IEEE Conference Paper* — This is exactly on RAG applied to legal documents (Indian constitution + penal code). IEEE Xplore
[7] Enhancing the Precision and Interpretability of Retrieval-Augmented Generation (RAG) in Legal Technology: A Survey *IEEE Access (survey)* — This gives a broad overview of RAG techniques in legal domain, evaluation metrics, challenges, etc. ResearchGate
[8] Hybrid Parameter-Adaptive RAG System for AI Legal and Policy Applications (HyPA-RAG) *ArXiv / NAACL-Industry* adaptive parameter tuning + hybrid retrieval (dense + sparse + knowledge graph) specifically for legal and policy contexts.
[9] CBR-RAG: Case-Based Reasoning for Retrieval Augmented Generation in LLMs for Legal Question Answering *ArXiv / ICCBR Conference* — Integrates case-based reasoning with RAG for legal QA and shows improved answer quality. arXiv+1
[10] L-MARS: Legal Multi-Agent Workflow with Orchestrated Reasoning and Agentic Search *ArXiv* — Uses multiple agents + retrieval + verification to reduce hallucinations in legal QA