ChatGPT

File: prompt.md

Next-Generation Indian Legal AI Assistant (MVP)

Overview

We are building a **state-of-the-art AI legal assistant** tailored for **Indian law** – an "Indian Harvey AI" that surpasses existing solutions in accuracy, reliability, and usefulness for lawyers in India. This MVP will be a **web-based application** integrating OpenAI's top-tier GPT model (GPT-4/GPT-5 via API) with a robust **Retrieval-Augmented Generation (RAG)** pipeline. The system will leverage a **legal knowledge base** (Indian statutes, case law, regulations, and user-provided documents) to ensure that answers are **grounded in authoritative sources** [1] . Our focus is on **accuracy, trust, and up-to-date knowledge**, making the assistant an indispensable tool for law practitioners. Scalability, security, and seamless user experience are key goals, as we plan to deploy on **Azure cloud** for enterprise-grade reliability and compliance.

Key Features and Use Cases (MVP Scope)

- **Intelligent Legal Q&A Chatbot:** A chat interface where users (lawyers) can ask **legal questions** and get **concise, accurate answers with citations** to Indian legal sources. The assistant handles queries about **case law, statutes, regulations, and procedures** in the Indian context (e.g. "What are the grounds for bail under the CrPC?"). It uses RAG to fetch relevant text from the knowledge base, ensuring responses are **grounded in actual law** [1] and not hallucinated.
- **Document Analysis and Summarization:** Ability to **upload legal documents** (e.g. contracts, petitions, case judgments) and ask questions about them or get summaries. For example, a lawyer could upload a lengthy contract and query "Identify any unusual indemnity clauses in this contract" – the system will analyze the document and provide an answer referencing the text. This feature helps with due diligence, contract review, and case preparation by extracting key insights quickly.
- **Drafting Assistance:** Assist in drafting legal documents (limited scope in MVP). The AI can generate a **first draft of a contract clause, legal notice, or brief** based on user instructions. For instance, "Draft a clause for termination in a service agreement under Indian law" would yield a draft clause which the lawyer can refine. The model will pull in relevant legal language or standards from the knowledge base (e.g., citing the Indian Contract Act if needed) to enhance the draft's quality.
- **Citations and Source Verification:** For every answer or drafted text, the system provides **references to source materials** (case reports, sections of acts, etc.) to enhance trust. Lawyers can click citations to view the referenced text. This not only builds confidence but also allows quick fact-checking. The assistant's prompt is designed to **require source attribution**, mitigating hallucinations. *In high-stakes legal usage, trust is as important as accuracy – our system is designed with verification in mind* [2] .
- **Indian Legal Knowledge Base:** The knowledge base will encompass **Indian central and state laws, recent judgments, and regulations**. It includes:
- Key **statutes** (e.g. Constitution of India, IPC, CrPC, Contract Act, etc.).
- **Case law** from Supreme Court and major High Courts, especially landmark and recent cases (with an update mechanism to continually add new judgments from 2024, 2025, etc.).
- **Regulations and notifications** from relevant authorities.

- Optionally, **firm's internal knowledge** (if a user/law firm uploads their own templates or past work, kept private to them).
- **Multilingual Query Support (future):** While the primary interface and knowledge base will be in English (the primary language of Indian judiciary and law), the system will tolerate queries in **common Indian languages** (like Hindi) by translating them to English before search, and can translate answers back, if needed. This ensures accessibility for users more comfortable with vernacular questions, though answers will largely reference English legal texts (MVP will focus on English; multilingual support can expand later).

System Architecture

The system follows a **client–server architecture** with a separation of concerns for scalability and maintainability: - **Frontend (Web App):** A dynamic web application (e.g. React/Next.js or Angular) providing an intuitive **chat interface** and document upload UI. Key UI elements: - Chatbox with conversation history, supporting **markdown** answers (so that the assistant's output – including lists or formatted text – appears clearly) and clickable citation links. - File upload widget to submit documents for analysis. The UI may allow selecting whether an answer should use only the uploaded file, the general knowledge base, or both. - A panel to display retrieved source texts when the user wants to inspect the context (for transparency). - **User controls**: e.g. option to regenerate answer, to stop the response, or to switch query mode (Q&A vs. drafting vs. summarization). - (Future/enterprise) Integration points for Microsoft 365: e.g., ability to import a document from SharePoint or save an answer to Word. *(This is beyond MVP, but we note it for roadmap given Harvey's success integrating into Word/SharePoint* [3] *.)* - **Backend (API Server):** A server (likely Python with FastAPI, or Node.js, etc.) that exposes RESTful endpoints for the frontend: - `/api/query`: Accepts a user query (and optionally context like selected document IDs) and orchestrates the RAG process (detailed below), returning the answer with citations. - `/api/upload`: Accepts file uploads, parses them (e.g. PDF, Word, text) into text, chunks and embeds the content, and stores it in the vector database (with metadata linking it to the user or session). Ensures quick ingestion so the user can query immediately [4]. - (Other endpoints: user auth, if any; health check; admin tools for indexing data, etc.) - **LLM Integration:** The backend will integrate with **OpenAI's ChatGPT (GPT-4/GPT-5) via API** (using Azure OpenAI service for data residency in India). We will use the **chat completion API** with a carefully designed system and user prompt. The prompt will include retrieved context and instructions such as *"You are an expert Indian legal assistant. Answer the question using the provided documents and factual knowledge. If you cite laws or cases, reference them. Do not fabricate information."* This ensures the model knows to use the context and refrain from unsupported claims. - **Knowledge Base and RAG Pipeline:** At the heart of our system is the **Retrieval-Augmented Generation** workflow: - All knowledge documents (whether pre-loaded legal corpora or user uploads) are **indexed in a Vector Database (Vector DB)** for semantic search [5]. We will likely use a **vector index** that balances performance and privacy – for MVP this could be an open-source solution (like **Postgres + PGVector** extension or a managed service such as **Azure Cognitive Search** with vector capabilities) for simplicity, and later upgrade to more advanced Vector DB (Harvey, for instance, uses **LanceDB** in production for its speed and decentralized storage [6]). - Each document is **chunked** (split into semantically coherent paragraphs or sections), and each chunk is converted to an **embedding vector** via OpenAI's embedding model (e.g. `text-embedding-ada-002`). Metadata (source name, date, etc.) is stored alongside for filtering and citation generation. - **Query workflow:** When a user asks a question, the system generates an embedding for the query and performs a **semantic search** in the Vector DB to retrieve the top relevant pieces of text (e.g. top 5 chunks) that might contain the answer. We may also use a **hybrid search** strategy: combine dense vector search with keyword filtering for important legal terms or citations (since dense embeddings alone might miss exact matches for case citations or section numbers [7]). The retrieved texts are then compiled (with some light processing, like removing irrelevant parts or merging overlapping chunks) into a prompt context. - The backend sends the **augmented prompt** (user query + retrieved context) to the GPT API. The model's response

(answer) is then post-processed to format citations properly (if the model provided raw references like "[Source 3]", the system might replace that with full case citations or act names). - **Iterative refinement:** The system can perform a quick **verification pass** on the answer. For MVP, this could be a simple check: if the answer makes a factual claim not obviously supported by the retrieved text, we flag it or add a disclaimer. Future versions will implement a more rigorous verification – decomposing the answer into claims and cross-checking each against the sources [8]. The goal is to **minimize hallucinations to near-zero** (Harvey's internal evals show ~0.2% hallucination with such methods [8]). - **Data Storage:** In addition to the vector index, we may use a relational database (e.g. PostgreSQL) or Azure Cosmos DB for storing: - User data (accounts, preferences) – if the MVP includes user accounts. - Conversation history (if we allow users to save or if needed for context continuity beyond a single query). - Document metadata (linking uploaded documents to users, access control lists, etc.). - Logging of queries and results for monitoring and further training (with strict privacy controls). - **Azure Cloud Infrastructure:** We will deploy the application on **Azure** for reliability and scalability. Key components: - **Azure Web App or Azure Kubernetes Service (AKS)** for hosting the backend API (containerized for portability). We ensure the setup can auto-scale to handle spikes in queries (common in enterprise use-cases [9]). - **Azure Storage (Blob)** for securely storing document files and possibly the vector index files, ensuring data residency in India. All sensitive data will remain in the Azure India region to meet data localization requirements. - **Azure Cognitive Search or Azure-managed Vector DB** for the knowledge index, if available, to simplify maintenance (else, manage our own vector DB instances on Azure VMs). - **Azure OpenAI Service** to access GPT models within Azure's compliance boundary, inheriting Microsoft's enterprise security and compliance (this is how Harvey integrates with Office 365 and gains IT trust [10]). - **Monitoring & Logging:** Use Azure Monitor/Application Insights for telemetry, logging errors, and tracking performance (latency of vector searches, API calls, etc.). This helps quickly identify issues and ensure a smooth, error-free operation.

Figure: High-level architecture for a Retrieval-Augmented Generation pipeline, similar to our design. Multiple knowledge sources (public legal data, private documents) are embedded and indexed in a vector database. A user query triggers a retrieval of relevant context which is then fed into the LLM to generate an answer [11] [7]. The system emphasizes grounded answers using authoritative data, crucial for legal AI.

RAG Methodology and Reliability

Retrieval-Augmented Generation (RAG) is central to our approach for accuracy and trust. By augmenting the LLM with a curated knowledge base, we ensure the output is **based on real sources** rather than the model's parametric memory [1]. Our RAG implementation will use **state-of-the-art techniques**: - **Semantic Search with Domain-Specific Tuning:** We utilize embedding-based search for semantic matching. However, legal queries often involve specific jargon, citations, or procedural terms (e.g. "Article 226 petition high court"). We plan to incorporate **custom legal embeddings** in the future – i.e. embeddings trained on Indian legal text – to better capture domain nuances, similar to how Harvey saw a 25% reduction in irrelevant results with custom legal embeddings [12]. For MVP, we use off-the-shelf embeddings and possibly augment with keyword search for names/sections to ensure we don't miss exact matches [7]. - **Document Ranking and Filtering:** Retrieved chunks will be relevance-ranked. We may apply a secondary **re-ranking model** (possibly an LLM itself or a smaller model) to improve the quality of chosen snippets. We'll also filter out any chunk that lacks substantive content (e.g. boilerplate) to avoid diluting the answer context window. - **Prompt Strategy:** The prompt given to GPT will clearly instruct it to use the retrieved info and **not to speculate beyond it**. For example: *"Answer the question using the provided documents. If the answer is not in those documents or known Indian law, say you cannot find the information. Quote any relevant law or case and cite the source."* This makes the output **evidence-backed and traceable**. In addition, we set a temperature parameter low (for deterministic, factual output) when answering fact-based queries, and slightly higher for creative drafting requests. - **Citations and Verification:** The system will format answers with inline citations

referencing the source documents (by title or a code). To implement this, one approach is to insert reference tags in the prompt context around each chunk (like "[1] ...text..") so the model learns to cite "[1]" when using that text. Alternatively, we may post-process by mapping sentences back to sources. The critical part is ensuring **every factual claim can be tied to a retrieved source** [13]. Any claim that isn't supported triggers either a warning or is pruned. We will also integrate mechanisms to verify cited case law is still valid (for instance, if a case was overruled, the system should flag it). In the US, Harvey does this via Shepardization with LexisNexis [13]; for India, a future integration with local legal databases (like SCC Online or Manupatra which track case validity) can be planned. - **Handling Ambiguous or Complex Queries:** Legal queries can be complex or multi-layered [9]. For instance, "Analyze the evolution of the Supreme Court's stance on privacy rights in India" is broad and involves multiple cases over time. The assistant may need to break down such queries. In future, we can employ a **chain-of-thought agent** that plans sub-queries (e.g., identify key cases, then fetch details) and synthesizes an answer. For MVP, we handle this by retrieving a diverse set of relevant texts (from constitution, landmark judgments like *Puttaswamy case*, etc.) and letting GPT summarize the trend. - **Continuous Learning and Improvement:** We will gather user feedback on answers (thumbs up/down) and have domain experts (experienced lawyers) review outputs periodically. This feedback loop will guide iterative improvements – e.g., adding missing sources, adjusting prompts, or extending the knowledge base. As Harvey's team noted, working closely with domain experts can significantly refine retrieval quality [14]. Our system is designed to incorporate such iterative tuning to steadily improve reliability and performance for Indian legal queries.

Scalability and Performance

The design ensures the system can **scale to enterprise usage**: - **Stateless Backend:** The API servers will be stateless (not relying on in-memory sessions for long-term data), which allows horizontal scaling behind a load balancer. Any state (conversation context, user profile) is stored in a fast database or cache accessible to all instances. - **Vector DB Performance:** For quick retrieval even as the knowledge base grows to millions of entries, we will use indexes (HNSW or IVF) that enable sub-second similarity search. We choose technologies proven in high-scale RAG systems. (For example, Harvey's solution handles millions of documents with <2s query latency [15] [16]. We aim for similar performance, perhaps by using **approximate nearest neighbor** search in the vector DB.) We will also monitor and optimize embedding dimensionality and index parameters for speed vs. accuracy trade-offs. - **Asynchronous Processing:** Uploading and indexing documents will be handled asynchronously. When a user uploads a large PDF, the server can immediately acknowledge receipt and process the text extraction and embedding in the background. The UI can show a "processing" status and notify when the doc is ready for query. This prevents blocking the user and enables handling many uploads in parallel using background worker threads or Azure Functions. - **Caching:** Implement caching of frequent queries or embeddings. If multiple users often ask about a popular statute section or a landmark case, we cache the answer or at least the retrieved context for faster subsequent responses. We will also cache embeddings of frequently used documents (like Constitution of India) in memory to avoid recomputation. - **Load Testing and Auto-Scaling:** We will perform load testing to estimate how many concurrent users/queries the system can handle (given GPT API limits and our infrastructure). Based on this, we configure auto-scaling rules on Azure (e.g., scale out to more instances if CPU or response time crosses a threshold). The vector database should also be scalable – if using a cloud service like Cognitive Search, we can increase replicas; if self-hosted, we can scale the VMs or use sharding if needed for very large corpora. - **Scalable Design for Knowledge Base:** The knowledge base is modular. Initially, we might load a curated subset of Indian legal docs (maybe a few thousand important documents). But our pipeline will be designed to ingest **millions of documents** in the future. We plan robust ETL scripts for various data sources (scraping court websites, converting legislation PDFs, etc.). The vector store choice will account for growth (some options allow horizontal scaling of storage/index easily [17] [18]). This ensures our assistant remains comprehensive as Indian law evolves.

Security and Privacy

Legal professionals require strict confidentiality and compliance: - **Data Privacy:** All user-provided data (queries, documents) will be kept **confidential**. We implement a **zero-retention policy** on user data for the AI model – i.e., we do not use or store any client documents or queries in the training pipeline without permission [19]. Data is only used transiently to answer the query and for logging within the secure environment. We will **not send sensitive content to third-party services** beyond the trusted LLM API. (Using Azure OpenAI ensures data does not leave the Azure environment and is not used to train OpenAI's base models, per their data policies.) - **Access Control:** If multi-user, ensure each firm's or individual's data is siloed. For the MVP, we might have a single-tenant deployment (one firm's instance). In future multi-tenant SaaS, strong access controls will isolate each client's document vault. We'll integrate Azure Active Directory or a secure auth system for user login if needed. - **Compliance and Hosting:** Host all services in **India-based Azure data centers** to comply with data localization (important for certain regulated data in India). The solution will be built with an eye toward relevant compliance standards (for example, ISO 27001, SOC 2, etc. as needed by enterprise IT). We will document our security architecture (encryption at rest for databases, encryption in transit with HTTPS, periodic security audits on the code) and potentially offer a **self-hosted option** (deployable in the client's own cloud or on-prem) for firms extremely sensitive about data (similar to Harvey's approach of allowing private cloud deployments [20]). - **Audit Logs:** Maintain detailed logs of system actions – e.g., which documents were retrieved for a query, what answer was given – without logging the actual confidential content. These logs provide an **audit trail** for transparency and can be shared with the user/admin if needed (for instance, to trace why the AI gave a certain answer, or to investigate any issue). This also aligns with the principle of building **trust through auditability** [2]. - **No Unauthorized Learning:** The system will **not learn from user data by default**. Each query is handled in isolation (aside from context in a conversation thread). If in future we allow fine-tuning or custom model improvement using client data, it will be strictly opt-in and done in a siloed way for that client (ensuring no cross-contamination of data between clients). This prevents inadvertent leakage of sensitive info across users and maintains trust.

Development Approach and Best Practices

We will follow a **modular, error-free development strategy**: - **Frameworks and Libraries:** Utilize reliable libraries to speed up development and reduce errors. For example, use **LangChain** or similar orchestration libraries for the RAG pipeline (these provide utilities for chaining LLM calls with retrieval, and can integrate with vector stores easily). Use proven PDF/text parsing libraries (like PyMuPDF or Azure Form Recognizer) to avoid writing brittle parsers from scratch. We choose frameworks that are well-tested to minimize bugs. - **Clean Code and Architecture:** Maintain a clear separation between components (as described in architecture). Follow best practices in coding – meaningful naming, error handling, logging, and comments for clarity. The project will be structured for readability since an AI developer agent will build it. For instance: - Use a **config file** for all keys and parameters (API keys, DB connection, etc.), making it easy to change environments. - Implement robust **error handling** around external calls (GPT API, DB queries). E.g., if the GPT API fails or times out, catch the exception and return a graceful error message to the user (perhaps suggesting to retry). - Write **unit tests** for critical functions: e.g., test the chunking and embedding pipeline on sample documents, test that retrieval returns expected results for known queries, etc. This ensures each part works as intended and prevents regressions. - **Iterative Development & Validation:** Begin with a **proof-of-concept** of the RAG loop (a simple script that given a sample query, can retrieve from a small set of docs and get an answer from GPT). Gradually integrate this into the full-stack app. After each major feature (upload, chat, etc.), conduct manual tests with typical use cases: e.g. ask it a known question like "What is the latest Supreme Court case on privacy?" and verify it finds the correct case and answers accurately. We'll

involve an actual lawyer to validate outputs on real questions to ensure the system's usefulness. - **Performance Testing:** Before deployment, test the system with increasing load and document sizes. Ensure that adding, say, 100K documents doesn't break retrieval (we might simulate with dummy data if actual corpus is smaller during MVP) and that response time stays acceptable (< a few seconds for answers). Also test concurrency (multiple queries at once) and memory usage (embedding large docs, etc.) to catch any bottlenecks or memory leaks. - **Deployment Strategy:** Use Infrastructure-as-Code (Azure ARM templates or Terraform) to provision resources consistently. Containerize the application using Docker – enabling easy deployment and rollback. Set up a staging environment where new versions are tested with a copy of the database before going live. Only deploy when all tests pass and the system is verified to be stable. Our goal is a **resilient and dependable** system from day one, even as an MVP. - **Documentation:** Document the code and system thoroughly. Provide a README (besides this prompt) for the repository that explains how to set up the dev environment, how to deploy, and how to use the features. In-line comments and function docstrings will describe complex logic (especially around prompt construction and verification steps). This helps future developers or AI agents quickly grasp the system and reduces misunderstanding-induced errors.

Future Enhancements (Beyond MVP)

While the MVP focuses on core capabilities, we design with a long-term vision: - **Fine-Tuned Legal LLM:** Collaborate with OpenAI/Azure to fine-tune an LLM on **Indian legal text** (court judgments, law textbooks, etc.), similar to Harvey's custom case law model [21] [22]. A fine-tuned model could understand Indian legal reasoning better and produce even more nuanced answers. This model could run alongside the base model (multi-model orchestration: choosing the best model per query type [23]). - **Advanced Verification System:** Implement a full **hallucination detection and claim verification module**. For each answer, automatically break it into claims (e.g., "The Supreme Court held X in case Y") and check against the knowledge base or external databases whether each claim is true and up-to-date. Flag or prevent any unverified statements from reaching the user [8]. This might involve additional AI agents or rules and will further **elevate trust** in the system. - **Workflow Automation (Legal Agents):** Move beyond single Q&A to guided workflows. For example, an **"AI research memo" agent** that can take a complex task (like preparing a legal memo on a topic) and perform multi-step research: gather relevant cases, summarize each, compare them, and produce a structured memo. Or a **contract analysis agent** that reviews an entire contract and produces an issue list. Harvey introduced such agents for proactive tasks in 2025 [24]; we aim to develop similar capabilities tailored to Indian law practice. - **Integration with Legal Databases & Tools:** Forge integrations with platforms like **SCC Online, Manupatra, or Bar & Bench** for direct access to comprehensive Indian legal data and citation networks. This could also include hooking into government gazettes for new legislation or the courts' cause lists for procedural updates. Additionally, integrate with Microsoft Word/Office 365 so lawyers can invoke the AI assistant while drafting or reviewing documents (e.g., an add-in to ask questions about the open Word document). Leverage the Azure ecosystem for this integration to ensure smooth single sign-on and data flow [3]. - **Enhanced UI and Collaboration:** Develop features for team collaboration, such as shared document repositories (a "Vault" like Harvey's [25]) where a firm's lawyers can collectively build a knowledge library. Add user feedback mechanisms (allow users to correct the AI or add notes to an answer, which the system can learn from for that organization). Possibly incorporate voice input for queries or mobile app access for on-the-go usage by lawyers. - **Comprehensive Indian Language Support:** Extend support to Indian regional languages deeply. This means not just translating queries, but also incorporating vernacular legal documents (some lower court judgments or older land records might be in local languages). Train or use multi-lingual embeddings/LLMs so the AI can interpret and translate such content. Provide answers in the user's preferred language when needed. - **Continuous Update Pipeline:** Set up an automated pipeline to keep the knowledge base current – e.g., periodically crawl official sources for new judgments or amendments, process and add them to the index. This ensures the AI's advice is always based on the **latest law** (critical in a field where new precedents or

regulations can change the answer). - **Benchmarking and Outperforming Competitors:** Regularly benchmark the system's performance against other legal AI tools (Harvey, JurisPilot, Legora, etc.) using real-world legal questions. Track metrics like answer accuracy, relevance, and lawyer satisfaction. Our aim is to **outperform all existing solutions in the Indian context** – by virtue of superior local data integration, nuanced understanding of Indian law, and rigorous verification. We will use these benchmarks to identify areas for improvement and validate that our AI indeed exceeds the competition in quality.

In summary, this project will deliver an MVP of an **Indian Legal AI Assistant** that is powerful yet safe: **feature-rich** in functionality (chat Q&A, document analysis, drafting help), underpinned by a cutting-edge **RAG system for reliability**, and built on a **scalable Azure infrastructure**. By focusing on the unique needs of Indian lawyers and embedding the latest techniques in retrieval and generation, we set the foundation for a product that can revolutionize legal work in India, much like Harvey AI is doing globally – and even go beyond it by addressing India-specific challenges and opportunities. The following project plan will break down the steps to achieve this.

File: project_plan.md

Project Plan – Indian Legal AI Assistant (MVP)

1. Project Overview and Objectives

Goal: Develop a web-based AI assistant for Indian legal professionals that can answer legal questions and analyze documents with high accuracy and trustworthiness. The MVP will demonstrate the core functionality using OpenAI GPT-4/5 and a Retrieval-Augmented Generation approach with an Indian law knowledge base. We aim to achieve a reliable, scalable, and secure system that can be piloted in a law firm setting, showing clear advantages over existing solutions (e.g. faster research, time savings, and higher confidence due to source-backed answers).

Key Objectives: - Implement a **chatbot interface** for legal Q&A that provides answers with citations to Indian legal sources. - Build a **RAG pipeline** with a vector database to integrate external knowledge (legal documents) into the LLM's responses, minimizing hallucinations and ensuring up-to-date information [1]. - Enable **document upload and analysis** for user-provided documents (contracts, case PDFs, etc.), allowing Q&A and summarization on those. - Ensure the system is **scalable** (can handle growth in users and data), **secure** (protects confidential info, no data leaks), and **extensible** for future enhancements (like fine-tuning, additional integrations). - Deliver the MVP with no critical errors, thoroughly tested and ready for real-world usage in an Indian legal context by end of development cycle.

2. Scope and Deliverables

In-Scope (MVP Features): - Web application (frontend + backend) supporting: - **User Query Interface:** Chat-style Q&A with the AI on Indian legal topics. - **Retrieval System:** Integration of a curated set of Indian legal documents (at least a few thousand entries: important laws and sample case rulings) into a vector database for context retrieval. - **OpenAI API Integration:** Using GPT-4 (or latest available model via OpenAI/Azure) for generating answers, with proper prompt construction. - **Document Upload:** Users can upload documents (PDF/DOCX/text) to incorporate into the knowledge base (either for their individual query or into a personal "vault"). Basic PDF/text parsing included. - **Citations in Answers:** The

AI's responses will reference source documents (e.g., "as per *Indian Contract Act, 1872*, Section 23…【source】"). The system will present these sources for user verification. - Infrastructure setup on Azure: - Backend service deployed (Azure App Service or AKS), front-end hosted (perhaps Azure Static Web Apps or part of App Service). - Azure OpenAI service usage for the model (or OpenAI API if Azure OpenAI is not available, but preference given to Azure region for compliance). - A vector store for document embeddings (could be an Azure service or a managed database on Azure VM). - Azure Blob Storage or database for storing uploaded files and metadata. - **Testing and Quality Assurance:** - Functional testing of all features. - Accuracy testing with a set of legal Q&A pairs (we will prepare ~20 sample questions that we know answers to and verify the system's outputs). - Security testing (basic checks like ensuring uploaded docs aren't accessible to unauthorized users, API keys are secure, etc.). - **Documentation:** - Developer docs: describing system architecture, how to run/deploy, and how to extend. - User guide (for pilot users): explaining how to use the chat, upload docs, interpret the AI answers and citations, and any limitations (e.g. "the AI is an assistant, not a lawyer – always review its advice" disclaimer).

Out-of-Scope (for MVP, possibly future): - Fine-tuning or custom model training on legal data (use pre-trained GPT via API for MVP). - Full integration with external legal DBs like SCC Online/Manupatra (MVP will rely on a manually gathered corpus due to time constraints). - Multi-tenancy and elaborate user management (MVP can assume a single organization context or simple login if needed; robust multi-organization support can come later). - Office 365/Word plugin integration (future phase). - Advanced workflow automation or multi-step AI agents for complex tasks (beyond basic Q&A and doc analysis in MVP). - Very large-scale deployment concerns (MVP will handle moderate usage; enterprise rollout considerations like multi-region support, CDN, etc., are future enhancements).

3. Timeline and Milestones

We propose a phased approach for development, with the following **milestones**:

- **Week 1-2: Requirements Refinement & Design**
- *Requirements Workshop:* Finalize details on the legal content scope for MVP (which laws/cases to include), with input from a legal advisor. Define user personas and core use cases in detail.
- *System Design:* Create detailed architecture diagrams and data flow for the RAG pipeline, component integration, and Azure infrastructure setup. Decide on tech stack specifics (e.g., choose between FastAPI vs Express.js for backend, select vector DB technology).
- *Design Review:* Present the design to stakeholders (or senior engineers) for feedback. Ensure the approach covers accuracy, performance, and security aspects properly.

- **Milestone:** Design sign-off – the team and stakeholders agree on the architecture and plan (no critical gaps).

- **Week 3-5: Backend Implementation (RAG Pipeline & API)**

- **Data Ingestion Module:** Implement scripts/services to ingest legal documents:
    - Convert source documents (e.g., text of Acts, case judgments) into plain text. If sources are PDFs, use a PDF parsing library; if HTML from websites, use scrapers.
    - Chunk the text into smaller sections (e.g., ~500 tokens each) while preserving meaningful boundaries (e.g., by sections for laws, by paragraph for judgments).
    - Generate embeddings for each chunk using OpenAI's embedding API. Store vectors and metadata in the chosen vector database.

    - Optimize ingestion for speed – e.g., batch process embeddings, multi-thread the parsing if needed [4]. For MVP, ingest an initial dataset (perhaps ~1000 documents); ensure the pipeline can later be re-run to add more.
- **Query Retrieval Service:** Develop a function that given a user query, performs:
    - Embedding of the query, vector similarity search in the DB, returning top N relevant chunks.
    - (If needed) a secondary keyword search: e.g., if the query contains a case citation or section number, explicitly filter or boost those in results.
    - Formatting of retrieved text and sources into a prompt-friendly format (like a compiled context string).
    - This component should be modular (e.g., a Python class or function) so it can be independently tested and optimized (we can unit test it with sample queries to see if it retrieves expected snippets).
- **OpenAI API Integration:** Set up an API client to call OpenAI's chat completion. Handle streaming vs non-streaming responses (for MVP, sync calls are fine). Include robust error handling (with retries or graceful failure messages).
- **Answer Construction:** Create a prompt template that combines the system message (instructions), retrieved context, and user query. Implement the logic to call the model and process its response. Ensure that the response is parsed for citations if needed (e.g., if the model returns reference tags, convert them to actual document names/links).
- **Upload Endpoint:** Develop the `/api/upload` endpoint:
    - Accept file (PDF, DOCX, etc.), authenticate user (if applicable), save file temporarily.
    - Extract text (using a library or service). If OCR is needed (scanned PDF), use an OCR engine (like Tesseract or Azure Form Recognizer).
    - Pass the text through the same chunk-and-embed pipeline, add to vector DB, associating with the user's ID or session.
    - Return a success response (and maybe a document ID or summary to the frontend).
    - Ensure security: scan or sanitize file contents if needed, and enforce file size/type limits to prevent abuse.
- **Testing (Phase 3 interim):** Write and run unit tests for the above. For example, test embedding+search on a small known set (maybe insert a known text and see if a query finds it). Test the upload pipeline with a sample PDF. Also test error conditions (invalid file, OpenAI API down, etc.).

- **Milestone:** Basic backend functionality in place – one can query the API (via curl or a temporary frontend) and get an answer from GPT that uses retrieved knowledge.

- **Week 6-7: Frontend Implementation (Web App UI)**

- **UI Design:** Quickly sketch the UI layout – a simple single-page app with a chat panel and a sidebar or header for upload and other actions. Ensure it's user-friendly for lawyers (clean, professional look; maybe the theme aligning with law firm branding).
- **Set up Frontend Project:** Initialize a React (or chosen framework) project with TypeScript for type safety. Set up basic routing if needed (though likely a single-page with sections).
- **Chat Interface:** Implement the chat component:
    - Display conversation history (user questions and AI answers).
    - Input box for new question, send button (and support pressing Enter to send).
    - When sending a query, show a loading indicator and disable input until response comes.
    - On receiving answer, render it (the answer might include markdown, e.g. lists or bold text, so use a markdown renderer or sanitize properly).

    - Parse and display citations: if the answer contains references like 【1】 or [Source: 123], ensure these appear as clickable links. When clicked, they could open a panel or modal showing the referenced text from the knowledge base. (Implement a method to fetch the full text from the vector DB or a pre-stored index by an ID to show to the user.)
- **File Upload Interface:** Implement an upload button or drag-and-drop area:
    - Allow multiple files maybe, but MVP can be one at a time.
    - After file selection, call the `/api/upload` endpoint. Show progress or at least a message "Indexing document..."
    - Once done, notify the user that the document is ready. Possibly list the uploaded document in a sidebar "My Documents" with an identifier. The user can then ask questions specifically about that document (the UI can allow selecting it as context or the system can automatically prioritize recently uploaded doc in retrieval for the next query).
- **Additional UI elements:** A toggle or option to switch between modes (Q&A vs Drafting vs Summarize). For MVP, we can implement these modes as simple variations:
    - If user selects "Drafting", the prompt to backend can include an instruction like "draft mode" so that the system knows to produce a longer, form-like output (e.g., generate a document clause rather than answer yes/no).
    - If "Summarize", and a document is selected, it triggers a summary of that doc rather than normal Q&A. (This could reuse the same pipeline: just have a different prompt: "Summarize the document X in bullet points." and feed the doc's content as context.)
    - These sub-features utilize the same backend but offer a more task-specific UI prompt for user convenience.
- **Frontend-Backend Integration:** Connect the frontend to the backend API:
    - Define a config for API base URL. Use fetch/Axios to call `/api/query` with the user question and perhaps an array of document IDs (empty if none, or a flag if user wants whole knowledge base vs personal docs). Handle the JSON response (containing answer text and citation metadata).
    - Handle errors: if the API returns an error (e.g., 500 or validation error), display an appropriate message in the chat (like "Sorry, something went wrong. Please try again."). Log errors to console for debugging.
    - Ensure CORS is configured so the frontend (if served separately) can call the backend.
- **UI Testing:** Manually test the UI:
    - Ask a simple question (without any uploaded doc): see that the answer comes with a citation and is rendered properly.
    - Upload a known document (maybe a sample contract), then ask a question about it to verify the pipeline end-to-end (the answer should reference the uploaded doc).
    - Test layout on different screen sizes (lawyers may open on laptop or tablet). Ensure it's responsive enough.
    - Fix any usability issues (e.g., long answers causing scrolling – ensure the chatbox scrolls, etc.).

- **Milestone:** Fully functional MVP web application – users can interact through the UI to get answers and upload docs. The core feature set is implemented.

- **Week 8: Refinement, Optimization, and QA**

- **Performance Optimization:** Profile the application:
    - Check average response time for a query. If it's, say, 8-10 seconds and that's considered too slow, investigate: perhaps reduce number of retrieved chunks or shorten them, or enable partial streaming of answer to user. Optimize wherever possible (maybe parallelize

the embedding and search steps if not already). Ensure that even if slightly slow, the UI gives feedback (loader) to manage user expectation.
    - Optimize indexing pipeline if needed: e.g., large files might take time to embed – we can add a notice for large files or break the work into smaller asynchronous tasks.
- **Accuracy and Validation Testing:** Using a set of prepared legal questions (covering various topics: constitutional law, corporate law, procedure, etc.), test the system's answers:
    - Check for correctness and completeness. Are the citations correct and relevant? Involve a legal expert if possible to review answers for any mistakes or misleading info.
    - Track how often the system says "I don't know" or refuses vs when it answers incorrectly. Adjust the prompt if needed to find a better balance (we prefer it to refuse rather than hallucinate wrong info).
    - Particularly test edge cases: ambiguous queries, very broad queries, or ones with no answer in the knowledge base. Ensure the system handles them gracefully (perhaps by saying it cannot find information, rather than guessing).
    - If any systematic issues are found (e.g., model tends to confuse certain legal terms or a certain law is missing from knowledge base), address them: update the data or add prompt reminders ("If question is about X law, make sure to consult Y document").
- **Security Review:** Double-check security aspects:
    - All API routes should have proper validation (no SQL injection or such since we use parameterized queries for DB, but ensure file paths are handled safely, etc.).
    - Ensure no sensitive info is logged. Sanitize user input if it might end up in logs.
    - If using any secrets (API keys), confirm they are not exposed in the frontend or error messages.
    - Run basic vulnerability scan tools on the web app and address any high risks.
    - Confirm compliance measures: data at rest encryption (Azure typically has this by default, but ensure it's enabled on databases), using HTTPS for all comms, etc.
- **User Experience Polish:** Make minor improvements to UI/UX:
    - Add instructions or placeholders (e.g., in the chat input "Ask a question about Indian law..." to guide users).
    - Ensure the interface is clean (maybe add the product name/logo, a consistent color scheme).
    - If time permits, implement a simple conversation saving (so the user can refresh without losing context or have a way to start a "New chat" and clear context).
    - Add a disclaimer in the UI footer or chat: "**Disclaimer:** AI can assist with legal research, but always consult a qualified lawyer for legal advice. The system provides information based on sources and cannot guarantee completeness." This is important to set the right expectations.

- **Milestone:** MVP Release Candidate – all features are working correctly, tested for accuracy, and the system is optimized for a smooth demo or pilot use. No known critical bugs remain.

- **Week 9: Deployment & Handover**

- **Azure Deployment:** Provision necessary Azure resources via IaC or Azure Portal:
    - Set up the web app/AKS cluster, database, storage accounts, Azure OpenAI resource, etc., as per architecture.
    - Deploy the backend and frontend. Ensure environment variables and settings (API keys, endpoints) are correctly configured in the cloud environment.
    - Run smoke tests on the deployed version (some basic queries) to ensure everything works in production mode.

- **Scalability Setup:** Configure autoscaling rules (if using AKS, set min and max pod count and CPU threshold; if App Service, enable auto-scale by CPU or queue length). Also set up Azure Monitor alerts for high error rates or high response times, so we get notified of any issue early.
- **Final Review & Client Demo:** Present the working MVP to the stakeholders (could be internal team or the client/law firm partners). Demonstrate the key use cases:
    - Ask it a few legal questions live, show how it cites sources.
    - Upload a document and query it.
    - Possibly show a draft generation (like drafting a simple contract clause).
    - Gather immediate feedback. If any minor adjustments are needed (e.g., tweak a prompt or add one more important case to the knowledge base), do them promptly.
- **Documentation & Handover:** Complete all documentation:
    - Ensure the **developer docs** are up to date with any changes. Include instructions to extend the knowledge base or retrain embeddings when new data comes.
    - Provide an **operations guide**: how to restart services, monitor system, where to find logs, how to scale if usage increases, etc., so the system can be maintained.
    - If an AI or another developer will continue development, make sure they have access to all info (code repositories, API keys in a secure vault, etc.). A quick knowledge transfer session can be conducted.
- **Milestone:** MVP Launched – The system is live on Azure, and ready for initial users. The project artifacts (code, documentation, plans for next steps) are handed over.

4. Technical Implementation Details

(This section provides further details on important technical choices and algorithms to be used, ensuring clarity for the implementing team.)

- **Language & Frameworks:** We will use **Python 3.x** for the backend due to its rich ecosystem for AI (FastAPI for API endpoints, and possibly libraries like LangChain for RAG orchestration). Python's `asyncio` can be leveraged in FastAPI to handle concurrent requests (like multiple file processing tasks). For the frontend, **React** with TypeScript is chosen for a robust, maintainable UI. This stack is widely understood by developers and AIs, minimizing any ambiguity in implementation.
- **Vector Database:** For MVP, a pragmatic choice is **Postgres with PGVector** extension or **Azure Cognitive Search** with vector indexing (if we want a managed solution). We decide on PGVector if self-hosting on Azure, due to simplicity and the fact it can handle our initial scale easily. As per Harvey's experience, PGVector offers high accuracy at small-to-medium scale and can index in parallel [26]. Should our corpus grow significantly, we plan to transition to a more scalable solution (like an open-source **LanceDB** deployment or a specialized service) that supports distributed storage and search [17] [18]. The code will be abstracted enough (using an interface for vector search) so we can swap the backend if needed.
- **Embedding Model:** Use OpenAI's `text-embedding-ada-002` (which is the current best general embedding model) through the OpenAI API (or Azure equivalent). This gives 1536-dimension embeddings suitable for semantic search of text. It's fast and cost-effective. We will normalize and store these vectors. If needed, we will also store a separate traditional index (like an inverted index on keywords) to support hybrid search, possibly using a library or even Postgres full-text search for certain fields (e.g., case names). This addresses the **"Sparse vs. Dense"** challenge in legal text [7].
- **Prompt Template & Parameters:** A careful prompt design will be documented (in the code and readme). For example:

```
System Message: "You are an AI legal assistant trained on Indian law.
You have access to the following relevant excerpts from legal sources.
Answer the user's question based on these sources, citing them. If you
don't find an answer in these, say you cannot answer."
Context: [insert top N retrieved texts, each labeled e.g. [Doc1] ...]
User Message: "[User's question]"
```

We will test a few variations to see what yields the best behavior (some experimentation might be needed to get the model to cite properly and refrain from outside knowledge unless sure). We'll keep temperature low (~0.2-0.3 for fact queries). For drafting tasks, we might allow slightly higher temperature for creativity (0.7) but always instruct the model to stay legally correct.

- **Citing Sources:** We plan to cite by a short identifier (like [1], [2] or [DocName]). The mapping from these IDs to actual references (like case name or act section) will be maintained. For example, if a retrieved chunk is from *K.S. Puttaswamy vs Union of India (2017)*, we might label it [Case1] in the prompt; the answer might say "the Supreme Court in *Puttaswamy* held that privacy is a fundamental right [Case1]". After getting the answer, we replace [Case1] with a formatted citation or hyperlink to a stored snippet of that case. This approach ensures the user sees a source, and if they click it, we show them the context from which the answer was drawn. This is akin to Harvey's "structured citation" approach [13] but simplified for MVP.
- **Concurrency and Async Tasks:** The system must handle potentially multiple users querying and uploading at once. We'll ensure:
- The vector DB can handle concurrent reads/writes (PGVector can, though heavy concurrent writes may slow queries – we might schedule large ingestions during off-peak hours if needed).
- The backend uses async I/O for waiting on API calls (OpenAI calls, DB queries) so that one request doesn't block others.
- For CPU-intensive tasks (like large PDF parsing), if using a single server, consider using background task workers (e.g., Celery or FastAPI BackgroundTasks) to offload these so the main API remains responsive. Alternatively, allocate sufficient CPU or separate service for ingestion.
- **Logging and Error Handling:** Implement structured logging (e.g., in JSON) for important events: each query will log query text, maybe the retrieved doc IDs (but not the full content to avoid sensitive data in logs), and whether the answer was successful or had an error. File uploads will log file name and status. Errors will log stack traces. These logs can be sent to Azure Application Insights for analysis. We'll also add catch-all error handlers in the API to return a friendly error message instead of a crash. The plan is to **avoid any unhandled exceptions** – all known failure points (OpenAI API errors, DB connection issues, file parse errors) will be caught and managed.
- **Monitoring & Analytics:** Beyond logs, set up metrics: e.g., count of queries answered, average response time, vector search time, etc. This can be done by instrumenting the code or via Azure's monitoring. Also, track OpenAI API usage (tokens) to watch cost and efficiency. If certain queries consume too many tokens (maybe because too many docs are being stuffed in), we adjust our approach (like limit context length).
- **Collaboration with Domain Experts:** Though more process than technical, it's worth highlighting: we will involve a **small group of lawyers** in testing the system with real questions. Their feedback will be crucial. For instance, if they find the AI's language not sufficiently formal or some answers missing context, we'll refine the prompt or add data accordingly. This echoes the approach of Harvey working with experts to fine-tune their system's performance in domain-specific tasks [14]. We consider this part of the development plan to ensure the product meets professional standards.

5. Risk Mitigation

Identify potential risks and how we'll mitigate them: - **Hallucination/Misinformation Risk:** The AI might fabricate plausible-sounding legal answers. *Mitigation:* Strict prompt instructions to only use provided context, implementation of citation requirement, and verification steps. If unsure, the AI should respond it cannot find the answer. We will test extensively to tune this. In the long run, adding an automated fact-checker will further reduce this risk [8] . - **Insufficient Knowledge Base:** The system is only as good as the data it has. If our Indian legal corpus misses crucial information, answers could be incomplete. *Mitigation:* Start with a solid base of important documents. Prioritize adding data for areas the pilot users need. Also allow users to add their own references via upload. In future, integrate with larger databases or regularly update the corpus from public sources. - **Scalability Bottleneck:** If usage grows rapidly (e.g., many queries or large documents), the system could slow down, especially due to the LLM API calls or vector searches. *Mitigation:* Use Azure scaling, and design the system statelessly to add servers easily. Monitor performance; if vector search becomes a bottleneck, consider sharding the index or upgrading infra (we have a plan to move to more scalable vector DB if needed). Also, the use of Azure OpenAI means we can request higher rate limits or provision dedicated capacity if usage demands. - **Cost Management:** OpenAI API and Azure resources incur costs. Unbounded use could be expensive. *Mitigation:* Implement usage limits or monitoring. For instance, for MVP/pilot, set a quota of queries per minute or restrict some expensive operations (like summarizing a huge document) to avoid runaway costs. Optimize prompts to be concise (to reduce token usage). Long-term, explore hosting our own model or fine-tuned smaller models for cost efficiency if feasible. - **Compliance and Ethical Concerns:** There might be legal/ethical issues using AI for legal advice (e.g., unauthorized practice of law, or data privacy concerns by clients). *Mitigation:* Position the product clearly as an "assistant" that helps with research, not a lawyer replacement. Always involve the human in final decisions. Obtain any necessary approvals for data usage (especially if using any proprietary case data, ensure licensing is respected or use public domain data). Keep personally identifiable information and client data safe – possibly anonymize inputs in logs, etc. By building in compliance (like the zero-retention policy for training [19] and strong security), we make the system more acceptable in a conservative domain like law.

6. Future Roadmap (Post-MVP)

(Briefly reiterating future plans, to contextualize that the MVP is built with these in mind.)

- **Phase 2 – Enhanced Data and Model:** Increase coverage of Indian legal data (target tens of thousands of documents, covering all major areas of law). Work on fine-tuning the LLM or integrating a specialty model for Indian legal reasoning. Possibly develop a citation resolver that can automatically convert a citation in text to a hyperlink (for a richer experience).
- **Phase 3 – Workflow Integration:** Develop MS Word and Outlook plugins so lawyers can use the AI from their existing workflow (e.g., ask questions while drafting or summarize an email contract attachment). Also, integrate with document management systems common in law firms (e.g., SharePoint, iManage) to pull in firm-specific knowledge with permission [10] .
- **Phase 4 – AI Agents and Automation:** Introduce multi-step agents for tasks like contract review (the AI goes through a contract and produces an issue list or annotation), litigation prep (summarize all cases relevant to a matter), or regulatory compliance tracking (alerting the user to new changes in law relevant to them). This will likely involve more complex planning algorithms and possibly scheduling tasks in the background.
- **Phase 5 – Productization and Scaling:** Harden the system for production use by multiple clients. This includes multi-tenancy, robust user management, billing (if offered as a service), and obtaining necessary certifications (ISO, SOC 2) to build client trust. Expand to other jurisdictions

as needed (e.g., support common law of other countries if an Indian firm has international needs, or at least be aware of cross-border law aspects).

- **Phase 6 – Continuous Improvement:** Based on user feedback and advancements in AI, continuously improve answer quality. If GPT-5/6 becomes available or open-source LLMs get better, evaluate switching or offering on-prem LLMs for clients who want that. Keep the knowledge base updated daily. Strive to remain the **most accurate and reliable AI legal assistant in the Indian market**, maintaining a lead over any competitors by superior tech and integration of domain expertise.

7. Conclusion

This project plan outlines a comprehensive yet agile path to building the **Indian Legal AI Assistant MVP**. By following the best practices in AI application development – especially focusing on RAG for accuracy, using a scalable cloud infrastructure, and integrating domain knowledge at every step – we will create a foundation for a transformative tool for lawyers. The plan emphasizes reliability (no errors, verified information) and relevance to the Indian context, which will be our key differentiators. With this plan, the development (whether by human engineers or an AI agent) should proceed in a structured manner, resulting in a resilient and dependable system. By the end of the MVP phase, we expect to have a working product that can be piloted with real users, showcasing capabilities that **meet or exceed what Harvey AI offers, tailored to India's legal landscape**, and setting the stage for further innovations in legal tech.

---

1 4 5 6 7 9 11 14 15 16 17 18 20 25 26 Enterprise-Grade RAG Systems
https://www.harvey.ai/blog/enterprise-grade-rag-systems

2 3 8 10 12 13 19 21 22 23 24 How Harvey Built Trust in Legal AI: A Case Study for Builders | by Takafumi Endo | Sep, 2025 | Medium
https://medium.com/@takafumi.endo/how-harvey-built-trust-in-legal-ai-a-case-study-for-builders-786cc23c3b6d