iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

CONTRACTSENSE AI: AN INTELLIGENT LEGAL DOCUMENT ANALYSIS SYSTEM USING RETRIEVAL-AUGMENTED GENERATION

Madhan D,
Research Scholar, School of Computer Sciences, VISTAS, Chennai, India.

Dr. Sangeetha Radhakrishnan,
Assistant Professor, School of Computer Sciences, VISTAS, Chennai, India.

ABSTRACT
ContractSense AI is a production-ready Retrieval-Augmented Generation (RAG) web application that analyzes legal contracts for risk, surfaces dangerous clauses, and enables interactive question-answering over uploaded PDF documents. The system combines FastAPI with LangChain orchestration, a locally running sentence-transformer embedding model (all-MiniLM-L6-v2), and the Groq-hosted Llama 3.3 70B large language model for fast inference. A custom NumpyVectorStore provides lightweight vector similarity search without external database dependencies. The frontend features an animated Three.js landing page and a fully interactive three-panel dashboard with risk gauge, red-flag cards, TL;DR summary, and RAG-powered conversational chat. All embedding computation runs locally — no cloud embedding API is required — making the system inherently privacy-preserving and cost-free to operate at the inference layer. The system is designed for deployment on standard personal computing hardware without requiring GPU acceleration, making it accessible to students, freelancers, and small organisations with limited IT infrastructure.

Index Terms
Retrieval-Augmented Generation, Legal Document Analysis, Large Language Models, FastAPI, LangChain, Sentence Transformers, Risk Assessment, Vector Similarity Search, Groq, Natural Language Processing.

I. INTRODUCTION

Legal contracts govern virtually every professional and business relationship. Yet the complexity of legal language makes it difficult for non-experts to identify risky clauses, understand obligations, or ask targeted questions about specific provisions. Traditional contract review is time-consuming, expensive, and requires specialised expertise unavailable to most individuals and small organisations.

ContractSense AI addresses this gap by combining modern natural language processing with a clean, interactive web interface. Users upload any PDF contract and receive an instant risk score, a plain-English summary of the three most critical clauses, a list of favorable and unfavorable terms, and the ability to ask open-ended questions — all powered by a local RAG pipeline requiring only one free API key (Groq).

The system is motivated by two principal observations. First, existing contract review tools are either expensive enterprise solutions costing thousands of dollars per year or cloud-based APIs that transmit sensitive legal documents to third-party servers, raising serious data privacy concerns. Second, recent advances in open-weight large language models and efficient sentence-transformer embedding models have made it feasible to deploy a capable legal analysis pipeline on commodity hardware at near-zero operational cost.

ContractSense AI runs the full embedding pipeline locally on CPU, requiring only the free Groq API for LLM inference. This design makes it accessible for students, researchers, freelancers, and small businesses. The system is architected around a four-stage RAG pipeline — extraction, chunking, embedding, and retrieval-generation — described in detail in Section IV.

The remainder of this paper is organised as follows. Section II presents a system overview. Section III documents the technology stack. Sections IV and V describe the RAG pipeline and core features. Sections VI and VII cover the API and frontend design. Section VIII presents performance evaluation. Section IX discusses engineering challenges and solutions. Section X concludes with directions for future work.

II. SYSTEM OVERVIEW

ContractSense AI follows a modular client-server architecture. The backend is a Python FastAPI application exposing REST endpoints for document upload, processing status, risk analysis, summarisation, and

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

chat. The frontend consists of two HTML pages: an animated Three.js landing page and an interactive three-panel dashboard. All AI processing occurs in the backend RAG pipeline, which is stateless with respect to the HTTP layer and communicates solely through session-keyed temporary files and in-memory stores.

The high-level data flow is linear: PDF Upload → Text Extraction → Chunk and Embed → NumpyVectorStore → Groq LLM → Structured Output. This pipeline is triggered once per document upload and results are cached for the duration of the session. Subsequent chat queries bypass re-embedding and interact directly with the persisted vector store, reducing per-query latency to the sum of similarity search and LLM inference time.

Session management is handled through UUIDs generated at upload time. Each session maintains its own vector store pickle file and extracted text file in a temporary directory, providing isolation between concurrent users. Sessions are cleaned up either on explicit DELETE request or after a configurable timeout, ensuring that sensitive document data does not persist indefinitely on the server.

III. TECHNOLOGY STACK

The system is implemented entirely using open-source components, with the sole exception of the Groq API for LLM inference (free tier). Table I summarises the technology stack across all system layers. All components are compatible with Python 3.10 on Windows, Linux, and macOS without platform-specific configuration.

TABLE I: Technology Stack

Layer	Technology	Purpose
Web Framework	FastAPI 0.111 + Uvicorn	Async REST API, background tasks
LLM	Groq — llama-3.3-70b-versatile	Risk analysis, summarisation, Q&A
Embeddings	all-MiniLM-L6-v2	Local semantic embedding, no API key
RAG Orchestration	LangChain 0.2	Prompt templates, text splitting
Vector Store	Custom NumpyVectorStore	Cosine similarity, pickle persistence
PDF Parsing	PyMuPDF + pdfplumber	Text extraction with fallback
Frontend	HTML5 + Tailwind CSS + Vanilla JS	Dashboard, drag-drop, chat
3D Animation	Three.js r128	Landing page particle field
Runtime	Python 3.10+	Windows / Linux / macOS

IV. RAG PIPELINE DESIGN

The RAG pipeline is the core of ContractSense AI. It converts a raw PDF into a queryable knowledge base enabling the LLM to answer questions grounded in actual contract text rather than parametric knowledge. The pipeline has four sequential stages: extraction, chunking, embedding, and retrieval-generation. Each stage is implemented as a discrete, testable function with explicit input and output contracts.

4.1. PDF Text Extraction

PyMuPDF (fitz) serves as the primary extraction engine for its speed and accuracy on standard PDF layouts. If fewer than 200 characters are extracted from a document page — indicating a scanned image or complex multi-column layout — pdfplumber is invoked as a fallback. Extracted text is cleaned by removing non-ASCII characters, collapsing repeated whitespace, and re-joining hyphenated line breaks to restore word continuity across pages.

4.2. Text Chunking

LangChain's RecursiveCharacterTextSplitter divides the cleaned text into overlapping chunks of 1,200 tokens with a 200-token overlap. The overlap ensures that context spanning chunk boundaries is preserved during retrieval, reducing the risk of incomplete clause representation. Separators are tried in descending priority: paragraph breaks, line breaks, sentence endings, spaces, and individual characters.

4.3. Local Embedding with Sentence-Transformers

Each chunk is converted into a dense 384-dimensional vector using the all-MiniLM-L6-v2 model. The model runs entirely on local CPU — no external API call is made at embedding time. Batches of 32 chunks are

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

processed simultaneously for throughput efficiency. The model is downloaded once (approximately 90 MB) on first run and cached locally by the Hugging Face hub, requiring no repeated downloads for subsequent sessions.

4.4. NumpyVectorStore

Rather than ChromaDB — which introduced gRPC and SQLite compatibility issues on Windows — ContractSense AI implements a lightweight custom vector store. Embeddings and corresponding text chunks are maintained as Python lists in memory and persisted to disk as a pickle (.pkl) file keyed to the session UUID. Retrieval uses cosine similarity computed in pure Python without NumPy array operations beyond basic dot products, returning the top-5 ranked chunks. This design eliminates all external database dependencies.

4.5. Retrieval and Generation

At query time the user question is embedded using the same MiniLM model to produce a 384-dimensional query vector. The top-5 most similar contract chunks are retrieved from the NumpyVectorStore by cosine similarity and concatenated as the context block. The context, together with the user question, is assembled into a LangChain PromptTemplate and dispatched to the Groq Llama 3.3 70B endpoint. The model is explicitly instructed not to hallucinate: if the requested information is not present in the provided contract context, the model returns a clearly stated negative response. Conversation history comprising the last six turns is maintained in session state, enabling contextual follow-up questions without re-uploading the document.

V. CORE FEATURES

5.1. Risk Score Engine

Risk scoring uses a blended approach: LLM semantic analysis contributing 60% of the final score is combined with a deterministic keyword heuristic contributing 40%. The LLM returns a 0–100 score derived from full-context reasoning over the complete contract text. In parallel, 33 high-risk legal terms are scanned and weighted by severity: terms such as indemnification carry three points, auto-renewal carries two points, and amendment carries one point. The two scores are blended and clamped to the 0–100 range. Table II documents the four risk classification levels and their associated gauge colours.

Risk Level	Score Range	Gauge Color	Typical Triggers
LOW	0 – 24	Green	Standard terms, balanced obligations
MEDIUM	25 – 49	Amber	Confidentiality, jurisdiction clauses
HIGH	50 – 74	Orange	Non-compete, IP assignment, arbitration
CRITICAL	75 – 100	Red	Unlimited indemnity, perpetual exclusivity

TABLE II: Risk Level Classification

5.2. Red Flag Detection

The LLM is prompted with a structured JSON schema requiring exactly three red flags, each containing a short clause name, a severity level (LOW, MEDIUM, or HIGH), a plain-English explanation suitable for non-legal readers, and an exact verbatim excerpt from the contract. These are rendered as styled cards in the dashboard with colour-coded severity badges. The structured JSON output format is enforced through prompt engineering rather than function calling, providing compatibility with the Groq API's standard completion endpoint.

5.3. TL;DR Summary

A separate LangChain chain summarises the full contract into four components: a two-to-three sentence plain-English overview of the agreement's purpose and parties, three critical clauses each annotated with their potential business impact and a recommended action for the reader, a bullet list of favorable terms, and a bullet list of unfavorable terms. This summary chain executes in parallel with risk analysis using FastAPI's Background Tasks scheduler, minimising total processing latency by overlapping independent LLM calls.

5.4. Interactive Q&A Chat

The conversational chat interface allows users to pose free-form questions about any aspect of the uploaded contract. Each message submission triggers a fresh vector similarity search against the session's NumpyVectorStore, retrieving the top-5 most relevant chunks as grounded context for the LLM response. The model is instructed to reference specific clause numbers or section headings when available, and to explicitly state when the requested information is absent from the contract, preventing hallucination in the chat context.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

VI. API REFERENCE

The backend exposes a RESTful API with seven endpoints, summarised in Table III. All endpoints accept and return JSON. The session_id returned by the upload endpoint is a UUID4 string that serves as the primary key for all subsequent operations within a user session. The health endpoint confirms that both the Groq API key and the embedding model are correctly configured.

Method	Endpoint	Description
POST	/api/upload	Upload PDF; returns session_id immediately
GET	/api/status/{id}	Poll status: processing / ready / error
GET	/api/analyze/{id}	Risk score, level, red flags, key parties
GET	/api/summary/{id}	TL;DR, critical clauses, favorable/unfavorable terms
POST	/api/chat	RAG Q&A: submit question, receive grounded answer
DELETE	/api/session/{id}	Clean up session and temporary files
GET	/api/health	Health check; confirms API keys configured

TABLE III: API Endpoints

VII. FRONTEND DESIGN

7.1. Landing Page

The landing page (index.html) features a Three.js WebGL particle field rendering 900 floating points connected by proximity-based line segments, two wireframe document meshes that orbit a central axis, and a rotating torus ring. Scroll-reveal animations are triggered using Intersection Observer to progressively expose feature sections as the user scrolls. A custom CSS cursor replaces the system pointer with a spring-interpolated ring effect that trails mouse movement. Animated counter elements tally system metrics such as analysis speed and supported document types as they enter the viewport.

7.2. Dashboard

The three-panel dashboard (app.html) is the primary user interface. The left panel contains the drag-and-drop file upload zone, a processing progress bar with stage labels, an animated SVG arc gauge displaying the computed risk score, and red-flag cards with colour-coded severity badges and quoted contract excerpts. The centre panel displays document metadata in a header block, the TL;DR summary, critical clause cards with recommended actions, and a two-column favorable/unfavorable terms grid. The right panel is the RAG chat interface, featuring message bubbles with role-differentiated styling, animated typing indicator dots during LLM generation, pre-populated suggestion chips for common queries, and an auto-resizing textarea that sends messages on Enter key press.

VIII. PERFORMANCE

Table IV presents observed latency measurements for each pipeline stage, collected during testing on a mid-range laptop with an Intel Core i7 processor and no GPU. All LLM calls were routed through the Groq API; all embedding operations were performed locally on CPU. The total first-analysis latency of under 12 seconds encompasses extraction, chunking, embedding, and two parallel LLM calls (risk analysis and TL;DR summary).

TABLE IV: System Performance Metrics

Metric	Observed Value	Notes
PDF text extraction	< 1 second	PyMuPDF on 10-page contract
Local embedding (10 chunks)	2 – 4 seconds	CPU-only, all-MiniLM-L6-v2
Risk analysis (Groq LLM)	3 – 6 seconds	llama-3.3-70b via Groq API
TL;DR summary (Groq LLM)	2 – 5 seconds	Parallel with risk analysis
RAG Q&A response	2 – 4 seconds	Similarity search + LLM generation
Total first analysis	< 12 seconds	End-to-end for average contract
Model download (first run)	~33 seconds	90 MB one-time download

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Metric	Observed Value	Notes
Max PDF file size	20 MB	Configurable in main.py

The parallel execution of risk analysis and TL;DR summary using FastAPI BackgroundTasks is the primary optimisation responsible for keeping total latency below 12 seconds. Sequential execution of both LLM calls would increase total latency to approximately 16–18 seconds for an average 10-page contract. The 90 MB model download occurs only on first launch and is thereafter served from the local Hugging Face cache, adding no overhead to subsequent runs.

IX. CHALLENGES AND SOLUTIONS

The development of ContractSense AI encountered several non-trivial engineering challenges arising from dependency conflicts and third-party API changes. Table V documents the four principal challenges encountered, their root causes, and the solutions implemented.

TABLE V: Engineering Challenges and Solutions

Challenge	Root Cause	Solution
gRPC SecretStr TypeError	langchain-google-genai always inits gRPC; Pydantic v1 SecretStr conflict	Replaced Google Gemini with local sentence-transformers; eliminated gRPC entirely
ChromaDB SQLite mismatch	ChromaDB 0.5.3 added "topic" column; posthog telemetry API conflict	Replaced ChromaDB with custom NumpyVectorStore; pickle persistence
Groq model deprecation	llama3-8b-8192 decommissioned by Groq	Updated to llama-3.3-70b-versatile
Google embedding 404	Generative Language API not enabled in Google Cloud Console	Migrated to local sentence-transformers; removed Google API dependency

The migration from ChromaDB to a custom NumpyVectorStore is the most architecturally significant decision made during development. While ChromaDB offers richer functionality including persistent collections and metadata filtering, its dependency on gRPC and a specific SQLite schema version introduced platform-specific failures that were not reliably reproducible across Windows and Linux environments. The custom NumpyVectorStore trades these features for zero-dependency operation and full cross-platform compatibility, at the cost of O(n) linear scan retrieval rather than approximate nearest-neighbour indexing.

X. CONCLUSION

ContractSense AI demonstrates a practical, production-ready application of Retrieval-Augmented Generation to the legal domain. By combining a locally running sentence-transformer embedding model with the Groq LLM API, the system delivers sub-12-second full contract analysis with zero cloud embedding cost and strong privacy guarantees. The custom NumpyVectorStore eliminates problematic database dependencies while maintaining effective semantic retrieval quality for typical legal documents.

The system's modular architecture and reliance on well-maintained open-source components makes it straightforward to extend. The most impactful future enhancement would be the integration of a domain-specific legal embedding model fine-tuned on contract corpora, replacing all-MiniLM-L6-v2 with a model that encodes legal semantics more precisely.

Future work could include multi-document comparison enabling side-by-side risk assessment of contract versions, clause-level negotiation suggestions powered by a fine-tuned instruction model, integration with e-signature platforms to create a complete contract lifecycle management tool, and replacement of the NumpyVectorStore with a production vector database such as Qdrant or Weaviate for deployments handling large document collections. The animated Three.js frontend demonstrates that AI-powered tools need not sacrifice user experience for technical depth.

REFERENCES

[1] Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 2020.
[2] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP 2019.
[3] LangChain Documentation. (2024). LangChain: Building Applications with LLMs. https://docs.langchain.com
[4] Groq Inc. (2024). Groq LPU Inference Engine. https://console.groq.com/docs

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

[5] FastAPI Documentation. (2024). FastAPI: High Performance Web Framework. https://fastapi.tiangolo.com
[6] Hugging Face. (2024). all-MiniLM-L6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[7] Johnson, J., Douze, M., & Jegou, H. (2019). Billion-scale similarity search with GPUs. IEEE Trans. Big Data.
[8] Three.js. (2024). Three.js JavaScript 3D Library. https://threejs.org