Architecting Sovereign Legal Intelligence: A Technical Framework for Indigenous NLP in the Indian Judiciary

1. The Imperative for Sovereign Legal AI in the Indian Context

The Indian judicial system, arguably one of the most prolific and complex legal ecosystems in the world, faces an unprecedented operational challenge. With pending cases crossing the 50 million mark—over 5.3 crore as of late 2025—the judiciary is burdened by an administrative weight that threatens the fundamental right to speedy justice.[1] This backlog is not merely a statistical anomaly but a structural crisis exacerbated by the manual processing of voluminous, unstructured legal texts. In this landscape, the deployment of Artificial Intelligence (AI), specifically Natural Language Processing (NLP), transitions from an experimental novelty to an infrastructural necessity. However, the adoption of generic, Western-centric Large Language Models (LLMs) presents significant risks regarding data sovereignty, linguistic alignment, and hallucination in high-stakes environments. Therefore, the development of custom, domain-specific models—mirroring the architectural successes of missions like **OpenNyAI** and platforms like **NyaySetu**—is critical for building a "Sovereign Legal Intelligence" infrastructure.[1]

This report delineates a comprehensive technical roadmap for engineering such systems. It moves beyond superficial API wrappings to explore the deep learning architectures required to navigate the linguistic labyrinth of the Indian Penal Code (IPC), the Code of Criminal Procedure (CrPC), and the distinct "Indian Legal English" found in Supreme Court and High Court judgments. We analyze the dichotomy and necessary convergence between discriminative architectures (like **BERT** and its derivative **InLegalBERT**) for structural extraction and generative architectures (like **Llama 3**) for legal reasoning and summarization. By synthesizing protocols for data engineering, model fine-tuning, and enterprise-grade deployment, we establish a blueprint for a digital judiciary that is efficient, transparent, and constitutionally aligned.

2. Data Engineering: The Bedrock of Legal Intelligence

The efficacy of any legal NLP model is deterministically bound to the quality and specificity of its training corpus. In the Indian context, this requires a departure from standard datasets like "Pile of Law" (US/EU centric) toward the curation of indigenous datasets that capture the syntactic idiosyncrasies of Indian courts—ranging from the usage of Latin maxims like *suo moto* and *res judicata* to the specific citation standards of the All India Reporter (AIR) and

Supreme Court Cases (SCC) journals.

2.1 Sovereign Data Acquisition Strategies

Constructing a robust training corpus necessitates the aggregation of millions of judgment texts, a task that requires navigating the fragmented digital infrastructure of Indian courts. While the **e-Courts** project has digitized vast archives, programmatic access remains challenging, necessitating sophisticated scraping and data standardization pipelines.

2.1.1 Web Scraping Architectures for Indian Kanoon and e-Courts

The primary repositories for Indian legal data are **Indian Kanoon** and the official **e-Courts** services. To build a training dataset, one must implement scraping architectures capable of traversing these repositories while respecting rate limits and handling varied document formats.

Python-based scraping frameworks utilizing Selenium and BeautifulSoup are the standard for this extraction.[3] A production-grade scraping pipeline typically involves:

- **Index Crawling:** Systematically iterating through year-wise and court-wise indices to harvest Document IDs. This is critical for ensuring temporal and jurisdictional coverage, preventing biases toward recent judgments or specific High Courts.
- **Content Extraction:** Fetching raw HTML content is preferred over PDF parsing for initial text acquisition. HTML structures often preserve semantic signals—such as blockquotes for precedents or bold tags for headers—that are vital for downstream tasks like Rhetorical Role Labeling.[5]
- **CAPTCHA Handling:** Accessing official e-Courts portals often triggers CAPTCHA challenges. Advanced pipelines integrate OCR-based solvers (using libraries like EasyOCR) or headless browser sessions to navigate these gateways without manual intervention, ensuring continuous data ingestion.[6]

2.1.2 Utilization of Pre-Compiled Corpora

For teams accelerating past the scraping phase, several high-quality, pre-compiled datasets serve as foundational benchmarks:

- **Indian Legal Documents Corpus (ILDC):** This corpus contains approximately 35,000 Supreme Court cases annotated with original court decisions. It is particularly valuable for training models on judgment prediction and outcome analysis, serving as a baseline for understanding judicial reasoning patterns.[7]
- **InJudgements:** This dataset offers a representative sample of Indian court judgments controlled for court hierarchy and case type. By balancing data across civil, criminal, constitutional, and tax domains, it prevents model overfitting to dominant case types (like criminal appeals) and ensures broader generalization.[8]
- **Aalap Instruction Dataset:** Tailored for generative tasks, this dataset structures legal data into instruction-response pairs (e.g., "Generate an argument for the petitioner

based on Section 302 IPC"). This format is essential for instruction-tuning LLMs like Llama 3 to perform reasoning rather than mere text completion.[9]

2.2 Advanced Text Cleaning and Regex Normalization

Raw legal text is notoriously "noisy," filled with procedural artifacts, scanning errors, and inconsistent formatting. A rigorous preprocessing pipeline using Regular Expressions (Regex) is non-negotiable for converting scraped HTML into clean, tokenizable text.

2.2.1 Regular Expressions for Legal Entity Extraction

Standardizing legal entities is crucial for the model to recognize references accurately.

- **Citation Standardization:** Indian legal citations appear in myriad formats. A model must understand that "AIR 1973 SC 1461" and "1973 AIR SC 1461" refer to the same document. Regex patterns such as r'\d{4}\s+AIR\s+SC\s+\d+' or dynamic patterns capable of capturing variable reporter names (SCC, SCR, SCALE) are used to normalize these citations into a canonical format before tokenization.[10]
- **Statutory Reference Parsing:** Extracting Acts and Sections requires patterns that account for abbreviations and varying spacing. A robust pattern like r'(?i)(?:section|sec\.?|u/s)\s+(\d+[A-Z]*)\s+(?:of\s+)?(?:the\s+)?([A-Za-z\s]+Act)' captures the section number (including alphanumeric variants like 304A) and the associated Act, enabling the model to link facts to statutes.[12]
- **Party Name Segmentation:** Distinguishing the "Petitioner" from the "Respondent" is fundamental for case analysis. Regex patterns utilizing the adversarial delimiters—r'(.+?)\s+(?:vs\.?|versus|v\.)\s+(.+)'—allow the system to parse case titles (e.g., "Kesavananda Bharati v. State of Kerala") into structured metadata fields.[13]

2.2.2 Artifact Removal and Text Normalization

Scraped text often contains non-semantic artifacts that waste context window space.

- **HTML and Whitespace Cleaning:** Functions using re.sub(r'<[^>]+>', '', text) remove residual HTML tags. Further, re.sub(r'\s+', ' ', text) collapses multiple spaces, tabs, and line breaks into single spaces, ensuring that the model sees continuous prose rather than fragmented lines typical of PDF-to-text conversions.[14]
- **Anonymization (Redaction):** To uphold privacy and ethical standards, especially in sensitive cases (e.g., POCSO or family law), regex patterns must identify and redact Personal Identifiable Information (PII) like mobile numbers (r'\d{10}'), email addresses, and home addresses before the data enters the training pipeline.[14]

2.3 Semantic Chunking for Long-Context Processing

Legal judgments often exceed the context limits of standard models (512 tokens for BERT, 8k for Llama 3). "Chunking" is the process of breaking these documents into smaller segments without losing semantic meaning.

- **Structure-Aware Chunking:** Naive splitting by character count often severs sentences or legal arguments. Semantic chunking leverages document structure—splitting at paragraph boundaries (\n\n) or section headers—to maintain coherence.
- **Overlapping Windows:** To preserve context across boundaries, a sliding window approach with 10-20% overlap is standard. This ensures that if a reference to a "Section" appears at the end of Chunk A, and the "Act" appears at the start of Chunk B, the model can still resolve the relationship.[16]
- **Hierarchical Indexing:** For advanced Retrieval-Augmented Generation (RAG) systems, chunks are not just stored flat but indexed hierarchically. A "Parent" node might represent the full case summary, while "Child" nodes represent specific arguments or facts. This allows the system to retrieve precise details while maintaining access to the broader case context.[18]

3. The OpenNyAI Architecture: A Structural Blueprint

The **OpenNyAI** mission provides the reference architecture for Indian Legal NLP. It systematically decomposes legal analysis into three specific sub-tasks: Named Entity Recognition (NER), Rhetorical Role Labeling (RRL), and Summarization. Replicating this modular architecture is essential for building systems that are both interpretable and effective.[19]

3.1 Named Entity Recognition (NER) for Indian Law

Unlike generic NER models that identify people and locations, Legal NER must extract specialized entities. OpenNyAI defines 14 specific entity classes for Indian judgments:

- **Primary Entities:** COURT, PETITIONER, RESPONDENT, JUDGE, LAWYER.
- **Legal Objects:** STATUTE (e.g., Indian Penal Code), PROVISION (e.g., Section 302), PRECEDENT (citations to past cases), CASE_NUMBER.
- **Temporal/Spatial:** DATE, GPE (Geopolitical Entity), ORG (Organization).

Architectural Insight: The baseline models for this task often utilize **RoBERTa** or **LegalBERT** fine-tuned on token classification. However, experimentation has shown that **InLegalBERT**—pretrained on Indian texts—significantly outperforms Western counterparts in recognizing Indian entity names and statutory abbreviations (e.g., "CrPC" vs "Criminal Procedure Code").[21]

3.2 Rhetorical Role Labeling (RRL)

RRL is the process of segmenting a judgment into its constituent functional parts. This is critical because a "fact" stated in the "Arguments" section is merely a claim, whereas a "fact" in the "Analysis" or "Ratio" section is a judicial finding.

- **The 13 Rhetorical Roles:** The schema typically includes PREAMBLE, FACTS, ISSUE, ARGUMENT_PETITIONER, ARGUMENT_RESPONDENT, ANALYSIS, STATUTE, PRECEDENT_RELIED, PRECEDENT_NOT_RELIED, RATIO, RULING_LOWER_COURT, RULING_PRESENT_COURT (RPC).
- **Sequential Modeling:** Since the role of a sentence is heavily dependent on its neighbors (e.g., a "Conclusion" follows "Analysis"), architectures that combine Transformers with sequential layers like **BiLSTM-CRF** are preferred. The **MARRO** model (Multi-Headed Attention for Rhetorical Role Labeling) represents the state-of-the-art here, using attention mechanisms to link distant sentences (e.g., linking the final decision back to the initial issue).[23]

3.3 Extractive Summarization via Rhetorical Roles

Legal summarization differs from news summarization; it must capture specific legal elements. By leveraging RRL, summarization pipelines can be rule-based or learned. For instance, a summary template might explicitly require: "The Issue (from ISSUE role) was X. The Court relied on (from PRECEDENT_RELIED) Y. The final order (from RPC) is Z." This ensures that the generated summary is legally sound and structured, avoiding the "black box" unpredictability of purely abstractive methods.[19]

4. Discriminative Modeling: Building Custom BERT Models (InLegalBERT)

For tasks requiring structural analysis—specifically NER and RRL—discriminative models like BERT remain the gold standard due to their precision, speed, and lower computational cost compared to LLMs.

4.1 InLegalBERT: The Sovereign Foundation

InLegalBERT serves as the foundational encoder for Indian Legal NLP. Unlike the standard bert-base-uncased, InLegalBERT has been pre-trained on a massive corpus of 5.4 million Indian legal documents, encompassing roughly 27GB of text. This domain-specific pre-training ensures that the model's vocabulary includes Indian legal terms and that its attention mechanisms are attuned to the syntactic structures of Indian judgments. Utilizing InLegalBERT as a starting point for fine-tuning yields significantly faster convergence and higher F1 scores than starting with generic BERT.[26]

4.2 Fine-Tuning Pipeline for Rhetorical Role Labeling

Training a model to identify Rhetorical Roles involves a specific pipeline:

1. **Architecture:** The model architecture typically stacks a classification head on top of InLegalBERT. For state-of-the-art performance, a **BiLSTM-CRF** layer is added. The BiLSTM (Bidirectional LSTM) aggregates context from the entire sequence of sentence

embeddings, while the CRF (Conditional Random Field) enforces valid transitions between labels (e.g., ensuring a PREAMBLE doesn't randomly appear in the middle of ANALYSIS).
2. **Dataset:** The **BUILD** (Benchmark for Understanding Indian Legal Documents) corpus is the standard dataset, containing sentence-level annotations for rhetorical roles.
3. **Training Dynamics:** Training involves minimizing a loss function (typically Cross-Entropy for Softmax or Negative Log-Likelihood for CRF). Hyperparameters must be tuned carefully; a lower learning rate (e.g., 2e-5) is used for the BERT layers to preserve pre-trained knowledge, while a higher rate (e.g., 1e-3) may be applied to the randomly initialized BiLSTM-CRF layers.[28]

4.3 Fine-Tuning Pipeline for Legal NER

NER is modeled as a token classification task.

- **Data Formatting:** The training data must be in **CoNLL** or Hugging Face Dataset format, where each word token is mapped to a BIO tag (e.g., B-STATUTE, I-STATUTE, O).
- **Tokenization Strategy:** A critical challenge is sub-word tokenization. Legal terms might be split into multiple sub-words by the tokenizer. The labeling strategy must account for this, typically by assigning the label of the word to its first sub-word token and ignoring (or labeling as X) subsequent sub-words.
- **Optimization:** Training utilizes the AdamW optimizer with a linear learning rate scheduler. Evaluation is performed using the seqeval library, which calculates strict F1 scores based on exact entity boundary matching, ensuring that "Section 302" is captured as a whole entity, not just "Section" or "302".[21]

Metric	**BERT-Base**	**InLegalBERT**	**Improvement**
NER F1 Score	~78%	~84%	+6%
RRL Accuracy	~72%	~79%	+7%
Convergence	Slower	Faster	~30% fewer epochs

Table 1: Comparative performance of Generic BERT vs. InLegalBERT on Indian Legal Tasks [27]

5. Generative Modeling: Training Custom Llama 3 Models

While BERT excels at extraction, **Llama 3** (8B and 70B variants) introduces the generative capabilities required for reasoning, drafting, and complex Q&A. Fine-tuning these models transforms them from generic predictors into specialized "Legal Assistants."

5.1 Instruction Tuning: The Paradigm Shift

Fine-tuning Llama 3 for legal tasks is not about "next token prediction" on raw text, but **Instruction Tuning**. The model is trained to follow specific instructions, bridging the gap between raw knowledge and task execution.

- **Dataset Structure:** Training data is formatted into JSON/JSONL entries containing instruction, input (context), and output. For example, the **Aalap** dataset transforms raw judgments into tasks like "Draft a counter-argument for the respondent based on these facts."
- **Legal Reasoning over Recall:** To minimize hallucinations, the training strategy emphasizes reasoning (e.g., "Apply Section 302 to these facts") rather than recall (e.g., "Recite Section 302"). The input context provides the necessary legal statutes, forcing the model to learn the *application* of law.[9]

5.2 Parameter-Efficient Fine-Tuning (PEFT) with LoRA

Full fine-tuning of an 8B parameter model is computationally prohibitive for many organizations. **LoRA (Low-Rank Adaptation)** offers a sovereign-friendly alternative.

- **Mechanism:** LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into the layers of the Transformer. This reduces the number of trainable parameters by 99%, allowing fine-tuning on consumer-grade GPUs (e.g., NVIDIA RTX 3090/4090 or A100s).
- **QLoRA (Quantized LoRA):** For even greater efficiency, QLoRA loads the base model in 4-bit precision (NF4 format) while keeping the LoRA adapters in higher precision (BFloat16). This technique allows an 8B model to be fine-tuned on a single GPU with less than 16GB VRAM.
- **Hyperparameters:** Key configurations include lora_alpha (scaling factor), lora_dropout, and the target modules. For legal reasoning, targeting all linear modules (q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj) typically yields better reasoning capabilities than targeting attention modules alone.[31]

5.3 Structured Output via Prompt Engineering

To integrate generative models into software pipelines (like NyaySetu), their output must be machine-readable. Prompt engineering techniques enforce structured outputs, typically JSON.

- **System Prompts:** A robust system prompt establishes the persona and output constraints.*Example:* "You are an AI legal assistant. Analyze the provided case text. Output your analysis strictly as a JSON object with the following keys: 'case_summary', 'statutes_cited' (list), 'final_verdict'. Do not include any conversational text outside the JSON."

- **Grammar-Constrained Decoding:** Advanced inference engines (like vLLM) allow for "grammar-constrained decoding," where the model is mathematically forced to generate tokens that conform to a specific JSON schema, eliminating syntax errors in the output.[34]

5.4 Case Study: Aalap – Indigenous Fine-Tuning

The **Aalap** project serves as a definitive case study. It involved fine-tuning a Mistral 7B (and later Llama) model on a curated dataset of Indian legal instructions.

- **Compute:** The training utilized 4x NVIDIA A100 GPUs and took approximately 88 hours.
- **Performance:** In human evaluations (and GPT-4 based auto-evaluations), the fine-tuned Aalap model outperformed base GPT-3.5 on specific Indian legal drafting tasks, validating the hypothesis that a smaller, domain-specialized model can beat a larger generalist model.[36]

6. Pipeline Integration: The NyaySetu Architecture

NyaySetu provides a practical reference architecture for deploying these models within a scalable, user-facing application. It moves beyond the model itself to the ecosystem required to support it.

6.1 Enterprise-Grade Backend Orchestration

The core of NyaySetu is an enterprise-grade backend built with **Spring Boot 3.2** (Java 17). This choice reflects the need for stability, security, and scalability in judicial infrastructure.

- **Microservices Pattern:** The backend acts as an orchestrator. It manages user authentication (Spring Security + JWT), case management logic, and database persistence (PostgreSQL).
- **Decoupled AI Services:** Crucially, the AI components are decoupled. The Spring Boot backend communicates with Python-based AI microservices (running FastAPI) via REST APIs or message queues. This allows the heavy AI inference workloads to scale independently of the transactional business logic.[38]

6.2 The "Vakil-Friend" Intelligence Layer

The **Vakil-Friend** module illustrates the application of generative AI for citizen empowerment.

- **Functionality:** It provides a conversational interface for citizens to understand their legal standing. It simplifies "Hard Legal Language" into vernacular (Hindi/English).
- **Inference Engine:** To address latency, NyaySetu migrated from generic APIs to **Groq LPU** (Language Processing Unit) inference. Groq's hardware deterministic execution allows Llama 3 models to generate text at speeds exceeding 500 tokens per second ("Blink-Speed"), making the chat experience real-time and responsive—a critical factor for user adoption.[38]

6.3 The "Evidence Vault": Blockchain-Anchored Integrity

A unique feature of NyaySetu is the integration of a **Digital Evidence Locker** anchored by Blockchain.

- **Mechanism:** When a document is uploaded, its SHA-256 hash is calculated and committed to a blockchain ledger. This ensures an immutable audit trail.
- **Trust:** In a judicial context, the integrity of the model's *input* is as important as the *output*. By guaranteeing that the document analyzed by the AI has not been tampered with, the system builds trust in the AI-generated summaries.[38]

7. Deployment and MLOps: Sovereign Serving Strategies

Deploying legal AI requires distinct considerations regarding data privacy ("Sovereignty") and computational efficiency.

7.1 Inference Servers: vLLM vs. Groq

Two primary paths exist for serving Llama 3 models:

1. **vLLM (Self-Hosted/Sovereign):** For deployments requiring absolute data isolation (e.g., inside a court's private data center), **vLLM** is the industry standard. Its "PagedAttention" algorithm manages GPU memory efficiently, enabling high-throughput serving of concurrent requests. It is significantly faster than standard Hugging Face pipelines and supports continuous batching.[39]
2. **Groq (Cloud/Hybrid):** For applications prioritizing speed, **Groq** offers unparalleled inference velocity. While traditionally a cloud service, Groq supports **LoRA Adapters**, allowing organizations to upload their custom fine-tuned weights (e.g., for Indian legal reasoning) and serve them via Groq's blazing-fast LPUs. This hybrid approach combines custom model intelligence with managed infrastructure speed.[40]

7.2 Scalable Integration Patterns

Integration between the Java/Spring Boot backend and the Python AI services follows robust patterns:

- **Asynchronous Communication:** Using **Spring WebClient** (Reactive) allows the backend to make non-blocking calls to the AI service. This prevents the server from freezing while waiting for a long legal summary to be generated.
- **FastAPI Wrappers:** The Python AI models are wrapped in **FastAPI** applications. These microservices expose endpoints like /analyze_evidence or /summarize_judgment. They handle the tokenization, model inference, and response formatting, returning clean JSON to the Java backend.[42]

Component	**Technology**	**Purpose in Legal AI Pipeline**
Orchestrator	Spring Boot 3.2	API Gateway, Auth, Business Logic
Database	PostgreSQL 16	Relational Data (Cases, Users, Metadata)
AI Serving	vLLM / Groq	High-throughput Model Inference
Model (Gen)	Llama 3 (8B/70B)	Reasoning, Drafting, Simplification
Model (Disc)	InLegalBERT	NER, Rhetorical Roles, Semantic Search
Integrity	Blockchain	Immutable Evidence Hashing

Table 2: Technology Stack for a Sovereign Legal AI System [38]

8. Future Directions and Ethical Guardrails

The trajectory of Indian Legal AI points toward increasingly autonomous agents, but this progress must be tempered with ethical guardrails.

8.1 "Human-in-the-Loop" Governance

Systems like NyaySetu explicitly adopt a **Human-in-the-Loop** (HITL) philosophy. AI outputs—whether summaries or drafted notices—are advisory, not binding. The final authority rests with the judge or lawyer. This is not just an ethical stance but a technical requirement to mitigate the risk of hallucination inherent in LLMs.[1]

8.2 Multilingual Justice (Bhashini Integration)

The future of Indian Legal AI is multilingual. Integrating models with **Bhashini** (India's National Language Translation Mission) APIs allows the system to bridge the gap between English-language High Court judgments and the vernacular needs of litigants in District Courts. Future fine-tuning efforts will likely focus on multilingual instruction tuning, enabling models to reason in English but explain in Hindi, Tamil, or Bengali.[20]

8.3 Retrieval-Augmented Generation (RAG)

To further reduce hallucinations, the next generation of models will move beyond pure fine-tuning to **RAG**. By connecting Llama 3 to a vector database (like Milvus or Chroma) containing the entire corpus of Indian Case Law, the model can cite specific paragraphs from retrieved precedents to support its generated answers. This moves the system from "creative writing" to "evidence-based reasoning".[38]

9. Conclusion

Building a custom legal NLP model for the Indian Judiciary is a sophisticated engineering endeavor that demands a harmonization of domain expertise and cutting-edge technology. It requires the meticulous curation of "Sovereign Data" from Indian Kanoon, the precision of discriminative models like **InLegalBERT** for structural analysis, and the reasoning capabilities of generative models like **Llama 3** for drafting and summarization.

The **OpenNyAI** and **NyaySetu** initiatives provide the architectural scaffolding for this transformation—demonstrating how modular pipelines, enterprise-grade backends, and secure deployment strategies can converge to solve real-world judicial challenges. By adhering to the technical protocols outlined in this report—from regex-based cleaning to LoRA fine-tuning and Groq-accelerated inference—technologists can build "Vakil-Friends" and judicial assistants that are not merely intelligent, but constitutionally aligned, secure, and truly sovereign. This is the path to clearing the "Case Mountain" and ensuring that in the digital age, justice remains accessible, transparent, and swift.

Works cited

1. Final_Project_Report_NyaySetu.pdf
2. Legal-NLP-EkStep/rhetorical-role-baseline: OpenNyAI is a mission aimed at developing open source software and datasets to catalyze the creation of AI-powered solutions to improve access to justice in India. BUILD is the first benchmark dataset created by OpenNyAI - GitHub, accessed January 22, 2026, https://github.com/Legal-NLP-EkStep/rhetorical-role-baseline
3. Web Scraping & NLP in Python - DataCamp, accessed January 22, 2026, https://www.datacamp.com/tutorial/web-scraping-python-nlp
4. Web-Scraping-of-Indian-Judgements/Indian Kanoon details web scrapingL.ipynb at main, accessed January 22, 2026, https://github.com/jasp9559/Web-Scraping-of-Indian-Judgements/blob/main/Indian%20Kanoon%20details%20web%20scrapingL.ipynb
5. Web Scraping & NLP - CLDSPN - Kaggle, accessed January 22, 2026, https://www.kaggle.com/code/zackakil/web-scraping-nlp-practical-exercise
6. vanga/indian-high-court-judgments - GitHub, accessed January 22, 2026, https://github.com/vanga/indian-high-court-judgments

7. ILDC (Indian Legal Documents Corpus) - OpenDataLab, accessed January 22, 2026, https://opendatalab.com/OpenDataLab/ILDC/download
8. opennyaiorg/InJudgements_dataset · Datasets at Hugging Face, accessed January 22, 2026, https://huggingface.co/datasets/opennyaiorg/InJudgements_dataset
9. opennyaiorg/aalap_instruction_dataset · Datasets at Hugging Face, accessed January 22, 2026, https://huggingface.co/datasets/opennyaiorg/aalap_instruction_dataset
10. Regular Expression HOWTO — Python 3.14.2 documentation, accessed January 22, 2026, https://docs.python.org/3/howto/regex.html
11. Regex for legal citations - Stack Overflow, accessed January 22, 2026, https://stackoverflow.com/questions/50263659/regex-for-legal-citations
12. re — Regular expression operations — Python 3.14.2 documentation, accessed January 22, 2026, https://docs.python.org/3/library/re.html
13. python - How to extract the substring between two markers? - Stack Overflow, accessed January 22, 2026, https://stackoverflow.com/questions/4666973/how-to-extract-the-substring-between-two-markers
14. Mastering Text Cleaning for NLP: Techniques and Applications | CodeSignal Learn, accessed January 22, 2026, https://codesignal.com/learn/courses/collecting-and-preparing-textual-data-for-classification/lessons/mastering-text-cleaning-for-nlp-techniques-and-applications
15. Exploring the capabilities of Natural Language Processing (NLP) in conducting legal analysis: An experiment using POCSO Judgments | by Sai Krishna Dammalapati | CivicDataLab | Medium, accessed January 22, 2026, https://medium.com/civicdatalab/exploring-the-capabilities-of-natural-language-processing-nlp-in-conducting-legal-analysis-88ef2b9dec9c
16. What are best practices for chunking lengthy legal documents for vectorization? - Milvus, accessed January 22, 2026, https://milvus.io/ai-quick-reference/what-are-best-practices-for-chunking-lengthy-legal-documents-for-vectorization
17. The Art of Document Chunking for LLM Applications - Agentset, accessed January 22, 2026, https://agentset.ai/blog/the-art-of-document-chunking-for-llm-applications
18. Long-Context Isn't All You Need: How Retrieval & Chunking Impact Finance RAG, accessed January 22, 2026, https://www.snowflake.com/en/engineering-blog/impact-retrieval-chunking-finance-rag/
19. Opennyai : An efficient NLP Pipeline for Indian Legal documents - GitHub, accessed January 22, 2026, https://github.com/OpenNyAI/Opennyai
20. AI, Justice, and the Ecosystem Approach – Notes from the OpenNyAI Mission By: Smita Gupta1 and Atreyo Banerjee2 - Social Innovations Journal, accessed January 22, 2026, https://socialinnovationsjournal.com/index.php/sij/article/download/7121/5944/218

90
21. En legal ner trf · Models - Dataloop, accessed January 22, 2026, https://dataloop.ai/library/model/opennyaiorg_en_legal_ner_trf/
22. [2211.03442] Named Entity Recognition in Indian court judgments - ar5iv - arXiv, accessed January 22, 2026, https://ar5iv.labs.arxiv.org/html/2211.03442
23. MARRO: Multi-headed Attention for Rhetorical Role Labeling in Legal Documents - arXiv, accessed January 22, 2026, https://arxiv.org/pdf/2503.10659
24. Neural architectures for rhetorical role labeling. - ResearchGate, accessed January 22, 2026, https://www.researchgate.net/figure/Neural-architectures-for-rhetorical-role-labeling_fig3_398384176
25. Indian Legal Judgment Summarization using LEGAL-BERT and BiLSTM model with Adaptive Length - EPJ Web of Conferences, accessed January 22, 2026, https://www.epj-conferences.org/articles/epjconf/pdf/2025/13/epjconf_icetsf2025_01043.pdf
26. InLegalBERT - PromptLayer, accessed January 22, 2026, https://www.promptlayer.com/models/inlegalbert
27. InLegalBERT · Models - Dataloop, accessed January 22, 2026, https://dataloop.ai/library/model/law-ai_inlegalbert/
28. legal-text-classification/IndianLegalBERT - Hugging Face, accessed January 22, 2026, https://huggingface.co/legal-text-classification/IndianLegalBERT
29. MARRO: Multi-headed Attention for Rhetorical Role Labeling in Legal Documents - arXiv, accessed January 22, 2026, https://arxiv.org/html/2503.10659v1
30. Fine-tune Llama 3 for text generation on Amazon SageMaker JumpStart - AWS, accessed January 22, 2026, https://aws.amazon.com/blogs/machine-learning/fine-tune-llama-3-for-text-generation-on-amazon-sagemaker-jumpstart/
31. The Ultimate Guide to Fine-Tune LLaMA 3, With LLM Evaluations - Confident AI, accessed January 22, 2026, https://www.confident-ai.com/blog/the-ultimate-guide-to-fine-tune-llama-2-with-llm-evaluations
32. Fine-Tuning Llama 3 with LoRA: Step-by-Step Guide - Neptune.ai, accessed January 22, 2026, https://neptune.ai/blog/fine-tuning-llama-3-with-lora
33. how to prepare dataset for fine tunining (llama 3.2 8b) : r/LocalLLaMA - Reddit, accessed January 22, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1ooajcl/how_to_prepare_dataset_for_fine_tunining_llama_32/
34. Named Entity Recognition - Mirascope, accessed January 22, 2026, https://mirascope.com/docs/mirascope/guides/more-advanced/named-entity-recognition
35. How to force LLama3.1 to respond with JSON only? : r/LocalLLaMA - Reddit, accessed January 22, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1eqayuq/how_to_force_llama31_to_respond_with_json_only/
36. OpenNyAI/aalap_legal_llm - GitHub, accessed January 22, 2026,

https://github.com/OpenNyAI/aalap_legal_llm
37. Aalap — A finetuned Mistral 7B legally trained model for Indian Legal System - Medium, accessed January 22, 2026, https://medium.com/@vanshajkerni/aalap-a-finetuned-mistral-7b-legally-trained-model-for-indian-legal-system-458b0dfde638
38. viru0909-dev/nyay-setu-working: Live Demo - GitHub, accessed January 22, 2026, https://github.com/viru0909-dev/nyay-setu-working
39. vLLM or llama.cpp: Choosing the right LLM inference engine for your use case, accessed January 22, 2026, https://developers.redhat.com/articles/2025/09/30/vllm-or-llamacpp-choosing-right-llm-inference-engine-your-use-case
40. LoRA Inference - GroqDocs, accessed January 22, 2026, https://console.groq.com/docs/lora
41. accessed January 22, 2026, https://console.groq.com/docs/lora#:~:text=Using%20the%20Fine%2DTuning%20API&text=This%20process%20involves%20two%20API,files%20to%20provide%20this%20service.
42. Integrate AI APIs with Spring Boot – Step-by-Step Guide - Credo Systemz, accessed January 22, 2026, https://www.credosystemz.com/blog/integrate-ai-apis-java-spring-boot-applications/
43. How to Call REST Services with WebClient in Spring Boot? - GeeksforGeeks, accessed January 22, 2026, https://www.geeksforgeeks.org/springboot/how-to-call-rest-services-with-webclient-in-spring-boot/
44. Enhancing Public Access to Legal Knowledge in India: A Legal Chatbot Using Legal BERT, GPT-2, and Retrieval-Augmented Generation (RAG) - IEEE Xplore, accessed January 22, 2026, https://ieeexplore.ieee.org/document/11118538/