iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** XI **Month of publication:** November 2026

DOI: https://doi.org/10.22214/ijraset.2026.85469

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

NyayaAI: An Explainable Judicial Delay-Risk Prediction and Similar-Case Retrieval System for Court Administration

Abhishek Devkatte[1], Ranjit Dhapate[2], Sanchit Pati[3], Rohit Patil[4], Omkar Shetake[5], Mr. V. V. Mangave[6]
Department of Computer Science and Engineering, D. Y. Patil College of Engineering and Technology, Kolhapur, Maharashtra, India

Abstract: ***Court backlogs in India now run into tens of millions of pending cases, but court staff have very few tools that show which of those cases are likely to drag on, why, and what to look at first. In this paper we present NyayaAI, a web-based decision-support prototype that brings gradient-boosted machine learning, explainable AI and text retrieval together in one workflow. Given a case record, it classifies the delay risk as LOW, MEDIUM or HIGH using an XGBoost classifier, estimates the delay in days with an XGBoost regressor, explains each prediction through SHAP feature contributions, finds similar cases using TF-IDF vectors and cosine similarity, and suggests administrative actions through a set of rules; the Google Gemini API can optionally restate these findings in plain language.***

The system is built with React, Node.js/Express, FastAPI and MongoDB, and it pulls real judgment records from the Indian Kanoon API. On 600 held-out cases from a synthetic corpus of 3,000 historical cases, the classifier reached 71.7% accuracy with a macro F1-score of 0.724, and the regressor had a mean absolute error of 97.69 days. These numbers should be read with care, since the training labels are synthetic and the procedural fields of the 89 Indian Kanoon records we ingested are derived, not observed. They show that the pipeline works; they do not measure accuracy on real court outcomes. Testing on independently sourced docket data is the most important next step.

Index Terms: Judicial delay prediction, court case pendency, explainable AI, SHAP, XGBoost, TF-IDF, similar case retrieval, decision-support system, Indian Kanoon

I. INTRODUCTION

A. Background

India has one of the largest case backlogs in the world. Data from the National Judicial Data Grid (NJDG) placed before Parliament show that close to 4.98 crore (about 49.8 million) cases were pending in district and subordinate courts on 28 July 2026 [10], [11]. Court portals and legal-information services have made records and judgments much easier to reach. Even so, having the records does not tell an administrator which pending cases are heading for long delay, what is causing it, or how a case compares with earlier ones.

Machine learning can pick up patterns in structured case data, language-processing methods can find cases with similar descriptions, and explainable-AI techniques can show which inputs shaped a prediction. NyayaAI puts these three things into a single web application meant for court administrators, registry staff and judicial researchers.

B. Problem Context

How long a case takes depends on its type, its stage, how often it is adjourned, the gap between hearings, the workload of the court and a good deal more, so no single indicator can be trusted on its own. When this information sits in different places, reviewers have to open several sources before they can pick out the cases that need attention. A risk score with no explanation is hard to trust. A similarity search that ignores risk does not tell anyone which cases need action. And a dashboard that hides how a score was reached invites people to treat that score as a verdict.

For that reason NyayaAI shows the risk class, the estimated delay, the reasons behind the prediction, comparable earlier cases and suggested follow-up actions side by side. It supports administration. It predicts the risk of procedural delay, and it says nothing about the merits or outcome of a case.

©IJRASET: All Rights are Reserved

C. Motivation

Public judgment data can now be fetched programmatically. The Indian Kanoon API offers search over court judgments and statutes [9], which let us demonstrate a risk-assessment workflow on real legal records and not only on made-up text. At the same time, open datasets that record the true delay of individual cases are hard to find, so a prototype has to be clear about what it learns from which data. We wanted to build a modular, explainable and honestly documented system that can be retrained later on validated docket data without redesigning the application.

D. Proposed Approach

NyayaAI is made of four cooperating services: a React web interface, a Node.js/Express API gateway, a MongoDB database and a Python FastAPI machine-learning service. The ML service extracts 15 case features, predicts the risk class and the delay in days with XGBoost, explains the classification with SHAP, and retrieves similar cases with TF-IDF and cosine similarity. The gateway adds authentication, role-based access control, rule-based recommendations and optional Gemini narratives, and it stores every prediction for audit.

We use two kinds of data for two different jobs. A synthetic historical corpus of 3,000 cases trains the models and also serves as the reference collection for retrieval, while real judgments fetched from Indian Kanoon fill the dashboard and show the workflow running on genuine legal records. The prediction target is therefore a modelled delay label, not an observed court outcome.

E. Contributions

Our main contributions are the following: (1) one workflow that covers delay-risk classification, delay estimation, explanation, similar-case retrieval and recommendations; (2) a shared feature-extraction module, so training and serving always use identical preprocessing; (3) a SHAP layer that turns multi-class tree explanations into a single risk-increasing or risk-decreasing view for each original feature; (4) a replaceable retrieval interface, currently built on TF-IDF and cosine similarity; (5) a rule engine that converts predictions and case context into prioritised administrative actions; (6) automatic ingestion of real judgments from the Indian Kanoon API; and (7) a secured, role-based dashboard that keeps working when the ML service or the Gemini API is unavailable.

II. RELATED WORK

A. Delay Prediction in Indian Courts

Bhatnagar and Huchhanavar built a classifier that predicts delay in Indian lower courts from information known when a case is filed. Their data cover about 4.2 million cases filed in 2010 and followed for ten years across more than 7,000 courts, and their best model reached 81.4% accuracy using AutoML and decision-forest classifiers [1]. A later study added hybrid machine-learning modelling and explainability to predict both case delays and durations [2]. These studies show that delay can be predicted from case attributes. They are modelling studies, though, and they do not offer a working interface that brings together prediction, explanation, comparable cases and recommended actions.

B. Legal Judgment Prediction and Legal NLP

A different line of work predicts how a case will be decided. Nigam et al. introduced PredEx, an expert-annotated collection of more than 15,000 annotations for judgment prediction and explanation in Indian courts, and showed that instruction-tuned language models improve both prediction and explanation [3]. NyayaAI asks a different question. It does not try to say who will win; it estimates how likely a case is to be delayed so that administrators know where to look.

C. Explainable Machine Learning

Gradient-boosted trees such as XGBoost work very well on structured data [4]. SHAP gives every input feature an additive contribution to a prediction, based on Shapley values [5], and TreeExplainer computes these values efficiently for tree ensembles [6]. In NyayaAI a reviewer can therefore see, for instance, that a high adjournment ratio pushed a case towards higher risk. It is worth remembering that SHAP values describe how the model behaves; they do not prove what causes delay in the real world.

D. Text Retrieval for Similar Cases

TF-IDF term weighting [7] combined with cosine similarity is still a transparent baseline for ranking documents by how much vocabulary they share.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

It is fast, it needs no GPU, and its matches are easy to interpret, for example a shared case type, stage or set of terms. Its weakness is that it cannot capture meaning beyond shared words, which is why NyayaAI keeps the retriever behind an abstract interface where dense legal embeddings can be added later.

E. Open Legal Data

The Indian Kanoon API gives access to search, documents and document fragments from its legal database [9], and the NJDG publishes pendency statistics for courts across the country [10]. Public judgments, however, are decided matters. They contain titles, courts, parties, dates and cited provisions, but no live docket history. Anyone who uses such records for delay modelling has to say how the missing procedural fields were obtained, and we do that in Section III-C.

F. Research Gap

Earlier work has shown delay prediction [1], [2], explainability [5], [6] and legal judgment prediction [3], each on its own. As far as we could tell from the literature we reviewed, none of it describes a prototype for Indian courts that combines delay-risk classification, delay estimation, SHAP explanations, similar-case retrieval and rule-based recommendations in one secured web application with live ingestion of legal records. NyayaAI is such a prototype. Its present limit is the data: the models are trained on a synthetic corpus, so the system shows how the workflow operates but does not prove real-world accuracy.

III. PROPOSED METHODOLOGY

A. System Architecture

NyayaAI follows a modular client–server design (Fig. 1). The React single-page application never talks to MongoDB or to the Python models directly. Every request goes through the Express gateway, which authenticates the user, validates the input with Zod schemas, reads and writes MongoDB through Mongoose, and calls the FastAPI ML service [12] over HTTP/JSON. Two external services are involved: the Indian Kanoon API for legal records and the Google Gemini API for optional plain-language narratives [13]. One prediction request travels through the following steps: the user selects a case, the Express API authenticates and validates the request, FastAPI extracts the features, the XGBoost classifier and regressor produce their outputs, SHAP computes the explanation, TF-IDF retrieves similar cases, the rule engine generates recommendations, the results are stored in MongoDB, and the dashboard displays them. If the ML service cannot be reached, the gateway returns a structured 503 error. If Gemini is unavailable, a deterministic template narrative is used, so an optional component never causes a prediction to be lost.

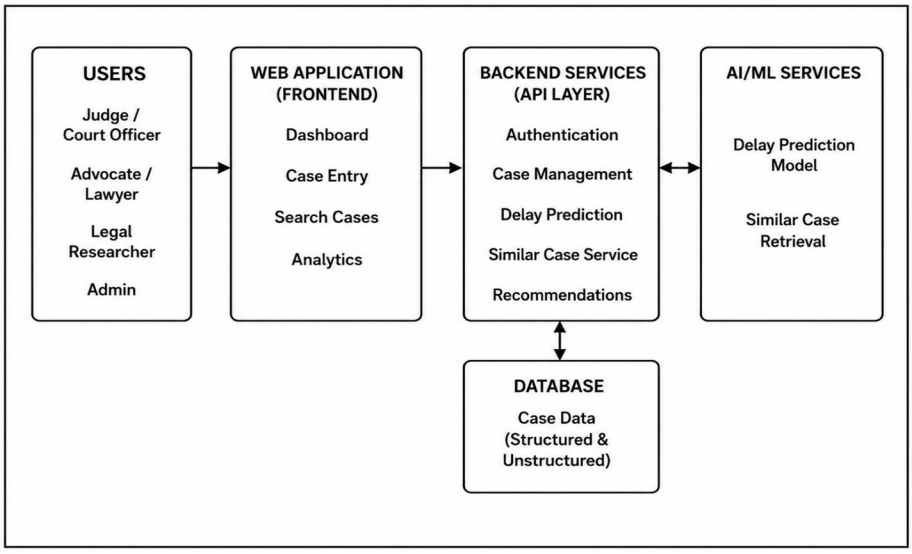


Fig. 1. Overall architecture of NyayaAI.

Fig. 2 gives the context view of the data exchange. Users submit case details and receive risk predictions, similar cases and recommendations, while the system reads and writes MongoDB, queries the Indian Kanoon API and can optionally call the Gemini API.

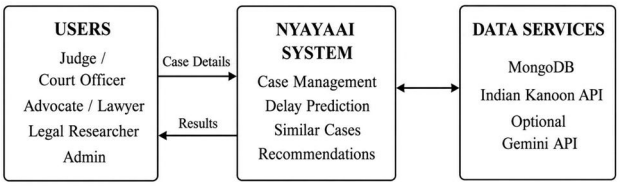


Fig. 2. Context diagram (data flow level 0) of NyayaAI.

B. Data Sources

The system holds three kinds of records (Table I). The synthetic historical corpus comes from the project's data-generation script and contains the structured fields the models need together with a modelled delay label. It is used in two ways: 2,400 cases train the models, 600 are held out for testing, and the whole corpus is the reference collection for similar-case retrieval. Real judgments from Indian Kanoon are stored separately and flagged as non-sample records. Cases entered by analysts through the web form make up the third kind.

TABLE I — DATA SOURCES AND THEIR ROLES

Source	Records	Role in NyayaAI
Synthetic historical corpus	3,000	Training (2,400) and testing (600) of the XGBoost models; reference corpus for similar-case retrieval
Indian Kanoon API judgments	89 stored	Real titles, courts, parties and cited provisions shown on the dashboard; demonstration of prediction and retrieval
Analyst-entered cases	variable	Created by authenticated users through the case form

C. Ingestion of Indian Kanoon Records

The synchronisation service sends the developer token, kept in the server environment, with a form-encoded POST request to the Indian Kanoon search endpoint. For every judgment it returns, the service strips HTML from the title, extracts the petitioner and respondent, maps the court to a court level, extracts the cited statutory provisions, assigns a dispute category (civil, criminal, property, cheque bounce, family, labour or commercial) and builds a unique case number of the form IK/year/document-id. Records are written with an upsert, so running the synchronisation again does not duplicate a case number. The service can be run from the command line or, if enabled, starts automatically a few seconds after the backend boots, with a configurable query and record limit.

A decided judgment has no live docket history. The procedural attributes the model needs, such as the number of hearings, adjournments, current stage and next hearing date, are therefore derived by the ingestion service and are not read from an official court record. Risk scores on these records illustrate the workflow and must not be taken as assessments of the actual cases. In line with the licence terms of the API, we acknowledge Indian Kanoon as the source of these records [9].

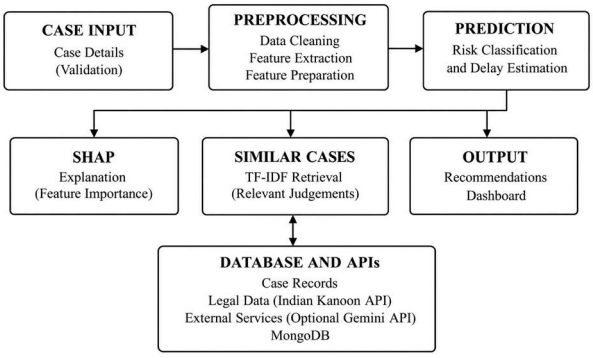


Fig. 3. End-to-end data flow from case input to dashboard output.

D. Feature Engineering

One shared Python module extracts the same 15 features during training and during inference, which removes any mismatch between the two. Nine features are numeric and six are categorical (Table II). The adjournment ratio is the number of adjournments divided by the number of hearings:

Equation (1): r_adj = n_adj / n_hear (r_adj = 0 when n_hear = 0)

where n_adj is the number of adjournments and n_hear the number of hearings. Numeric columns get median imputation and standardisation with scikit-learn [8]; categorical columns get constant imputation and one-hot encoding, with unseen categories ignored. The fitted pipeline is saved with Joblib and loaded by the service at start-up.

TABLE II — MODEL INPUT FEATURES

Type	Features
Numeric (9)	caseAgeDays, numberOfHearings, numberOfAdjournments, adjournmentRatio, hearingsPerYear, daysSinceLastHearing, daysToNextHearing, numLegalSections, descriptionLength
Categorical (6)	caseType, court, state, currentStage, status, priority

E. Delay-Risk Classification

The classifier is an XGBoost model with a multi-class soft-probability objective. It outputs the probability of LOW, MEDIUM and HIGH, and the most probable class is reported together with its probability as the model's confidence. Table III lists the configuration. The service can also train a Random Forest baseline for comparison.

TABLE III — MODEL CONFIGURATION

Component	Setting
Classifier	XGBoost, multi:softprob, 300 trees, maximum depth 5, learning rate 0.06; classes LOW, MEDIUM, HIGH
Regressor	XGBoost regression on ln(1 + delay days)
Explainer	SHAP TreeExplainer on the classifier
Retriever	TF-IDF (unigrams and bigrams, sublinear term frequency) with cosine similarity
Model version	xgb-v1, trained on 21 October 2026

F. Delay Estimation

Delay in days is positive and strongly right-skewed, so the regressor is trained on a logarithmic target and its output is converted back:

Equation (2): $y = \ln(1 + d)$, $\hat{d} = \exp(\hat{y}) - 1$

where d is the delay in days and $\hat{y}$ is the regressor output. This keeps the estimate positive and stops very long cases from dominating the training loss.

G. SHAP Explanation Layer

The classifier has three classes, so SHAP returns one contribution vector per class. To get one simple explanation, the service takes the difference between the HIGH and LOW contributions and adds up the one-hot columns that belong to the same original feature (for example, every court_* column becomes court):

Equation (3): $\Phi_f = \Sigma_{j \in f} (\varphi_j^{HIGH} - \varphi_j^{LOW})$

where φ_j^c is the SHAP value of encoded column j for class c, and f is the original feature. A positive Φ_f is labelled "increases risk" and a negative value "decreases risk". The interface turns the largest contributions into short sentences such as "adjournment ratio (45%) pushed the model toward higher risk".

H. Similar-Case Retrieval

Each case is turned into a text built from its case type, court, stage, legal sections and description, and the more informative fields are given more weight. Terms are weighted with TF-IDF using sublinear term frequency:

Equation (4): $w(t, d) = (1 + \ln f(t, d)) \cdot idf(t)$

where f(t, d) is the number of times term t appears in document d and idf(t) is its inverse document frequency [7]. Cases are then ranked by the cosine similarity between the query vector q and a candidate vector c:

Equation (5): $sim(q, c) = (q \cdot c) / (\|q\| \|c\|)$

The top-K matches come back with their similarity, their risk class if they have been assessed, and a short list of matching attributes. Retrieval sits behind an abstract class, so a SentenceTransformer or vector-database retriever can replace TF-IDF without touching the API routes.

I. Recommendation Engine

A provider-based rule engine in the gateway looks at the prediction together with the case record and creates prioritised, categorised suggestions (Table IV). The engine is advisory: each recommendation states its reason, and the interface shows a notice that recommendations are not legal advice or judicial orders. When it is switched on, the Gemini API only rewrites the existing findings as a neutral summary; it cannot change the predicted class or the rules.

TABLE IV — RULE-BASED RECOMMENDATION CHECKS

Check	Trigger	Suggested action
Priority mismatch	HIGH predicted risk but priority Low or Medium	Review the listing priority
Adjournment anomaly	3 or more adjournments, or ratio of 35% or more	Review the adjournment pattern
Hearing inactivity	Critical stage (evidence, pleadings, judgment reserved) and more than 90 days since the last hearing	Schedule the next hearing
Stay-order audit	Case status is Stayed	Check the review date of the stay
Monitoring and ageing	MEDIUM risk or long pendency	Monitor progress at regular intervals

J. Persistence, Security and Roles

MongoDB holds four collections (Table V). Every prediction is saved as a record that is never edited, with its probabilities, a snapshot of the input features, the SHAP contributions and the explanation, and the latest risk indicators are copied onto the case so that filtering is fast. The gateway hashes passwords with bcrypt (12 rounds), issues stateless JWT tokens and enforces three roles: admin (full access and role management), analyst (create cases and run predictions) and viewer (read-only). Helmet security headers, a CORS whitelist, request-size limits, MongoDB operator sanitisation and rate limits (500 requests per 15 minutes in general, 30 per 15 minutes for authentication) protect the API.

TABLE V — MONGODB COLLECTIONS

Collection	Main fields
users	name, email, hashed password, role, last login
cases	case number, title, court, state, type, dates, stage, status, priority, parties, hearings, adjournments, legal sections, sample flag, latest risk and delay
predictions	case, risk level, class probabilities, delay days, model version, feature contributions, explanation, input features, requesting user
recommendations	case, prediction, priority, category, recommendation text, reason, source

K. User Interface

The interface has a dashboard with key indicators and charts, a filterable and paginated case list with a selectable page size, a case-details page, an AI Prediction page with a risk gauge and SHAP chart, a similar-case page, a recommendations page, an analytics page and a settings page that shows system health and the metadata of the loaded model. If a case is edited after its last prediction, it is marked as outdated so the user knows to run the analysis again.

L. Prediction Workflow

Algorithm 1 summarises how one prediction request is handled.

Algorithm 1: Prediction workflow

Input: case identifier c from an authenticated request
Output: risk class, delay estimate, explanation, similar cases, recommendations
1. Load case c from MongoDB and check the user role.
2. Send the case attributes to the ML service (POST /predict).
3. Extract the 15 features and apply the stored preprocessing pipeline.
4. Predict the class probabilities and take the most probable class.
5. Predict ln(1 + d) with the regressor and convert it back to days (Eq. 2).
6. Compute SHAP contributions and aggregate them per feature (Eq. 3).
7. Build the narrative with Gemini, or with the template if Gemini is unavailable.
8. Save the prediction record and copy the latest risk indicators to the case.
9. Request the top-K similar cases (POST /similar) using Eqs. (4) and (5).
10. Apply the recommendation rules and store the suggestions.
11. Return all results to the interface.

IV. RESULTS AND DISCUSSION

A. Experimental Setup

We split the synthetic corpus of 3,000 cases into 2,400 training cases and 600 test cases. The models were trained with the configuration in Table III on Python 3.10 using the training script of the ML service, and the resulting metadata are stored with the model files and shown on the settings page (Fig. 4). Every figure we report refers to the 600 held-out test cases.

B. Classification and Delay-Estimation Performance

Table VI gives the results. The classifier sorts each case into one of three risk classes, so random guessing would give about 33% accuracy; the model reaches 71.7%, with macro precision of 0.729 and macro recall of 0.723, which suggests that the errors are not concentrated in a single class. The mean absolute error of the regressor is 97.69 days, so on average its estimate is roughly three months away from the modelled delay.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

TABLE VI — PERFORMANCE ON THE HELD-OUT TEST SET (600 CASES)

Metric	Value
Classifier accuracy	71.7%
Macro precision	0.729
Macro recall	0.723
Macro F1-score	0.724
Delay regression MAE	97.69 days

These values show how well the pipeline reproduces the delay labels of the synthetic corpus. They do not tell us how the system would do on real court dockets, and accuracy on its own says nothing about class-wise behaviour or calibration. The next evaluation steps are a class-wise confusion matrix, per-class precision and recall, and a comparison of the regressor with a simple baseline such as the median delay.

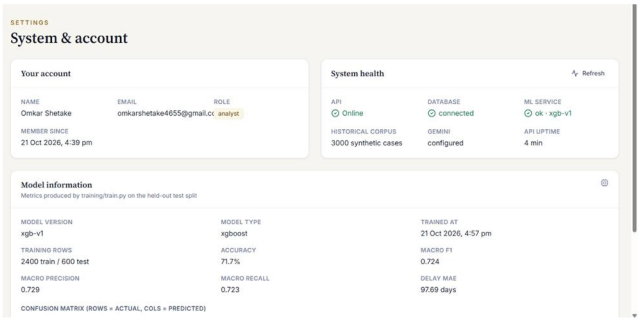


Fig. 4. Settings page showing system health and model metadata (version xgb-v1).

C. Similar-Case Retrieval

Fig. 5 shows the similar-case page for a civil case. The top five matches have cosine similarities between 76% and 91%, and each match lists why it was chosen, such as the same case type, the same stage, the same court level and shared terms. Matches that have been assessed also show their own risk class, so an administrator can compare how similar cases were rated. The retrieved set did contain repeated judgment titles that came in under different identifiers, so duplicate detection during ingestion is listed as future work.

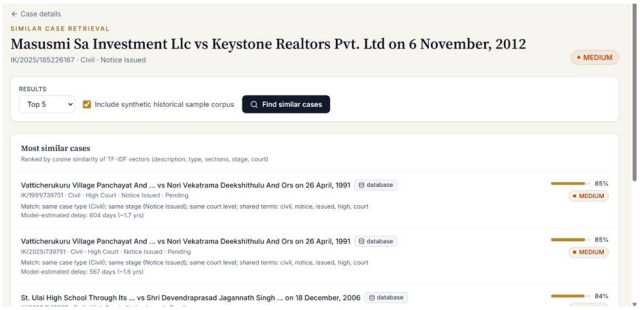


Fig. 5. Similar-case retrieval ranked by TF-IDF cosine similarity.

D. Dashboard on Indian Kanoon Records

With 89 Indian Kanoon records ingested and assessed, the dashboard (whose overview page can be filtered by filing date, case type, risk level and court) summarises the portfolio (Table VII and Figs. 6–8). 25 cases are classified HIGH, 38 MEDIUM and 26 LOW, the average case age is 708 days and the average estimated delay is 512 days. Civil matters are the largest group, followed by criminal, property, cheque-bounce, family, labour and commercial cases. The charts show cases by risk and by type and a monthly trend of filed cases against HIGH-risk cases, and a panel lists upcoming hearings for HIGH-risk cases. Since the procedural fields of these records are derived, these figures show that the filters, charts and workflow work on real legal records; they say nothing about the state of those courts.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

TABLE VII — DEMONSTRATION SNAPSHOT OF STORED INDIAN KANOON RECORDS

Indicator	Value
Total assessed cases	89
HIGH / MEDIUM / LOW risk	25 / 38 / 26
Pending cases	68
Average case age	708 days (about 1.9 years)
Average hearings / adjournments	8.7 / 2.6
Average estimated delay	512 days

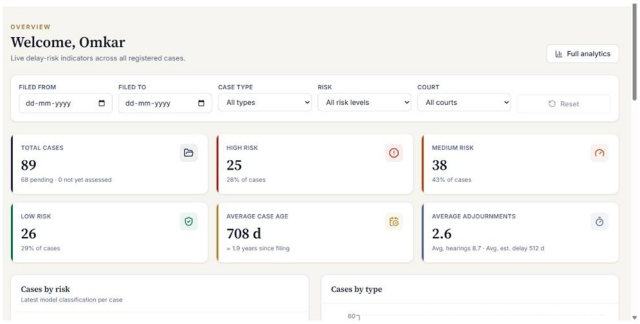


Fig. 6. Dashboard overview with filters and key delay-risk indicators.

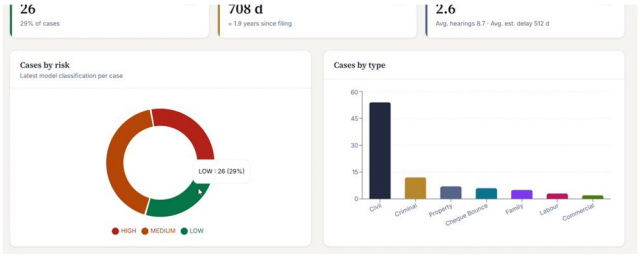


Fig. 7. Dashboard charts: cases by risk class and by case type.

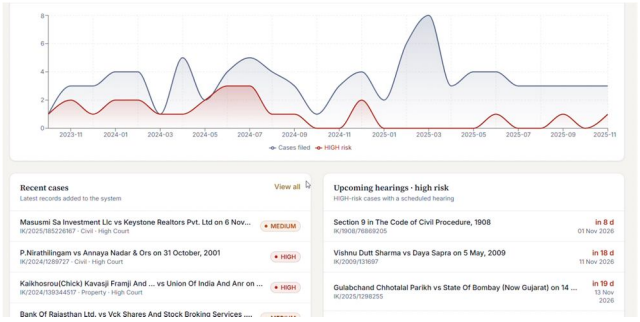


Fig. 8. Monthly trend of filed and HIGH-risk cases, recent cases and upcoming high-risk hearings.

E. Recommendations

Fig. 9 shows the recommendations panel for a MEDIUM-risk case with 47% model confidence that has been pending for 916 days. The rule engine suggests monitoring progress at regular intervals and tracking case ageing, and each suggestion carries a priority, a category and the reason that triggered it. The panel also states that the suggestions come from a rule engine and are not legal advice or judicial orders.

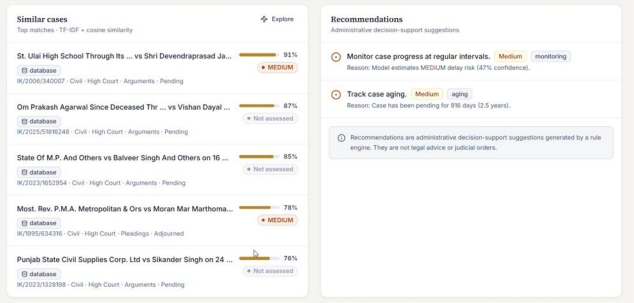


Fig. 9. Similar cases and recommendations for a MEDIUM-risk case.

F. System Functionality and Testing

A signed-in user can create or search cases, filter them by court, state, stage and risk, run a prediction, read the SHAP explanation, open similar cases, review recommendations and explore analytics, and an administrator can also manage roles. The repository includes an end-to-end flow script (npm run test:flow) that exercises authentication, the API, MongoDB, the ML service, SHAP, similarity search and recommendations in sequence.

G. Deployment and Reproducibility

The system can be started with Docker Compose or as three local services: the ML service on port 8000 (started with uvicorn), the API gateway on port 5000 and the React interface on port 5173 (Vite development server). Seed scripts create the first administrator and some demonstration cases, and the Indian Kanoon synchronisation can be run by hand (npm run sync:kanoon) or automatically when the backend starts. Secrets such as the JWT key and the Indian Kanoon and Gemini tokens are read from environment files and never sent to the browser. Table VIII lists the main interfaces between the components.

TABLE VIII — MAIN SERVICE INTERFACES

Service	Endpoint	Purpose
API gateway	POST /api/predictions	Run a prediction for a case (body: case identifier)
API gateway	POST /api/cases/sync-kanoon	Start an on-demand Indian Kanoon synchronisation
API gateway	GET /api/cases/sync-kanoon/status	Report the synchronisation status
API gateway	GET /api/health	Health of API, database and ML service
ML service	POST /predict	Risk class, delay estimate and SHAP contributions
ML service	POST /similar	Top-K similar cases
ML service	GET /health	Model and retriever status

H. Ethical and Responsible-Use Considerations

NyayaAI is a decision-support tool for administration and research. It is not an automated judge. It estimates the risk of procedural delay, never the merits or the verdict of a case, and the interface carries a visible notice that its outputs are not legal advice. Models can inherit bias from their training data, so a risk label must never be used to deprioritise, penalise or judge litigants, and the recommendations stay advisory. Each prediction is stored with its inputs, model version and explanation so it can be audited. Because judgments contain the names of real parties, a real deployment would also need a privacy review, data-minimisation rules, a bias audit across courts and case types, and continuing human oversight.

©IJRASET: All Rights are Reserved

I. Discussion

The results show that the whole pipeline works from end to end. Features are extracted consistently, a risk class with probabilities and a delay estimate come out, the reasons for the prediction can be inspected, comparable cases are found and actionable suggestions appear in the same interface. Because the design is modular, each part can be tested and improved on its own, and the fallbacks keep the system usable when the ML service or the Gemini API fails. Four limitations matter when reading the numbers. First, the training labels are modelled, so accuracy measures agreement with the synthetic generator and not with real court outcomes. Second, the procedural attributes of the ingested Indian Kanoon records are derived, so their risk scores are for demonstration only. Third, SHAP contributions explain the model; they do not prove what causes delay. Fourth, TF-IDF retrieval depends on shared vocabulary and on how well the reference corpus covers the case. For these reasons we present NyayaAI as a decision-support prototype whose predictions need human review.

V. CONCLUSION

NyayaAI is an integrated decision-support prototype for judicial administration. It predicts the delay-risk class of a case, estimates the delay in days, explains the prediction with SHAP, retrieves similar cases with TF-IDF and cosine similarity, and proposes administrative actions. It runs as a secured React, Express, FastAPI and MongoDB application and ingests real judgments from the Indian Kanoon API. On the held-out synthetic test set the classifier achieved 71.7% accuracy and a macro F1-score of 0.724, and the regressor a mean absolute error of 97.69 days. These results show that the workflow is technically sound; they do not yet show predictive accuracy on real court data. Before any institutional use, the models must be trained and validated on independently sourced, properly licensed docket records.

VI. LIMITATIONS AND FUTURE WORK

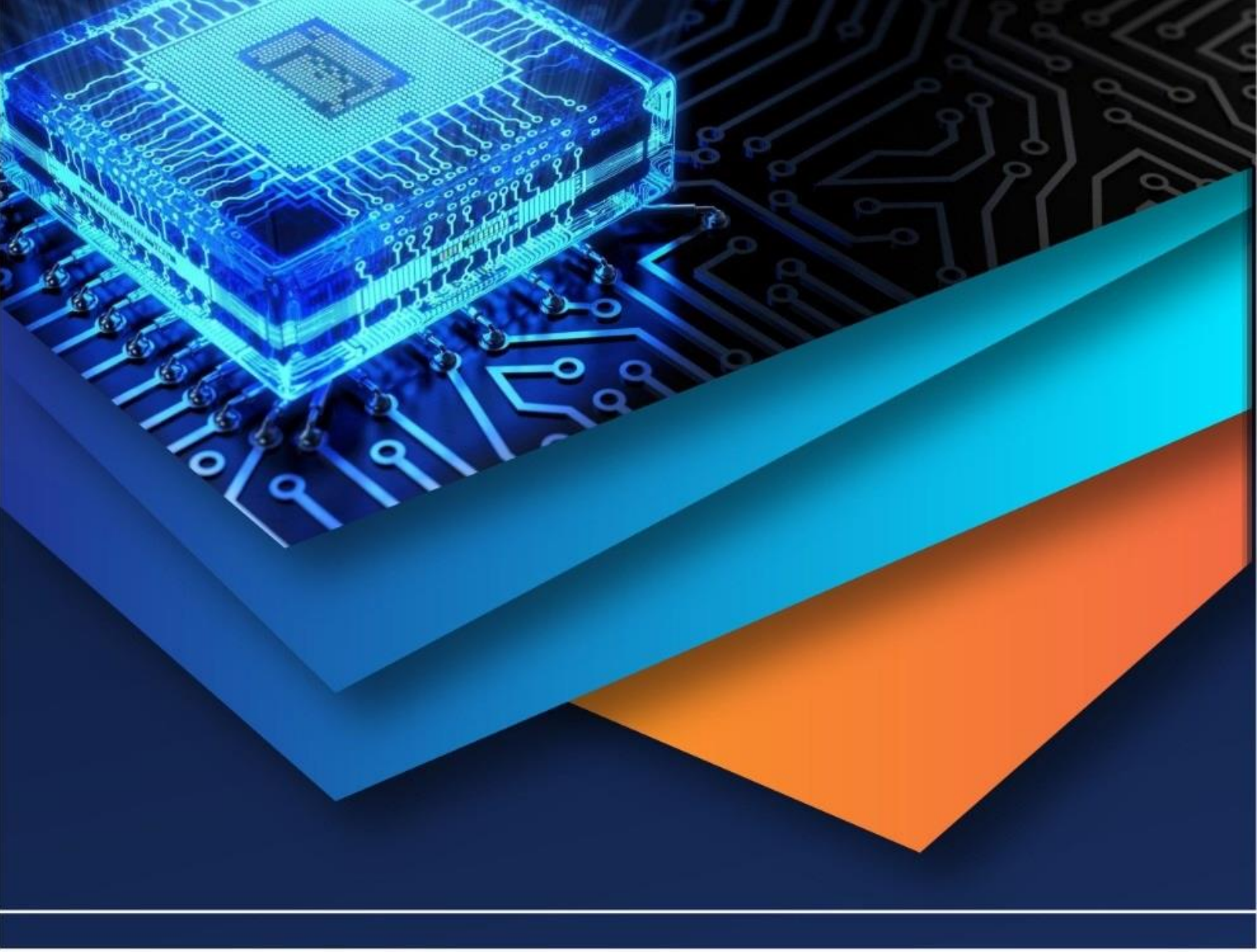
In future work we plan to replace the synthetic training data with real docket records that contain true filing, hearing, adjournment and disposal dates, for example through an authorised eCourts/NJDG integration, and to evaluate the models with a time-based split, class-wise metrics, calibration and prediction intervals for the delay estimate. Baselines such as a Random Forest and the median delay should be reported alongside XGBoost. The retriever can be upgraded to legal-domain sentence embeddings and reviewed by legal experts, ingestion should gain duplicate detection and provenance tracking, and a scheduled background job could refresh records and recompute risks. On the engineering side, JSON Web Tokens should move from browser local storage to short-lived httpOnly cookies, audit logging and monitoring should be added, and the models should be checked for bias across courts and case types before any real deployment.

VII. ACKNOWLEDGMENT

The authors thank the Department of Computer Science and Engineering, D. Y. Patil College of Engineering and Technology, Kolhapur, for the facilities and guidance provided during this work. Judgment records used in the demonstration come from the Indian Kanoon API (powered by IKanoon, indiankanoon.org).

REFERENCES

[1] M. Bhatnagar and S. S. Huchhanavar, “Predicting delays in Indian lower courts using AutoML and decision forests,” in Computing, Internet of Things and Data Analytics (ICCIDA 2023), Studies in Computational Intelligence, vol. 1145. Cham, Switzerland: Springer, 2024.
[2] M. Bhatnagar and S. S. Huchhanavar, “Hybrid machine learning modelling with explainability for predicting case delays and durations in Indian lower courts,” Journal of Big Data, vol. 13, no. 4, pp. 1–30, 2026.
[3] S. K. Nigam, A. Sharma, D. Khanna, N. Shallum, K. Ghosh, and A. Bhattacharya, “Legal judgment reimagined: PredEx and the rise of intelligent AI interpretation in Indian courts,” in Findings of the Association for Computational Linguistics: ACL 2024, Bangkok, Thailand, 2024, pp. 4296–4315, doi: 10.18653/v1/2024.findings-acl.255.
[4] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[5] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in Advances in Neural Information Processing Systems, vol. 30, 2017.
[6] S. M. Lundberg et al., “From local explanations to global understanding with explainable AI for trees,” Nature Machine Intelligence, vol. 2, no. 1, pp. 56–67, 2020.
[7] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” Information Processing & Management, vol. 24, no. 5, pp. 513–523, 1988.
[8] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
[9] IKanoon Software Development Pvt. Ltd., “Indian Kanoon API.” [Online]. Available: https://api.indiankanoon.org/ (accessed Oct. 2026).
[10] Supreme Court of India, eCourts Services, “National Judicial Data Grid (NJDG).” [Online]. Available: https://njdg.ecourts.gov.in/ (accessed Oct. 2026).
[11] “26 cases pending in Supreme Court for 30+ years, 558 cases for 20+ years: Centre in Rajya Sabha,” Bar & Bench, 2026. [Online]. Available: https://www.barandbench.com/news/26-cases-pending-in-supreme-court-for-30-years-558-cases-for-20-years-centre-in-rajya-sabha
[12] S. Ramírez, “FastAPI documentation.” [Online]. Available: https://fastapi.tiangolo.com/ (accessed Oct. 2026).
[13] Google, “Gemini API documentation.” [Online]. Available: https://ai.google.dev/ (accessed Oct. 2026).



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)