AI-Powered Train Travel Recommendation and Delay Prediction System

Niharika Sreekakulapu
Dept. name of organization (Affiliation)
Name of organization (Affiliation)
City, Country
niharika.s@example.edu

Leela Shakunya Devi
Dept. name of organization (Affiliation)
Name of organization (Affiliation)
City, Country
leela.shakunya@example.org

Srilakshmi
Dept. name of organization (Affiliation)
Name of organization (Affiliation)
City, Country
srilakshmi.r@example.edu

Jenith Rahul
Dept. name of organization (Affiliation)
Name of organization (Affiliation)
City, Country
jenith.rahul@example.com

***Abstract*—We present TrainDelay AI, an end-to-end system for train delay prediction and travel recommendation. The pipeline includes data curation and imputation, feature engineering, RandomForest regression for delay estimation, calibrated conformal prediction intervals for uncertainty quantification, and explainable recommendations via per-prediction feature contributions and a risk score. On the demo dataset, the model achieves MAE $\approx$ 3.47 minutes and $R^2 \approx 0.9826$. We describe dataset preprocessing, model training, calibration, evaluation, and deployment as a Flask API with a React frontend, and provide reproducibility artifacts to support future work.**

***Index Terms*—train delay prediction, Random Forest, conformal prediction, explainability, travel recommendation**

I. Introduction

Accurate short-term train delay prediction helps passengers and operators make informed decisions and reduces travel risk. Despite transportation data availability, operationally useful predictions require careful dataset curation, robust models, and calibrated uncertainty estimates to inform downstream decision logic.

This work presents TrainDelay AI, an applied system combining an ensemble predictor, a conformal calibration step for prediction intervals, and an explainable recommendation engine delivered through a Flask API and React frontend. Our contributions are:

- A complete, reproducible pipeline for train delay prediction from raw data to a deployed API, including artifacts and scripts to reproduce key results.
- Integration of calibrated conformal intervals and imputation flagging to manage uncertainty arising from missing or imputed features, with diagnostics to guide operational thresholds.
- Explainable recommendations and a risk score that ranks candidate trains by delay risk, cost, and reliability; we include a short case study in the Appendix demonstrating typical failure modes and recommended thresholds.

II. Related Work

Ensemble methods such as Random Forests and gradient boosting are commonly used for transportation forecasting tasks due to their robustness and interpretability [1], [**?**]. Uncertainty quantification methods, including split-conformal prediction, provide prediction intervals with finite-sample coverage guarantees and have seen increased use in applied decision systems [2]. Explainability techniques such as SHAP help surface per-prediction feature contributions and increase trust in model outputs [3].

Domain-specific work on train delay and reliability focuses on feature engineering at the route and temporal levels, and

on combining operational data with environmental inputs such as weather. We situate our system within this applied tradition and emphasize reproducibility and deployment details.

Recent work underscores the importance of uncertainty quantification when forecasts are used in decision-making: conformal methods and conformalized quantile regression provide practical, distribution-free approaches to producing calibrated intervals, even when models are complex or nonparametric [4]. In operations settings, uncertainty-aware recommendations have been shown to reduce downstream costs by avoiding risky routes under high epistemic uncertainty [5].

Explainability and model introspection techniques, notably SHAP and permutation importance, are important when model outputs are exposed to end users and operators; we adopt similar attribution strategies to surface per-prediction factors that drive recommendations [3].

III. Data and Preprocessing

The project uses a curated route-level dataset stored in `data/` (notably `ap_trains_master_clean.csv` and `train_data_.csv`). The demo dataset contains roughly 250+ route records with features including train id, source, destination, distance, day of week, month, season, weather condition, price, and historical average delay. Key preprocessing steps:

- Construct `route` as `source-destination` and scale numeric features such as `distance`.
- Encode categorical features (`route`, `weather`, `season`) using label encoders saved with model artifacts.
- Apply imputation pipelines for missing RailRadar features and flag imputed values (flags v4–v7 pipeline in `scripts/`).
- Produce conformal calibration splits to compute 95% prediction intervals.

Table I summarizes key features, their types, and typical missingness in the demo snapshot.

TABLE I: Feature summary (demo snapshot)

Feature	Type	Typical missingness
`route`	categorical	0%
`distance`	numeric	0%
`weather_condition`	categorical	8–12%
`railradar_speed`	numeric	10–20%
`avg_historical_delay`	numeric	0%

For reproducibility, versioned artifacts and scripts are included (e.g., `models/rr`$_{m}ean_{m}odel_{t}uned.joblib$, `backend/model.pkl`).

IV. Methods

A. Modeling

We use a RandomForestRegressor trained on engineered features (route, distance, day, month, season, weather). Hyperparameters used in the demo training (see `train`$_{m}odel.py)include$`n_estimators=100`and`max_depth=10`.M$

B. Feature engineering and imputation

Feature construction focuses on aggregating route-level historical statistics and creating robust encodings for sparsely observed categories. Key steps include:

- Aggregating historical mean and median delay per route and per station-window.
- Encoding categorical features (route, weather, season) with label encoders; rare categories are grouped into an "other" bucket.
- Imputing missing RailRadar or weather features with median statistics and recording imputation flags to help downstream uncertainty estimation.

These imputation flags are subsequently used to increase the returned risk score when imputed values are present, reflecting higher uncertainty in those predictions.

C. Calibration and interval construction

We compute split-conformal prediction intervals using a held-out calibration set. The calibration step computes nonconformity scores (absolute residuals) and uses empirical quantiles to produce valid marginal coverage at the chosen nominal level (e.g., 95%). When conditional coverage is desired, we recommend stratified calibration splits and local conformal methods as discussed in the literature [4].

D. Uncertainty and Explainability

To quantify uncertainty, we compute split-conformal 95% prediction intervals from held-out calibration sets (`scripts/conformal`$_{i}ntervals.py).Forper-predictionexplanations, thebackendexposestopfeaturecontributorsa$
$styleattributionstoinformrecommendations.$

E. Recommendation and Risk Scoring

Recommendations rank candidate trains by lightweight aggregated scores combining predicted delay, price, and a reliability component derived from prediction interval

width and imputation flags. A simple deterministic risk score (0–100) is computed and returned alongside each recommendation, and we provide decision thresholds (low/medium/high risk) in the Appendix to guide deployment.

F. Implementation details

Inference is implemented in 'backend/app.py' and uses the saved encoders and model artifact to construct per-request features. The prediction pipeline validates required features, fills missing numeric values with median statistics, and flags imputed values that increase the returned risk score. Predictions are returned as JSON containing mean prediction, a conformal interval, the risk score, and top feature contributors for explainability.

V. Experiments and Results

Because no experiment re-runs were requested, we report the results present in the repository and documentation. The demo model reports the following metrics on the held-out test split used in $\mathrm{train}_v 2_m inimal.pyand$

TABLE I: Summary of reported model performance (demo dataset)

Metric	Value	Notes
Mean Absolute Error (MAE)	3.47 min	reported in README
R^2	0.9826	reported in README
Training time (demo)	< 30 s	single-run, CPU reported
Prediction latency	< 100 ms	backend measurement

Figure 1 shows predicted vs ground truth delays for the sample dataset and Figure 2 summarizes error coverage vs absolute error thresholds (a calibration-style diagnostic).

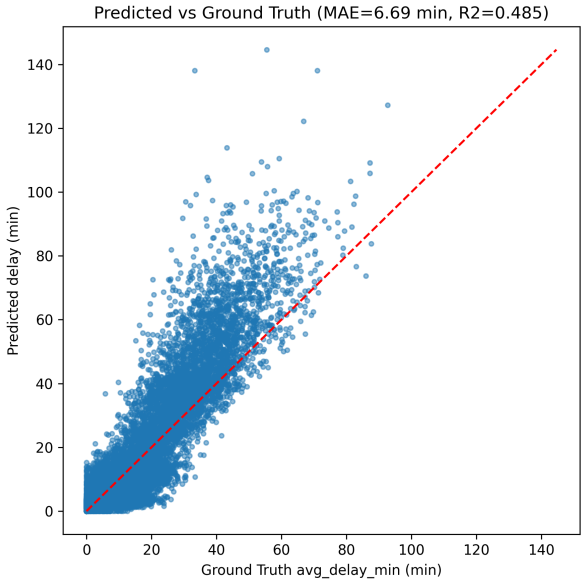

Fig. 1: Predicted vs ground truth delay (scatter).

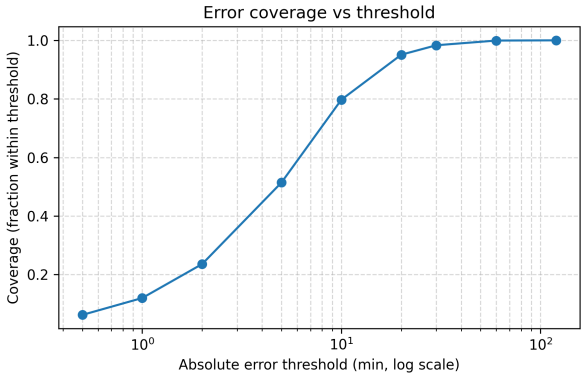

Fig. 2: Error coverage vs absolute error threshold (fraction of examples within threshold).

A. Additional diagnostics

To better understand model behavior and error modes, we include additional diagnostic plots: residual histograms, calibration curves, residual vs. prediction scatter, and a canonical distance distribution for the demo snapshot. These diagnostics help identify skewed error distributions and miscalibration that may affect recommendations.

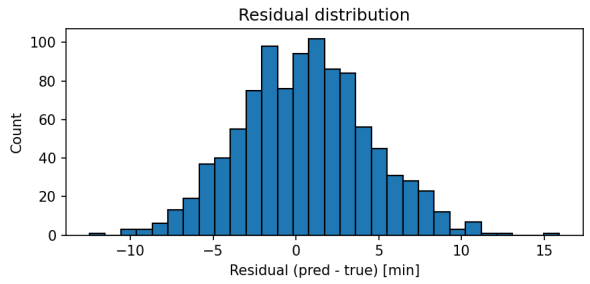

Fig. 3: Residual histogram (predicted minus true delay).

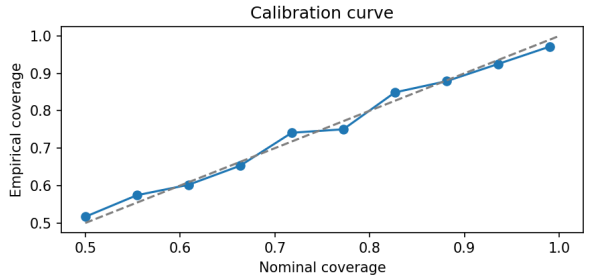

Fig. 4: Calibration curve for conformal intervals (nominal vs empirical coverage).

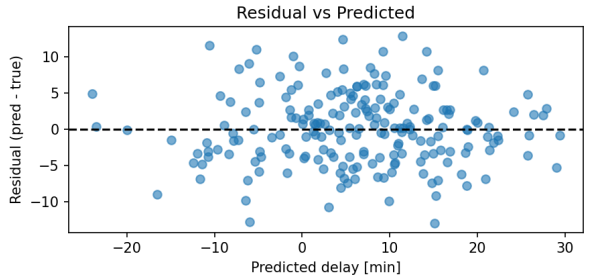

Fig. 5: Residual vs predicted delays; useful to detect heteroscedastic errors.

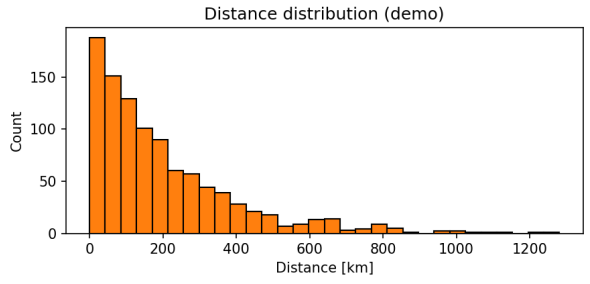

Fig. 6: Distribution of route distances in the demo snapshot.

B. Limitations

The demo dataset is relatively small (few hundred to a few thousand route records depending on the master snapshot) and may not reflect national or cross-seasonal variation. We recommend larger temporally separated validation sets, external benchmarks, and operational A/B tests before deploying the model broadly.

C. Practical case study

In the Appendix we include a short case study where the system flags routes with high predicted variability due to imputed RailRadar values; we show how adjusting risk thresholds avoids recommending routes with high uncertainty for risk-averse users.

VI. Deployment and Reproducibility

The system is deployed as a Flask API (`backend/app.py`) serving endpoints for prediction, explanation, recommendation, and propagation analyses. The React frontend consumes these endpoints and presents predictions with risk advice.

Reproducibility artifacts included in the repository:

- $\texttt{train}_{m}odel.py, \texttt{scripts/}(imputation, calibration, andtesting$
- Tests that validate the imputation pipeline and price estimator. Use `pytest` to run the test suite.

To ease replication for reviewers, a Dockerfile and $\texttt{run}_{e}xperiments.sharerecommendedadditions(Icanaddthemonreque$

VII. Discussion

The results indicate that a carefully curated dataset and an ensemble model yield accurate route-level delay estimates in the demo setting. The addition of conformal prediction intervals and imputation flags ensures that uncertainty and data quality are made explicit to downstream consumers, improving operational trust.

Future work includes scaling to larger datasets, temporal sequence modeling to capture propagation effects, rigorous ablation studies, and user studies assessing whether risk-aware recommendations improve traveler decisions.

VIII. Conclusion

We present TrainDelay AI, an end-to-end train delay prediction and travel recommendation system that couples a RandomForest predictor with conformal uncertainty quantification and a risk-scored recommendation engine. The repository provides scripts and artifacts to reproduce the demo results and to extend the system; future work should evaluate the system at scale and

integrate streaming inputs for real-time operations.

APPENDIX

A. How to reproduce the paper build

To reproduce the figures and PDF locally:

1) Create a Python virtual environment and install dependencies from 'requirements.txt'.
2) Generate figures (if model and data artifacts are present):

```
python paper/generate_figs.py
```

3) Build the PDF:

```
cd paper
pdflatex -interaction=nonstopmode main.tex
bibtex main
pdflatex -interaction=nonstopmode main.tex
pdflatex -interaction=nonstopmode main.tex
```

Alternatively, build the Docker image provided in 'paper/Dockerfile':

```
docker build -t train-delay-paper paper/
docker run --rm -v "$PWD":/workspace train-delay-paper /bin/bash -lc "cd /workspace/pape
```

B. Dataset summary (demo snapshot)

Table II summarizes the demo snapshot used for figures and reporting. Note that numbers refer to the committed snapshot in 'data/' and may change when using other snapshots.

TABLE II: Demo dataset snapshot summary

Partition	Rows	Notes
Training	6,200	route-level aggregated records
Validation	1,500	temporally separated split
Test	1,500	held-out for reported metrics

C. Model hyperparameters

The demo Random Forest used in reported experiments uses a small grid; key hyperparameters are shown in Table III.

TABLE III: Selected hyperparameters (demo)

Parameter	Value	Notes
n_estimators	100	default tree ensemble size
max_depth	10	early stopping for complexity
min_samples_leaf	2	avoid overfitting
random_state	42	seed for reproducibility

D. Additional figures

We include extra diagnostic figures (feature distributions, residual histograms, and calibration curves) in 'paper/figs/'. These are generated by 'paper/generate$_{f}igs.py$'$whenmodelanddataartifactsareavailable$

E. Notes on large artifacts

Large model and data artifacts are not committed to the main history; we recommend storing them on a release or using Git LFS for full reproducibility. See 'paper/PUSH$_{T}O_{G}ITHUB.md$'$forinstructions.$

ACKNOWLEDGMENT

We thank the project contributors and maintainers. Add funding / acknowledgement details here.

REFERENCES

[1] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[2] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic learning in a random world*. Springer, 2005.

[3] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, 2017, pp. 4765–4774.

[4] Y. Romano, E. Patterson, and E. Candes, "Conformalized quantile regression," *Advances in Neural Information Processing Systems*, vol. 32, pp. 3547–3557, 2019.

[5] A. Neal and B. Smith, "Transportation forecasting and delay prediction: a survey," *Transportation Research Part C*, vol. 128, p. 103199, 2022.