

Comparing Machine Learning and Soft-Voting Ensemble Models for Credit Card Fraud Detection in Imbalanced Transaction Data

Christian D. Barravecchio

Delbarton School, 230 Mendham Road, Morristown, NJ 07960, United States

ABSTRACT

Credit card fraud detection is a highly imbalanced classification problem in which headline accuracy can be misleading. In the held-out Kaggle test file used in this study, only 2,145 of 555,719 transactions (0.386%) were fraudulent; a classifier that labeled every transaction legitimate would therefore achieve approximately 99.61% accuracy while detecting no fraud. The analysis evaluated four configured machine-learning pipelines - Decision Tree, Random Forest, XGBoost, and a Multilayer Perceptron (MLP) - together with equal and validation-AP-weighted soft-voting ensembles. Average Precision (AP) was the primary ranking metric, along with precision, recall, F1 score, PR AUC, false positives (FP), and false negatives (FN) used to characterize operating tradeoffs. The selected XGBoost configuration produced the highest observed test AP (0.9084) and F1 score (0.8534) on this synthetic dataset. The soft-voting ensembles reduced some false positives but did not exceed XGBoost in AP. These findings apply to the specific configured pipelines, feature engineering choices, and recurring synthetic customer/merchant population studied here; they do not establish that XGBoost is generally the best fraud-detection method. The study also identifies limitations related to repeated validation use, unequal imbalance treatments, synthetic-data artifacts, probability calibration, and the absence of uncertainty estimates or entity-disjoint evaluation.

Keywords: credit card fraud; machine learning; XGBoost; ensemble learning; imbalanced classification; Average Precision; temporal validation

INTRODUCTION

The rapid growth of digital payments has increased both transaction convenience and the scale at which fraud can occur. The Federal Trade Commission reported more than 1.1 million identity-theft reports in 2024, with credit card fraud among the most frequently reported identity-

theft categories; consumers reported more than \$12.5 billion in fraud losses across categories (1). Modern payment networks therefore use automated risk scoring, rules, authentication, network intelligence, and human review together rather than relying on a single static rule or classifier (2-6).

Class imbalance is a central technical difficulty. Legitimate transactions greatly outnumber fraudulent ones, so accuracy alone can reward a model that simply predicts the majority class. In the final test file used here, fraud prevalence is 0.386%; an all-legitimate rule would be 99.61% accurate with zero fraud recall. For that reason, this study emphasizes AP and precision-recall behavior, while also reporting F1 and confusion-matrix

Corresponding author: Christian D. Barravecchio, E-mail: cbarravecchio@yahoo.com

Copyright: © 2026 Christian D. Barravecchio. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Accepted September 24, 2026

<https://doi.org/10.70251/HYJR2348.45829841>

errors at model-specific thresholds.

This study evaluates four configured base-model pipelines—Decision Tree, Random Forest, XGBoost, and MLP—and two soft-voting ensembles. It asks how these six configured fraud-detection pipelines compare on a highly imbalanced synthetic transaction dataset when evaluated with precision-recall-focused metrics and validation-selected decision thresholds. Boosting and ensemble approaches were expected to outperform a single Decision Tree. Because the pipelines use different imbalance treatments and hyperparameter constraints, the analysis compares specific configurations rather than isolating the effect of model family.

LITERATURE REVIEW

Research on credit card fraud has moved from manually specified rules toward machine-learning systems that learn nonlinear patterns from transaction data. Bhattacharyya et al. compared multiple data-mining methods and demonstrated that model choice and sampling strategy materially affect fraud detection (7). Dal Pozzolo et al. emphasized the practical consequences of severe class imbalance and nonstationarity in card fraud data (8).

Tree ensembles have become prominent because of their performance on structured data. Breiman introduced Random Forests as a bagging-based method that aggregates randomized trees (9), while Chen and Guestrin introduced XGBoost as a scalable gradient-boosting framework (10). More recent fraud-detection studies continue to investigate resampling, deep-learning ensembles, soft voting, and comparative model pipelines (11-14).

These studies additionally reinforce that fraud detection is not solved by one algorithm alone. Performance depends on representation, class-imbalance treatment, temporal design, thresholding, calibration, and the deployment population. The present study therefore reports the selected pipelines in detail and treats differences among them as empirical differences between configurations rather than proof that one model family is universally superior.

Machine learning is already used in commercial payment-fraud systems, but it is typically one layer within a broader decision architecture. Mastercard has described large-scale AI-based transaction scoring, Visa describes AI-supported fraud prevention and risk services, and Stripe Radar combines learned risk scores with rules and platform signals (2, 4-6). Survey

evidence also suggests substantial but incomplete industry adoption: industry reports have found approximately half of surveyed payments or financial-risk professionals using AI or machine learning for fraud-related work (2, 3).

METHODS AND MATERIALS

Dataset and Preprocessing

The exact dataset is the Credit Card Transactions Fraud Detection Dataset uploaded by Kartik Shenoy to Kaggle, version 1, containing `fraudTrain.csv` and `fraudTest.csv` (15). The Kaggle data card states that the data are synthetic and were generated with the Sparkov simulator created by Brandon Harris (16). Sparkov uses predefined customer/merchant profiles and generated transaction distributions; these design choices can create simulator-specific regularities that may be easier to learn than patterns in real payment networks (15, 16). The source training file contains 1,296,675 transactions from 2019-01-01 00:00:18 through 2020-06-21 12:13:37, including 7,506 fraud cases (0.5789%). The source test file contains 555,719 transactions from 2020-06-21 12:14:25 through 2020-12-31 23:59:34, including 2,145 fraud cases (0.3860%) (15). The final pipeline divided `fraudTrain` chronologically at the 80% row boundary: the model-training subset contained 1,037,340 transactions from 2019-01-01 00:00:18 through 2020-03-06 07:15:17, including 5,968 fraud cases (0.5753%); the validation subset contained 259,335 transactions from 2020-03-06 07:16:43 through 2020-06-21 12:13:37, including 1,538 fraud cases (0.5931%). The resulting data partitions are summarized in Table 1.

A feature is an input variable used by a model to estimate fraud probability. Each transaction can be represented as a feature vector containing measurements such as amount, merchant category, time of day, recent transaction count, elapsed time since the prior transaction, age, and geographic context. The label is fraud is the outcome to be predicted and is never used as an input feature.

Historical Feature Construction

For the finalized feature-engineering design, `fraudTrain` and `fraudTest` were concatenated only for label-free, chronological history construction while preserving a dataset-origin indicator. Rows were sorted by card number, transaction timestamp, and transaction number. This allows a later transaction to use information that would already exist at scoring time. Fraud labels

Table 1. Data partitions used in the study. The table reports the transaction counts, fraud totals, prevalence, and timestamp ranges from *fraudTrain.csv* and *fraudTest.csv*. The chronological model-training/validation cutoff occurs between 2020-03-06 07:15:17 and 2020-03-06 07:16:43.

Partition	Transactions	Fraud cases	Fraud prevalence	Date range
Source <i>fraudTrain.csv</i>	1,296,675	7,506	0.5789%	2019-01-01 00:00:18 to 2020-06-21 12:13:37
Model-training subset	1,037,340	5,968	0.5753%	2019-01-01 00:00:18 to 2020-03-06 07:15:17
Validation subset	259,335	1,538	0.5931%	2020-03-06 07:16:43 to 2020-06-21 12:13:37
Held-out <i>fraudTest.csv</i>	555,719	2,145	0.3860%	2020-06-21 12:14:25 to 2020-12-31 23:59:34

were not used to construct historical features. Rolling transaction counts used left-closed windows, so the current transaction was excluded from its own 1-day, 7-day, and 30-day history. Shifted and cumulative calculations similarly used only earlier rows for hours-since-previous-transaction and prior mean spending. The first observed transaction for a card had no prior history; count features were initialized to zero, missing history-derived numeric values were filled with zero, and a `has_previous_transaction` indicator distinguished genuine zero values from the absence of history. Although the simulated dataset contained no missing values, zero-filling was performed to ensure reproducibility. Rows were sorted by `cc_num`, transaction timestamp, and `trans_num`. For shifted or cumulative features, `trans_num` provided a deterministic tie-break when timestamps were identical. For the left-closed rolling windows, transactions occurring at exactly the same timestamp as the current record were excluded from that record's rolling count because the right endpoint of the time window was open. Histories were updated sequentially across time. A validation transaction could use earlier transactions already observed before it, and a test transaction could use earlier training history and earlier test transactions, but never later transactions or any fraud label. This design approximates an online scoring system in which behavioral history grows as transactions arrive. It differs from a strict batch-transform setting in which all test records are isolated from one another.

Features, Encoding, and Entity Leakage Risk

Direct identifiers or redundant raw fields - first name, last name, street address, `trans_num`, `cc_num`, `dob`, `trans_date_trans_time`, and `unix_time` - were removed from the final predictor matrix after relevant derived features were created. Date of birth was converted to age; timestamp

fields generated temporal and history variables; and `cc_num` was used only as a grouping key before removal. The retained conceptual feature set included transaction amount, location variables, city population, merchant coordinates, hour/day/month indicators, weekend/night indicators, recent transaction counts, previous transaction count, hours since the previous transaction, prior mean spending, amount deviations from prior spending, log amount, round-amount indicator, approximate customer-to-merchant distance, age, category, gender, state, merchant, job, and city. Category, gender, and state were one-hot encoded; merchant, job, and city were frequency encoded. Encoding rules and frequency maps were fit on the model-training subset only and applied unchanged to validation and test data. The final conceptual set contained 31 predictors. After preprocessing, one-hot encoding produced 14 category columns, 2 gender columns, and 51 state columns. Together with 25 numeric or engineered columns and 3 frequency-encoded columns, the final model matrix contained exactly 95 input columns in `X_train`, `X_validation`, and `X_test`.

Removing direct identifiers does not guarantee entity independence. Geographic, demographic, merchant-frequency, and historical behavior variables can collectively act as quasi-identifiers for recurring synthetic customers. In the model-training subset, there were 964 unique card numbers and 693 unique merchants. The held-out test file contained 924 unique card numbers, of which 908 (98.27%) had already appeared in the model-training subset; those seen cardholders accounted for 555,556 of 555,719 test transactions (99.9707%). Only 16 test card numbers were completely unseen during model fitting. All 693 test merchants had already appeared in the model-training subset. The reported evaluation therefore measures later transactions from an overwhelmingly recurring synthetic population

rather than cold-start performance for new customers or merchants. The temporal test file is not suitable for a conventional cardholder-disjoint or merchant-disjoint sensitivity analysis without constructing a new group-held-out split. The 16 unseen test cardholders generated only 163 test transactions (0.0293% of the test file), and all 163 of those transactions were labeled fraudulent; they therefore provide no legitimate unseen-card comparison group for measuring discrimination. They account for 7.60% of the test fraud cases. Merchant-disjoint evaluation is impossible on the temporal test file because every one of its 693 merchants was already present during model fitting. These patterns reflect the structure of the generated data and motivate custom cardholder-disjoint and merchant-disjoint resplitting in future work.

Missing Values, Duplicates, and Data Governance

Neither fraudTrain.csv nor fraudTest.csv contained missing values in the raw columns. The transaction identifier trans_num was unique for all 1,296,675 training rows and all 555,719 test rows, so no duplicate transaction identifiers were present. No separate row-deduplication step was applied because each transaction record was unique. Missing history-derived values created by feature engineering were handled explicitly as described above, and the final matrix-building code checked that no missing or infinite numeric values remained before fitting.

Configured Model Pipelines

The four base models were chosen to represent different practical tradeoffs. A Decision Tree offers a transparent learned-threshold baseline. Random Forest tests whether bagging many trees improves stability. XGBoost tests sequential boosting on tabular data. The MLP provides a neural-network comparison that can learn nonlinear interactions after standardization. The study compares four configured pipelines rather than claiming a controlled comparison of model family alone. The Decision Tree and Random Forest used balanced class weights, XGBoost used scale_pos_weight = 5, and the MLP did not use an equivalent explicit class-weight parameter. The models also differed in regularization and structural constraints. These choices reflect the final pipelines that were developed, but they prevent the performance differences from being attributed solely to the underlying algorithm. The Decision Tree (17) used entropy, maximum depth 9, minimum split size 8, minimum leaf size 6, balanced class weights, and random seed 42. The Random Forest (9) used 300 trees, maximum depth 12, minimum split size 5, minimum

leaf size 2, square-root feature sampling, balanced class weights, and seed 42. XGBoost (10) used maximum depth 9, learning rate 0.01, 400 estimators, scale_pos_weight 5, row and column subsampling of 0.8, min_child_weight 1, gamma 0, histogram tree method, and seed 42. The MLP (18) used hidden layers (256,128), tanh activation, learning rate 0.0001, alpha 0.001, maximum 300 iterations, early stopping, and standardized inputs fit only on the model-training subset.

Soft-Voting Ensembles

Equal soft voting averaged the four base-model fraud probabilities with equal weights. AP-weighted soft voting assigned weights proportional to each base model's validation AP. Because the component probabilities were not explicitly calibrated to a common probability scale, these ensembles should be interpreted as heuristic probability combinations rather than calibrated risk estimates.

Model Development and Hyperparameter Sensitivity

The final configurations emerged from iterative development rather than a single preregistered search with equal budgets across model families. A separate diagnostic sampled 15 hyperparameter configurations for each base model using Python random seed 42; each sampled model used seed 42 + trial number. Validation AP was the diagnostic score. The sampled search spaces are reported in Table 2, and every sampled configuration is listed in Appendix B. The 15-trial diagnostic was not the procedure that selected the final models and should therefore be interpreted as sensitivity to the chosen search space, not as a confidence interval, robustness estimate, or proof of generalization. The same validation subset was reused during iterative model development, ensemble weighting, and F1-threshold optimization. This can produce optimistic development-stage estimates because repeated decisions adapt to the same holdout set. No nested temporal cross-validation or rolling-origin validation was performed. The held-out fraudTest file remained separate for the reported final test results, but future work should use nested or rolling temporal validation before making stronger claims about model selection.

Evaluation and Threshold Selection

Five metrics were reported: precision, recall, F1, AP, and PR AUC. A true positive (TP) is fraud correctly flagged; a true negative (TN) is a legitimate transaction correctly allowed; a false positive (FP) is a legitimate

Table 2. Hyperparameter spaces used in the 15-trial sensitivity diagnostic. Each trial drew one value per listed parameter. Final model configurations were not selected by an equal-budget search across model families.

Model	Sampled hyperparameter space
Decision Tree	criterion={gini, entropy}; max_depth={5,7,9,11,13,15}; min_samples_split={2,5,8,12,16}; min_samples_leaf={1,3,6,9}
Random Forest	n_estimators={100,150,200,300}; max_depth={8,10,12,15,20}; min_samples_split={2,5,8,12}; min_samples_leaf={1,2,4,8}; max_features={sqrt,log2,None}; class_weight={None,balanced}
XGBoost	max_depth={4,6,8,9,10}; learning_rate={0.01,0.03,0.05,0.10}; n_estimators={200,300,400}; scale_pos_weight={1.5,3,5,8}; subsample={0.8,1.0}; colsample_bytree={0.8,1.0}
MLP	hidden_layer_sizes={({64}),(128),(128,64),(256,128),(128,64,32)}; activation={relu,tanh}; learning_rate_init={0.0001,0.0003,0.001}; alpha={0.0001,0.001,0.01}

transaction incorrectly flagged; and a false negative (FN) is a fraudulent transaction missed by the model. Precision = $TP/(TP+FP)$, recall = $TP/(TP+FN)$, and F1 is their harmonic mean. Each pipeline received a model-specific threshold chosen only on validation predictions. scikit-learn's `precision_recall_curve` enumerated the available thresholds. F1 was calculated at each threshold, and the threshold with the highest validation F1 was selected; if multiple thresholds tied, the first maximum returned by the implementation was retained. The selected threshold was then frozen and applied to final test probabilities, so test labels were not used to select thresholds. The final notebook reports the selected threshold and maximum validation F1 for all six methods, as well as validation AP for each of the four base models. These development-set values are reported in Table 3. Validation AP for the two ensemble probability combinations was not printed in the final notebook, so those two cells are shown as NR (not recorded) rather than estimated later. AP was computed with `average_precision_score`, which uses a step-wise weighted summary of precision as recall changes. PR AUC was computed separately as trapezoidal area using `auc(recall, precision)` (19). Because both summarize the same precision-recall curve, their values can be nearly identical when a model produces many distinct scores; they are not duplicate implementations but should not be treated as independent evidence.

SHAP Analysis

SHAP was used to interpret the final XGBoost model, which produced the highest observed AP and F1 among the configured pipelines (20). The reported analysis used `shap.TreeExplainer` on up to 2,500 validation observations, with no custom approximation settings. Mean absolute SHAP values were calculated for each

encoded input feature. One-hot encoded category, gender, and state columns were aggregated back to their conceptual source variables, while the merchant, job, and city frequency-encoded columns were relabeled to their source variables. To avoid treating SHAP magnitudes from different model families as directly comparable, the reported visualization is limited to XGBoost. Grouped mean absolute SHAP values were normalized within that model by dividing each feature's importance by the total grouped mean absolute SHAP importance. The figure reports the 12 highest-ranked conceptual features. Excluded raw columns are not shown because SHAP assigns contributions only to variables supplied to the fitted model; their absence should not be interpreted as evidence that they have zero predictive value.

Reproducibility and Computing Environment

The analysis was developed in Google Colab/Python using pandas, NumPy, scikit-learn, XGBoost, SHAP, Matplotlib, Seaborn, gdown, and Plotly. Random seed 42 was used for the main stochastic models, and diagnostic trials used seed 42 + trial number. The saved `Credit_Card_Fraud_Project.ipynb` contains the final model outputs, base-model validation AP values, validation-max F1 values, selected thresholds, data-partition statistics, and visualization outputs. Because the original runtime did not save a frozen requirements file, the exact package versions and hardware allocation from the final model run are not available. A later reproducibility check used Python 3.13.5, pandas 2.2.3, NumPy 2.3.5, scikit-learn 1.8.0, XGBoost 3.1.3, SHAP 0.50.0, Matplotlib 3.10.8, and Seaborn 0.13.2. These versions document the environment used for that check, not necessarily the versions used in the original final run.

RESULTS

Final Test Performance

Tables 3 and 4 summarize the validation-set and held-out test results. On the validation set, XGBoost had the highest base-model AP (0.9455) and the highest maximum validation F1 (0.8931). The MLP followed with validation AP = 0.8803 and maximum validation F1 = 0.8447; the Decision Tree had validation AP = 0.8401 and maximum validation F1 = 0.8019; and the Random Forest had validation AP = 0.8241 and maximum validation F1 = 0.7462. On the held-out test set, the selected XGBoost configuration had the highest observed AP (0.9084) and F1 (0.8534). The AP-weighted soft-voting ensemble ranked second by AP (0.8877) and had F1 = 0.8504, while equal soft voting achieved the highest precision (0.9285) but lower recall (0.7804). These are

descriptive results for the selected pipelines; possible explanations for the model ranking are discussed later.

All six methods exceeded 99.7% test accuracy, but this did not provide a meaningful model ranking because an all-legitimate classifier would already achieve approximately 99.61% accuracy. As a result, the interpretation focuses on AP, precision, recall, F1, FP, and FN rather than headline accuracy. Figure 1 compares the precision-recall curves of all six configured methods. XGBoost maintained the strongest overall precision-recall performance, while the two voting ensembles followed closely.

Classification Errors and Model Disagreement

Figure 2 translates the validation-selected thresholds into concrete TP, TN, FP, and FN outcomes. The AP-weighted ensemble produced 414 FN, not 424. Figure 3

Table 3. Validation-set performance used during model development. Validation AP was reported for the four base models and was also used to weight the AP-weighted ensemble. For all six methods, the final notebook contains the maximum validation F1 and the corresponding selected probability threshold. Ensemble validation AP was not separately reported, so those values are shown as NR (not recorded).

Method	Validation AP	Max validation F1	Selected threshold
XGBoost	0.9455	0.8931	0.5198
AP-Weighted Soft Voting (4)	NR	0.8820	0.5723
Equal Soft Voting (4)	NR	0.8810	0.6309
MLP Classifier	0.8803	0.8447	0.3046
Decision Tree	0.8401	0.8019	0.9867
Random Forest	0.8241	0.7462	0.8047

Table 4. Held-out test performance of the four configured base pipelines and two soft-voting ensembles. Thresholds were selected on validation data by maximizing F1 and then applied unchanged to the test set. FP = false positives; FN = false negatives. The AP-weighted ensemble FN count is 414, consistent with recall = 0.8070 among 2,145 test fraud cases.

Method	Type	AP	PR AUC	Precision	Recall	F1	Val.-selected threshold	FP	FN
XGBoost	Individual	0.9084	0.9084	0.8918	0.8182	0.8534	0.5198	213	390
AP-Weighted Soft Voting (4)	Ensemble	0.8877	0.8877	0.8988	0.8070	0.8504	0.5723	195	414
Equal Soft Voting (4)	Ensemble	0.8859	0.8858	0.9285	0.7804	0.8480	0.6309	129	471
MLP Classifier	Individual	0.8063	0.8063	0.8769	0.7072	0.7830	0.3046	213	628
Decision Tree	Individual	0.7969	0.8116	0.8710	0.7021	0.7775	0.9867	223	639
Random Forest	Individual	0.7652	0.7652	0.7474	0.6606	0.7013	0.8047	479	728

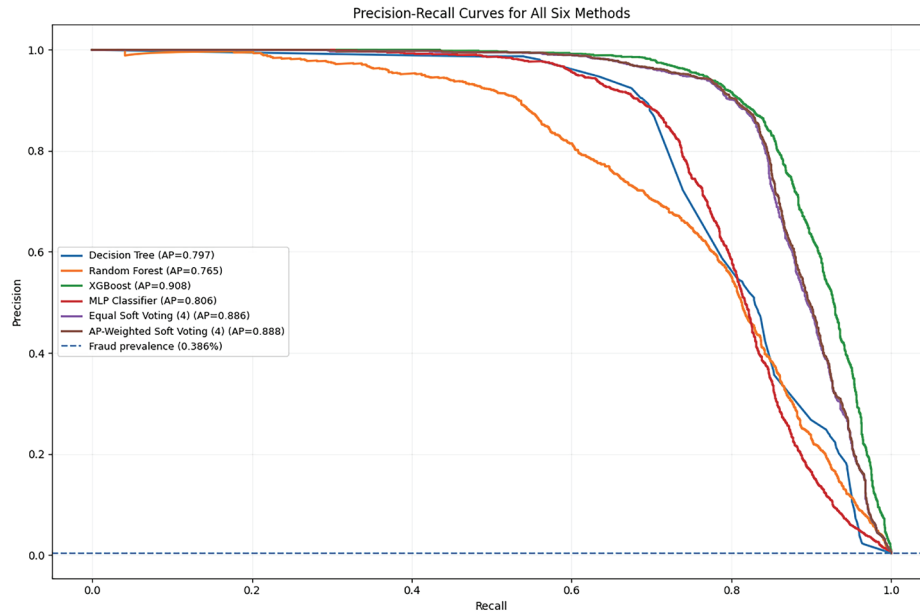


Figure 1. Precision-recall curves for all six configured methods on the held-out test set. Each point on a curve corresponds to a possible probability threshold; methods that remain closer to the upper-right region preserve higher precision while recovering more fraud. The dashed horizontal line is the fraud-prevalence baseline. This figure evaluates ranking behavior across thresholds and is separate from the single validation-selected threshold used for Table 4. Source: Author's analysis of data from Shenoy [15].

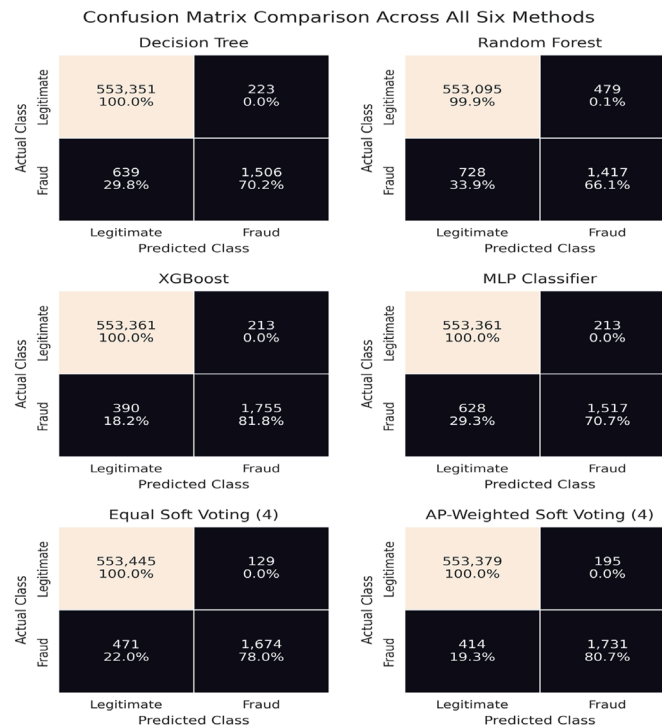


Figure 2. Confusion matrices for the four base models and two ensembles at their validation-selected thresholds. Rows are actual classes and columns are predicted classes. The upper-right cell in each panel is FP (legitimate transactions incorrectly flagged), and the lower-left cell is FN (fraud missed). Counts and within-row percentages make the tradeoff between customer friction and missed fraud explicit. Source: Author analysis of data from Shenoy [15].

shows that the four base models did not identify exactly the same fraud cases; overlap provides a rationale for testing ensembles but does not guarantee that probability averaging improves AP. The base models detected overlapping but nonidentical groups of fraud cases, demonstrating meaningful model disagreement.

Hyperparameter Sensitivity

Figure 4 shows validation AP across the 15 sampled diagnostic configurations. The dispersion reflects sensitivity to the sampled search space, not repeated-data statistical uncertainty, robustness, or out-of-sample generalization. The final model configurations were not

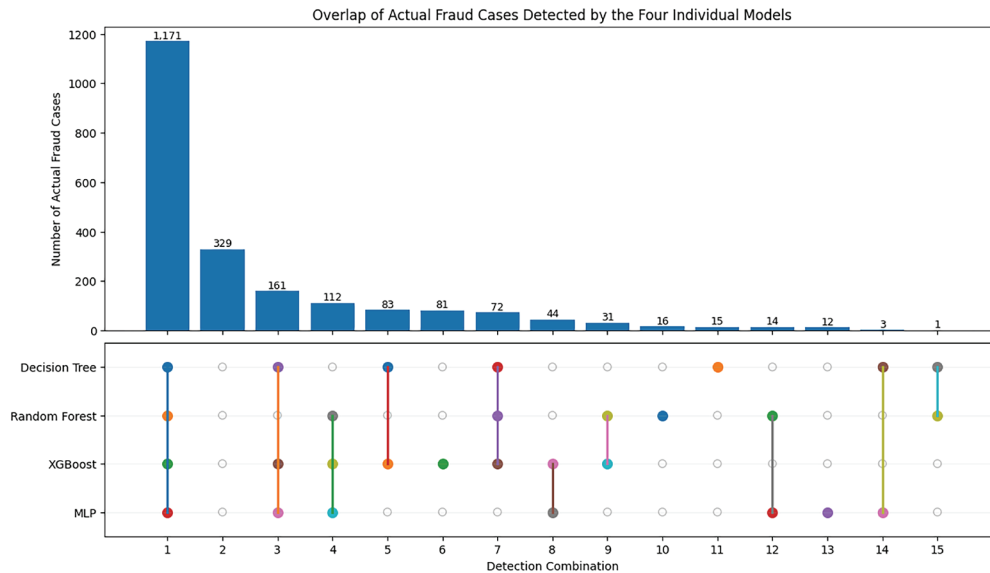


Figure 3. Overlap of actual fraud cases detected by the four individual base models. Bars show the number of fraud cases associated with each unique detection pattern, while the connected-dot matrix identifies which models participated in each pattern. The plot demonstrates model disagreement, but it does not by itself show that ensembling is superior; complementary detections can be offset by diluted probabilities from weaker or differently calibrated models. Source: Author's analysis of Shenoy [15].

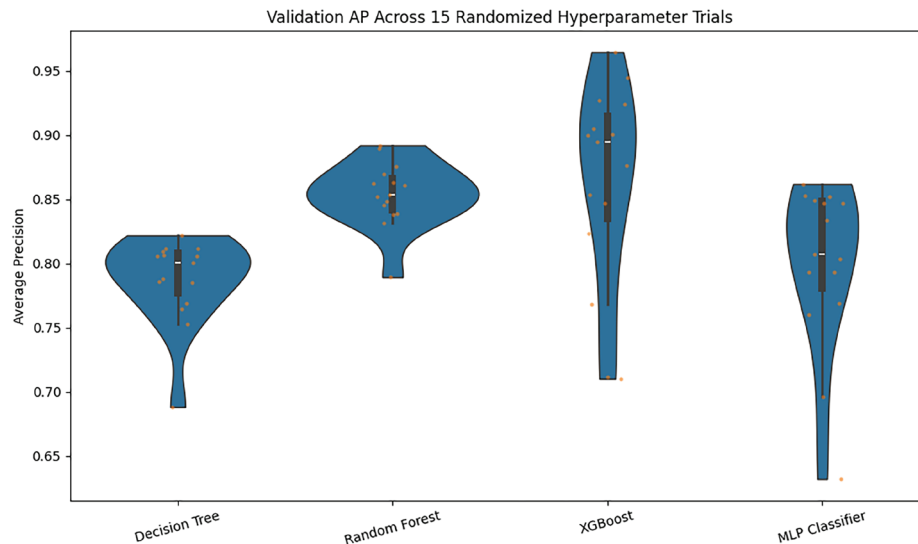


Figure 4. Validation AP observed across 15 randomized hyperparameter configurations per base model. Violin width summarizes the distribution of the 15 observed AP values within the chosen search space, while individual points correspond to specific sampled configurations. Because the trials reuse the same validation subset and vary model settings rather than data samples, the figure should be interpreted as hyperparameter sensitivity only. Source: Author's analysis of Shenoy [15].

selected through this equal-budget diagnostic. Appendix B lists every sampled configuration so the diagnostic can be reproduced.

Temporal Performance

Figure 5 reports monthly AP within the held-out test period. Performance varies across months and falls for several pipelines in December, illustrating that a single aggregate score can hide time-specific weakness. Because these monthly values come from one synthetic test period, they are descriptive evidence of temporal variation rather than a general estimate of production concept drift.

Feature Interpretation with SHAP

Figure 6 shows that transaction amount was the dominant feature in the final XGBoost model, accounting for 34.7% of normalized mean absolute SHAP importance. Category (18.0%) and transaction hour (15.2%) were the next largest contributors, followed by log-transformed amount (7.7%). Historical behavior features - including hours since the previous transaction, one-day transaction count, deviation from previous mean

spending, and previous mean amount - also appeared among the highest-ranked features. This pattern suggests that the fitted XGBoost model relied heavily on transaction size, purchase context, timing, and recent card behavior. The percentages describe relative importance within this fitted XGBoost model only; they do not show the direction of each effect, establish causality, or support direct comparisons with SHAP magnitudes from other model families.

DISCUSSION

The selected XGBoost pipeline produced the highest observed AP and F1 on this one synthetic held-out test file. That result supports the hypothesis that boosting could outperform a single tree in this setting, but it does not establish that XGBoost is generally the strongest fraud detector. The ranking may reflect the particular feature representation, imbalance strategy, constraints, threshold selection, and simulator distribution used in this experiment.

The hypothesis was only partly supported. Random Forest, despite being an ensemble method, achieved

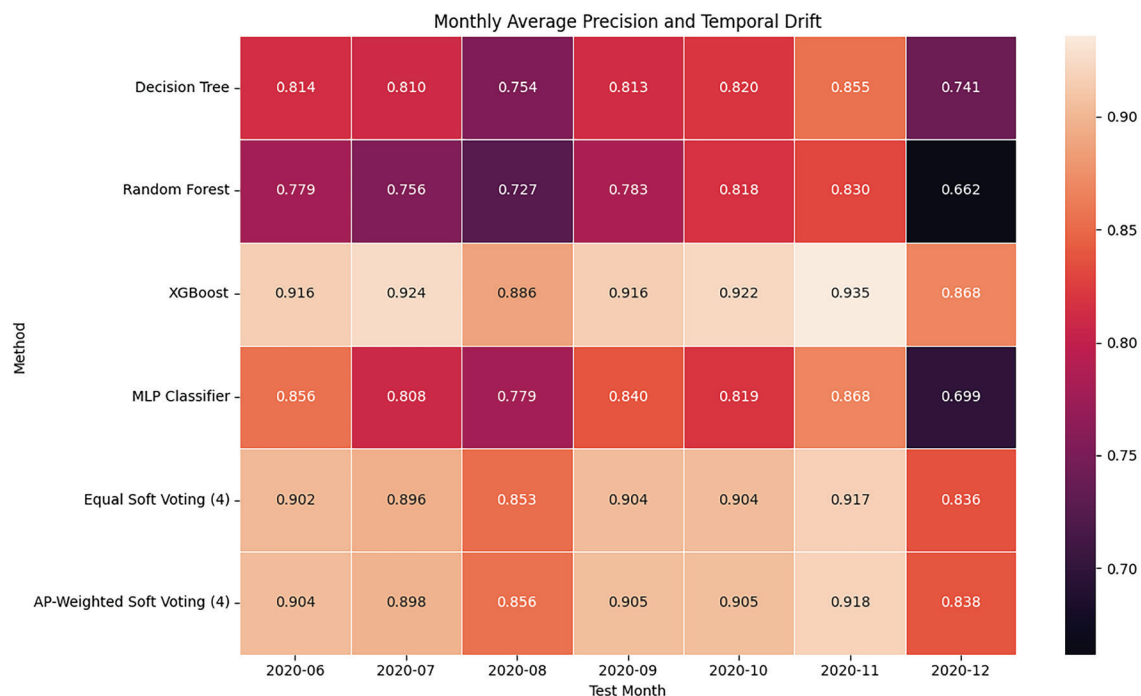


Figure 5. Monthly Average Precision for each configured method during the held-out test period. Rows represent models or ensembles and columns represent calendar months. Each cell is AP calculated using transactions from that month only. Month-to-month changes show that ranking quality was not temporally constant, motivating rolling monitoring in future work. Source: Author's analysis of Shenoy [15].

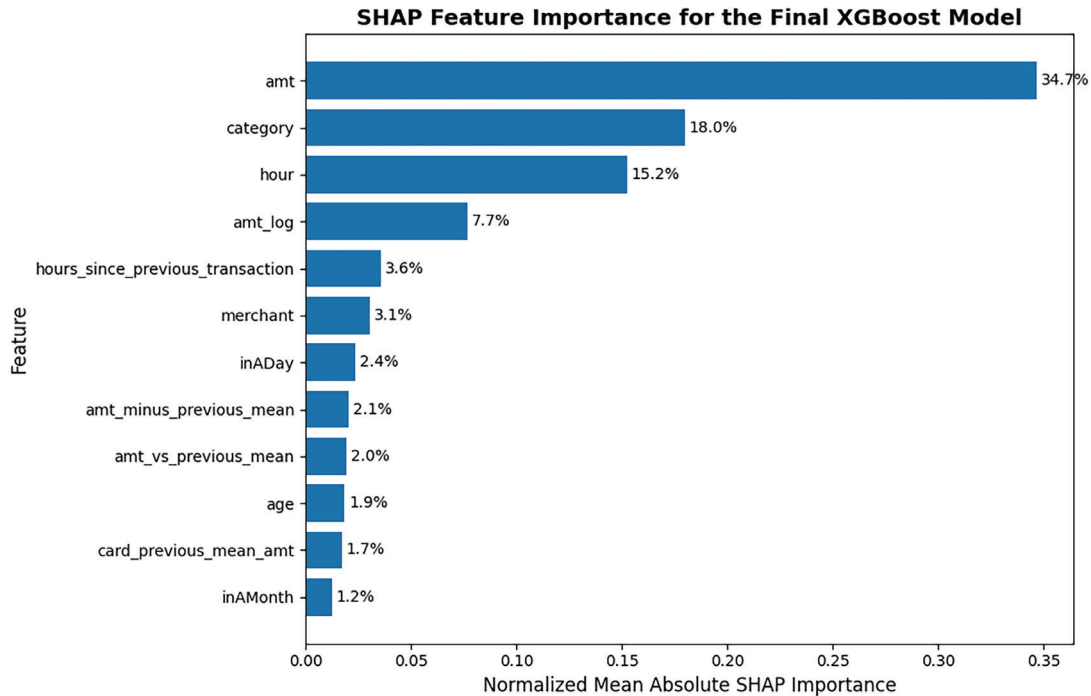


Figure 6. Normalized mean absolute SHAP feature importance for the final XGBoost model. SHAP values were computed with TreeExplainer on up to 2,500 validation observations, aggregated from encoded columns to conceptual source features, and normalized so each feature's grouped mean absolute SHAP contribution is expressed as a share of total XGBoost SHAP importance. The 12 highest-ranked features are shown; lower-ranked features account for the unshown portion, so the displayed percentages do not sum to 100%. Higher values indicate greater average influence on the magnitude of XGBoost predictions, not whether a feature increases or decreases predicted fraud risk. Source: Author's analysis of Shenoy [15] using SHAP [20].

lower AP than the single Decision Tree. This should not be taken as proof that bagging is inherently worse for fraud detection. The Random Forest and Decision Tree were differently constrained; the full study did not use an equal tuning budget across model families, and the final feature set may interact differently with each learner. A stronger follow-up would use nested rolling validation and comparable optimization budgets to determine whether the observed Random Forest deficit persists.

The soft-voting results show an operational tradeoff rather than a universal gain. The ensembles reduced some FP relative to XGBoost but increased FN. Because the component probabilities were not calibrated before averaging, one model's numerical probability scale could have more influence than intended. Calibration and out-of-fold ensemble weighting should therefore be tested before drawing stronger conclusions about voting strategies.

Generalizability is also narrower than the phrase

“unseen test data” can imply. The test transactions are later observations from the same synthetic ecosystem. Of 924 unique test cardholders, 908 (98.27%) were already present in the model-training subset, and those recurring cardholders generated 99.9707% of all test transactions. Every one of the 693 test merchants had also been seen during model fitting. Geographic, demographic, merchant-frequency, and card-history variables can therefore preserve a strong fingerprint of recurring entities even after direct identifiers are removed. The experiment primarily evaluates future-transaction generalization for a previously observed synthetic population, not cold-start performance for new customers or merchants.

The Sparkov generator itself can create artifacts. Customer profiles, merchant pools, category-specific amount distributions, day-of-week patterns, and fraud-generation rules are simulated from predefined parameters (15, 16). One especially important pattern

appeared among the 16 cardholders that were unseen in the model-training subset: they generated 163 held-out test transactions, and every one of those transactions was labeled fraud. This makes the temporal test file unsuitable for a balanced unseen-card discrimination analysis and suggests that entity appearance itself can be entangled with the simulation process. A model may learn simulator-specific structure more easily than it would learn adaptive fraud behavior in a live network, so the numerical scores should not be presented as estimates of real-industry performance.

The main limitations are the use of a single synthetic dataset; the overwhelming recurrence of customers and complete recurrence of merchants across temporal partitions; a generator artifact in which all 163 transactions from the 16 unseen test cardholders are fraudulent; unequal imbalance treatments and tuning budgets across model pipelines; reuse of one validation subset for iterative development, ensemble weighting, and threshold selection; absence of nested or rolling temporal validation; uncalibrated soft-voting probabilities; lack of confidence intervals or repeated-sample uncertainty estimates; and incomplete preservation of the exact original package versions and validation probability vectors.

Future work should prioritize 1) cardholder-disjoint and merchant-disjoint group-held-out evaluations constructed from the raw files rather than relying on the temporal test split; 2) rolling-origin or nested temporal validation; 3) equal tuning budgets and comparable imbalance strategies across model families; 4) probability calibration before soft voting; 5) bootstrap or temporal-block uncertainty estimates; 6) feature ablation rather than relying only on SHAP ranking; 7) validation on a second dataset or real de-identified operational data; and 8) a fully reproducible project package containing pinned package versions, hardware details, validation predictions, test predictions, fitted preprocessing objects, and all figure-generation code.

The Total Operational Risk Cost (TORC) expression in Appendix A remains a conceptual future-work proposal only. It was not used to rank the models and has not been validated against real institutional costs. Its weights and scaling would need empirical definition before use in model selection.

CONCLUSION

Among the six configured pipelines evaluated on the held-out Sparkov/Kaggle test file, the selected XGBoost

configuration had the highest observed AP and F1. The ensembles changed the FP/FN tradeoff but did not surpass XGBoost in AP. These findings are specific to one synthetic dataset, one recurring customer/merchant population, and the particular feature engineering, imbalance treatments, hyperparameters, and validation procedure used here. The Random Forest result partially disconfirmed the original hypothesis and underscores that ensemble structure alone does not guarantee improvement. The most important next step is not to generalize the ranking, but to test whether it survives entity-disjoint evaluation, stronger temporal validation, calibrated ensembling, comparable tuning budgets, and additional datasets.

ETHICS, IRB, AND INFORMED CONSENT STATEMENT

This study used a publicly available synthetic dataset and involved no recruitment, intervention, direct interaction with human participants, or access to private identifiable human-subject records. Therefore, institutional review board (IRB) review and informed consent were not required for this analysis. The dataset includes synthetic names, addresses, card numbers, and demographics generated by the simulator rather than real cardholder records (15, 16).

DATA AND CODE AVAILABILITY

The dataset is Credit Card Transactions Fraud Detection Dataset, Kaggle version 1, by Kartik Shenoy (15). The exact source files used are `fraudTrain.csv` and `fraudTest.csv`. Their SHA-256 hashes are `fd7139200dbfcbcd0b6742bbe05a4f1abce532c4fef20918228a651647a3e75d` for `fraudTrain.csv` and `12d553ab19440c752d2531ee1af44bb64f12cc3d3839f1649f19e81c230545f0` for `fraudTest.csv`. The files contain 1,296,675 training rows and 555,719 test rows, with zero raw missing values and unique `trans_num` values for every row. The preprocessing procedure produced the exact chronological 80/20 cutoff, the split-level fraud counts and prevalence reported in Table 1, and a 95-column final encoded model matrix. The saved `Credit_Card_Fraud_Project.ipynb` contains the base-model validation AP values, validation-max F1 values, and selected thresholds for all six methods, and the final held-out test metrics reported in Tables 3 and 4. The analysis code used Python/Google Colab with `pandas`, `NumPy`, `scikit-learn`, `XGBoost`, `SHAP`, `Matplotlib`, `Seaborn`, `Plotly`, and `gdown`; random seed 42

was used for the main pipeline. A frozen requirements file and original probability arrays were not saved, which remains a reproducibility limitation. Future work should preserve pinned package versions, validation/test probability vectors, fitted preprocessing objects, and figure-generation outputs.

ACKNOWLEDGMENTS

The author thanks Kartik Shenoy for publishing the synthetic dataset on Kaggle and Brandon Harris for developing the Sparkov simulator (15, 16). The author also acknowledges academic mentorship and feedback received during development of the project. Open-source Python libraries were essential to the analysis.

CONFLICT OF INTEREST

The author declares no known financial, commercial, or personal conflict of interest that could have influenced the design, analysis, interpretation, or presentation of this study.

AI DISCLOSURE STATEMENT

Generative AI tools, including OpenAI ChatGPT, were used to assist with code debugging. AI assistance was used alongside human mentorship and author review; the work is therefore not described as having been completed without assistance. The author remains responsible for the final methodological choices, reported results, citations, interpretations, and conclusions.

REFERENCES

1. Federal Trade Commission. Consumer Sentinel Network Data Book 2024 [Internet]. Washington (DC): Federal Trade Commission; 2025 [cited 2026-08-27]. Available from: https://www.ftc.gov/system/files/ftc_gov/pdf/csn-annual-data-book-2024.pdf
2. Mastercard. AI perspectives: Transaction fraud [Internet]. [cited 2026-08-27]. Available from: <https://b2b.mastercard.com/news-and-insights/report/ai-perspectives-transaction-fraud/>
3. Federal Reserve Bank of Atlanta. What risk officers are saying about 2024 [Internet]. 2024 May 6 [cited 2026-08-27]. Available from: <https://www.atlantafed.org/research-and-data/publications/take-on-payments/2024/05/06/what-risk-officers-are-saying-about-2024>
4. Visa. AI solutions for fraud prevention and detection [Internet]. [cited 2026-08-27]. Available from: <https://corporate.visa.com/en/solutions/visa-protect/insights/ai-fraud-detection.html>
5. Stripe. Radar: AI-powered fraud detection solution [Internet]. [cited 2026-08-27]. Available from: <https://stripe.com/radar>
6. Mastercard. Mastercard supercharges consumer protection with gen AI [Internet]. 2024 Feb 1 [cited 2026-08-27]. Available from: <https://www.mastercard.com/news/press/2024/february/mastercard-supercharges-consumer-protection-with-gen-ai>
7. Bhattacharyya S, Jha S, Tharakunnel K, Westland JC. Data mining for credit card fraud: a comparative study. *Decis Support Syst.* 2011; 50 (3): 602-613. doi:10.1016/j.dss.2010.08.008.
8. Dal Pozzolo A, Caelen O, Le Borgne YA, Waterschoot S, Bontempi G. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Syst Appl.* 2014; 41 (10): 4915-4928. doi:10.1016/j.eswa.2014.02.026.
9. Breiman L. Random forests. *Mach Learn.* 2001; 45 (1): 5-32. doi:10.1023/A:1010933404324.
10. Chen T, Guestrin C. XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM; 2016. p. 785-794. doi:10.1145/2939672.2939785.
11. Mienye ID, Sun Y. A deep learning ensemble with data resampling for credit card fraud detection. *IEEE Access.* 2023; 11: 30628-30638. doi:10.1109/ACCESS.2023.3262020.
12. Mim MA, Majadi N, Mazumder P. A soft voting ensemble learning approach for credit card fraud detection. *Heliyon.* 2024; 10 (3): e25466. doi:10.1016/j.heliyon.2024.e25466.
13. Wijaya MG, Pinaringgi MF, Zakiyyah AY, Meiliana. Comparative analysis of machine learning algorithms and data balancing techniques for credit card fraud detection. *Procedia Comput Sci.* 2024; 245: 677-688. doi:10.1016/j.procs.2024.10.297.
14. Hernandez Aros L, Bustamante Molano LX, Gutierrez-Portela F, Moreno Hernandez JJ, Rodriguez Barrero MS. Financial fraud detection through the application of machine learning techniques: a literature review. *Humanit Soc Sci Commun.* 2024; 11: 1130. doi:10.1057/s41599-024-03606-0.
15. Shenoy K. Credit Card Transactions Fraud Detection Dataset, version 1 [dataset on the Internet]. Kaggle; 2020 [cited 2026-08-27]. Available from: <https://www.kaggle.com/datasets/kartik2112/fraud-detection>
16. Harris B. Sparkov data generation [software on the Internet]. GitHub; 2020 [cited 2026-08-27]. Available from: <https://github.com/brandonharris/sparkov>

17. Quinlan JR. Induction of decision trees. *Mach Learn.* 1986; 1 (1): 81-106. doi:10.1007/BF00116251.
18. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. *Nature.* 1986; 323: 533-536. doi:10.1038/323533a0.
19. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: machine learning in Python. *J Mach Learn Res.* 2011; 12: 2825-2830.
20. Lundberg SM, Lee SI. A unified approach to interpreting model predictions. *Adv Neural Inf Process Syst.* 2017; 30.

APPENDIX A. PROPOSED TOTAL OPERATIONAL RISK COST (TORC) METRIC

The TORC formulation is presented only as a proposed direction for future cost-sensitive evaluation. It was not used to select or rank the models in this study and has not been empirically validated. The proposed TORC formulation is shown in Figure A1.

$$\begin{aligned}
 TORC &= w_1 \cdot \sum_{FP} \alpha \cdot (1 - p_i) + w_2 \cdot \sum_{customers\ c} \gamma \cdot [(n_c + 1)^r - 1] + w_3 \cdot \sum_{FN} V_i + w_4 \cdot \sum_{FN} \beta \cdot (1 - p_i) + w_5 \cdot \delta \cdot \sum_{FP} \text{norm}(V_i) \cdot \mathbf{1}_{V_i > T} = w_1 \cdot \text{FPC} + w_2 \cdot \text{CAS} + w_3 \cdot \text{MFL} + w_4 \cdot \text{CFNP} + w_5 \cdot \text{FPIT} \\
 \text{Base False Positive Cost (FPC)} &= \sum_{FP} \alpha \cdot (1 - p_i) \\
 \text{Customer Annoyance Score (CAS)} &= \sum_{customers\ c} \gamma \cdot [(n_c + 1)^r - 1] \\
 \text{Total Missed Fraud Loss (MFL)} &= \sum_{FN} V_i \\
 \text{Confidence-based False Negative Penalty (CFNP)} &= \sum_{FN} \beta \cdot (1 - p_i) \\
 \text{False Positive Impact Term (FPIT)} &= \delta \cdot \sum_{FP} \text{norm}(V_i) \cdot \mathbf{1}_{V_i > T} \\
 \sum_{FP} &\text{-- Sum over all false positives} \\
 \sum_{FN} &\text{-- Sum over all false negatives} \\
 \alpha &\text{-- Base cost of a false positive} \\
 \beta &\text{-- Confidence based penalty for false negatives} \\
 p_i &\text{-- Model's predicted probability that transaction } i \text{ is fraudulent} \\
 r &\text{-- Quadratic scaling annoyance function } c_i \\
 n_c &\text{-- Number of false positives for customer } c \\
 V_i &\text{-- Dollar value of transaction } i \text{ (for missed fraud)} \\
 \gamma &\text{-- Penalty per annoyed customer transaction} \\
 \delta &\text{-- Weight for high-value false positive outliers} \\
 \text{norm}(V_i) &\text{-- Normalized value of transaction } i \\
 \mathbf{1}_{V_i > T} &\text{-- Indicator function: 1 if } V_i = T, \text{ else 0}
 \end{aligned}$$

Figure A1. Original prototype Total Operational Risk Cost (TORC) formulation. This exploratory expression is reproduced as originally proposed; it is not part of the reported model evaluation.

APPENDIX B. HYPERPARAMETER SENSITIVITY CONFIGURATIONS

Tables B1–B4 list the exact parameter draws produced by random.Random(42) in the 15-trial sensitivity code for the Decision Tree, Random Forest, XGBoost, and MLP, respectively. Each model used random_state = 42 + trial where applicable. These configurations were diagnostic samples; they were not an equal-budget final model-selection procedure.

Table B1. Exact Decision Tree configurations sampled in the 15-trial sensitivity diagnostic.

Trial	Configuration	Model seed
1	criterion=gini; max_depth=5; min_samples_split=8; min_samples_leaf=3	43
2	criterion=entropy; max_depth=7; min_samples_split=12; min_samples_leaf=6	44
3	criterion=gini; max_depth=15; min_samples_split=12; min_samples_leaf=1	45
4	criterion=entropy; max_depth=15; min_samples_split=8; min_samples_leaf=3	46
5	criterion=gini; max_depth=15; min_samples_split=8; min_samples_leaf=1	47
6	criterion=entropy; max_depth=15; min_samples_split=16; min_samples_leaf=9	48
7	criterion=entropy; max_depth=13; min_samples_split=8; min_samples_leaf=1	49
8	criterion=gini; max_depth=7; min_samples_split=8; min_samples_leaf=3	50

Continued Table B1. Exact Decision Tree configurations sampled in the 15-trial sensitivity diagnostic.

Trial	Configuration	Model seed
9	criterion=gini; max_depth=7; min_samples_split=12; min_samples_leaf=3	51
10	criterion=entropy; max_depth=5; min_samples_split=16; min_samples_leaf=3	52
11	criterion=entropy; max_depth=7; min_samples_split=12; min_samples_leaf=9	53
12	criterion=gini; max_depth=7; min_samples_split=16; min_samples_leaf=9	54
13	criterion=entropy; max_depth=11; min_samples_split=5; min_samples_leaf=9	55
14	criterion=gini; max_depth=15; min_samples_split=8; min_samples_leaf=1	56
15	criterion=gini; max_depth=13; min_samples_split=2; min_samples_leaf=9	57

Table B2. Exact Random Forest configurations sampled in the 15-trial sensitivity diagnostic.

Trial	Configuration	Model seed
1	n_estimators=150; max_depth=10; min_samples_split=2; min_samples_leaf=1; max_features=None; class_weight=balanced	43
2	n_estimators=100; max_depth=10; min_samples_split=12; min_samples_leaf=4; max_features=log2; class_weight=None	44
3	n_estimators=300; max_depth=8; min_samples_split=8; min_samples_leaf=4; max_features=None; class_weight=None	45
4	n_estimators=200; max_depth=12; min_samples_split=5; min_samples_leaf=4; max_features=None; class_weight=None	46
5	n_estimators=150; max_depth=8; min_samples_split=8; min_samples_leaf=8; max_features=log2; class_weight=None	47
6	n_estimators=300; max_depth=12; min_samples_split=5; min_samples_leaf=2; max_features=None; class_weight=balanced	48
7	n_estimators=100; max_depth=20; min_samples_split=8; min_samples_leaf=4; max_features=sqrt; class_weight=balanced	49
8	n_estimators=100; max_depth=20; min_samples_split=8; min_samples_leaf=8; max_features=sqrt; class_weight=None	50
9	n_estimators=200; max_depth=20; min_samples_split=12; min_samples_leaf=2; max_features=None; class_weight=None	51
10	n_estimators=150; max_depth=8; min_samples_split=2; min_samples_leaf=1; max_features=sqrt; class_weight=None	52
11	n_estimators=150; max_depth=8; min_samples_split=2; min_samples_leaf=8; max_features=log2; class_weight=balanced	53
12	n_estimators=150; max_depth=15; min_samples_split=5; min_samples_leaf=4; max_features=log2; class_weight=None	54
13	n_estimators=100; max_depth=10; min_samples_split=12; min_samples_leaf=1; max_features=log2; class_weight=balanced	55
14	n_estimators=100; max_depth=20; min_samples_split=12; min_samples_leaf=2; max_features=sqrt; class_weight=None	56
15	n_estimators=200; max_depth=12; min_samples_split=5; min_samples_leaf=4; max_features=sqrt; class_weight=balanced	57

Table B3. Exact XGBoost configurations sampled in the 15-trial sensitivity diagnostic.

Trial	Configuration	Model seed
1	max_depth=4; learning_rate=0.01; n_estimators=200; scale_pos_weight=3; subsample=0.8; colsample_bytree=0.8	43
2	max_depth=6; learning_rate=0.05; n_estimators=200; scale_pos_weight=1.5; subsample=1.0; colsample_bytree=0.8	44
3	max_depth=4; learning_rate=0.01; n_estimators=400; scale_pos_weight=3; subsample=1.0; colsample_bytree=0.8	45
4	max_depth=10; learning_rate=0.03; n_estimators=400; scale_pos_weight=3; subsample=0.8; colsample_bytree=1.0	46
5	max_depth=6; learning_rate=0.05; n_estimators=200; scale_pos_weight=8; subsample=1.0; colsample_bytree=1.0	47
6	max_depth=4; learning_rate=0.01; n_estimators=200; scale_pos_weight=3; subsample=0.8; colsample_bytree=1.0	48
7	max_depth=9; learning_rate=0.03; n_estimators=300; scale_pos_weight=1.5; subsample=1.0; colsample_bytree=0.8	49
8	max_depth=8; learning_rate=0.05; n_estimators=200; scale_pos_weight=1.5; subsample=0.8; colsample_bytree=0.8	50
9	max_depth=8; learning_rate=0.1; n_estimators=400; scale_pos_weight=5; subsample=1.0; colsample_bytree=1.0	51
10	max_depth=4; learning_rate=0.05; n_estimators=200; scale_pos_weight=3; subsample=1.0; colsample_bytree=1.0	52
11	max_depth=9; learning_rate=0.1; n_estimators=400; scale_pos_weight=1.5; subsample=0.8; colsample_bytree=0.8	53
12	max_depth=4; learning_rate=0.1; n_estimators=400; scale_pos_weight=1.5; subsample=0.8; colsample_bytree=0.8	54
13	max_depth=9; learning_rate=0.05; n_estimators=300; scale_pos_weight=8; subsample=0.8; colsample_bytree=0.8	55
14	max_depth=6; learning_rate=0.01; n_estimators=400; scale_pos_weight=1.5; subsample=0.8; colsample_bytree=1.0	56
15	max_depth=9; learning_rate=0.03; n_estimators=400; scale_pos_weight=5; subsample=1.0; colsample_bytree=1.0	57

Table B4. Exact MLP configurations sampled in the 15-trial sensitivity diagnostic.

Trial	Configuration	Model seed
1	hidden_layer_sizes=(128, 64, 32); activation=relu; learning_rate_init=0.001; alpha=0.01	43
2	hidden_layer_sizes=(128, 64); activation=tanh; learning_rate_init=0.001; alpha=0.001	44
3	hidden_layer_sizes=(128,); activation=relu; learning_rate_init=0.0003; alpha=0.001	45
4	hidden_layer_sizes=(256, 128); activation=tanh; learning_rate_init=0.001; alpha=0.01	46
5	hidden_layer_sizes=(128,); activation=tanh; learning_rate_init=0.0001; alpha=0.0001	47
6	hidden_layer_sizes=(128, 64, 32); activation=relu; learning_rate_init=0.0003; alpha=0.001	48
7	hidden_layer_sizes=(128, 64, 32); activation=relu; learning_rate_init=0.001; alpha=0.001	49
8	hidden_layer_sizes=(64,); activation=tanh; learning_rate_init=0.0001; alpha=0.01	50
9	hidden_layer_sizes=(64,); activation=relu; learning_rate_init=0.0001; alpha=0.0001	51
10	hidden_layer_sizes=(128,); activation=relu; learning_rate_init=0.001; alpha=0.01	52
11	hidden_layer_sizes=(256, 128); activation=tanh; learning_rate_init=0.0001; alpha=0.0001	53
12	hidden_layer_sizes=(64,); activation=relu; learning_rate_init=0.0001; alpha=0.001	54
13	hidden_layer_sizes=(128, 64); activation=relu; learning_rate_init=0.0001; alpha=0.01	55
14	hidden_layer_sizes=(64,); activation=relu; learning_rate_init=0.001; alpha=0.01	56
15	hidden_layer_sizes=(64,); activation=relu; learning_rate_init=0.0003; alpha=0.01	57