

Temporal Evaluation of Gradient Boosting and Autoencoder Models for Credit Card Fraud Detection

Arnav Thakur

Abstract

Payment card fraud losses exceeded US\$33 billion worldwide in 2022, yet fraud detection models are often summarized by accuracy values that conceal how much fraud they miss. This study compares a Naive Bayes baseline, three gradient boosting libraries (LightGBM, XGBoost and CatBoost), an autoencoder anomaly detector, and a hybrid model that supplies the autoencoder's reconstruction error to LightGBM, using the IEEE-CIS Fraud Detection dataset. The 590,540 labeled transactions were ordered in time and divided into training (60%), validation (20%) and test (20%) periods, and all preprocessing was fitted on the training period only. On the later test period (118,108 transactions, 4,064 of them fraudulent), LightGBM reached an average precision (AP) of 0.501 and a ROC-AUC of 0.889. XGBoost reached an AP of 0.506 when text features were supplied as integer codes but only 0.325 with its native categorical handling, a larger gap than any observed between libraries. CatBoost reached 0.460 at its 800-iteration cap. The autoencoder alone was weaker than Naive Bayes (AP 0.111 versus 0.132), and adding its reconstruction error to LightGBM changed AP by +0.003 (95% day-bootstrap interval −0.002 to 0.009). At thresholds selected on validation data, the strongest models detected 43–46% of test fraud at 53–55% precision, whereas a classifier that flags nothing reached 96.6% accuracy. Every model performed worse on the test period than on validation. These results indicate that categorical encoding, threshold choice and temporal evaluation affect reported fraud detection performance at least as much as the choice among modern boosting libraries.

Keywords: credit card fraud, IEEE-CIS, gradient boosting, autoencoder, temporal validation, class imbalance, average precision

1. Introduction

Payment card fraud is a large and growing cost for the financial system. According to the Nilson Report, card issuers, merchants and acquirers worldwide incurred gross fraud losses of \$32.34 billion in 2021, equal to 6.61¢ for every \$100 of card volume, and the industry was projected to lose a cumulative \$397 billion over 2022–2031 [1]. Losses rose again to \$33.45 billion in 2022, or 6.81¢ per \$100 [2]. Because fraudulent transactions are rare compared with legitimate ones, and because a missed fraud and a wrongly blocked customer both carry costs, detecting fraud automatically is a difficult classification problem that has attracted a large machine learning literature [3].

Two properties of fraud data make evaluation especially important. First, fraud is rare. A model that labels every transaction as legitimate can reach very high accuracy while detecting nothing, so accuracy alone is misleading. Precision (the share of alerts that are truly fraud) and recall (the share of fraud that is detected) describe what a model actually does, and precision–recall analysis is more informative than ROC analysis

when positives are rare [4]. Second, transactions arrive over time. Customer behavior, fraud strategies and the availability of confirmed labels change between the period used to build a model and the period in which it is used. Dal Pozzolo and colleagues identify class imbalance, concept drift and verification latency as central constraints in realistic fraud detection [5]. Evaluating a model on transactions that occur after its training data (a chronological holdout) addresses part of this problem.

Gradient boosted decision trees are among the most widely used methods for tabular data such as transaction records. XGBoost, LightGBM and CatBoost implement boosting with different tree-growing strategies and different approaches to categorical features [6, 7, 8], and large benchmarks continue to find tree ensembles highly competitive with deep learning on tabular problems [9, 10]. Autoencoders offer a different approach: a neural network trained to reconstruct normal transactions should reconstruct unusual transactions poorly, so its reconstruction error can serve as an anomaly score that does not require fraud labels [11]. Combining an autoencoder with gradient boosting has also been proposed [12].

Published results on the IEEE-CIS dataset used here are usually reported as ROC-AUC or accuracy, are often obtained after resampling, and use different train–test protocols (Table 1). This makes it hard to compare models or to understand what a reported score means for a fraud team deciding which transactions to review. This project originally asked how the integration of LightGBM, XGBoost and an autoencoder could improve the accuracy and efficiency of fraud detection. Refined to reflect the evaluation issues above, the study addresses three research questions:

RQ1. Under a shared chronological split, how do a Naive Bayes baseline and the LightGBM, XGBoost and CatBoost pipelines compare in their ability to rank fraud above legitimate transactions?

RQ2. Does an autoencoder, used on its own or integrated with LightGBM through its reconstruction error, improve fraud detection on later transactions?

RQ3. How does choosing the decision threshold on validation data, rather than using the conventional 0.5, change precision, recall and the number of alerts on later transactions?

The contribution is an empirical comparison with a documented, leakage-controlled evaluation procedure, uncertainty intervals, and a sensitivity analysis showing that the treatment of categorical features can change the ranking of algorithms. No new learning algorithm is proposed.

2. Related Work

2.1 Models applied to the IEEE-CIS dataset

Table 1 summarizes studies that informed this project. Zhang et al. applied XGBoost with SMOTE oversampling, statistical feature engineering and recursive feature elimination and reported a ROC-AUC of 0.952, compared with 0.925 for random forest and 0.907 for a support vector machine [13]. Kewei et al. proposed a five-layer neural network with feature engineering, memory compression and a combined cross-entropy and focal loss, reporting a ROC-AUC of 0.910 compared with 0.889 for a support vector machine and 0.857 for Naive Bayes [14]. Prabha and Priscilla used an autoencoder for dimensionality reduction before an XGBoost classifier with a probabilistic threshold and reported 90.4% recall [12]. On the Kaggle competition itself, public notebooks such as Deotte's showed that engineered identifiers linking transactions from the same client were central to top scores [15]. These studies differ in data splits, resampling, feature engineering and metrics, so their scores are not directly comparable with one another or with this study.

Table 1. Prior studies on the IEEE-CIS dataset that informed this work. Reported values are as published and were obtained under different protocols.

Study	Methods	Reported result	Protocol notes
Zhang et al., 2020 [13]	XGBoost vs. logistic regression, SVM, random forest	ROC-AUC 0.952; accuracy 0.981	SMOTE oversampling; feature selection to 100 features
Kewei et al., 2021 [14]	Hybrid deep neural network vs. Naive Bayes, SVM	ROC-AUC 0.910 (NB 0.857, SVM 0.889)	Feature engineering; BCE + focal loss
Prabha & Priscilla, 2023 [12]	Autoencoder + XGBoost with probabilistic threshold	Recall 90.4%; F1 90.5%	Autoencoder used for dimensionality reduction
Deotte, 2019 [15]	XGBoost with engineered client identifiers	Competition score 0.9600 (per notebook title)	Kaggle competition leaderboard, unlabeled test set
This study	Naive Bayes, LightGBM, XGBoost, CatBoost, autoencoder, hybrid	Best test AP 0.506; ROC-AUC 0.891	Chronological 60/20/20 split; no resampling

2.2 Datasets used in credit card fraud research

Real transaction data are confidential, so only a few public datasets are available, and a large share of the literature relies on the same small set [3]. An informal review of recent articles carried out for this project found that most used the European cardholder dataset released by the Machine Learning Group of the Université Libre de Bruxelles [16]. That dataset contains 284,807 transactions made over two days in September 2013, of which 492 (0.172%) are fraudulent, and most of its features are anonymized principal components. Synthetic datasets such as PaySim, which simulates mobile money transfers from a sample of real logs, avoid confidentiality restrictions but may not reproduce real fraud behavior [17]. Table 2 compares these datasets with the IEEE-CIS dataset used in this study.

Table 2. Commonly used public datasets for payment fraud research.

Dataset	Source	Type	Transactions	Fraud share	Features
European cardholders (ULB) [16]	Kaggle, 2013 data	Real	284,807 (2 days)	0.172%	30 (28 PCA components, time, amount)
PaySim [17]	Simulator, 2016	Synthetic	Several million	Varies by simulation	Transaction type, amount, balances
IEEE-CIS Fraud Detection [18]	Kaggle / Vesta, 2019	Real	590,540 labeled + 506,691 unlabeled	3.50% (labeled)	433 raw and engineered features in two tables

The IEEE-CIS dataset was chosen for three reasons. It is larger and more recent than the European dataset. Many of its features, such as card type, e-mail domain, product code and device, are directly interpretable rather than principal components. Its timestamps also span about six months, which allows a chronological split, whereas the European dataset covers only two days.

2.3 Tree ensembles and deep learning on tabular data

Grinsztajn et al. showed that tree-based models still outperformed deep networks on typical medium-sized tabular datasets and attributed this partly to irregular target functions and uninformative features [10]. Shmuel et al. benchmarked 20 models on 111 tabular datasets and found that an automated ensemble (AutoGluon) was best on 35% of datasets and that CatBoost was the best single model, winning on 19 datasets (17.1%), while deep learning models were favored only under narrower conditions [9]. These findings motivated the focus on LightGBM, XGBoost and CatBoost here, with a neural autoencoder included to test whether an unsupervised deep model adds information.

2.4 Autoencoders for fraud detection

An autoencoder compresses each input into a small latent representation and then reconstructs it. When it is trained only on legitimate transactions, a high reconstruction error suggests that a transaction is unlike those seen in training. Pumsirirat and Yan applied autoencoders and restricted Boltzmann machines to credit card fraud and highlighted their ability to detect unusual patterns without labeled fraud [11]. This property is attractive because fraud labels are often delayed or incomplete [5]. Whether an autoencoder helps when many labels are available, and whether it adds information to a strong supervised model, are empirical questions that this study tests.

3. Methods

3.1 Dataset

The IEEE-CIS Fraud Detection dataset was released in 2019 for a Kaggle competition organized by the IEEE Computational Intelligence Society, using e-commerce transactions provided by the payment company Vesta [18]. It consists of a transaction table and an identity table linked by TransactionID. The labeled transaction table has 590,540 rows and 394 columns, and the identity table has 144,233 rows and 41 columns, so identity information (device and network details) is available for 24.4% of transactions. A left join on TransactionID produced 434 columns including the label isFraud. There are 20,663 fraudulent transactions (3.499%). The competition's unlabeled test files were not used. Table 3 summarizes the feature groups as described by the data provider [18].

Table 3. Feature groups in the IEEE-CIS dataset. Many meanings are masked by the data provider.

Group	Columns	Description
Time	TransactionDT	Seconds elapsed from an undisclosed reference time (not a calendar date)
Amount	TransactionAmt	Transaction payment amount in USD
Product	ProductCD	Product code (categorical)
Card	card1–card6	Payment card information such as card type, category and issuer

Group	Columns	Description
Address and distance	addr1–addr2, dist1–dist2	Address and distance information
E-mail	P_emaildomain, R_emaildomain	Purchaser and recipient e-mail domains
Counting	C1–C14	Counts, such as how many addresses are associated with the card (masked)
Time deltas	D1–D15	Time since events such as the previous transaction
Match	M1–M9	Match flags, such as whether names on the card and address agree
Vesta features	V1–V339	Engineered features including ranks, counts and entity relations
Identity	id_01–id_38, DeviceType, DeviceInfo	Network, browser and device information (24.4% of rows)

3.2 Exploratory data analysis

Exploratory data analysis (EDA) was performed in Python with pandas and matplotlib. It examined the class balance, the weekly fraud rate across the time span, the distribution of transaction amounts by class, fraud rates within categories of selected categorical features, and the proportion of missing values per feature. Results are reported in Section 4.1.

3.3 Chronological partition

The joined observations were sorted by TransactionDT and then TransactionID. The earliest 60% formed the training set, the next 20% the validation set and the latest 20% the test set (Table 4). Transactions with identical timestamps at a boundary were to be assigned to the later partition; no ties occurred at either boundary. The test period covers 42 elapsed-day groups. The validation set was used for early stopping and for choosing decision thresholds, and the test set was used only for final evaluation.

Table 4. Composition of the chronological partitions. Time ranges are TransactionDT offsets in seconds.

Partition	Transactions	Fraud cases	Fraud %	Elapsed days	Time range (s)
Training	354,324	11,988	3.38	101	86,400–8,745,772
Validation	118,108	4,611	3.90	41	8,745,798–12,192,842
Test	118,108	4,064	3.44	42	12,192,900–15,811,131

3.4 Predictors and preprocessing

TransactionID and TransactionDT were excluded from the predictors, leaving 431 features: 400 numerical and 31 text (categorical) columns. All original features were kept, and no additional aggregate features were engineered. Missing numerical values were replaced with the training-set median, and the values were stored as 32-bit floats. For each text column, the set of categories was learned from the training

period only. Missing values received a dedicated "missing" category, and categories that first appeared after the training period were mapped to a reserved "unseen" category. Numerical-coded identifiers such as card1 were treated as numbers. No resampling, class weighting or other rebalancing was applied. Fitting all preprocessing on training data only follows standard practice for avoiding data leakage [19].

3.5 Supervised models

Table 5 lists the models and their settings. The Naive Bayes baseline combined Gaussian likelihoods for standardized numerical features (variance smoothing 10^{-9}) with categorical likelihoods for text features (Laplace smoothing $\alpha = 1$), adding the class prior only once [20]. Its ranking score was the log-odds of fraud, which avoids ties when probabilities round to 0 or 1; a log-odds of zero corresponds to probability 0.5.

Each boosting model used a learning rate of 0.05 and at most 800 iterations. Training stopped when validation ROC-AUC did not improve for 50 iterations, and predictions used the best validation iteration. XGBoost used histogram-based tree construction with depth 6 and 80% row and column subsampling. LightGBM used 31 leaves, at least 20 samples per leaf and no subsampling. CatBoost used depth 6 and the log-loss objective. Other settings were library defaults, and the random seed was 42. In the primary analysis, each library used its native categorical handling: pandas categorical columns for XGBoost and LightGBM, and declared categorical features for CatBoost. Because XGBoost behaved unexpectedly under this setting, a sensitivity analysis refitted XGBoost and LightGBM with the text columns supplied as integer category codes, holding everything else constant. No hyperparameter search was performed, and no model was refitted on the combined training and validation data.

Table 5. Models, key settings, iterations retained by early stopping and training time. Times are single runs on two CPU threads and are indicative only.

Model	Description	Key settings	Iterations	Train time
Naive Bayes	Probabilistic baseline that assumes features are independent given the class	Gaussian + categorical likelihoods	—	8 s
LightGBM [7]	Gradient boosting with leaf-wise tree growth and histogram binning	31 leaves, min. 20 per leaf	433	40 s
XGBoost [6]	Regularized gradient boosting with level-wise trees	Depth 6, 0.8 row/column sampling	788	5.3 min
CatBoost [8]	Gradient boosting with ordered target statistics for categorical features	Depth 6, log-loss	799	23.5 min
Autoencoder	Neural network trained to reconstruct legitimate transactions (Section 3.6)	431–32–16–8–4–8–16–32–431	15 / 17 epochs	5.5 min

Model	Description	Key settings	Iterations	Train time
LightGBM + AE feature	LightGBM with the autoencoder reconstruction error as an extra feature	As LightGBM	536	45 s + AE
XGBoost (integer codes)	Sensitivity: text columns as integer codes	As XGBoost	784	2.8 min
LightGBM (integer codes)	Sensitivity: text columns as integer codes	As LightGBM	421	33 s

Iterations are the number of trees retained at the best validation iteration. CatBoost retained 799 and both XGBoost variants retained more than 780 of a maximum of 800 trees, so their results may be limited by the iteration budget. The autoencoder row gives the best epochs for its two cross-fitted models.

3.6 Autoencoder and hybrid model

The autoencoder used the architecture developed earlier in this project (Table 6). Its input had 431 values per transaction. The 400 numerical features were transformed with a signed logarithm, $\text{sign}(x) \cdot \log(1 + |x|)$, to reduce the influence of extreme values. Each text feature was replaced by the frequency of its category in the training period. All inputs were then standardized with training means and standard deviations. The network was implemented in PyTorch [21] and trained only on legitimate training transactions to minimize mean squared reconstruction error plus an L2 weight penalty of 0.001, using the Adam optimizer (learning rate 0.001), batches of 256 and up to 30 epochs. Training stopped when the reconstruction error on legitimate validation transactions did not improve for three epochs, and the best epoch was restored. The fraud score of a transaction was its mean squared reconstruction error.

Table 6. Autoencoder architecture. L2 denotes a weight penalty of 0.001 on the layer.

Block	Layer	Units	Activation	Regularization
Encoder	Dense	32	ReLU	L2; dropout 0.2 after layer
Encoder	Dense	16	ReLU	L2
Encoder	Dense	8	ReLU	L2
Latent	Dense	4	ReLU	—
Decoder	Dense	8	ReLU	L2; dropout 0.2 after layer
Decoder	Dense	16	ReLU	L2
Decoder	Dense	32	ReLU	L2
Output	Dense	431	Linear	—

To use the reconstruction error as a feature without leaking information, the legitimate training transactions were randomly split into two halves and one autoencoder was trained on each half (cross-fitting). Each legitimate training transaction was scored by the model that had not seen it. For the stand-alone autoencoder, all other transactions (training fraud, validation and test) were scored by the average of the two models. For the hybrid feature, each of those transactions was instead scored by one randomly

chosen model, so that the feature had the same single-model noise level in every partition. The hybrid model was LightGBM with the same settings as above plus this one additional feature.

3.7 Evaluation

Fraud was the positive class. ROC-AUC measures how well scores separate fraud from legitimate transactions across all thresholds. Average precision (AP) summarizes the precision–recall curve as a weighted mean of precision over increases in recall, using the non-interpolated scikit-learn implementation [22]; its value for a random ranking equals the fraud prevalence (0.034 in the test set). Accuracy, precision, recall and F1 were calculated from the confusion matrix, with $F1 = 2TP / (2TP + FP + FN)$, where TP, FP and FN are true positives, false positives and false negatives.

For each model, the decision threshold that maximized F1 on the validation set was chosen from the observed validation scores, with ties resolved toward the higher threshold. That threshold was then applied unchanged to the test set. Results at the conventional probability threshold of 0.5 (log-odds 0 for Naive Bayes) are also reported; the autoencoder has no natural 0.5 threshold.

Uncertainty was estimated with 300 paired bootstrap resamples of the 42 test days, defined as $\text{floor}(\text{TransactionDT} / 86,400)$. Each replicate sampled days with replacement and included every transaction from each sampled day. The same resamples were used for all models (seed 2026), so differences between models are paired. The 2.5th and 97.5th percentiles give approximate 95% intervals. These intervals reflect variation across the observed test days for fixed fitted models. They do not include variation from retraining or from future shifts in the data, and the pairwise comparisons are exploratory and not adjusted for multiple testing.

3.8 Software and reproducibility

The analysis used Python 3.11 with pandas 3.0.2, scikit-learn 1.8.0, LightGBM 4.7.0, XGBoost 3.2.0, CatBoost 1.2.10 and PyTorch 2.14.0 (CPU), on two CPU threads. The SHA-256 hashes of the input files were 3a5c83ab...d642 (train_transaction.csv) and b63c725d...c37c (train_identity.csv). The analysis scripts, which reproduce every table and figure from the raw data, are provided as supplementary material.

4. Results

4.1 Exploratory data analysis

Figure 1A shows the class distribution. Of the 590,540 labeled transactions, 569,877 (96.5%) are legitimate and 20,663 (3.5%) are fraudulent. This strong imbalance means that accuracy is not a useful summary of model quality, which is why AP and precision–recall curves are the main metrics in this study. Figure 1B shows that the weekly fraud rate was not constant over time: among weeks with more than 1,000 transactions it ranged from 1.85% to 5.06%. The validation period had a higher fraud rate (3.90%) than either the training (3.38%) or test (3.44%) periods. This variation is one reason to evaluate models on later data rather than on a random sample.



Figure 1. Class balance and fraud over time. (A) Number of legitimate and fraudulent labeled transactions. (B) Weekly fraud rate across elapsed days, with the training, validation and test periods shaded; the dashed line is the overall rate of 3.5%.

Figure 2 compares transaction amounts. The distributions overlap heavily: the median fraudulent amount was \$75.00 and the median legitimate amount was \$68.50. Fraud is over-represented among very small amounts (below \$1) and in the \$5–\$20 range, which may reflect small test purchases, but amount alone clearly cannot separate the classes.

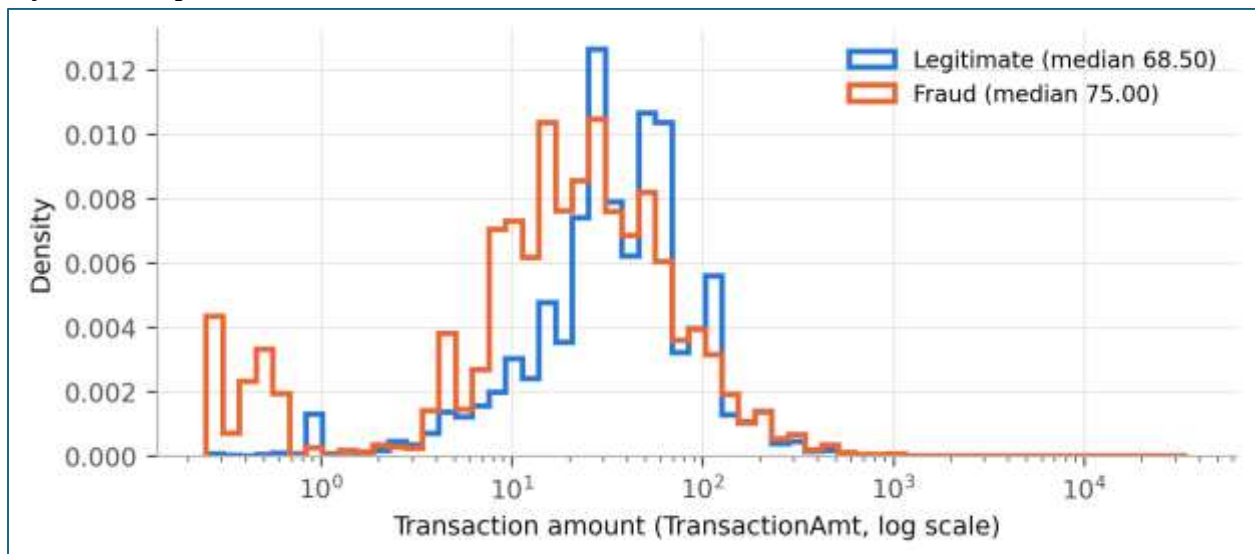


Figure 2. Distribution of transaction amounts for legitimate and fraudulent transactions (density, logarithmic horizontal axis).

Figure 3 shows that fraud rates differ strongly between categories. Product code C had a fraud rate of 11.7%, almost six times that of product code W (2.0%), which accounts for most transactions. Credit cards had a higher fraud rate (6.7%) than debit cards (2.4%). Among the 12 most frequent purchaser e-mail domains, outlook.com had the highest rate (9.5%) and att.net the lowest (0.7%). Some rarer domains had even higher rates, for example 19.0% for mail.com, but these were based on few transactions (559 for mail.com), so the estimates are uncertain. Fraud also varied by time of day relative to the unknown reference time, peaking at 10.6% in one hourly bin compared with about 2.3% at its lowest, but the true clock time cannot be recovered from TransactionDT.

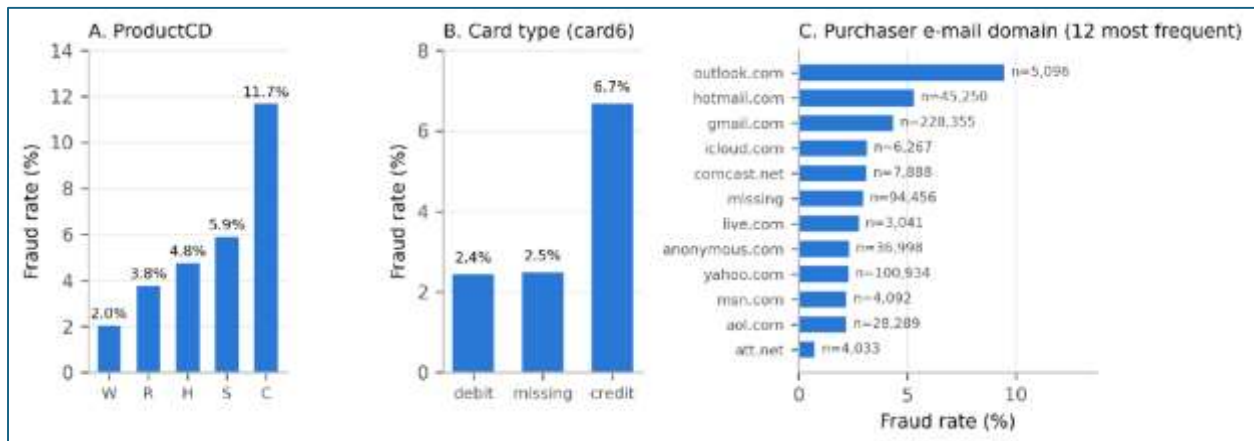


Figure 3. Fraud rate by (A) product code, (B) card type (categories with at least 100 transactions) and (C) purchaser e-mail domain for the 12 most frequent domains, with the number of transactions in each domain.

Figure 4 shows that missing data are widespread. Of the 431 predictors, 414 have at least one missing value, 214 are missing for more than half of transactions and 74 are missing for more than 80%. Most of the heavily missing features come from the identity table, which covers only about a quarter of transactions, and from groups of V features that share the same missing pattern. An earlier version of this project removed columns with more than 80% missing values. The final analysis kept all features so that every model used the same fixed feature set; the effect of removing sparse features is left for future ablation.

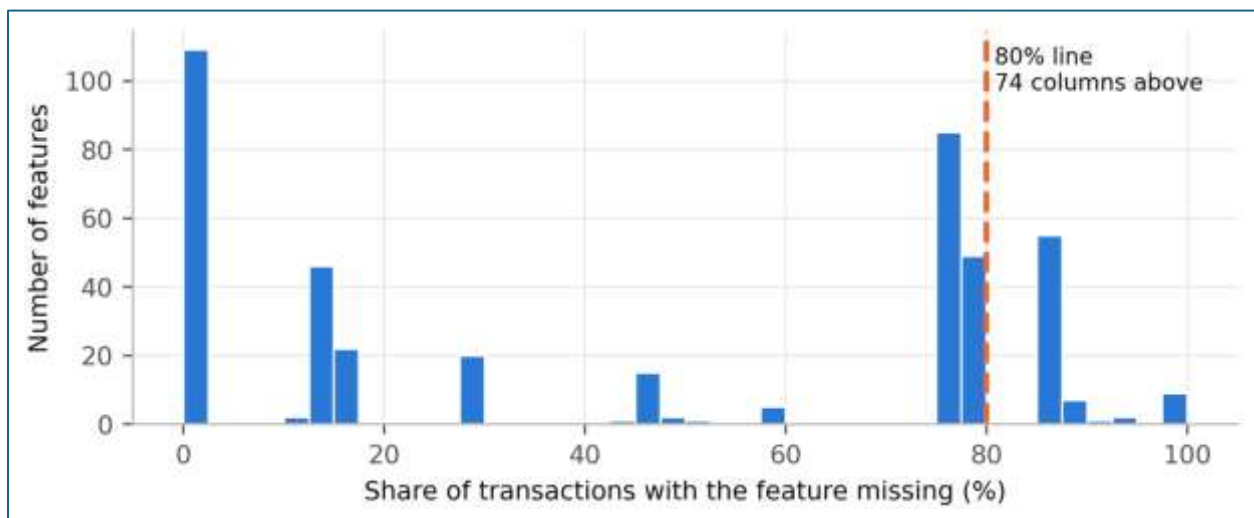


Figure 4. Number of features by share of transactions with the feature missing. The dashed line marks the 80% threshold considered in an earlier version of the project.

4.2 Ranking performance on later transactions (RQ1)

Table 7 reports ROC-AUC and AP on the validation and test periods, with bootstrap intervals for the test estimates. A classifier that labels every transaction as legitimate reached 96.56% test accuracy but has ROC-AUC 0.5 and AP equal to the prevalence, 0.034. Among the three libraries in their primary (native categorical) configuration, LightGBM had the highest test AP (0.501; interval 0.472–0.531) and ROC-AUC (0.889), followed by CatBoost (AP 0.460) and XGBoost (AP 0.325). Naive Bayes and the autoencoder were far behind. When XGBoost received integer codes instead of native categorical

columns, its test AP rose to 0.506, the highest observed value in the study, but its difference from LightGBM (+0.004; interval -0.003 to 0.013) was small and its interval included zero (Table 10).

Table 7. Validation and test discrimination. Parentheses contain approximate 95% intervals from 300 day-level bootstrap resamples of the test period.

Model	Validation ROC-AUC	Test ROC-AUC (95% interval)	Validation AP	Test AP (95% interval)
All legitimate (trivial)	0.500	0.500	0.039	0.034
Naive Bayes	0.811	0.790 (0.774–0.804)	0.276	0.132 (0.101–0.193)
LightGBM	0.920	0.889 (0.880–0.896)	0.604	0.501 (0.472–0.531)
XGBoost (native categorical)	0.907	0.871 (0.862–0.880)	0.443	0.325 (0.293–0.360)
CatBoost	0.903	0.873 (0.863–0.881)	0.580	0.460 (0.428–0.496)
Autoencoder	0.776	0.761 (0.746–0.775)	0.190	0.111 (0.089–0.152)
LightGBM + AE feature	0.921	0.890 (0.881–0.897)	0.604	0.504 (0.474–0.534)
Sensitivity analysis				
XGBoost (integer codes)	0.926	0.891 (0.884–0.898)	0.620	0.506 (0.477–0.536)
LightGBM (integer codes)	0.923	0.888 (0.879–0.896)	0.603	0.499 (0.467–0.532)

Figure 5 shows the full precision–recall and ROC curves on the test set. The curves for LightGBM, the hybrid model and XGBoost with integer codes almost overlap. At low recall these models are very precise: roughly 90% of their highest-scored transactions are fraud up to a recall of about 20%. Precision then falls steadily, reaching about 30% at a recall of 60%. XGBoost with native categorical handling had a noticeably lower curve, with precision below about 60% across nearly all recall levels. The ROC curves are much closer together than the precision–recall curves, which illustrates why ROC-AUC can hide large differences in the precision of the alerts that a fraud team would actually review [4].

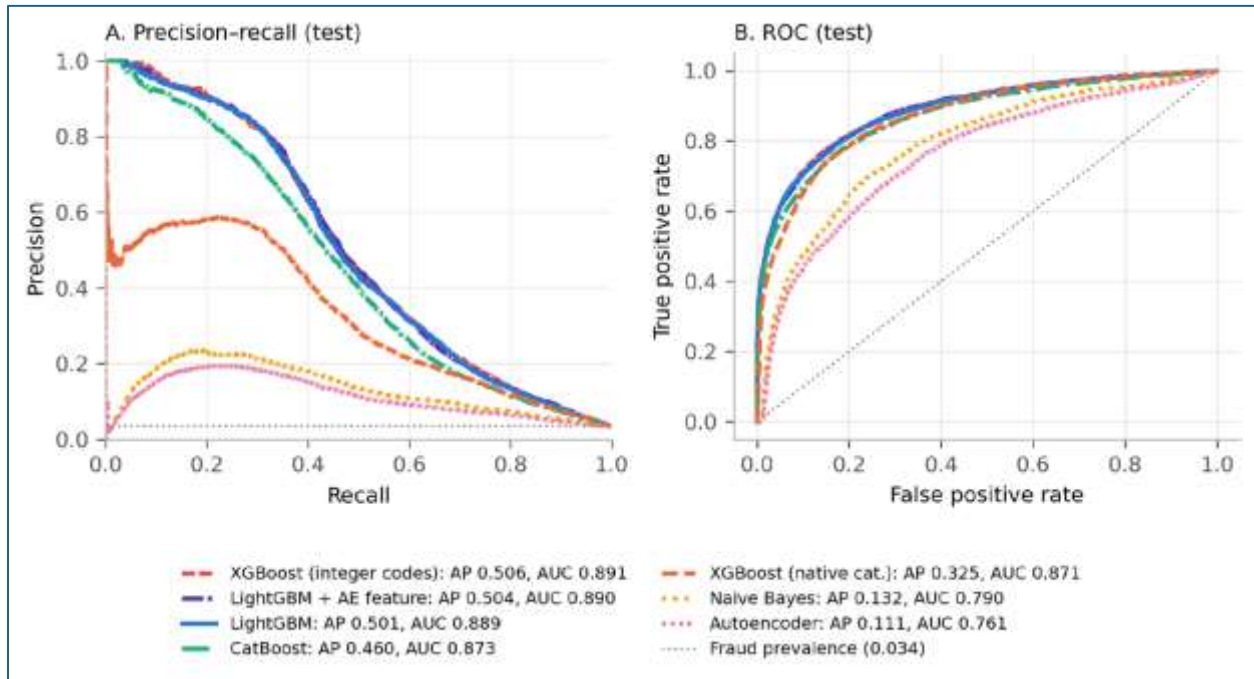


Figure 5. (A) Precision–recall and (B) ROC curves on the chronological test set. The dotted horizontal line in panel A is the test fraud prevalence, the AP of a random ranking.

Every model scored lower on the test period than on validation (Figure 6). For LightGBM, AP fell from 0.604 to 0.501 and ROC-AUC from 0.920 to 0.889. The drop was proportionally largest for Naive Bayes, whose AP roughly halved. Part of this gap is expected because the validation set was used for early stopping, but a consistent drop across all models, including Naive Bayes which does not use early stopping, suggests that the later period is also harder to predict. The split alone cannot determine whether this is due to changing fraud patterns, new categories (Section 4.5) or other differences between periods.

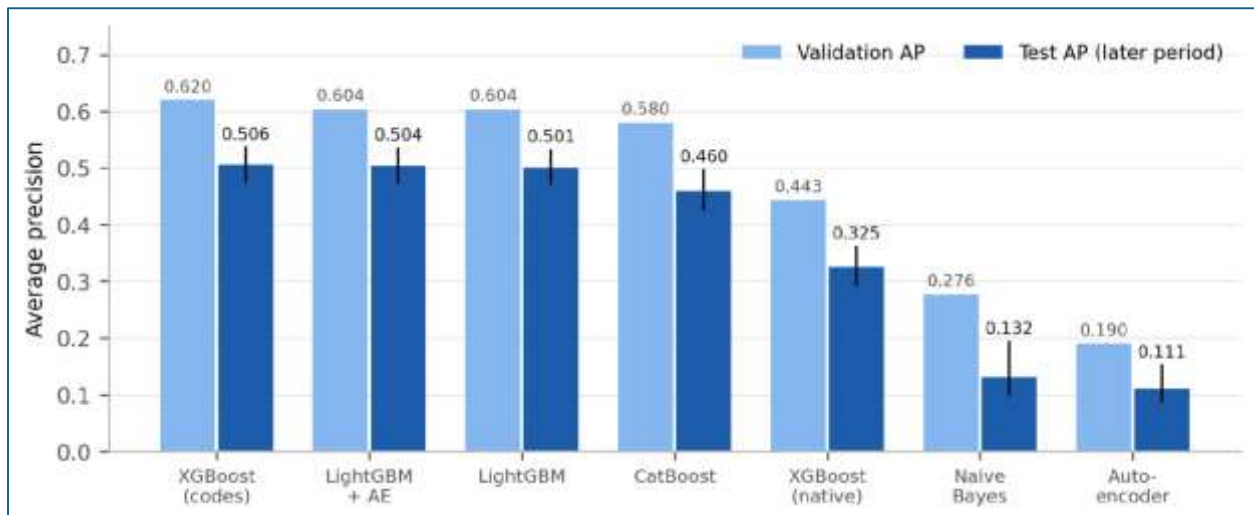


Figure 6. Average precision on the validation period and the later test period. Error bars show approximate 95% day-bootstrap intervals for the test estimates.

4.3 Threshold selection and detected fraud (RQ3)

Table 8 reports test performance at the thresholds that maximized validation F1. Among the main models, the hybrid model detected the most fraud (1,849 of 4,064 cases, recall 45.5%) with 1,644 false alerts, about the same as the integer-coded LightGBM sensitivity model (1,852 cases), while LightGBM had the highest

precision (55.3%) with 1,750 detected cases and 1,415 false alerts. Even the strongest models missed more than half of the fraud at these operating points. All of the boosting models except native-categorical XGBoost had test accuracy between 96.6% and 96.8%, barely above the 96.6% of the trivial classifier, which again shows how uninformative accuracy is for this problem.

Table 8. Test performance at thresholds chosen to maximize validation F1. Precision, recall and accuracy are percentages. Naive Bayes uses a log-odds threshold and the autoencoder a reconstruction-error threshold.

Model	Threshold	Alerts	TP	FP	FN	Precision	Recall	F1	Accuracy
Naive Bayes	327.56	6,104	1,277	4,827	2,787	20.9	31.4	0.251	93.55
LightGBM	0.185	3,165	1,750	1,415	2,314	55.3	43.1	0.484	96.84
XGBoost (native categorical)	0.513	3,418	1,540	1,878	2,524	45.1	37.9	0.412	96.27
CatBoost	0.149	3,518	1,765	1,753	2,299	50.2	43.4	0.466	96.57
Autoencoder	2.56	6,532	1,209	5,323	2,855	18.5	29.7	0.228	93.08
LightGBM + AE feature	0.160	3,493	1,849	1,644	2,215	52.9	45.5	0.489	96.73
XGBoost (integer codes)	0.169	3,437	1,823	1,614	2,241	53.0	44.9	0.486	96.74
LightGBM (integer codes)	0.176	3,589	1,852	1,737	2,212	51.6	45.6	0.484	96.66

At the conventional threshold of 0.5 (Table 9), the boosting models were much more conservative. LightGBM, for example, produced only 1,501 alerts with 82.5% precision but detected only 30.5% of fraud. Figure 7 shows that moving to the validation-selected threshold raised recall by 12–15 percentage points for LightGBM, XGBoost with integer codes, CatBoost and the hybrid model, at the cost of roughly doubling the number of alerts and lowering precision to 50–55%. For native-categorical XGBoost, the selected threshold (0.513) was close to 0.5 and changed little. The choice of threshold is therefore a trade-off between missed fraud and review workload, not a free improvement.

Table 9. Test performance at probability threshold 0.5 (log-odds 0 for Naive Bayes). Precision, recall and accuracy are percentages.

Model	Alerts	TP	FP	Precision	Recall	F1	Accuracy
Naive Bayes	19,954	2,289	17,665	11.5	56.3	0.191	83.54
LightGBM	1,501	1,238	263	82.5	30.5	0.445	97.38
XGBoost (native categorical)	3,470	1,549	1,921	44.6	38.1	0.411	96.24
CatBoost	1,576	1,178	398	74.7	29.0	0.418	97.22
LightGBM + AE feature	1,510	1,244	266	82.4	30.6	0.446	97.39

Model	Alerts	TP	FP	Precision	Recall	F1	Accuracy
XGBoost (integer codes)	1,615	1,300	315	80.5	32.0	0.458	97.39
LightGBM (integer codes)	1,641	1,315	326	80.1	32.4	0.461	97.40

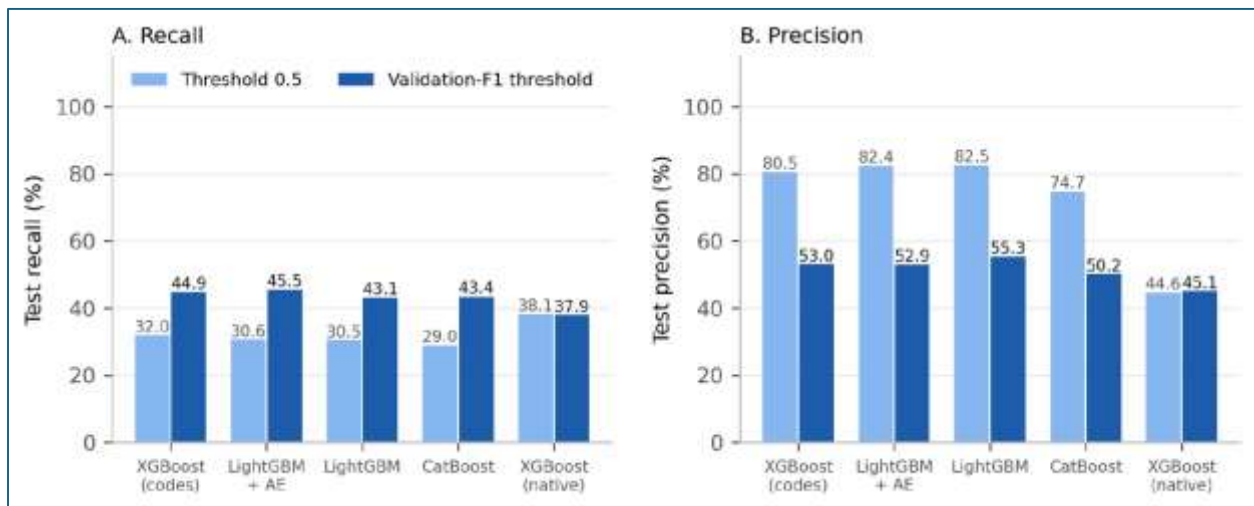


Figure 7. Test recall (A) and precision (B) of the boosting models at threshold 0.5 and at the threshold that maximized validation F1. Thresholds were fixed before being applied to the test set.

Figure 8 shows the corresponding confusion matrices. The boosting models correctly cleared about 98.5% of legitimate transactions, flagging roughly 1.2–1.6% of them, but still let 54–62% of fraudulent transactions through. Naive Bayes flagged 4.2% of legitimate transactions and caught only 31.4% of fraud.

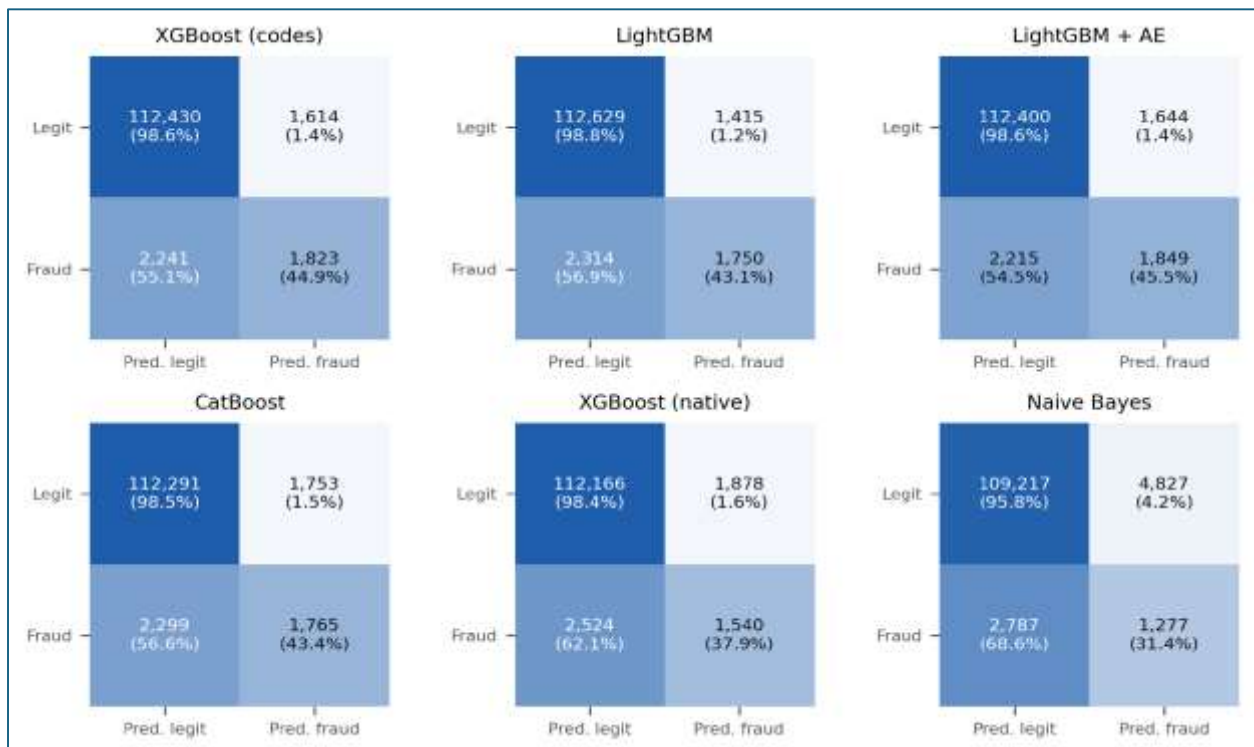


Figure 8. Test confusion matrices at validation-F1 thresholds. Each cell shows the count and, in parentheses, the percentage of its true class (row).

4.4 Autoencoder and hybrid model (RQ2)

The autoencoder learned a representation in which fraud tends to be harder to reconstruct: on the test set the median reconstruction error was 0.930 for fraud and 0.226 for legitimate transactions (Figure 9). However, the two distributions overlap heavily, and many fraudulent transactions were reconstructed as well as typical legitimate ones. As a stand-alone detector it reached a test ROC-AUC of 0.761 and AP of 0.111, below even Naive Bayes (AP 0.132) and 0.390 below LightGBM. At its validation-selected threshold it detected 29.7% of fraud at 18.5% precision.

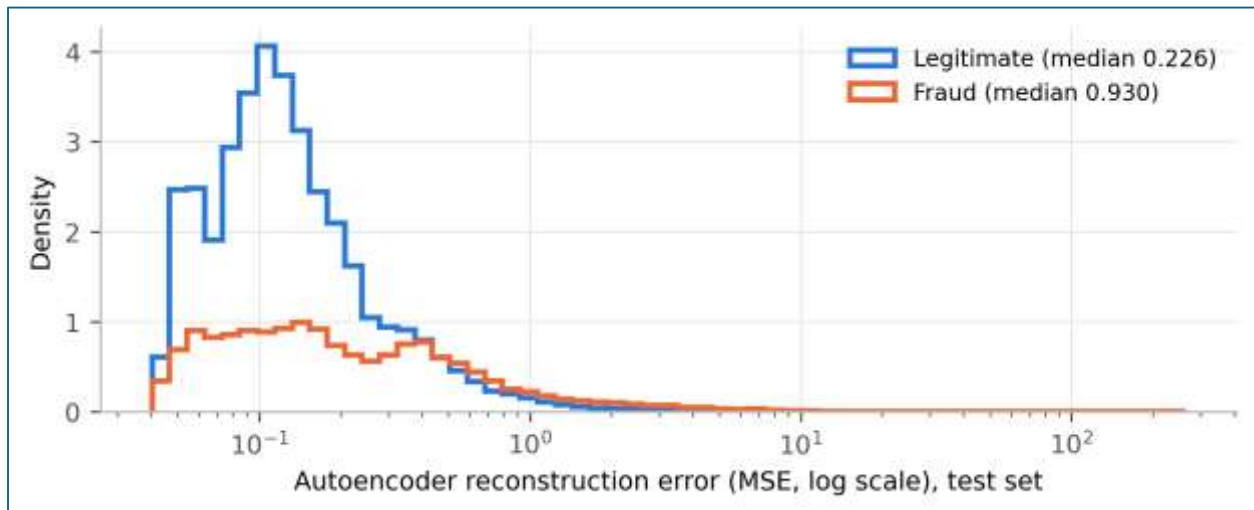


Figure 9. Distribution of autoencoder reconstruction error for legitimate and fraudulent test transactions (logarithmic horizontal axis). Fraud has a higher median error, but the distributions overlap substantially. When the reconstruction error was added to LightGBM, it became the single most-used feature, accounting for 9.4% of the total split gain. Yet the test AP changed only from 0.501 to 0.504 (difference +0.003, interval -0.002 to 0.009), and ROC-AUC from 0.889 to 0.890 (interval for the difference -0.001 to 0.003). Both intervals include zero. The feature appears to summarize information that LightGBM could already extract from the raw features, rather than adding new information.

The way the feature is constructed also matters. In an initial version of the hybrid, validation and test transactions received the average error of the two cross-fitted autoencoders while training transactions received the error of a single model. This small mismatch made the feature less noisy outside the training data than inside it. That version stopped after 135 iterations and reached a test AP of only 0.476, lower than LightGBM without the feature. Scoring every transaction with a single model, as described in Section 3.6, removed this problem. This shows that combining models requires care to keep features identically constructed across partitions.

4.5 Sensitivity to categorical encoding

Table 10 reports paired differences between models. The largest difference among the boosting pipelines was not between libraries but between two versions of the same library. XGBoost with integer codes exceeded XGBoost with native categorical handling by 0.181 in AP (interval 0.158 to 0.200). For LightGBM, the two encodings gave almost identical results (difference +0.002, interval -0.007 to 0.010). Native-categorical XGBoost already had a validation AP of only 0.443, so the problem was visible before the test period. Several text features have many categories (DeviceInfo had 1,446 distinct values in the training period, id_33 had 163 and id_31 had 101), and new categories continued to appear after training: 18,994 validation and test transactions had an id_31 (browser) value never seen in training, 1,337 had an

unseen DeviceInfo and 1,009 an unseen id_33. A plausible explanation, not tested directly here, is that XGBoost's partition-based splits on such high-cardinality features overfit the training period.

Table 10. Paired differences between models on the test set. Intervals are approximate 95% day-bootstrap intervals and are exploratory; an interval that excludes zero indicates separation under this resampling procedure.

Comparison	AP difference	AP interval	ROC-AUC difference	ROC-AUC interval
LightGBM – CatBoost	+0.041	0.029 to 0.053	+0.015	0.008 to 0.022
LightGBM – XGBoost (native)	+0.177	0.152 to 0.196	+0.017	0.012 to 0.022
CatBoost – XGBoost (native)	+0.135	0.110 to 0.156	+0.002	–0.005 to 0.009
LightGBM + AE feature – LightGBM	+0.003	–0.002 to 0.009	+0.001	–0.001 to 0.003
LightGBM – Autoencoder	+0.390	0.356 to 0.416	+0.127	0.114 to 0.142
LightGBM – Naive Bayes	+0.370	0.307 to 0.398	+0.098	0.086 to 0.112
XGBoost (codes) – XGBoost (native)	+0.181	0.158 to 0.200	+0.020	0.015 to 0.025
XGBoost (codes) – LightGBM	+0.004	–0.003 to 0.013	+0.003	–0.002 to 0.008
XGBoost (codes) – LightGBM + AE	+0.001	–0.006 to 0.009	+0.001	–0.004 to 0.006
LightGBM (native) – LightGBM (codes)	+0.002	–0.007 to 0.010	+0.000	–0.006 to 0.007

4.6 Feature importance

Figure 10 lists the 15 features with the largest share of LightGBM's total split gain. The Vesta-engineered feature V258 was the most important (10.5%), followed by the counting features C1 (7.2%) and C14 (5.0%) and the device description DeviceInfo (5.4%). Card identifiers (card1, card2), transaction amount, address and both e-mail domains also appear. Because the meanings of the C and V features are masked, these importances show which signals the model relies on rather than explaining why they indicate fraud. Split-gain importance can also favor features with many distinct values, so it should be read as descriptive.

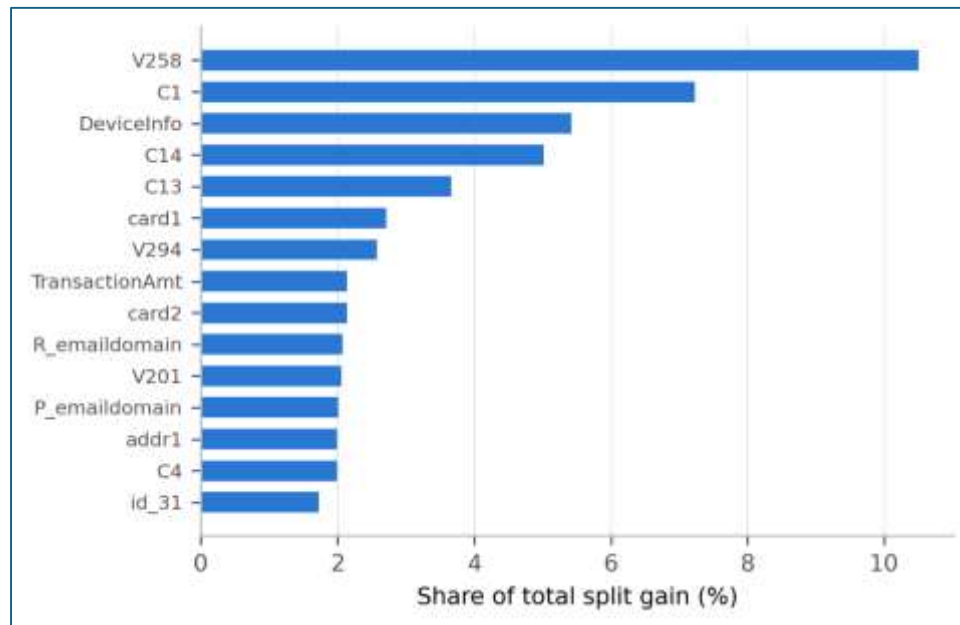


Figure 10. The 15 most important LightGBM features, measured as the share of total split gain.

5. Discussion

5.1 Accuracy hides missed fraud

The clearest finding concerns evaluation. A classifier that detects no fraud already exceeds 96% accuracy on the test period, and the boosting models reach 97.2–97.4% accuracy at the default threshold while detecting only about 30% of fraud. Reporting accuracy alone would therefore hide most of the detection problem. AP and precision–recall curves describe ranking quality across all thresholds, while confusion matrices at a stated threshold show the concrete result: how many fraud cases are caught, how many are missed and how many customers are incorrectly flagged. An earlier draft of this project reported accuracies of 85.7–92.3% and ROC-AUC values of 0.91–0.97 that could not be reproduced in the final analysis; the results reported here replace them.

5.2 Which model performed best?

Three configurations formed a top group with test AP between 0.501 and 0.506: LightGBM, LightGBM with the autoencoder feature and XGBoost with integer codes. Their differences were small and their paired intervals included zero, so the data do not identify a single best model among them. LightGBM was clearly better than CatBoost under these settings (AP difference 0.041, interval 0.029 to 0.053), but CatBoost used nearly all of its 800 iterations and its validation score was still improving, so a larger iteration budget or higher learning rate might narrow this gap. CatBoost was also by far the slowest model to train (23.5 min versus 40 s for LightGBM on the same hardware). These results do not contradict benchmark findings that CatBoost is often the strongest single model on tabular data [9]; they show that its advantage depends on the dataset and on the training budget.

The larger lesson is that pipeline details can matter more than the choice of library. Changing only how text features were passed to XGBoost moved its AP from 0.325 to 0.506, far more than the spread among the best models. A comparison of algorithms is therefore a comparison of complete pipelines, including encoding, missing-value handling and default regularization. Claims that one boosting library is superior for fraud detection should be tested under more than one reasonable configuration.

5.3 Why the autoencoder added little

The original motivation for this project was that an autoencoder could improve fraud detection. The results do not support that hypothesis for this dataset. The autoencoder is unsupervised: it learns what typical legitimate transactions look like but never sees a fraud label. Many fraudulent transactions in this dataset are not unusual overall; they differ from legitimate ones in specific combinations of features, such as a particular product code, card type and e-mail domain, which a supervised model can learn directly from 11,988 labeled training frauds. When the reconstruction error was given to LightGBM, the model used it heavily but did not become more accurate, suggesting that the error was largely redundant with patterns already available in the raw features. Autoencoders may be more useful when labels are scarce or delayed [5, 11], or for detecting new types of fraud that have no labeled examples. Those settings were not tested here.

5.4 Comparison with earlier studies

The best test ROC-AUC here (0.891) is lower than several values reported on the same dataset, such as 0.952 for XGBoost [13], and the best recall at a validation-F1 threshold (about 45–46%) is far below the 90.4% reported for an autoencoder–XGBoost combination [12]. These numbers should not be compared directly. The earlier studies used different splits, resampling, feature engineering and thresholds, and some did not evaluate on a later time period. Random splits place transactions from the same days and the same cards in both training and test sets, which generally makes prediction easier than predicting a later period. The competition's top solutions also relied on engineered identifiers that link transactions from the same client [15], which this study deliberately did not add. The chronological design used here is intended to approximate the more difficult task of predicting future transactions.

5.5 Limitations

This study used one public dataset, one chronological split, one training seed and fixed hyperparameters without tuning. The test period can share cards, devices and other entities with earlier periods, so observations are not independent. A chronological split approximates prediction on later transactions but does not reproduce delayed label availability, and it cannot confirm that every anonymized feature would have been available at the time of each decision. The day-level bootstrap captures variation between days but not longer-range dependence or variation from retraining. Preprocessing was deliberately simple: numerical-coded identifiers were treated as numbers, missing values were imputed with medians even though the boosting libraries can handle missing values directly, and no missingness indicators or client-level aggregates were created. Naive Bayes assumes conditional independence and Gaussian numerical features, which are unrealistic here. The autoencoder used one architecture, one input representation and no hyperparameter search. Validation F1 is a convenient threshold criterion but not a financial utility function; choosing an operating point in practice would require investigation costs, customer friction and recoverable losses, none of which are available in this dataset. For these reasons, the results describe the specified pipelines on this dataset and period, not the best achievable performance of any model family, and they do not support claims about monetary savings or real-time deployment.

5.6 Future work

Several extensions follow directly from these results. Rolling time windows and multiple seeds would show how stable the model ranking is. Hyperparameters, including CatBoost's iteration budget, should be tuned within the training and validation periods and checked on a new final holdout. Ablation studies of categorical encoding, missing-value handling and sparse-feature removal would clarify why XGBoost's native categorical mode performed poorly. Client-level features built from card, address and e-mail

combinations, and sequence models such as recurrent neural networks that read each card's transaction history, could capture behavior that single-transaction models miss. Finally, calibration, alert budgets, delayed labels and evaluation on other datasets would be needed before making deployment recommendations.

6. Conclusion

On a chronological holdout of the IEEE-CIS dataset, LightGBM, LightGBM with an autoencoder feature and XGBoost with integer-coded categories performed best, with test AP of about 0.50 and ROC-AUC of about 0.89, and could not be separated statistically. At validation-selected thresholds these models detected 43–46% of later fraud with 53–55% precision. CatBoost was weaker under a fixed 800-iteration budget, and XGBoost with native categorical handling was much weaker, showing that encoding choices can outweigh the choice of library. An autoencoder trained on legitimate transactions performed poorly on its own and did not measurably improve LightGBM. Across all models, accuracy exceeded 96% even when most fraud was missed, and performance on the later test period was lower than on validation. Fraud detection studies should therefore report precision–recall metrics, state their thresholds and evaluate on later data before drawing conclusions about which algorithm is best.

Acknowledgments

The author thanks Jesus Valdiviezo for mentorship throughout this project through the Polygence research program, including guidance on dataset selection, study design and manuscript structure.

Use of Generative AI

Generative AI tools assisted substantially with this work. OpenAI ChatGPT assisted with grammar, the outline of earlier drafts and an earlier version of the analysis design. Anthropic Claude re-implemented and executed the complete analysis reported here from the raw data (including the autoencoder, hybrid model and sensitivity analyses), produced the figures, checked references and drafted the manuscript text. The author reviewed the code, results, interpretation and references and is responsible for the final content.

Data and Code Availability

The IEEE-CIS Fraud Detection data are available from the Kaggle competition page [18] under its access terms and are not redistributed. The analysis scripts (data preparation, models, autoencoder, sensitivity analyses, evaluation and figures), the summary results and a description of the protocol are provided as supplementary material. Exploratory work from the earlier stage of the project is available in the author's Kaggle notebook (kaggle.com/code/arnavt1/notebook-frauddet-arnav-polygence). Public Kaggle notebooks by other authors informed early exploration [15, 23, 24].

References

1. Nilson Report. (2022, December 22). Payment card fraud losses reach \$32.34 billion [Press release]. GlobeNewswire. <https://www.globenewswire.com/news-release/2022/12/22/2578877/0/en/Payment-Card-Fraud-Losses-Reach-32-34-Billion.html>
2. Nilson Report. (n.d.). Card fraud losses worldwide in 2022. Retrieved September 25, 2026, from <https://nilsonreport.com/articles/card-fraud-losses-worldwide-2/>
3. Cherif, A., Badhib, A., Ammar, H., Alshehri, S., Kalkatawi, M., & Imine, A. (2023). Credit card fraud detection in the era of disruptive technologies: A systematic review. *Journal of King Saud University – Computer and Information Sciences*, 35(1), 145–174. <https://doi.org/10.1016/j.jksuci.2022.11.008>

4. Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3), e0118432. <https://doi.org/10.1371/journal.pone.0118432>
5. Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., & Bontempi, G. (2018). Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8), 3784–3797. <https://doi.org/10.1109/TNNLS.2017.2736643>
6. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). <https://doi.org/10.1145/2939672.2939785>
7. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30*.
8. rokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems 31*. <https://arxiv.org/abs/1706.09516>
9. Shmuel, A., Glickman, O., & Lazebnik, T. (2024). A comprehensive benchmark of machine and deep learning across diverse tabular datasets. arXiv:2408.14817. <https://arxiv.org/abs/2408.14817>
10. Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems 35* (Datasets and Benchmarks Track).
11. Pumsirirat, A., & Yan, L. (2018). Credit card fraud detection using deep learning based on auto-encoder and restricted Boltzmann machine. *International Journal of Advanced Computer Science and Applications*, 9(1). <https://doi.org/10.14569/IJACSA.2018.090103>
12. Prabha, D. P., & Priscilla, C. V. (2023). Probabilistic XGBoost threshold classification with autoencoder for credit card fraud detection. *International Journal on Recent and Innovation Trends in Computing and Communication*, 11(8s), 528–537. <https://doi.org/10.17762/ijritcc.v11i8s.7234>
13. Zhang, Y., Wang, Z., Tong, J., & Gao, F. (2020). Customer transaction fraud detection using XGBoost model. In *2020 International Conference on Computer Engineering and Application (ICCEA)* (pp. 554–558). IEEE. <https://doi.org/10.1109/ICCEA50009.2020.00122>
14. Kewei, X., Peng, B., Jiang, Y., & Lu, T. (2021). A hybrid deep learning model for online fraud detection. In *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*. <https://doi.org/10.1109/ICCECE51280.2021.9342110>
15. Deotte, C. (2019). XGB fraud with magic – [0.9600] [Kaggle notebook]. <https://www.kaggle.com/code/cdeotte/xgb-fraud-with-magic-0-9600>
16. Dal Pozzolo, A., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence* (pp. 159–166). <https://doi.org/10.1109/SSCI.2015.33>
17. Lopez-Rojas, E. A., Elmir, A., & Axelsson, S. (2016). PaySim: A financial mobile money simulator for fraud detection. In *Proceedings of the 28th European Modeling and Simulation Symposium (EMSS)* (pp. 249–255).
18. IEEE Computational Intelligence Society & Vesta Corporation. (2019). *IEEE-CIS Fraud Detection* [Data set and competition]. Kaggle. <https://www.kaggle.com/competitions/ieee-fraud-detection/data>

19. Scikit-learn developers. (n.d.). Common pitfalls and recommended practices: Data leakage. Retrieved September 25, 2026, from https://scikit-learn.org/stable/common_pitfalls.html
20. Scikit-learn developers. (n.d.). Naive Bayes. Retrieved September 25, 2026, from https://scikit-learn.org/stable/modules/naive_bayes.html
21. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035).
22. Scikit-learn developers. (n.d.). `sklearn.metrics.average_precision_score`. Retrieved September 25, 2026, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html
23. kirankunapuli. (n.d.). IEEE fraud: LightGBM with GPU [Kaggle notebook]. Retrieved September 25, 2026, from <https://www.kaggle.com/code/kirankunapuli/ieee-fraud-lightgbm-with-gpu>
24. plasticgrammer. (n.d.). IEEE-CIS fraud detection EDA [Kaggle notebook]. Retrieved September 25, 2026, from <https://www.kaggle.com/code/plasticgrammer/ieee-cis-fraud-detection-eda>