Eurasia Journal of Science and Technology
Print ISSN: 2663-1024
Online ISSN: 2663-1016
DOI: https://doi.org/10.61784/ejst4160

AN AI-DRIVEN INTELLIGENT TRANSACTION EXECUTION FRAMEWORK FOR INTERNET FINANCE PLATFORMS: COST-SENSITIVE REAL-TIME INTEGRITY INTERDICTION

YuXuan Qin[1*], JiTong Zou[2], Minjae Rhee[3]
[1]*University College London, Gower St, London WC1E 6BT, United Kingdom.*
[2]*Case Western Reserve University, 10900 Euclid Avenue, Cleveland, OH 44106, USA.*
[3]*University of Illinois Urbana-Champaign, 601 E John Street, Champaign, IL 61820, USA.*
[*]*Corresponding Author: YuXuan Qin*

Abstract: Internet finance platforms must decide, in real time and for every incoming transaction, whether to approve it, subject it to additional verification, or decline it. Most machine-learning research on transaction Integrity stops at producing a Integrity score and is judged with threshold-independent ranking metrics, leaving a gap between the score and the concrete execution action, and ignoring that different transactions carry very different monetary stakes. This paper presents IntelliExec, an AI-driven intelligent transaction execution framework that closes this gap. A class-weighted gradient-boosting model produces a Integrity probability that is calibrated with Platt scaling so it can be treated as a true probability; the calibrated probability and the transaction amount are then mapped to one of three execution actions—approve, step-up (one-time-password / 3-D Secure challenge), or decline—by minimising an example-dependent expected cost. Because the decision is derived analytically from a cost model, it needs no threshold tuning and adapts the operating point to each transaction's amount. On a public benchmark of 284,807 real card transactions (0.172% Integrity) evaluated under a strict temporal split, the framework lowers total expected cost by 71.8% relative to approving every transaction and by 45.4% relative to the best cost-aware global-threshold baseline, while hard-declining only three legitimate customers (versus thirty-seven for a binary cost-sensitive rule) by routing borderline cases to the low-friction step-up tier. The gradient-boosting scorer runs in under one millisecond per transaction, and a sensitivity analysis confirms the cost advantage is robust to the choice of cost parameters.
Keywords: Transaction integrity detection; Cost-sensitive learning; Gradient boosting; Probability calibration; Imbalanced classification; Real-time decision-making; Internet finance

1 INTRODUCTION

The rapid expansion of internet finance—mobile wallets, online lending, and card-not-present payments—has moved an enormous share of everyday financial activity onto digital platforms that must authorise transactions instantly. This scale and immediacy make such platforms an attractive target for Integrity, which costs the financial industry billions of dollars every year and erodes customer trust [1-3]. Every incoming transaction therefore triggers a real-time execution decision: the platform must approve it, subject it to additional verification, or decline it, within tens of milliseconds and without a human in the loop.

A large body of machine-learning research addresses this problem as a supervised classification task and reports strong results with ranking metrics such as the area under the ROC or precision–recall curve [4-6]. Yet three gaps stand between these models and a working execution system. First, ranking metrics do not reflect the business action that is ultimately taken or the money that is actually lost; a model with a high AUC can still be operated at a poor point. Second, deployed systems typically apply a single global probability threshold, treating a two-dollar and a two-thousand-dollar transaction identically, even though the financial stakes differ by three orders of magnitude. Third, the raw scores produced by many classifiers—especially those trained with class weighting or resampling to counter imbalance—are not calibrated probabilities, so they cannot be plugged directly into a cost calculation.

We argue that the execution decision should be made by minimising expected financial cost on a per-transaction basis, using a calibrated probability together with the transaction amount. We present IntelliExec, a modular framework that (i) produces a real-time Integrity probability with a class-weighted gradient-boosting model, (ii) calibrates that probability with Platt scaling, and (iii) maps the calibrated probability and the amount to one of three execution actions by minimising an example-dependent cost. The third action—a step-up challenge such as a one-time password (OTP) or a 3-D Secure prompt—models a mechanism that real platforms already use: it imposes little friction on genuine customers, who simply complete the challenge, while blocking the large majority of Integritysters, who cannot [3]. Adding this low-friction tier lets the system protect revenue without hard-declining borderline customers.

The main contributions of this paper are:

1) A real-time transaction-execution framework that explicitly connects a calibrated Integrity score to a concrete execution action, rather than stopping at a score.

© By the Author(s) 2022, under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0).

2) An example-dependent, amount-aware cost model and a closed-form three-way (approve / step-up / decline) execution policy that minimises expected cost and, crucially, requires no threshold tuning—the operating point adapts automatically to each transaction's amount.
3) A rigorous evaluation on a public benchmark of real transactions under a strict temporal split, showing large expected-cost reductions, a drastic drop in false declines, sub-millisecond scoring latency, robustness to the cost parameters, and model interpretability, together with an honestly reported negative ablation.
The remainder of the paper is organised as follows. Section 2 reviews related work. Section 3 formalises the problem and describes the dataset and evaluation protocol. Section 4 presents the framework and the cost-sensitive policy. Section 5 reports experiments, and Section 6 discusses limitations. Section 7 concludes.

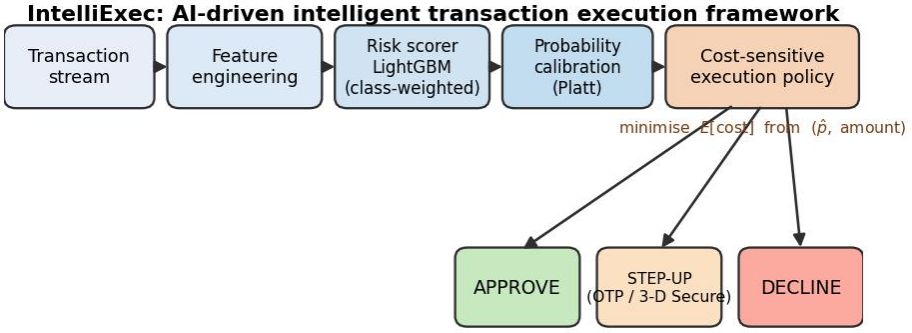


Figure 1 The IntelliExec Execution Framework
Note: A calibrated Integrity probability and the transaction amount are mapped to one of three execution actions by minimising expected cost.

2 RELATED WORK

2.1 Machine Learning for Transaction Integrity Detection

Surveys of financial-Integrity detection catalogue the standard toolkit—logistic regression, decision trees, random forests, support-vector machines, neural networks, and ensembles—and stress that class imbalance and data quality, rather than model choice alone, dominate performance [1-2, 7]. Comparative studies on real card data confirm that ensemble methods such as random forests are strong baselines [4, 8]. Dal Pozzolo et al. emphasise that realistic Integrity detection is a non-stationary streaming problem with concept drift and delayed labels, and propose learning strategies tailored to these conditions [5, 9]. More expressive models have followed: sequence classifiers that model a card's transaction history, generative adversarial networks that augment the minority class, and graph neural networks that expose coordinated Integrity through relational structure while resisting Integrityster camouflage [10-12]. Ensemble and boosting approaches remain competitive and efficient [13-14]. These works optimise detection or ranking; they do not model the execution action or its monetary consequences, which is the focus of the present paper.

2.2 Learning from Imbalanced Data

Integrity is extremely rare, so imbalance-handling is central. SMOTE and its variants synthesise minority examples [15]; broad reviews compare resampling, cost-sensitive, and algorithmic approaches [16]. Resampling, however, distorts the base rate and therefore the output probabilities, which is problematic when those probabilities feed a cost calculation. We instead use class weighting, which preserves the data distribution, and restore probability quality with an explicit calibration step.

2.3 Cost-Sensitive Learning

Elkan established the theory of cost-sensitive classification and showed that the optimal decision threshold is a function of the misclassification costs [17]. Bahnsen et al. extended this to example-dependent costs—where each transaction carries its own cost, driven by its amount—through cost-sensitive decision trees and cost-aware feature engineering and evaluation [18-19]. Transaction-aggregation features further encode short-term spending behaviour [20]. These methods define binary cost-sensitive classifiers. We build on the example-dependent view but extend the decision from a binary accept/reject to a three-action execution policy that adds a low-friction step-up tier, and we integrate probability calibration as a prerequisite.

2.4 Probability Calibration

Zadrozny and Elkan describe transforming classifier scores into accurate probabilities [21]; Niculescu-Mizil and Caruana compare calibration methods across learners [22]. In the Integrity setting specifically, Bahnsen et al. show that

calibrated probabilities improve cost-based Integrity decisions [23]. We adopt Platt scaling because it is monotone and therefore preserves the ranking quality of the scorer while improving calibration, both of which our policy relies on.

2.5 Real-Time and Streaming Systems

At the systems level, SCARFF integrates big-data tooling for scalable streaming Integrity detection [24], and Carcillo et al. combine unsupervised anomaly scores with supervised learning [25]. Our contribution is complementary: rather than the ingestion pipeline, we target the decision layer that turns a score into an action, and we report per-transaction latency to confirm the approach is compatible with real-time execution.

3 PROBLEM FORMULATION AND DATASET

3.1 The Transaction-Execution Decision

For each incoming transaction x with monetary amount a, the platform must choose an action k from the set {approve, step-up, decline}. Let y = 1 denote Integrity and y = 0 a legitimate transaction, and let each (action, outcome) pair incur a cost that combines Integrity losses, operational overhead, and customer-friction. The objective is to choose, for every transaction and in real time, the action that minimises expected cost over the transaction stream. Because the true label is unknown at decision time, the framework estimates the Integrity probability and takes the expectation over it.

3.2 Dataset

We use a widely adopted public benchmark of real credit-card transactions made by European cardholders over two days, comprising 284,807 transactions of which 492 (0.172%) are Integrityulent—an imbalance ratio of roughly 1:578. To protect confidentiality, twenty-eight of the features (V1–V28) are the principal components of the original attributes; only the elapsed time (Time) and the transaction amount (Amount) are given in the clear, along with the binary label. The amount is essential to our study because it defines the example-dependent cost of a missed Integrity. The data contain no missing values. We note two limitations that we return to in Section 6: the PCA anonymisation precludes domain-specific feature engineering, and the two-day window is short; nevertheless the dataset reflects genuine Integrity with a realistic base rate and non-stationarity, and is a standard reference for this task [5, 9].

3.3 Temporal Evaluation Protocol

A deployed system trains on past transactions and scores future ones, so a uniformly random train/test split leaks future information and inflates results. We instead sort transactions by time and split them chronologically into 60% for training, the next 20% for calibration and baseline-threshold tuning, and the final 20% for testing—170,884, 56,961, and 56,962 transactions respectively. The Integrity prevalence falls from 0.211% in the training period to 0.100% and 0.132% in the calibration and test periods, an example of the non-stationarity that motivates a realistic protocol. All model selection, calibration, and baseline tuning use only the training and calibration periods; the test period is untouched until final evaluation.

3.4 Feature Engineering

We apply a log transform, $\log(1 + a)$, to the heavy-tailed amount, and derive a cyclical time-of-day representation—the sine and cosine of the hour extracted from the Time field—so that 23:00 and 00:00 are adjacent. Together with V1–V28 this yields 31 features. No resampling is applied; imbalance is handled through class weighting inside the learner.

4 THE INTELLIEXEC FRAMEWORK

4.1 Overview

Figure 1 shows the pipeline. An incoming transaction is turned into features, scored by a risk model, and the score is calibrated into a probability; the probability and the amount then drive a cost-sensitive policy that emits one of the three execution actions. The design is modular: the scorer and the cost model are independent components that can be replaced without changing the rest of the system.

4.2 Risk-Scoring Model

We compare logistic regression, a random forest, XGBoost, and LightGBM (Section 5.2) and deploy LightGBM, a histogram-based gradient-boosting method, for its balance of ranking accuracy, calibration, and inference speed [14]. Imbalance is addressed with balanced class weights rather than resampling, so the empirical base rate—and hence the meaning of the output score—is preserved. The model uses 500 trees with 64 leaves, a learning rate of 0.05, and column and row subsampling of 0.9.

4.3 Probability Calibration

A class-weighted booster ranks transactions well but its raw scores are not probabilities: weighting inflates the apparent Integrity likelihood. Since the execution policy computes expected costs, it needs true probabilities. We therefore fit Platt scaling—a one-dimensional logistic map from the score to a probability—on the calibration period. Platt scaling is monotone, so it leaves the ranking (and thus ROC-AUC and PR-AUC) unchanged while markedly improving calibration, as confirmed in Section 5.3.

4.4 Example-Dependent Cost Model

Let $\hat{p}$ be the calibrated Integrity probability of a transaction with amount a. We assign each action an expected cost. Approving a Integrityulent transaction loses its amount, while approving a legitimate one costs nothing, giving (1). Declining a Integrityulent transaction incurs only a small administrative cost C_d, whereas declining a legitimate one incurs a false-decline cost C_fp that captures customer friction and lost revenue, giving (2). A step-up challenge imposes a friction cost C_su; a legitimate customer completes it with probability π_L (otherwise abandoning, at cost C_fp), while a Integrityster completes it with a small probability π_F (then causing loss a, otherwise being blocked), giving (3).

$$E[\text{approve}] = \hat{p} \cdot a \quad (1)$$

$$E[\text{decline}] = \hat{p} \cdot C_d + (1 - \hat{p}) \cdot C_fp \quad (2)$$

$$E[\text{step-up}] = C_su + \hat{p} \cdot \pi_F \cdot a + (1 - \hat{p})(1 - \pi_L) \cdot C_fp \quad (3)$$

The step-up term formalises the intuition that a challenge is cheap for genuine customers yet blocks most Integrity (π_F is small), making it attractive for medium-risk transactions where an outright decline would needlessly reject many legitimate users.

4.5 Cost-Optimal Execution Policy

The policy selects, for every transaction, the action of minimum expected cost, (4). It has no free threshold: the operating point follows directly from the cost model and the calibrated probability.

$$\text{action}^* = \arg\min_\{k \in \{\text{approve, step-up, decline}\}\} \; E[k] \quad (4)$$

Because E[approve] grows linearly with the amount a while E[decline] does not, the decision boundary is amount-dependent. For a fixed probability, larger amounts push the decision from approve toward step-up and then decline; equivalently, the effective probability threshold for blocking falls as the amount rises. Figure 2 visualises the resulting regions. This amount-awareness is the key departure from a single global threshold and is what allows the policy to tolerate small-value risk while aggressively protecting high-value transactions.

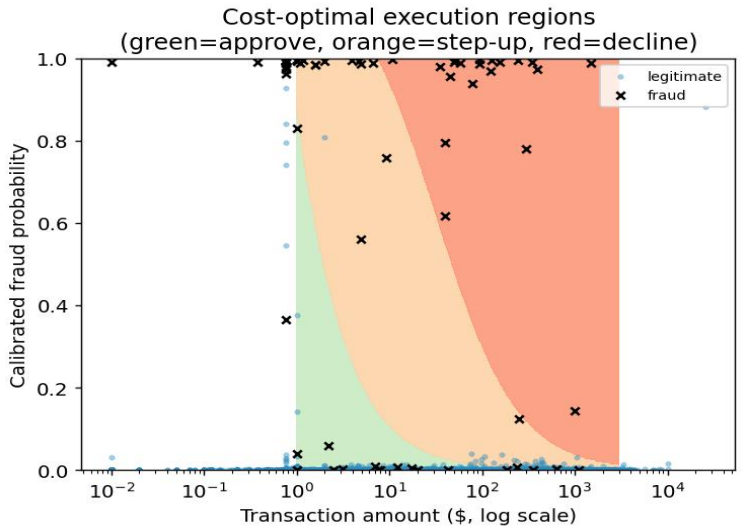


Figure 2 Cost-Optimal Execution Regions in the Plane of Transaction Amount (x-axis) and Calibrated Integrity Probability (y-axis): Approve (Green), Step-Up (Orange), and Decline (Red)

Note: Markers show test transactions (legitimate and Integrityulent). The blocking boundary moves to lower probabilities as the amount grows.

5 EXPERIMENTS

5.1 Setup

All experiments use the temporal split of Section 3.3 and run on a single CPU core with scikit-learn, XGBoost, and LightGBM. We report ROC-AUC, PR-AUC (average precision), and the Brier score for ranking and calibration quality,

and, for the execution decision, the total expected cost together with the number of false declines. Unless stated otherwise the cost parameters are C_fp = 5.0, C_d = 1.5, C_su = 0.75, π_L = 0.95, and π_F = 0.10; these are illustrative values that we vary in Section 5.5. Costs are in the dataset's currency units.

5.2 Risk-Scoring Comparison

Table 1 compares the four scorers on the test period. All exceed 0.977 ROC-AUC. The random forest attains the highest PR-AUC (0.803), with LightGBM close behind (0.800) and XGBoost at 0.792; logistic regression ranks transactions competitively (0.982 ROC-AUC) but has a poor Brier score, reflecting badly scaled scores. Given that LightGBM matches the best PR-AUC to within noise while scoring a transaction more than an order of magnitude faster than the random forest (Section 5.6), we deploy LightGBM. Figure 3 shows the ROC and precision–recall curves.

Table 1 Risk-Scoring Model Comparison on the Test Period

Model	ROC-AUC	PR-AUC	Brier	Train (s)	Latency (ms)
Logistic regression	0.982	0.748	0.0386	2.0	0.09
Random forest	0.985	0.803	0.00055	123.6	11.71
XGBoost	0.977	0.792	0.00049	9.6	0.57
LightGBM (deployed)	0.980	0.800	0.00046	15.8	0.79

Note: Latency is mean per-transaction inference time on one CPU core. Best PR-AUC is the random forest; LightGBM is deployed for its accuracy–latency trade-off.

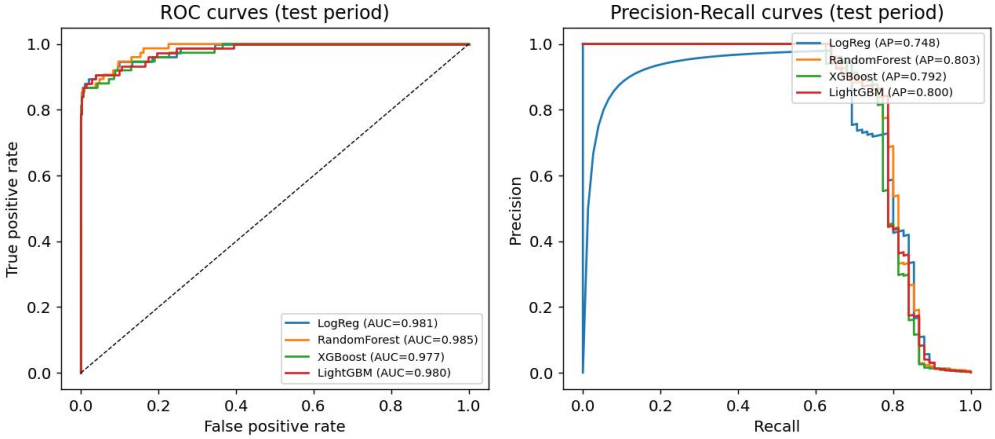


Figure 3 ROC (Left) and Precision–Recall (Right) Curves on the Held-Out Test Period for the Four Risk-Scoring Models

5.3 Probability Calibration

Platt scaling lowers the Brier score of the LightGBM scorer from 0.000459 to 0.000437 while leaving PR-AUC unchanged at 0.800, exactly as expected for a monotone map. Isotonic regression reaches a marginally lower Brier (0.000427) but, by tying many scores together, drops PR-AUC to 0.770; we therefore prefer Platt scaling, which improves probability quality at no cost to ranking. Figure 4 shows the reliability diagram before and after calibration.

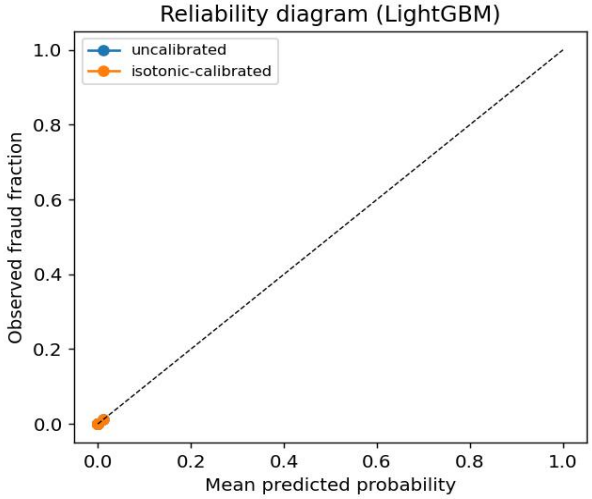


Figure 4 Reliability Diagram for the LightGBM Scorer Before and After Platt Calibration on the Test Period

5.4 Execution-Policy Comparison

Table 2 reports the core result. Approving every transaction costs 7,729 (the total Integrityulent amount in the test period). The three global-threshold baselines—the default 0.5 threshold, the F1-optimal threshold (0.416), and the cost-optimal global threshold (0.451), the latter two tuned on the calibration period—all map to the same test decisions and cost 3,999, a 48.3% reduction. Our example-dependent binary policy lowers cost to 2,890 (a 62.6% reduction, and 27.7% below the best global-threshold baseline). The full three-way policy reaches 2,182: a 71.8% reduction versus approving all, and 45.4% below the best baseline. Figure 5 summarises these costs.

Two findings stand out. First, the three-way policy prevents 5,835 of Integrity value (75.5% of the total) while hard-declining only three legitimate customers and routing 247 transactions—228 legitimate and 19 Integrityulent—to a step-up challenge. Compared with the binary cost policy, which hard-declines thirty-seven legitimate customers, the step-up tier cuts false declines more than tenfold and still lowers total cost. Second, because the policy minimises money rather than counts, its Integrity count-recall (0.52) is lower than the global threshold's (0.72), yet it prevents more Integrity value and costs less: it rationally lets small-value Integritys through when the expected loss is smaller than the cost of declining a legitimate customer, and concentrates interdiction on high-value risk. This is precisely the behaviour that a cost-agnostic metric would penalise but a platform's finances reward.

Table 2 Execution-Policy Comparison on the Test Period (Base Cost Parameters)

Execution policy	Total cost	Legit. declined	Step-ups	Integrity $ prevented	Cost saving
Approve all (no model)	7,729	0	0	0	—
Global threshold (0.5 / F1 / cost-opt)	3,999	7	0	3,846	48.3%
Example-dependent cost, binary	2,890	37	0	5,067	62.6%
Cost-sensitive three-way (proposed)	2,182	3	247	5,835	71.8%

Note: Cost saving is relative to approving all transactions. The three-way policy also lowers cost by 45.4% relative to the best global-threshold baseline.

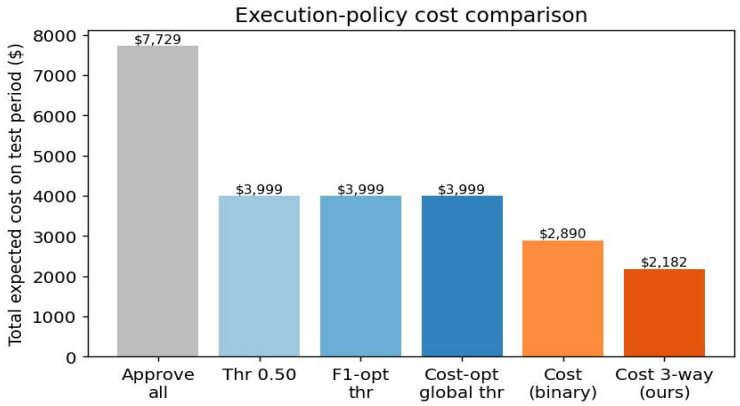


Figure 5 Total Expected Cost of Each Execution Policy on the Test Period (Lower Is Better)

5.5 Sensitivity Analysis

The cost parameters are platform-specific, so we test robustness. Table 3 varies the false-decline cost C_fp from 1 to 20. The three-way policy beats the global-threshold baseline for every value and is the overall cheapest across the realistic mid-range C_fp ∈ 2–12; only at the extremes (C_fp = 1, where declines are almost free, and C_fp = 20, where they are very costly) does the binary variant become marginally cheaper. As C_fp rises the policy shifts work from hard declines to step-ups. Varying the Integrityster challenge pass-rate π_F from 0.02 to 0.30 changes the three-way cost only from 2,124 to 2,460, while the number of step-ups falls from 283 to 168—when challenges leak more Integrity, the policy issues fewer of them. The qualitative advantage is therefore stable across a wide range of assumptions.

Table 3 Sensitivity of Total Cost to the False-Decline Cost C_fp

C_fp	Global threshold	Cost binary	Cost three-way	Step-ups (three-way)
1	2,683	2,098	2,176	208
2	2,910	2,922	2,170	292
3	3,137	2,885	2,164	278
5	3,999	2,890	2,182	247
8	4,020	2,873	2,184	216
12	4,048	2,825	2,391	176
20	4,104	2,825	3,045	133

Note: Lowest cost in each row in bold context: the three-way policy is cheapest for C_fp ∈ 2–12 and always beats the global-threshold baseline.

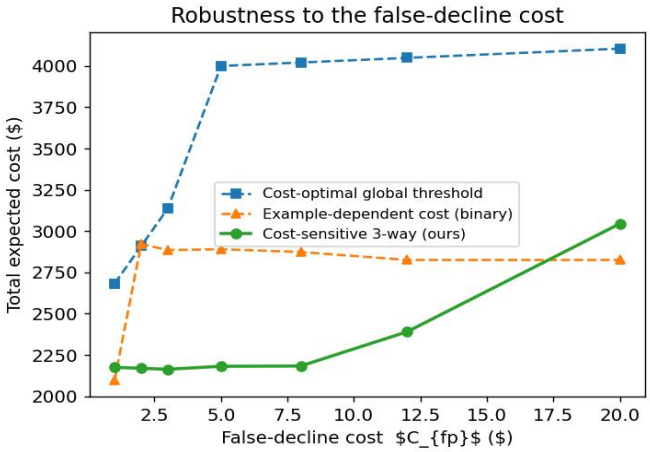


Figure 6 Total Expected Cost Versus the False-Decline Cost C_fp for the Three Policies

5.6 Inference Latenc

Real-time execution demands low latency. On a single CPU core the deployed LightGBM scorer takes 0.79 ms per transaction on average and 1.58 ms at the 99th percentile; XGBoost is faster still at 0.57 ms, while the random forest needs 11.7 ms (Table 1). The calibration and policy layer is a handful of arithmetic operations and is negligible. Sub-millisecond scoring with a gradient-boosting model leaves ample headroom within a typical authorisation budget.

5.7 Interpretability

Figure 7 ranks features by mean absolute SHAP value for the LightGBM scorer. The latent component V14 dominates, followed by the engineered time-of-day feature and the components V4, V8, and V1. The prominence of the time feature confirms that when a transaction occurs carries genuine signal, validating the feature engineering of Section 3.4, while the concentration of importance in a few components is consistent with Integrity occupying specific regions of the anonymised feature space.

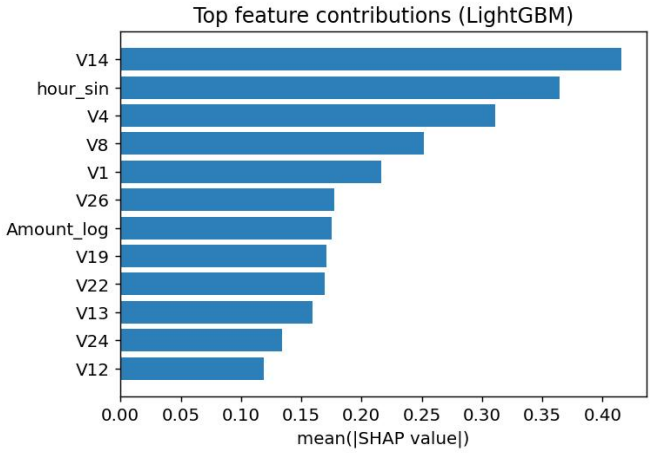


Figure 7 Top Feature Contributions to the LightGBM Scorer, by Mean Absolute SHAP Value

5.8 Ablation: Unsupervised Fusion

Motivated by hybrid detectors, we added an Isolation-Forest anomaly score, learned from legitimate transactions, as an extra feature [25]. On this dataset it did not help: PR-AUC moved from 0.800 to 0.796 and ROC-AUC from 0.980 to 0.975. We interpret this as the class-weighted booster already capturing the relevant anomaly structure, and we report the negative result for completeness rather than omitting it.

6 DISCUSSION AND LIMITATIONS

The cost parameters are illustrative and must be estimated from a platform's own history; our sensitivity analysis shows the qualitative advantage is robust, but absolute savings will differ by platform. Step-up outcomes are modelled in expectation through the pass-rates π_L and π_F, which real deployments should measure directly, ideally alongside customer-experience metrics such as challenge-abandonment. The benchmark's PCA anonymisation prevents domain feature engineering, and the temporal split leaves only 75 Integritys in the test period, so absolute figures carry variance;

we therefore emphasise relative comparisons under an identical protocol. The study covers a single card channel over two days—generalising to other internet-finance products such as wallets and lending, and to longer horizons with pronounced concept drift, is important future work. Finally, because Integritysters adapt and score distributions drift, a production system should recalibrate online and monitor for distribution shift; graph- and sequence-based risk features and online cost estimation are natural extensions of the framework [10, 12].

7 CONCLUSION

We presented IntelliExec, an AI-driven intelligent transaction-execution framework for internet finance platforms. Rather than stopping at a Integrity score, it calibrates that score into a probability and, together with the transaction amount, maps it to a concrete, amount-aware, cost-optimal execution action—approve, step-up, or decline—by minimising an example-dependent expected cost, without any threshold tuning. On a public benchmark of real transactions evaluated under a strict temporal split, the framework reduced total expected cost by 71.8% relative to approving every transaction and by 45.4% relative to the best cost-aware global-threshold baseline, while cutting false declines more than tenfold through a low-friction step-up tier, and it scores transactions in under a millisecond. Future work will estimate costs and pass-rates online, add graph and sequence risk features, handle concept drift, and validate the framework in a live A/B setting.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Ngai E W T, Hu Y, Wong Y H, et al. The application of data mining techniques in financial Integrity detection: A classification framework and an academic review of literature. Decision Support Systems, 2011, 50(3): 559–569.
[2] West J, Bhattacharya M. Intelligent financial Integrity detection: A comprehensive review. Computers & Security, 2016, 57: 47–66.
[3] Abdallah A, Maarof M A, Zainal A. Integrity detection system: A survey. Journal of Network and Computer Applications, 2016, 68: 90–113.
[4] Bhattacharyya S, Jha S, Tharakunnel K, et al. Data mining for credit card Integrity: A comparative study. Decision Support Systems, 2011, 50(3): 602–613.
[5] Dal Pozzolo A, Boracchi G, Caelen O, et al. Credit card Integrity detection: A realistic modeling and a novel learning strategy. IEEE Transactions on Neural Networks and Learning Systems, 2018, 29(8): 3784–3797.
[6] Randhawa K, Loo C K, Seera M, et al. Credit card Integrity detection using AdaBoost and majority voting. IEEE Access, 2018, 6: 14277–14284.
[7] Dal Pozzolo A, Caelen O, Le Borgne Y A, et al. Learned lessons in credit card Integrity detection from a practitioner perspective. Expert Systems with Applications, 2014, 41(10): 4915–4928.
[8] Xuan S, Liu G, Li Z, et al. Random forest for credit card Integrity detection. Proceedings of the IEEE 15th International Conference on Networking, Sensing and Control (ICNSC), 2018: 1–6.
[9] Dal Pozzolo A, Caelen O, Johnson R A, et al. Calibrating probability with undersampling for unbalanced classification. Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), 2015: 159–166.
[10] Jurgovsky J, Granitzer M, Ziegler K, et al. Sequence classification for credit-card Integrity detection. Expert Systems with Applications, 2018, 100: 234–245.
[11] Fiore U, De Santis A, Perla F, et al. Using generative adversarial networks for improving classification effectiveness in credit card Integrity detection. Information Sciences, 2019, 479: 448–455.
[12] Dou Y, Liu Z, Sun L, et al. Enhancing graph neural network-based Integrity detectors against camouflaged Integritysters. Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM), 2020: 315–324.
[13] Chen T, Guestrin C. XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016: 785–794.
[14] Ke G, Meng Q, Finley T, et al. LightGBM: A highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems (NeurIPS), 2017: 3146–3154.
[15] Chawla N V, Bowyer K W, Hall L O, et al. SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 2002, 16: 321–357.
[16] He H, Garcia E A. Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 2009, 21(9): 1263–1284.
[17] Elkan C. The foundations of cost-sensitive learning. Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), 2001: 973–978.
[18] Bahnsen A C, Aouada D, Ottersten B. Example-dependent cost-sensitive decision trees. Expert Systems with Applications, 2015, 42(19): 6609–6619.
[19] Bahnsen A C, Aouada D, Stojanovic A, et al. Feature engineering strategies for credit card Integrity detection. Expert Systems with Applications, 2016, 51: 134–142.

[20] Whitrow C, Hand D J, Juszczak P, et al. Transaction aggregation as a strategy for credit card Integrity detection. Data Mining and Knowledge Discovery, 2009, 18(1): 30–55.
[21] Zadrozny B, Elkan C. Transforming classifier scores into accurate multiclass probability estimates. Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002: 694–699.
[22] Niculescu-Mizil A, Caruana R. Predicting good probabilities with supervised learning. Proceedings of the 22nd International Conference on Machine Learning (ICML), 2005: 625–632.
[23] Bahnsen A C, Stojanovic A, Aouada D, et al. Improving credit card Integrity detection with calibrated probabilities. Proceedings of the SIAM International Conference on Data Mining (SDM), 2014: 677–685.
[24] Carcillo F, Dal Pozzolo A, Le Borgne Y A, et al. SCARFF: A scalable framework for streaming credit card Integrity detection with Spark. Information Fusion, 2018, 41: 182–194.
[25] Carcillo F, Le Borgne Y A, Caelen O, et al. Combining unsupervised and supervised learning in credit card Integrity detection. Information Sciences, 2021, 557: 317–331.