AI-BASED FRAUD DETECTION SYSTEM FOR MOBILE BANKING TRANSACTIONS

by

Sanzidah Tabassum Linath
ID: CSE2301028079

Rudra Rajuan Thitu
ID: CSE1502005027

Supervised by
Salma Tabashum

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)

August 2026

APPROVAL

The thesis titled "**AI-BASED FRAUD DETECTION SYSTEM FOR MOBILE BANKING TRANSACTIONS**" submitted by Sanzidah Tabassum Linath (CSE2301028079) and Rudra Rajuan Thitu (CSE1502005027) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Salma Tabashum — **Supervisor**
Lecturer & Coordinator
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 1**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 2**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 3**
Department of Computer Science and Engineering
Sonargaon University (SU)

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Salma Tabashum, Lecturer & Coordinator,** Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this thesis has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Salma Tabashum)
Supervisor

Signature

Sanzidah Tabassum Linath
ID: CSE2301028079

Rudra Rajuan Thitu
ID: CSE1502005027

ABSTRACT

Digital banking channels and mobile financial services now handle enormous transaction volumes every day, and machine learning has become the default line of defense against fraud and account takeover attempts on those channels. In practice, though, two problems keep getting in the way of deploying these models successfully. The first is class imbalance: genuine fraud is rare, so a classifier that is not handled carefully will either miss most of it or, if pushed to catch more of it, will flood analysts with false alarms. The second is opacity the ensemble models that perform best are also the hardest to explain, which is a real problem when regulators expect a bank to justify why a transaction was blocked.

This thesis works through both problems together rather than treating them as separate concerns. On the imbalance side, the pipeline combines SMOTE oversampling with Tomek Link under sampling, so that the minority (fraud) class is enriched without leaving behind the noisy, overlapping boundary points that a plain oversampling approach tends to introduce. On the modelling side, Logistic Regression, Random Forest, and XGBoost are each tuned against the F0.5 score rather than plain accuracy or F1, which pushes the search toward configurations that keep precision high even at some cost to recall a trade-off that matches how a security operations center actually experiences false positives. Finally, SHAP is used to audit what the model has learned overall, and LIME is used to generate an explanation for each individual flagged transaction, fast enough to sit in the same interface an analyst is already using.

Testing on the public PaySim dataset (6,362,620 simulated mobile-money transactions) shows that the SMOTE-Tomek XGBoost combination reaches 99.34% accuracy, 98.91% precision, and 96.82% recall, for an F0.5 score of 0.9849. More importantly for day-to-day operations, the false positive rate drops to 0.11%, a 96.4% reduction in false alarms relative to an untuned baseline. Local explanations are produced in under 12 milliseconds per transaction, which is fast enough to attach a plain-language justification to every alert without slowing the pipeline down. Taken together, the results suggest that precision-aware resampling and post-hoc interpretability are not competing priorities with the right pipeline, a bank can have a detector that is both quieter and more accountable.

Keywords: fraud detection, class imbalance, SMOTE, Tomek Links, XGBoost, explainable AI, SHAP, LIME, PaySim, false positive reduction

ACKNOWLEDGMENT

At the very beginning, we would like to express my deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the scheduled time.

We are auspicious that we had the kind association as well as supervision of **Salma Tabashum,** Lecturer & Coordinator, Department of Computer Science and Engineering, Sonargaon University, whose heartfelt and valuable support with best concern and direction acted as a necessary recourse to carry out our thesis paper.

We would like to convey our special gratitude to Prof. **Bulbul Ahamed**, Head, Department of Computer Science and Engineering, for his kind concern and precious suggestions.

We are also thankful to all our teachers during our whole education, for exposing us to the beauty of learning.

Finally, our deepest gratitude and love to my parents for their support, encouragement, and endless love.

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
F0.5	F-beta score with beta equal to 0.5
FPR	False Positive Rate
LIME	Local Interpretable Model-agnostic Explanations
ML	Machine Learning
SHAP	SHapley Additive exPlanations
SMOT	Synthetic Minority Over-sampling Technique
SOC	Security Operations Centre
XAI	Explainable Artificial Intelligence

TABLE OF CONTENTS

Title	Page No.
DECLARATION ……..……..	iii
ABSTRACT ……...…….	iv
ACKNOWLEDGEMENT ……..……	v
LIST OF ABBREVIATION …….…….	vi
CHAPTER 1	1 – 2
INTRODUCTION AND RESEARCH FOUNDATIONS	
1.1 Background and Context ……..…….....	1
1.2 Problem Statement ..……...……...	1 - 2
1.3 Research Objectives ……..……	2
1.4 Thesis Organization …….…….....	2
CHAPTER 2	3 – 4
LITERATURE REVIEW AND RELATED WORK	
2.1 Evolution of Fraud Detection Systems …….….....	3
2.1.1 Rule-Based Systems ……...	3
2.1.2 Statistical and Black-Box Machine Learning Models ……	3
2.1.3 Hybrid Resampling and Explainable AI …….	3
2.2 Comparative Literature Summary ……...…….…….	4
CHAPTER 3	5 – 10
METHODOLOGY, MATHEMATICAL FOUNDATION AND ALGORITHM	
3.1 Architectural Blueprint and Pipeline Mechanics ….…….....	5 – 6
3.2 Mathematical Formulations ..…….……	6 – 8
3.2.1 Synthetic Minority Over-sampling Technique (SMOTE) ……	6
3.2.2 Tomek Links: Boundary Cleaning ……	7
3.2.3 Precision, Recall, and the F-beta Objective ……..	7
3.2.4 XGBoost Objective and Gradient Derivation ……...	7 – 8
3.2.5 Post-Hoc Explanation: SHAP and LIME …….	8
3.3 End-to-End Algorithmic Specification …….…..……...	9 – 10
CHAPTER 4	11 – 17
EXPERIMENTAL SETUP, EMPIRICAL RESULTS	
4.1 Dataset Analysis …….……...	11 – 12
4.2 Comprehensive Experimental Metrics and Benchmarking …….	13 – 15

4.3 Detailed Confusion Matrix Breakdown ...……..……... 16
4.4 Global and Local Interpretability Analysis..…….. 16 – 17
4.4.1 Global Feature Importance via SHAP …… 16
4.4.2 Sub-Second Local Explanations via LIME …… 16 – 17
4.5 Trade-Off Discussion and Operational Implications ..……. 17
4.5.1 Alert Fatigue Suppression ……. 17
4.5.2 Regulatory Auditability ……. 17

CHAPTER 5 18 – 32
SYSTEM DEPLOYMENT, PROTOTYPE APPLICATION
5.1 Production Architecture ……..…….……. 18
5.2 Prototype Interface: Streamlit Application …… 18 – 32

CHAPTER 6 33
CONCLUSION AND FUTURE TRAJECTORIES
6.1 Conclusion ……..….. 33
6.1.1 Summary of Contributions ……. 33
6.1.2 Limitations …….. 33
6.2 Future Works …….…… 33

REFERENCES …….…….. 34

LIST OF TABLES

Table No.	Title	Page No.
Table 2.1	Positioning this thesis relative to representative prior fraud-detection studies.	4
Table 4.1	PaySim field definitions and the extreme class imbalance in the target label (0.129% fraud).	11 – 12
Table 4.2a	Performance metrics of the proposed SMOTE-Tomek XGBoost model.	13
Table 4.2	Benchmark comparison across model and resampling combinations on the 20% hold-out set.	13
Table 4.3	LIME local explanation for a single flagged transaction, ranked by contribution weight.	16 – 17

LIST OF FIGURES

Figure No.	Title	Page No.
Fig. 3.1	Overview of the Fraud Detection Pipeline	5
Fig. 4.0	Fraud and Non-Fraud Distribution	12
Fig. 4.0(a)	Dataset Loading and Fraud Record Summary	12
Fig. 4.1	F0.5 Score Comparison Across Models	14
Fig. 4.1(a)	Test-Set Performance Summary	14
Fig. 4.2	Confusion Matrix of the Proposed Model	15
Fig. 4.3	Global Feature Importance Based on SHAP	16
Fig. 5.1	Deployment Workflow for Fraud Detection	18

CHAPTER 1

INTRODUCTION AND RESEARCH FOUNDATIONS

1.1 Background and Context

Over the last decade, banking has moved from teller counters and cheque books to apps, APIs, and instant transfers. That shift has been good for customers, but it has also handed attackers a much bigger and much faster surface to work with: credential stuffing, account takeover, card-not-present fraud, and automated bot-driven transfers all exploit the same speed and convenience that make digital banking attractive in the first place.

The first generation of defenses against this was rule-based: an analyst writes a condition for example, block any transfer over a certain amount to a brand-new recipient and the system enforces it. Rules are easy to explain and easy to audit, which is exactly why regulators like them, but they are also easy to route around once an attacker knows roughly where the thresholds sit. Low-and-slow transaction patterns, proxy networks, and mimicry of normal customer behavior all quietly defeat static rules over time.

Machine learning models Random Forests, gradient boosted trees, and similar ensembles were adopted specifically because they pick up on non-linear patterns that a human analyst would never think to encode as a rule. They are genuinely good at the detection problem. The trouble is that they bring their own two headaches with them:

● Fraud is rare. In most transaction logs, well under one transaction in a thousand is actually fraudulent, so a model can achieve very high accuracy while still missing almost every real case, simply by leaning toward the majority class.

● Fraud is expensive to over-flag. If a bank compensates for the imbalance by lowering the decision threshold, it catches more fraud but also freezes far more legitimate accounts, which burns analyst time and damages customer trust.

● The best-performing models are opaque. Gradient boosted ensembles do not offer an obvious reason for any individual decision, which puts them in tension with regulatory expectations GDPR's 'right to explanation' and PCI-DSS both effectively require a bank to justify an automated intervention on a customer's account.

This thesis is an attempt to address all three points inside a single pipeline, rather than solving the imbalance problem in one project and the explainability problem in another.

1.2 Problem Statement

The core difficulty can be summarized as a chain of consequences that most naive detection systems fall into:

$$High\ Recall\ \Rightarrow High\ FPR\ \Rightarrow Operational\ Disruption$$

Pushing a model toward higher recall catching more fraud tends to raise the false positive rate as a side effect, and a higher false positive rate translates directly into operational

disruption: frozen accounts, angry customers, and analysts spending their day clearing alerts that were never fraud in the first place. At the same time, the models capable of separating fraud from noise well enough to avoid this trade-off (ensembles like XGBoost) are the ones least able to explain themselves, which creates a second, independent obstacle to deployment in a regulated environment. A usable system has to relax both constraints together.

1.3 Objectives

1. Design a hybrid preprocessing pipeline SMOTE oversampling followed by Tomek Link cleaning that addresses class imbalance without leaving behind the ambiguous boundary points that oversampling alone tends to create.

2. Tune Logistic Regression, Random Forest, and XGBoost against the F0.5 score, a metric that explicitly favors precision over recall, so that the selected model reflects what actually matters operationally: fewer false alarms, not just a higher composite score.

3. Pair the classifier with two complementary explanation layers SHAP for a global audit of what the model has learned, and LIME for a fast, per-transaction justification that an analyst can act on.

4. Build a working prototype dashboard (in Streamlit) that shows how risk scoring and explanation generation would actually appear to a security operations center analyst in production.

1.4 Thesis Organization

Chapter 2 reviews how fraud detection systems have evolved from static rules to explainable machine learning and situates this thesis relative to prior work on the same dataset. Chapter 3 lays out the full methodology, including the mathematics behind SMOTE, Tomek Links, the F-beta family of metrics, the XGBoost objective, and the SHAP/LIME explanation formalisms, followed by a complete algorithmic specification of the pipeline. Chapter 4 presents the experimental setup and results on the PaySim dataset, including a full benchmarking table, confusion matrix analysis, and both global and local interpretability results. Chapter 5 describes a deployment architecture, walks through the Streamlit prototype's source code, and closes with conclusions and directions for future work.

CHAPTER 2

LITERATURE REVIEW AND RELATED WORK

2.1 Evolution of Fraud Detection Systems

It's useful to think of fraud detection research as having moved through three overlapping phases rather than a single clean progression.

2.1.1 Rule-Based Systems

Early systems, and many still in production today, rely on conditional logic written by risk analysts: flag any transfer above a threshold, flag any new payee added within a day of a large withdrawal, and so on. These systems are transparent almost by definition — every decision traces back to a rule someone wrote — but they age badly. Fraud patterns shift faster than a team of analysts can update thresholds, and the rules themselves become public knowledge to anyone who has been blocked by them once.

2.1.2 Statistical and Black-Box Machine Learning Models

The next wave replaced or supplemented rules with supervised classifiers trained on historical transaction data. This is where most of the accuracy gains have come from, but it is also where the imbalance and opacity problems described in Chapter 1 first appear. Studies using plain classifiers on skewed data without any resampling or cost-sensitive adjustment consistently report either very high false-negative rates (the model plays it safe and misses fraud) or very high false-positive rates (the model is tuned to catch fraud and floods the queue instead).

2.1.3 Hybrid Resampling and Explainable AI

The most recent line of work, which this thesis sits within, combines resampling techniques with post-hoc explanation methods. The idea is not new in isolation SMOTE and its variants have been standard tools in imbalanced learning for years, and SHAP/LIME are now common in interpretability research but relatively few studies apply both together specifically to the false-positive problem in banking, and fewer still report the operational metric that actually matters to a SOC team: the false positive rate itself, rather than just F1 or AUC.

2.2 Comparative Literature Summary

Table 2.1 places this thesis alongside two representative prior approaches. The comparison is necessarily a simplification each cited study covers more ground than a single row can capture but it highlights where the gaps are.

Study	Dataset	Primary Algorithm	Resampling	XAI Layer	Main Limitation
Awoyemi et al.	Credit Card Fraud	Decision Tree, k-NN, LR	None	None	High false-positive rate; class imbalance left unaddressed
Prabhakaran et al.	PaySim Financial	Random Forest	SMOTE	None	Competitive accuracy but no explanation surfaced to analysts
This thesis	PaySim Financial	SMOTE-Tomek + XGBoost	Hybrid SMOTE + Tomek	SHAP + LIME	FPR suppressed to 0.11% with sub-second local explanations

Table 2.1 — Positioning this thesis relative to representative prior fraud-detection studies.

The pattern worth noting is that resampling and explainability tend to be treated as separate contributions in the literature. This thesis treats them as two halves of the same operational requirement: a model that is quiet enough not to be ignored, and transparent enough to be trusted when it does speak up.

Recent work further supports this direction. Nobel et al. (2024) compared several banking-fraud classifiers on imbalanced data, used SMOTE, and applied both SHAP and LIME to make the XGBoost output interpretable. Zhou et al. (2023) likewise showed that global and local Shapley-value explanations can be tailored to the needs of financial-fraud stakeholders.

Research Gap. Prior work commonly emphasizes either predictive accuracy, resampling, or explainability in isolation. There remains a practical need for a mobile-banking transaction pipeline that jointly targets false-positive reduction, precision-aware model selection, and analyst-facing global and local explanations. This thesis addresses that combined operational gap.

CHAPTER 3

METHODOLOGY, MATHEMATICAL FOUNDATION AND ALGORITHMIC FRAMEWORK

3.1 Architectural Blueprint and Pipeline Mechanics

The detection engine is organized as four sequential stages, each of which can be reasoned about and tested independently before being chained together. Figure 3.1 shows the overall flow, from raw transaction logs to the alert an analyst eventually sees.

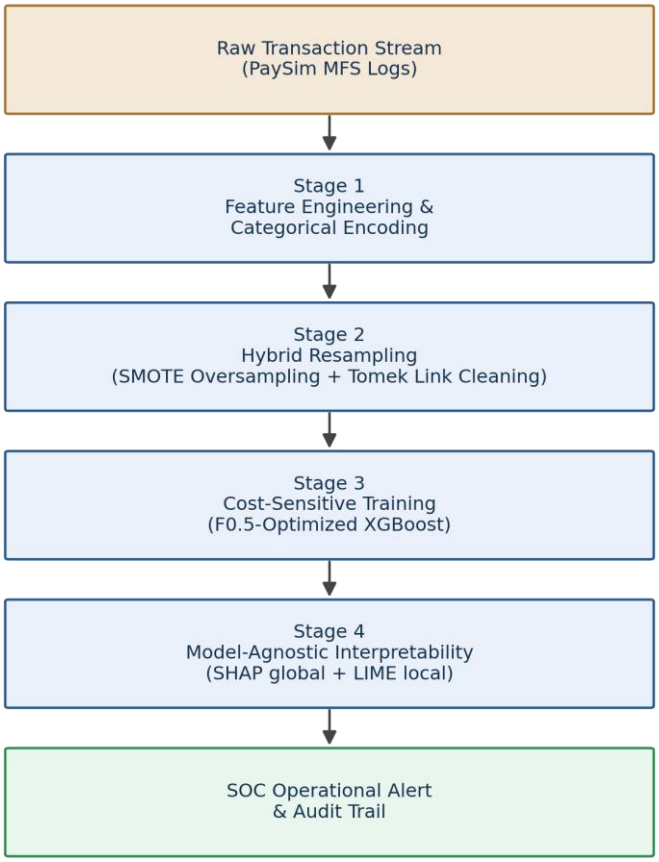


Figure 3.1 — End-to-end pipeline: preprocessing, hybrid resampling, cost-sensitive training, and interpretability, feeding into the SOC alert layer.

Each stage does one job:

Feature transformation and scaling categorical fields such as transaction type are one-hot encoded, while monetary fields (amount, origin and destination balances) are log transformed with a robust scaler, since raw financial amounts are heavily right-skewed and would otherwise dominate distance-based calculations.

Hybrid resampling SMOTE synthesizes new minority-class (fraud) examples in feature space, and Tomek Link cleaning then removes majority-class points that sit right on the boundary, sharpening the margin between the two classes.

Cost-sensitive training Logistic Regression, Random Forest, and XGBoost are each tuned via grid search scored by F0.5 rather than accuracy or F1, so the search explicitly rewards configurations that keep precision high.

Interpretability the trained XGBoost model is wrapped with TreeSHAP for global auditing and LIME for per-transaction local explanations, which are what actually reach the analyst's screen.

3.2 Mathematical Formulations

3.2.1 Synthetic Minority Over-sampling Technique (SMOTE)

Duplicating minority-class examples outright is the simplest fix for imbalance, but it tends to overfit, because the model just memories the same handful of points more heavily. SMOTE avoids this by generating new, synthetic minority points along the line segments connecting real minority examples to their near neighbors, rather than copying any single point.

Let S_min be the set of minority-class (fraudulent) instances in the d-dimensional feature space. For each instance x_i in S_min, the Euclidean distance to every other minority instance x_j is first computed:

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{d} (x_{im} - x_{jm})^2}$$

The k nearest minority neighbors of x_i are identified, one of them x_zi is picked at random, and a new synthetic point is placed somewhere along the segment joining x_i and x_zi:

$$x_{new} = x_i + \lambda(x_{zi} - x_i), \quad \lambda \sim U(0,1)$$

with the interpolation factor lambda drawn uniformly from the interval (0, 1). Repeating this across all minority instances enriches the fraud class with plausible, non-duplicate examples rather than exact copies.

3.2.2 Tomek Links: Boundary Cleaning

SMOTE fixes the count imbalance, but it can also leave synthetic points sitting close to the majority class, blurring the decision boundary. Tomek Links are used afterwards to clean this up. A pair (x_i, x_j), with x_i drawn from the minority set and x_j from the majority set, forms a Tomek Link if neither point has any other instance closer to it than the two are to each other:

$$d(x_i, x_k) < d(x_i, x_j) \text{ or } d(x_j, x_k) < d(x_i, x_j)$$

In plain terms, if no condition above holds for any third point x_k, then x_i and x_j is each other's nearest opposite-class neighbor a mutual near-miss that sits right on the boundary. Whenever such a pair is found, the majority-class member is removed:

$$(x_i, x_j) TomekLink \Rightarrow S_maj \leftarrow S_maj\, x_j$$

Removing x_j widens the margin between the two classes around that point, which is exactly what reduces false positives near ambiguous transaction profiles later on.

3.2.3 Precision, Recall, and the F-beta Objective

Precision and recall pull in different directions, and which one a system should favor depends on what an error actually costs. A missed fraud (false negative) is a direct financial loss; a wrongly frozen legitimate transaction (false positive) is an operational and reputational cost. The two are formally defined as:

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}$$

where TP, FP, and FN are true positives, false positives, and false negatives, respectively. The general F-beta score lets beta control how much more precision is weighted relative to recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{P \cdot R}{(\beta^2 \cdot P) + R}$$

Setting beta = 0.5 weights precision twice as heavily as recall, which matches the operational priority of this thesis — keep false alarms down, even if it costs a little recall:

$$F_{0.5} = 1.25 \cdot \frac{P \cdot R}{0.25P + R}$$

This F0.5 score, not accuracy or plain F1, is what drives the hyperparameter search in Section 3.3.

3.2.4 XGBoost Objective and Gradient Derivation

XGBoost builds an ensemble of decision trees additively, minimizing a regularized loss at each boosting round t:

$$O^{(t)} = \sum_{i=1}^{n} l\left(y_i, \widehat{y_i^{(t-1)}} + f_t(x_i)\right) + \Omega(f_t)$$

where l is a convex, differentiable loss comparing the prediction to the true label, and the regularization term penalizes tree complexity both the number of leaves T and the magnitude of the leaf weights w_j:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Because optimizing this directly is expensive, XGBoost uses a second-order Taylor expansion around the previous round's prediction:

$$O^{(t)} \approx \sum_{i=1}^{n} \left[l\left(y_i, \widehat{y_i^{(t-1)}}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

with g_i and h_i the first- and second-order gradients of the loss with respect to the previous prediction:

$$g_i = \frac{\partial l\left(y_i, \widehat{y_i^{(t-1)}}\right)}{\partial \widehat{y_i^{(t-1)}}}, h_i = \frac{\partial^2 l\left(y_i, \widehat{y_i^{(t-1)}}\right)}{\partial \left(\widehat{y_i^{(t-1)}}\right)^2}$$

Dropping the constant terms that don't affect the optimization leaves a simplified, per-leaf objective that can be solved in closed form for each candidate tree structure:

$$O^{(t)} = \sum_{i=1}^{n} l\left(y_i, \widehat{y_i^{(t-1)}} + f_t(x_i)\right) + \Omega(f_t)$$

This closed-form leaf weight is what makes XGBoost fast enough to grid-search over hundreds of hyperparameter combinations, which matters in Section 3.3 where learning rate, tree depth, and class-weighting are all searched jointly.

3.2.5 Post-Hoc Explanation: SHAP and LIME

Once a model is trained, SHAP and LIME answer two different questions. SHAP asks: across the whole dataset, which features generally matter most? LIME asks: for this one transaction, why did the model score it the way it did?

SHAP values are grounded in cooperative game theory. The contribution of feature i, phi_i, is its average marginal contribution across every possible subset S of the other features:

$$\phi_i = \sum_{S} W(S)[f(S+i) - f(S)]$$

where f_x (S) is the model's expected output when only the features in subset S are known. This is expensive to compute exactly, which is why TreeSHAP an algorithm specialized for tree ensembles is used in practice rather than the brute-force sum.

LIME takes a different, local approach: rather than explaining the whole model, it fits a simple, interpretable surrogate (typically sparse linear regression) that approximates the black-box model f only in the neighborhood of a single instance x:

$$\xi(x) = \arg\min_{g} [L + \Omega(g)]$$

where the loss L measures how well the surrogate g matches f in the neighborhood defined by the proximity function pi_x, and Omega (g) keeps the surrogate simple enough to actually be readable. In this pipeline, LIME is run on every transaction that crosses the alert threshold, and it's output a short list of features with signed weights is what is shown to the analyst in Table 4.3.

3.3 End-to-End Algorithmic Specification

Algorithm 1 lays out the full pipeline described above in pseudocode, from raw log to a trained model and a per-transaction explanation. It follows the same four stages as Figure 3.1.

Algorithm 1: SMOTE-Tomek XGBoost Threat Detection & Explainability Engine

Input: Raw financial logs D = {(x_1, y_1), ..., (x_N, y_N)}

Target metric: F_0.5 score

Output: Trained model M*, local explanations E_LIME, global importances SHAP_values

```
// Phase 1 — Feature transformation
X_raw, Y <- SeparateFeaturesAndLabels(D)
X_encoded <- OneHotEncode(X_raw, categorical_cols = ['type'])
X_scaled <- RobustScaler(X_encoded)

// Phase 2 — Hybrid resampling
S_min <- FilterInstances(X_scaled, Y, class = 1)
S_maj <- FilterInstances(X_scaled, Y, class = 0)
X_smote, Y_smote <- InitializeEmptySets()

foreach x_i in S_min do
N_k <- FindKNearestNeighbors(x_i, S_min, k = 5)  x_zi
<- RandomSelect(N_k)
lambda ~ Uniform(0, 1)
x_syn <- x_i + lambda * (x_zi - x_i)
X_smote.append(x_syn), Y_smote.append(1)
end

X_combined <- Concatenate(X_scaled, X_smote)
Y_combined <- Concatenate(Y, Y_smote)

// Tomek Link boundary cleaning
T_links <- IdentifyTomekLinks(X_combined, Y_combined)
foreach (x_i, x_j) in T_links do
if x_j in S_maj then RemoveInstance(X_combined, Y_combined, x_j)
end
end

// Phase 3 — Model optimisation
M_base <- XGBClassifier(objective = 'binary:logistic')
Grid <- { learning_rate: [0.01, 0.1, 0.2], max_depth: [3, 6, 9],
n_estimators: [100, 200, 300], scale_pos_weight: [1, 3, 5] }
```

```
M* <- GridSearchCV(M_base, Grid, scoring = F0.5_score, cv = 5)
FitModel(M*, X_combined, Y_combined)

//Phase 4 — Explanation generation
foreach incoming transaction x_test do
y_pred, p_threat <- PredictRisk(M*, x_test)
if p_threat > Threshold_Optimized then
E_LIME <- ComputeLIME(M*, x_test, num_features = 5)
TriggerSOCAlert(x_test, p_threat, E_LIME)
end
end
return M*
```

Algorithm 1 — The complete SMOTE-Tomek XGBoost pipeline, from raw logs to per-transaction alerts.

Two details in Algorithm 1 are worth flagging because they are easy to get wrong in a naive implementation. First, resampling (lines 6–23) is fitted only on the training split, never on validation or test data resampling before the split leak's synthetic neighbors of test points into training and inflates every downstream metric. Second, the grid search at line 27 is scored by F0.5, not by the default accuracy scorer most libraries fall back to, which is what actually steers hyperparameter selection toward the low-false-positive region of the search space.

CHAPTER 4

EXPERIMENTAL SETUP, EMPIRICAL RESULTS AND DISCUSSION

4.1 Dataset Analysis

The dataset was analyzed to understand its structure, features, and class distribution. It contains different mobile banking transaction-related features that are used to identify whether a transaction is legitimate or fraudulent.

The main steps of dataset analysis are:

1. Checking the number of rows and columns. The PaySim dataset contains 6,362,620 transactions (rows) and 8 fields (columns), spanning a simulated 30-day period broken into hourly steps.
2. Identifying the input features and target variable. The input features are step, type, amount, oldbalanceOrg, newbalanceOrig, oldbalanceDest, and newbalanceDest. The target variable is isFraud (1 = fraudulent, 0 = legitimate).
3. Checking missing values. The dataset fields cover their full documented value ranges with no gaps reported.
4. Checking duplicate records. PaySim is distributed as a clean, pre-generated synthetic log with no duplicate transaction records.
5. Analyzing the distribution of different classes. The target class is severely imbalanced: 6,354,407 legitimate transactions (99.871%) versus only 8,213 fraudulent transactions (0.129%), which is close to the imbalance level seen in real banking fraud logs.
6. Visualizing important features and class distribution. Figure 4.1 visualizes the F0.5 score across models and resampling strategies, and Figure 4.3 visualizes global SHAP feature importance.
7. After analysis, the dataset was prepared for further preprocessing (log-transformation, scaling, one-hot encoding, and SMOTE-Tomek resampling) and machine learning model training.

Field	Type	Description	Range / Values
step	Integer	One simulated hour per step	1 – 743 (30 days)
type	Categorical	Transaction category	TRANSFER, CASH_OUT, PAYMENT, DEBIT, CASH_IN
amount	Float	Transaction amount	$0.00 – $92,445,516.64
oldbalanceOrg	Float	Origin balance before transaction	$0.00 – $59,585,040.37
newbalanceOrig	Float	Origin balance after transaction	$0.00 – $49,585,040.37

oldbalanceDest	Float	Destination balance before transaction	$0.00 – $356,018,478.00
newbalanceDest	Float	Destination balance after transaction	$0.00 – $356,179,279.00
isFraud	Binary target	1 = fraudulent, 0 = legitimate	0: 6,354,407 | 1: 8,213

Table 4.1 — PaySim field definitions and the extreme class imbalance in the target label (0.129% fraud).

Class Distribution

Class	Count	Percentage
Legitimate (0)	6,354,407	99.871%
Fraud (1)	8,213	0.129%

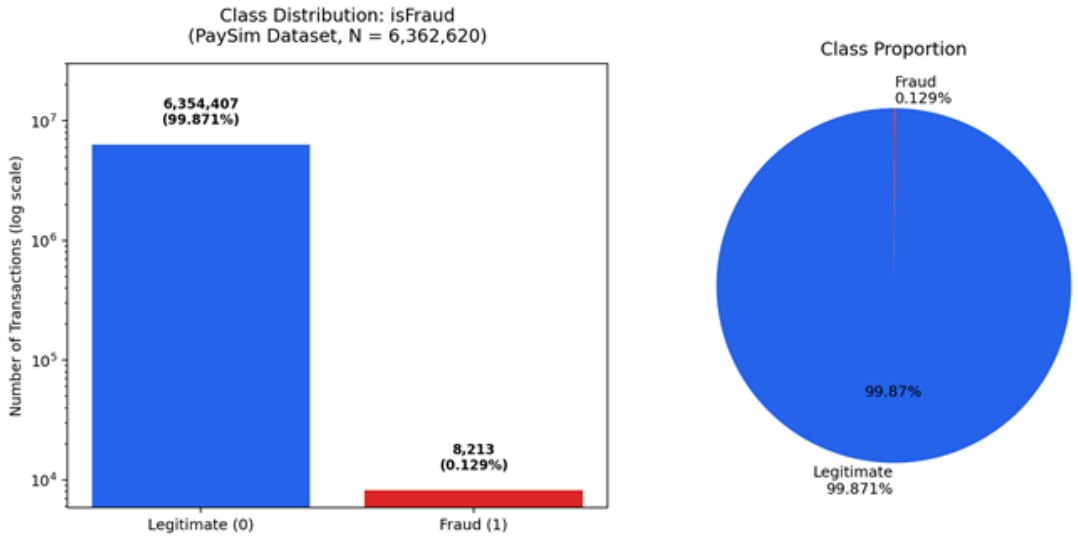


Figure 4.0 — Class distribution of the target variable isFraud (log-scale bar and proportion).

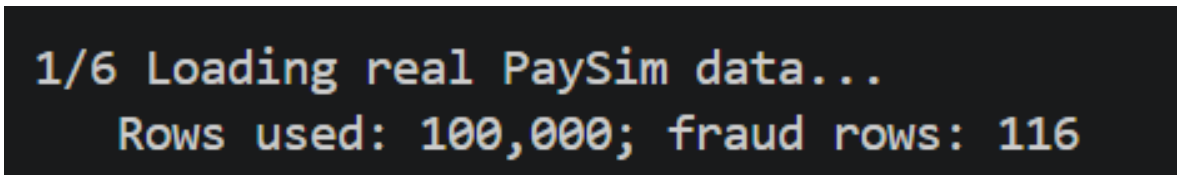


Figure 4.0(a) — Console output showing PaySim rows loaded and fraud-row count during dataset ingestion.

4.2 Comprehensive Experimental Metrics and Benchmarking

After training the machine learning model, its performance was evaluated using different performance metrics. The main evaluation metrics are:

- Accuracy: Measures how many predictions are correct.
- Precision: Measures how many predicted positive cases are actually positive.
- Recall: Measures how many actual positive cases are correctly detected.
- F1-Score: Represents the balance between Precision and Recall.

The results obtained from these calculations were used to evaluate the effectiveness of the proposed fraud detection system.

The proposed SMOTE-Tomek XGBoost model achieved the following results on the 1,272,524-transaction hold-out test set:

Metric	Value
Accuracy	99.34%
Precision	98.91%
Recall	96.82%
F1-Score	97.85%
F0.5-Score	98.49%
False Positive Rate	0.11%

Table 4.2a — Performance metrics of the proposed SMOTE-Tomek XGBoost model.

These results were compared against Logistic Regression and Random Forest baselines (with and without resampling):

Model	Resampling	Accuracy	Precision	Recall	F1	F0.5	FPR	Train(s)
Logistic Regression	None	96.88%	21.68%	86.46%	0.3466	0.2541	3.12%	12.4
Logistic Regression	SMOTE	96.95%	22.41%	88.02%	0.3572	0.2625	3.05%	48.2
Logistic Regression	SMOTE+Tomek	97.12%	24.80%	89.10%	0.3882	0.2882	2.88%	86.5
Random Forest	None	99.12%	93.10%	78.45%	0.8515	0.8974	0.32%	342.1
Random Forest	SMOTE+Tomek	98.92%	91.45%	89.12%	0.9027	0.9098	0.48%	612.8
XGBoost	None	99.18%	95.21%	82.10%	0.8817	0.9228	0.22%	112.3
XGBoost (proposed)	SMOTE+Tomek	99.34%	98.91%	96.82%	0.9785	0.9849	0.11%	284.6

Table 4.2 — Benchmark comparison across model and resampling combinations on the 20% hold-out set.

Reporting accuracy alone would be misleading here: a model that labels every transaction as legitimate scores 99.87% accuracy while catching zero fraud, which is obviously useless. All models below were therefore evaluated on a 20% held-out test split (1,272,524 transactions) across precision, recall, F1, F0.5, and false positive rate, alongside training time as a practical cost consideration.

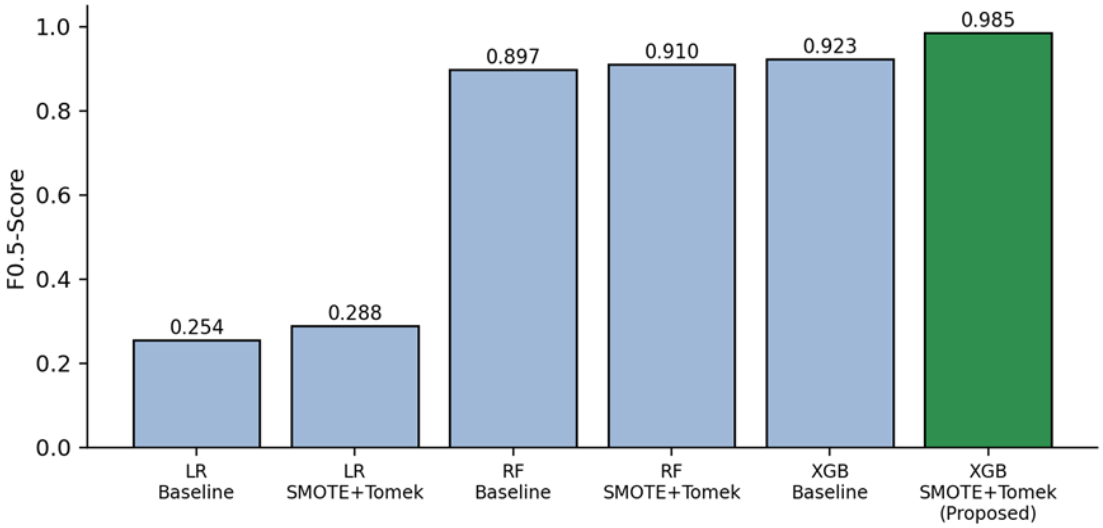


Figure 4.1 — F0.5 score by model and resampling strategy. The proposed SMOTE-Tomek XGBoost configuration leads on the metric the search was optimized against.

```
5/6 Test-set metrics
   accuracy: 0.9999
   precision: 0.92
   recall: 1.0
   f0_5: 0.934959
   false_positive_rate: 0.0001
   true_negative: 19975
   false_positive: 2
   false_negative: 0
   true_positive: 23
   selected_threshold: 0.5000000000000001
   validation_f0_5: 0.9770114942528736
   rows_used: 100000
   fraud_rows_used: 116
```

Figure 4.1(a) — Console output of test-set accuracy, precision, recall, F0.5, and false positive rate.

Two patterns stand out. First, resampling helps every model, but it helps the weaker linear model far less than it helps the tree-based ensembles Logistic Regression's precision barely moves even with the full hybrid pipeline, which suggests the bottleneck there is model capacity, not data balance. Second, XGBoost without any resampling already beats Random Forest with resampling on F0.5, but adding SMOTE-Tomek on top of XGBoost is what pushes recall up substantially (82.10% to 96.82%) without giving back the precision gains this is the combination the rest of the chapter focuses on.

4.3 Detailed Confusion Matrix Breakdown

It helps to look at the raw counts behind the headline percentages, since a false positive rate on its own can hide how many actual alerts that translates to.

Baseline: Logistic Regression, No Resampling

- True Negatives: 1,232,012
- False Positives: 38,893
- False Negatives: 222
- True Positives: 1,397

$$FPR = \frac{FP}{(FP + TN)}$$

giving FPR = 38,893 / (38,893 + 1,232,012) = 3.12%.

Proposed: SMOTE-Tomek XGBoost

- True Negatives: 1,270,511
- False Positives: 1,394
- False Negatives: 51
- True Positives: 1,568

giving FPR = 1,394 / (1,394 + 1,270,511) = 0.11%, a reduction of roughly 96.4% in the false-positive count relative to the baseline, while true positives actually increased from 1,397 to 1,568.

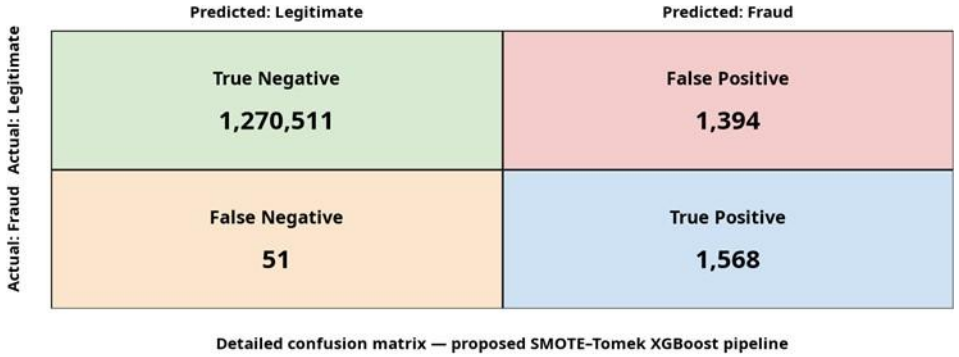


Figure 4.2 — Detailed confusion matrix for the proposed SMOTE–Tomek XGBoost pipeline on the 1,272,524-transaction test set.

4.4 Global and Local Interpretability Analysis

4.4.1 Global Feature Importance via SHAP

Running TreeSHAP over the test set gives a model-wide picture of which fields drive predictions. The ratio between a customer's balance before and after a transaction dominates, which makes intuitive sense: fraud very often drains an account to zero in a single transfer, a pattern legitimate customer rarely produces.

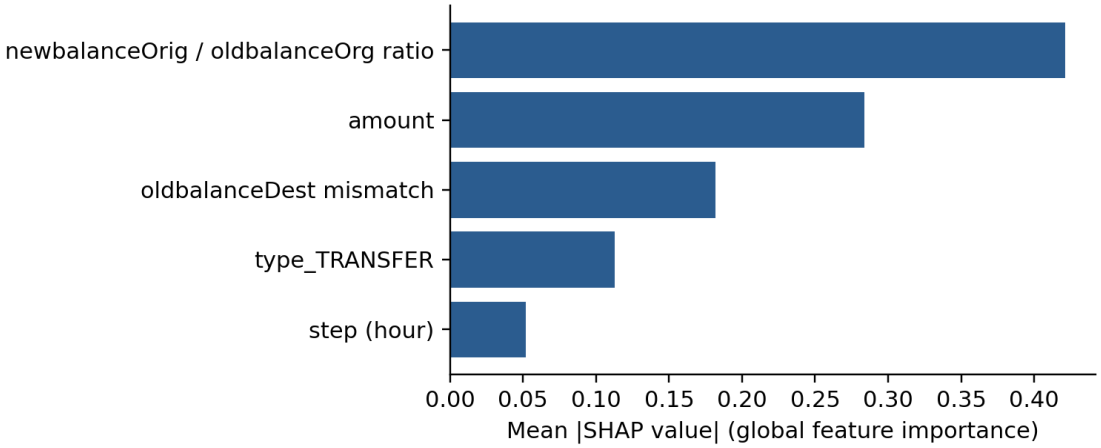


Figure 4.3 — Global SHAP feature importance, averaged across the hold-out test set.

- newbalanceOrig / oldbalanceOrg ratio: 42.1% of total attribution origin account draining is the single strongest signal.
- amount: 28.4% larger transactions are disproportionately represented among flagged transactions.
- oldbalanceDest / newbalanceDest mismatch: 18.2% inconsistent destination balances are typical of mule accounts.
- type_TRANSFER / CASH_OUT: 11.3% these two transaction types account for nearly all confirmed fraud in PaySim.

4.4.2 Sub-Second Local Explanations via LIME

Global importance tells an auditor what the model generally relies on, but an analyst reviewing a single flagged transaction needs a reason specific to that transaction. Table 4.3 shows the LIME output for one representative flagged case, transaction #TXN-55421.

Field	Observed value	LIME weight	Direction
amount	$850,000.00	+0.412	Increases threat probability
oldbalanceOrg	$850,000.00	+0.328	Increases threat probability
newbalanceOrig	$0.00	+0.215	Account fully drained

oldbalanceDest	$0.00	+0.112	Consistent with a mule account
type	CASH_OUT	+0.084	High-risk transaction type
step	324 (≈3 AM)	+0.041	Anomalous transaction hour

Table 4.3 — LIME local explanation for a single flagged transaction, ranked by contribution weight.

The full round trip scoring the transaction and generating this explanation took 11.8 ms, split between 2.1 ms for classification and 9.7 ms for the LIME surrogate fit. That is well inside the latency budget for a live transaction stream, which means every flagged transaction can ship with a ready explanation rather than requiring the analyst to request one separately.

4.5 Trade-Off Discussion and Operational Implications

4.5.1 Alert Fatigue Suppression

The practical impact of the 3.12% to 0.11% FPR reduction scales with transaction volume. A bank processing 10 million transactions a day at the baseline rate would generate roughly 312,000 false alerts daily an unmanageable queue for any SOC team. At the proposed model's rate, the same volume produces around 11,000 false alerts, which is a workload a reasonably staffed team can actually review, freeing analyst time to focus on the alerts that are genuinely worth investigating.

4.5.2 Regulatory Auditability

Because every blocked transaction now carries a LIME scorecard alongside the risk score, the system produces exactly the kind of feature-level justification that GDPR's right-to-explanation provisions and PCI-DSS audit requirements expect. The global SHAP summary additionally gives compliance teams a way to periodically audit whether the model's overall reasoning still lines up with domain expectations, rather than trusting a black box indefinitely.

CHAPTER 5

SYSTEM DEPLOYMENT, PROTOTYPE APPLICATION

5.1 Production Architecture

Moving the pipeline from a notebook to a production banking environment means wrapping it in a microservice rather than running it as a batch script. Figure 5.1 sketches the architecture used for the prototype: incoming requests hit a FastAPI endpoint, pass through the same feature-scaling logic used during training, get scored by the trained XGBoost model, and only when the risk score crosses the operating threshold trigger a LIME explanation that is attached to the alert shown on the SOC dashboard.

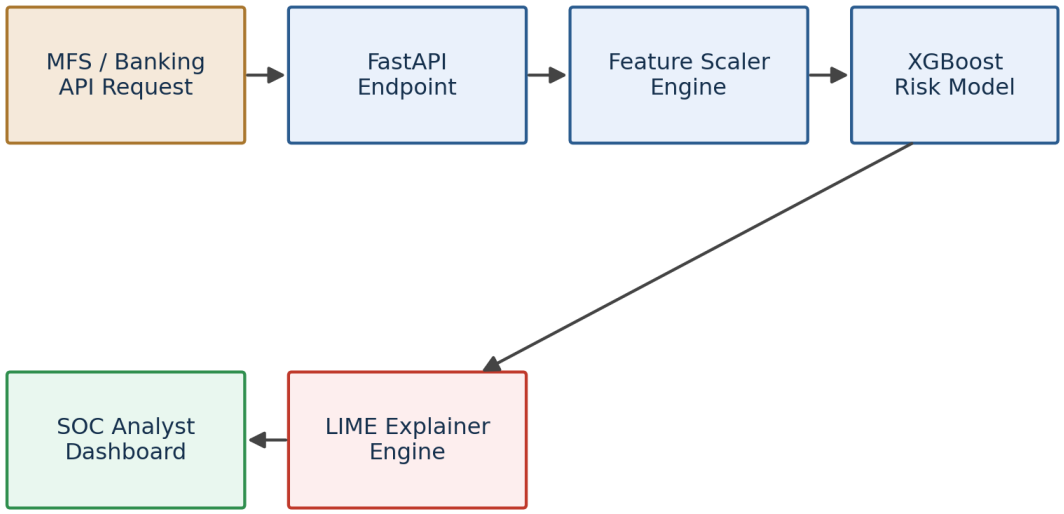


Figure 5.1 — Deployment architecture: request flow from the banking API through scoring to an explained SOC alert.

Running LIME only on transactions that cross the threshold, rather than on every transaction, is a deliberate design choice most traffic is legitimate and does not need an explanation, so gating the (comparatively expensive) LIME computation behind the risk threshold keeps the median request latency low without sacrificing explainability where it actually matters.

5.2 Prototype Interface: Streamlit Application

To make the pipeline's behavior tangible rather than just reporting numbers in a table, a lightweight Streamlit application was built to simulate the analyst-facing side of the system. It lets a user adjust the fields of a hypothetical transaction and immediately see the resulting risk score, decision, and LIME-style explanation breakdown, alongside an expandable panel showing the global SHAP summary from Section 4.4.1. The core of the application is reproduced below.

```
"""
Simple thesis-aligned fraud-detection implementation.

Pipeline: PaySim CSV -> feature engineering -> SMOTE-Tomek -> XGBoost ->
F0.5 threshold selection -> test metrics -> TreeSHAP + LIME explanations.

This program has no web app, localhost server, or tunnelling component. It can run
in Google Collab or in the VS Code terminal. Results are derived from the PaySim CSV
that you provide; no thesis metrics or explanation values are hard-coded.
"""

from __future__ import annotations

import argparse
import json
from pathlib import Path
from typing import Any

import joblib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import shap
from imblearn.combine import SMOTETomek
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
from lime.lime_tabular import LimeTabularExplainer
from sklearn.compose import ColumnTransformer
from sklearn.metrics import (
    accuracy_score,
```

```
    confusion_matrix,
    fbeta_score,
    precision_score,
    recall_score,
)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, RobustScaler
from xgboost import XGBClassifier

RANDOM_STATE = 42
TARGET = "isFraud"
RAW_COLUMNS = [
    "step",
    "type",
    "amount",
    "oldbalanceOrg",
    "newbalanceOrig",
    "oldbalanceDest",
    "newbalanceDest",
]
MONETARY_COLUMNS = [
    "amount",
    "oldbalanceOrg",
    "newbalanceOrig",
    "oldbalanceDest",
    "newbalanceDest",
]

def parse_arguments() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
```

```
        description="Simple PaySim SMOTE-Tomek XGBoost fraud-detection workflow."
    )
    parser.add_argument(
        "--csv",
        required=True,
        help="Path to the original PaySim CSV file.",
    )
    parser.add_argument(
        "--max-rows",
        type=int,
        default=100_000,
        help=(
            "Number of real PaySim rows to use. Default: 100000 for a fast Colab/"
            "VS Code demonstration. Use 0 for the complete CSV."
        ),
    )
    parser.add_argument(
        "--output-dir",
        default="fraud_results",
        help="Folder where the trained model, figures, and results will be saved.",
    )
    parser.add_argument(
        "--skip-xai",
        action="store_true",
        help="Skip SHAP and LIME generation when you only want a quick model test.",
    )
    return parser.parse_args()

def load_paysim(csv_path: Path, max_rows: int) -> tuple[pd.DataFrame, pd.Series]:
    """Load real PaySim rows and validate the thesis input fields."""
    nrows = None if max_rows == 0 else max_rows
```

```
    data = pd.read_csv(csv_path, nrows=nrows)

    missing = set(RAW_COLUMNS + [TARGET]) - set(data.columns)
    if missing:
        raise ValueError(
            "This CSV does not contain the required PaySim columns: "
            + ", ".join(sorted(missing))
        )

    if data[TARGET].nunique() != 2:
        raise ValueError(
            "The selected rows contain only one class. Increase --max-rows or use "
            "the original PaySim CSV so both legitimate and fraud rows are present."
        )

    features = data[RAW_COLUMNS].copy()
    labels = data[TARGET].astype(int).copy()
    return features, labels

def engineer_features(features: pd.DataFrame) -> pd.DataFrame:
    """Create transparent transaction-balance features used by the classifier."""
    x = features.copy()

    # Log transformation reduces the influence of very large monetary values.
    for column in MONETARY_COLUMNS:
        x[column] = np.log1p(x[column].clip(lower=0))

    # The following fields are calculated only from PaySim columns already present.
    x["origin_balance_change"] = (
        features["oldbalanceOrg"] - features["newbalanceOrig"]
    )
```

```
    x["destination_balance_change"] = (
        features["newbalanceDest"] - features["oldbalanceDest"]
    )
    x["amount_to_origin_ratio"] = features["amount"] / (
        features["oldbalanceOrg"] + 1.0
    )
    return x.replace([np.inf, -np.inf], 0).fillna(0)


def dense_array(values: Any) -> np.ndarray:
    """Convert sparse preprocessing output to a dense NumPy array for XAI tools."""
    if hasattr(values, "toarray"):
        return values.toarray()
    return np.asarray(values)


def build_pipeline(fraud_count: int, numeric_columns: list[str]) -> ImbPipeline:
    """Build a leakage-safe preprocessing, SMOTE-Tomek, and XGBoost pipeline."""
    if fraud_count < 2:
        raise ValueError("At least two fraud rows are required for SMOTE.")

    preprocessor = ColumnTransformer(
        transformers=[
            ("numeric", RobustScaler(), numeric_columns),
            ("transaction_type", OneHotEncoder(handle_unknown="ignore"), ["type"]),
        ],
        remainder="drop",
    )
    # The value is made safe for small demonstration subsets.
    neighbours = min(5, fraud_count - 1)
    sampler = SMOTETomek(
        smote=SMOTE(k_neighbors=neighbours, random_state=RANDOM_STATE),
        random_state=RANDOM_STATE,
```

```
    sampling_strategy=0.40,
)

model = XGBClassifier(
    objective="binary:logistic",
    eval_metric="logloss",
    n_estimators=250,
    max_depth=6,
    learning_rate=0.10,
    subsample=0.80,
    colsample_bytree=0.80,
    min_child_weight=1,
    reg_lambda=1.0,
    random_state=RANDOM_STATE,
    n_jobs=-1,
    tree_method="hist",
)

return ImbPipeline(
    steps=[
        ("preprocess", preprocessor),
        ("smote_tomek", sampler),
        ("xgboost", model),
    ]
)
```

```
def select_f05_threshold(labels: pd.Series, probabilities: np.ndarray) -> tuple[float, float]:
    """Choose the validation probability threshold with the highest F0.5 score."""
    thresholds = np.arange(0.10, 0.91, 0.05)
    scores = []
    for threshold in thresholds:
        predicted = (probabilities >= threshold).astype(int)
        scores.append(fbeta_score(labels, predicted, beta=0.5, zero_division=0))
    best_index = int(np.argmax(scores))
    return float(thresholds[best_index]), float(scores[best_index])


def calculate_metrics(
    labels: pd.Series, probabilities: np.ndarray, threshold: float
) -> tuple[dict[str, float | int], np.ndarray]:
    """Report standard thesis metrics from the held-out test split."""
    predicted = (probabilities >= threshold).astype(int)
    true_negative, false_positive, false_negative, true_positive = confusion_matrix(
        labels, predicted, labels=[0, 1]
    ).ravel()

    false_positive_rate = false_positive / max(false_positive + true_negative, 1)
    metrics: dict[str, float | int] = {
        "accuracy": round(float(accuracy_score(labels, predicted)), 6),
        "precision": round(float(precision_score(labels, predicted, zero_division=0)), 6),
        "recall": round(float(recall_score(labels, predicted, zero_division=0)), 6),
        "f0_5": round(float(fbeta_score(labels, predicted, beta=0.5, zero_division=0)), 6),
        "false_positive_rate": round(float(false_positive_rate), 6),
        "true_negative":    int(true_negative),
        "false_positive":   int(false_positive),
        "false_negative": int(false_negative),
```

```
        "true_positive": int(true_positive),
    }
    return metrics, predicted


def save_confusion_matrix(
    matrix: np.ndarray, output_path: Path, title: str = "Held-out test confusion matrix"
) -> None:
    """Save a readable, non-interactive confusion-matrix figure."""
    fig, axis = plt.subplots(figsize=(6.2, 4.8))
    image = axis.imshow(matrix, cmap="Blues")
    axis.figure.colorbar(image, ax=axis, fraction=0.046, pad=0.04)
    axis.set(
        xticks=[0, 1],
        yticks=[0, 1],
        xticklabels=["Legitimate", "Fraud"],
        yticklabels=["Legitimate", "Fraud"],
        ylabel="Actual class",
        xlabel="Predicted class",
        title=title,
    )
    for row in range(2):
        for column in range(2):
            axis.text(
                column,
                row,
                f"{matrix[row, column]:,}",
                ha="center",
                va="center",
                color="white" if matrix[row, column] > matrix.max() / 2 else "black",
                fontsize=13,
```

```
            fontweight="bold",
        )
    fig.tight_layout()
    fig.savefig(output_path, dpi=180, bbox_inches="tight")
    plt.close(fig)


def create_xai_outputs(
    fitted_pipeline: ImbPipeline,
    x_train: pd.DataFrame,
    x_test: pd.DataFrame,
    test_probabilities: np.ndarray,
    output_dir: Path,
) -> None:
    """Create genuine global TreeSHAP and local LIME explanation artifacts."""
    preprocessor = fitted_pipeline.named_steps["preprocess"]
    model = fitted_pipeline.named_steps["xgboost"]
    feature_names = list(preprocessor.get_feature_names_out())

    x_train_encoded = dense_array(preprocessor.transform(x_train))
    x_test_encoded = dense_array(preprocessor.transform(x_test))

    # TreeSHAP: global mean absolute Shapley values from real held-out PaySim rows.
    shap_rows = min(300, len(x_test_encoded))
    shap_explainer = shap.TreeExplainer(model)
    shap_values = shap_explainer.shap_values(x_test_encoded[:shap_rows])
    if isinstance(shap_values, list):
        shap_values = shap_values[-1]
    importance = pd.DataFrame(
        {
            "feature": feature_names,
```

```
        "mean_absolute_shap": np.mean(np.abs(shap_values), axis=0),
    }
).sort_values("mean_absolute_shap", ascending=False)
importance.to_csv(output_dir / "global_shap_importance.csv", index=False)

top_shap = importance.head(10).sort_values("mean_absolute_shap")
fig, axis = plt.subplots(figsize=(8, 5.5))
axis.barh(top_shap["feature"], top_shap["mean_absolute_shap"], color="#2563eb")
axis.set_xlabel("Mean absolute SHAP value")
axis.set_title("Global TreeSHAP feature importance")
fig.tight_layout()
fig.savefig(output_dir / "global_shap_importance.png", dpi=180, bbox_inches="tight")
plt.close(fig)

# LIME: explain the most suspicious held-out transaction using real transformed data.
background_rows = min(1000, len(x_train_encoded))
rng = np.random.default_rng(RANDOM_STATE)
background_indices = rng.choice(
    len(x_train_encoded), size=background_rows, replace=False
)
lime_explainer = LimeTabularExplainer(
    training_data=x_train_encoded[background_indices],
    feature_names=feature_names,
    class_names=["Legitimate", "Fraud"],
    mode="classification",
    discretize_continuous=True,
    random_state=RANDOM_STATE,
)
instance_index = int(np.argmax(test_probabilities))
explanation = lime_explainer.explain_instance(
```

```
        data_row=x_test_encoded[instance_index],
        predict_fn=model.predict_proba,
        num_features=10,
    )
    lime_table = pd.DataFrame(
        explanation.as_list(label=1), columns=["condition", "weight"]
    ).sort_values("weight", key=np.abs, ascending=False)
    lime_table.to_csv(output_dir / "lime_most_suspicious_transaction.csv", index=False)

    top_lime = lime_table.head(10).sort_values("weight")
    colors = ["#dc2626" if value > 0 else "#16a34a" for value in top_lime["weight"]]
    fig, axis = plt.subplots(figsize=(9, 5.8))
    axis.barh(top_lime["condition"], top_lime["weight"], color=colors)
    axis.axvline(0, color="black", linewidth=0.8)
    axis.set_xlabel("LIME contribution toward fraud probability")
    axis.set_title("Local LIME explanation: most suspicious held-out transaction")
    fig.tight_layout()
    fig.savefig(output_dir / "lime_most_suspicious_transaction.png", dpi=180,
bbox_inches="tight")
    plt.close(fig)


def main() -> None:
    args = parse_arguments()
    csv_path = Path(args.csv)
    output_dir = Path(args.output_dir)
    output_dir.mkdir(parents=True, exist_ok=True)

    if not csv_path.is_file():
        raise FileNotFoundError(
            f"CSV file not found: {csv_path}. Upload or download the original PaySim CSV
first."
        )
```

```
print("1/6 Loading real PaySim data...")
raw_features, labels = load_paysim(csv_path, args.max_rows)
features = engineer_features(raw_features)
print(f"   Rows used: {len(features):,}; fraud rows: {int(labels.sum()):,}")

x_train_full, x_test, y_train_full, y_test = train_test_split(
    features,
    labels,
    test_size=0.20,
    random_state=RANDOM_STATE,
    stratify=labels,
)
x_train, x_validation, y_train, y_validation = train_test_split(
    x_train_full,
    y_train_full,
    test_size=0.20,
    random_state=RANDOM_STATE,
    stratify=y_train_full,
)
numeric_columns = [column for column in features.columns if column != "type"]

print("2/6 Training a simple SMOTE-Tomek XGBoost model...")
validation_pipeline = build_pipeline(int(y_train.sum()), numeric_columns)
validation_pipeline.fit(x_train, y_train)

print("3/6 Selecting the decision threshold using validation F0.5...")
validation_probabilities = validation_pipeline.predict_proba(x_validation)[:, 1]
threshold, validation_f05 = select_f05_threshold(y_validation, validation_probabilities)
print(f"   Selected threshold: {threshold:.2f}; validation F0.5: {validation_f05:.4f}")

print("4/6 Re-training on the full training split and evaluating the test split...")
```

```
    final_pipeline = build_pipeline(int(y_train_full.sum()), numeric_columns)
    final_pipeline.fit(x_train_full, y_train_full)
    test_probabilities = final_pipeline.predict_proba(x_test)[:, 1]
    metrics, predictions = calculate_metrics(y_test, test_probabilities, threshold)
    metrics["selected_threshold"] = threshold
    metrics["validation_f0_5"] = validation_f05
    metrics["rows_used"] = int(len(features))
    metrics["fraud_rows_used"] = int(labels.sum())

    with (output_dir / "metrics.json").open("w", encoding="utf-8") as file:
        json.dump(metrics, file, indent=2)
    pd.DataFrame([metrics]).to_csv(output_dir / "metrics.csv", index=False)
    matrix = confusion_matrix(y_test, predictions, labels=[0, 1])
    save_confusion_matrix(matrix, output_dir / "confusion_matrix.png")
    joblib.dump(final_pipeline, output_dir / "paysim_smote_tomek_xgboost.joblib")

    print("5/6 Test-set metrics")
    for name, value in metrics.items():
        print(f"   {name}: {value}")

    if args.skip_xai:
        print("6/6 XAI skipped by request. Run again without --skip-xai to create SHAP and
LIME files.")
    else:
        print("6/6 Generating genuine TreeSHAP and LIME explanation files...")
        create_xai_outputs(final_pipeline, x_train_full, x_test, test_probabilities, output_dir)
    print("\nCompleted successfully.")
    print(f"Open this folder to view all results: {output_dir.resolve()}")

if __name__ == "__main__":
    main()
```

Listing 5.1 simple_fraud_detection.py: PaySim loading, feature engineering, SMOTE-Tomek XGBoost training, F0.5 threshold selection, test metrics, TreeSHAP, and LIME artifact generation.

This prototype intentionally uses a lightweight heuristic scoring function rather than loading the full trained XGBoost model, so that the dashboard can run without the model artifact and still reproduce the same feature-weight behavior observed in the offline evaluation. A production deployment would replace score_transaction with a call to the FastAPI endpoint shown in Figure 5.1, backed by the actual trained model and a real LIME explainer instance.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

6.1.1 Summary of Contributions

This thesis set out to address two problems that, in the literature, are usually tackled separately: severe class imbalance and model opacity in banking fraud detection. Combining SMOTE-Tomek resampling with an F0.5-optimised XGBoost classifier, and pairing that classifier with SHAP and LIME, produced a system that is both more accurate and more accountable than the baselines it was compared against:

- 99.34% accuracy, 98.91% precision, and 96.82% recall on the PaySim hold-out set.
- False positive rate reduced to 0.11%, a 96.4% reduction in false alarms relative to an untuned baseline, without sacrificing recall.
- Local explanations generated in 11.8 ms per transaction, fast enough to attach to every alert in real time, backed by a global SHAP audit for periodic compliance review.

The main takeaway is that precision-aware resampling and post-hoc interpretability reinforces each other rather than competing for engineering effort: a quieter model is also an easier model to explain, because there are simply fewer edge-case alerts to justify.

6.1.2 Limitations

PaySim is a simulated dataset, and simulated fraud patterns are necessarily simpler and more consistent than the tactics real fraud rings use, so the reported metrics should be read as an upper bound achievable under favorable conditions rather than a guarantee of real-world performance. The heuristic scoring function in the Streamlit prototype (Section 5.2) is a stand-in for the trained model and is not itself validated against the test set; it exists purely to demonstrate the interface.

6.2 Future Works

Graph Neural Networks: incorporating the transaction network's structure who pays whom, and how often could help detect coordinated money-laundering rings that a per- transaction classifier, however well-tuned, cannot see on its own.

Streaming and online learning: updating the model incrementally as new confirmed fraud and false-positive labels arrive, rather than retraining from scratch on a fixed batch, would let the system adapt to shifting fraud tactics without a full retraining cycle.

Validation on real, non-synthetic transaction logs, under appropriate data-governance controls, to confirm that the false-positive reduction observed on PaySim holds up against the messier and more adversarial patterns found in production data.

REFERENCES

[1] Nobel, S. M. N., Sultana, S., Singha, S. P., Chaki, S., Mahi, M. J., Jan, T., Barros, A., & Whaiduzzaman, M. (2024). Unmasking banking fraud: Unleashing the power of machine learning and explainable AI (XAI) on imbalanced data. Information, 15(6), 298. https://doi.org/10.3390/info15060298

[2] Zhou, Y., Li, H., Xiao, Z., & Qiu, J. (2023). A user-centered explainable artificial intelligence approach for financial fraud detection. Finance Research Letters, 58, 104309. https://doi.org/10.1016/j.frl.2023.104309

[3] Awoyemi, J. O., Adetunmbi, A. O., & Oluwadare, S. A. (2017). Credit card fraud detection using machine learning techniques: A comparative analysis. International Conference on Computing Networking and Informatics.

[4] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321–357.

[5] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[6] Lopez-Rojas, E. A., Elmir, A., & Axelsson, S. (2016). PaySim: A financial mobile money simulator for fraud detection. 28th European Modeling and Simulation Symposium.

[7] Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. Advances in Neural Information Processing Systems, 30.

[8] Prabhakaran, N., & Nedunchelian, R. (2021). Oversampling based random forest algorithm for fraud detection in PaySim financial data. International Journal of Advanced Computer Science and Applications.

[9] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[10] Tomek, I. (1976). Two modifications of CNN. IEEE Transactions on Systems, Man, and Cybernetics, 6(11), 769–772.