END-TO-END FINANCIAL FRAUD CASE STUDY & SOLUTION

Objective:
This project focuses on financial fraud that happens in banking systems. As we know, these frauds cause losses for both users and organizations. They not only result in money loss for customers but also damage the reputation of financial institutions.

The main goal of this project is to understand these fraud cases and build effective strategies and solutions that can help prevent and control fraudulent transactions.

About the Data:

In this project, the data was provided by an organization and contains around **63 lakh (6.3 million) transactions** collected over a period of **one month**. The dataset has **11 features** related to transaction behavior.

A sample of the dataset is shown in the image below.

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	9839.64	C1231006815	170136	160296.36	M1979787155	0	0	0	0
1	PAYMENT	1864.28	C1666544295	21249	19384.72	M2044282225	0	0	0	0
1	TRANSFER	181	C1305486145	181	0	C553264065	0	0	1	0
1	CASH_OUT	181	C840083671	181	0	C38997010	21182	0	1	0
1	PAYMENT	11668.14	C2048537720	41554	29885.86	M1230701703	0	0	0	0
1	PAYMENT	7817.71	C90045638	53860	46042.29	M573487274	0	0	0	0
1	PAYMENT	7107.77	C154988899	183195	176087.23	M408069119	0	0	0	0
1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M633326333	0	0	0	0
1	PAYMENT	4024.36	C1265012928	2671	0	M1176932104	0	0	0	0
1	DEBIT	5337.77	C712410124	41720	36382.23	C195600860	41898	40348.79	0	0
1	DEBIT	9644.94	C1900366749	4465	0	C997608398	10845	157982.12	0	0
1	PAYMENT	3099.97	C249177573	20771	17671.03	M2096539129	0	0	0	0
1	PAYMENT	2560.74	C1648232591	5070	2509.26	M972865270	0	0	0	0
1	PAYMENT	11633.76	C1716932897	10127	0	M801569151	0	0	0	0
1	PAYMENT	4098.78	C1026483832	503264	499165.22	M1635378213	0	0	0	0
1	CASH_OUT	229133.94	C905080434	15325	0	C476402209	5083	51513.44	0	0
1	PAYMENT	1563.82	C761750706	450	0	M1731217984	0	0	0	0
1	PAYMENT	1157.86	C1237762639	21156	19998.14	M1877062907	0	0	0	0
1	PAYMENT	671.64	C2033524545	15123	14451.36	M473053293	0	0	0	0
1	TRANSFER	215310.3	C1670993182	705	0	C1100439041	22425	0	0	0
1	PAYMENT	1373.43	C20804602	13854	12480.57	M1344519051	0	0	0	0
1	DEBIT	9302.79	C1566511282	11299	1996.21	C1973538135	29832	16896.7	0	0
1	DEBIT	1065.41	C1959239586	1817	751.59	C515132998	10330	0	0	0

The meaning of each column is explained below:

- **step** – Represents a unit of time in the real world. In this dataset, **1 step = 1 hour**. The total number of steps is **744**, which represents a **30-day simulation**.
- **type** – The type of transaction. It can be one of:
  CASH-IN, CASH-OUT, DEBIT, PAYMENT, TRANSFER.
- **amount** – The amount of money involved in the transaction in local currency.
- **nameOrig** – The customer who initiated the transaction.
- **oldbalanceOrg** – The initial balance of the sender before the transaction.
- **newbalanceOrig** – The balance of the sender after the transaction.
- **nameDest** – The customer who receives the transaction.
- **oldbalanceDest** – The initial balance of the receiver before the transaction. (This information is not available for merchant accounts, which start with "M".)
- **newbalanceDest** – The balance of the receiver after the transaction. (This is also not available for merchant accounts.)
- **isFraud** – Indicates whether the transaction is fraudulent. In this dataset, fraudulent agents attempt to take control of user accounts and empty funds by transferring money to another account and then cashing it out.
- **isFlaggedFraud** – A rule-based system used by the business to flag suspicious transactions. A transaction is flagged when a **single transfer is greater than 200,000** in amount.

Project Workflow:

After understanding the problem and the data, the project was carried out using the following workflow:

1. **Basic Preprocessing**
   - Handling a very large dataset
   - Data cleaning and basic preparation
2. **Exploratory Data Analysis (EDA)**
   - Univariate analysis
   - Bivariate analysis
   - Multivariate analysis
3. **EDA-Based Data Cleaning and Feature Preparation**
   - Removing multicollinearity
   - Feature engineering
4. **Model Building**

    - Model selection
    - Threshold selection
    - Handling class imbalance
    - Model tuning
    - Model evaluation
5. **Model Explainability**
    - Using SHAP to understand model behavior
    - Model weakness Analysis
6. **Model Improvements**
    - Rule-based system
    - Recall booster model
    - Re-evaluation of performance
7. **Final Model Selection and Evaluation**
    - Choosing the best-performing model based on business and model trade-offs.

1. Basic Preprocessing :

1. Handling a very large dataset
The dataset contained approximately **62 lakh rows** and **11 features**, with a total size of around **500 MB**. To handle this efficiently:

- First, I optimized the data types of each column (for example, using int32, float32, and boolean instead of default types). This significantly reduced memory usage.
- Second, I observed that whenever a **merchant account** was involved in a transaction, there was **0% chance of fraud**. To reduce unnecessary data and memory overhead, I removed these rows, which reduced the dataset by about **25%**.

By applying these two steps, the dataset size was reduced from **534 MB to 264 MB**, making further analysis and modeling much more manageable.

2. Removing unnecessary columns
I also removed columns such as **nameOrig** and **nameDest** because they contain individual customer names, which are not useful for numerical analysis or machine learning modeling.

2. Exploratory Data Analysis (EDA) :

Before performing EDA, the dataset still contained around **42 lakh rows**, which made plotting and summarization very slow. To handle this, I used a **sampling strategy**:

- I included **all fraud cases**
- I randomly sampled **25% of non-fraud cases**

This allowed me to analyze patterns efficiently without losing important fraud information.

---

1. Univariate Analysis

For numerical columns, I performed descriptive analysis including:
mean, median, 25%, 50%, 75%, min, and max, along with distribution plots to understand:

- Data distribution
- Skewness
- Presence of outliers

For categorical columns, I analyzed:

- Category counts
- Category distributions

This helped in understanding how values are spread across different transaction types.

---

2. Bivariate and Multivariate Analysis

I analyzed the relationship between:

- Numerical and categorical variables
- Categorical variables and the fraud label

For example:

- **diffOrg vs isFraud**
- **Transaction time vs isFraud**
- Correlation between numerical features

This helped in understanding how fraud behavior changes across different categories and time periods.

Key insights from EDA

From this analysis, I obtained several important insights:

- Most fraud transactions happen during **late-night hours**
- Fraud cases mainly occur in **CASH-OUT** and **TRANSFER** transactions
- High values of **diffOrg** and **diffDest** are strongly associated with fraud

These insights gave me a **clear understanding of fraud patterns** and guided the next steps of feature engineering and model building.

3. EDA-Based Cleaning and Feature Engineering

Based on insights from EDA, I performed feature engineering and data cleaning to make the data more meaningful for modeling.

1. Feature Engineering

a) diffOrg and diffDest
While checking multicollinearity among numerical features, I found that:

- newbalanceOrg and oldbalanceOrg
- newbalanceDest and oldbalanceDest

were **highly collinear**. This makes sense because they represent the same account balance before and after a transaction.

Instead of keeping both values, I created:

- **diffOrg = newbalanceOrg – oldbalanceOrg**
- **diffDest = newbalanceDest – oldbalanceDest**

This captured the **actual balance change** in a single feature and removed redundant information.

---

b) Transaction Time Feature

The **step** column represents time in hours over a one-month period (around 700 steps).
To make this more meaningful:

- I converted step into **time of day** by using step % 24
- Then I categorized it into:
    - **Morning**
    - **Afternoon**
    - **Night**
    - **Late Night**

This allowed the model to learn **time-based fraud patterns** instead of raw step values.

4. Model Building

For model building, I used **all fraud cases** and **50% of non-fraud cases** to reduce class imbalance while still keeping enough normal transactions.

Throughout the model development, I used a **machine learning pipeline** that consisted of three main steps:

1. **Preprocessing**
    - Prepared the data for training
    - Applied **log transformation** on numerical columns to reduce skewness

    - Applied **one-hot encoding** on categorical columns
2. **Imbalance Handling**
    - Applied different imbalance handling techniques such as:
        - Random under-sampling
        - SMOTE
        - SMOTE-Tomek
        - ADASYN
3. **Model Training**
    - Trained the model on the processed and balanced data

---

1. Model Selection

I trained multiple classification models using default parameters, including:

- Logistic Regression
- KNN
- Decision Tree
- Random Forest
- Extra Trees
- Gradient Boosting
- AdaBoost
- Bagging
- SVM
- Gaussian Naive Bayes
- XGBoost
- LightGBM

Although most models showed very high **accuracy**, their **precision was very poor**, which meant the fraud predictions were not reliable. This happened because the dataset was highly imbalanced, making accuracy a misleading metric.

---

2. Optimal Threshold Selection

To solve this issue, I applied **optimal threshold selection** instead of using the default 0.5 cutoff.

The process was:

1. Train the model

2. Compute **precision and recall** at different probability thresholds using precision_recall_curve
3. Compute **F1 score** at each threshold using

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

4. Select the threshold where **F1 score is maximum**

This allowed me to find the best balance between **precision and recall**, instead of relying on misleading accuracy.

---

3. Final Model Choice

After applying optimal thresholding, I compared all models again.
The two best performing models were:

- **XGBoost**
- **LightGBM**

Both achieved approximately:

- **~93% precision**
- **~72% recall**
- **~81% F1-score**

I selected **LightGBM** as the primary model because it:

- Trains faster
- Uses less memory
- Scales better on large tabular datasets

XGBoost was kept as a **backup model**.

Model Tuning & Validation

After selecting **LightGBM** as the final model, I further improved its performance using **hyperparameter tuning with Optuna**.

The objective of tuning was not accuracy (because the dataset is highly imbalanced), but to **maximize F1-score**, which balances both **precision and recall**.

The tuning process searched over a wide range of LightGBM parameters and the best configuration found was:

learning_rate = 0.0545

num_leaves = 74

max_depth = 10

min_child_samples = 91

subsample = 0.401

colsample_bytree = 0.787

reg_alpha = 1.209

reg_lambda = 4.968

n_estimators = 300

Best F1 Threshold = 0.986

Imbalance Handling

Because fraud is extremely rare, I tested multiple imbalance handling techniques:

- Random Under-Sampling
- SMOTE
- SMOTE-Tomek
- ADASYN

Among all of them, **SMOTE produced the best precision–recall balance**, so it was used in the final training pipeline.

Final Test Set Performance

Using the tuned LightGBM model with the optimal threshold:

Confusion Matrix (Test Set)

[[227431 54]

[317 1077]]

This means:

- 1077 frauds were correctly detected
- 54 normal transactions were wrongly flagged
- 317 frauds were missed
- 227431 normal transactions were correctly allowed

Classification Report (Test Set)

Fraud Precision = 95%

Fraud Recall = 77%

Fraud F1-score = 85%

Validation Set Performance

The same tuned model was evaluated on the **validation dataset** to check if it generalizes well.

Confusion Matrix (Validation Set)

[[420173 119]

[174 647]]

Classification Report (Validation Set)

Fraud Precision = 84%

Fraud Recall = 79%

Fraud F1-score = 82%

Conclusion from Validation

The validation performance is **very close to the test performance**:

- Precision ≈ 95–84%
- Recall ≈ 77–79%
- F1 ≈ 85-82%

This confirms that the model is **not overfitting** and is **stable across unseen data**.

Final Model Choice

Based on:

- Strong fraud precision
- Good recall
- Stable validation results
- Fast training and low memory usage

LightGBM with SMOTE and optimized threshold was selected as the final production model.

---

5. Model Explainability (Using SHAP)

Even after extensive tuning, the model performance reached a limit. Since this is a **financial fraud detection system**, the predictions **cannot be a black box** — we must understand **why** a transaction is flagged as fraud or non-fraud.

To achieve this, I used **SHAP (SHapley Additive exPlanations)**.

SHAP is a powerful explainability library based on **Shapley values** from game theory. It explains model predictions by adding features one by one and measuring how each feature **pushes the prediction toward fraud or toward non-fraud**.
It allows both:

- **Global explanation** → how the model behaves overall
- **Local explanation** → why a single transaction was predicted as fraud

---

Key Findings from SHAP Analysis

1. Primary Fraud Driver

- Sender account balance change (num__diffOrg) is the most dominant feature.
- Large reductions in sender balance strongly push predictions toward fraud.
- Small reductions or balance increases shift predictions toward non-fraud.
- This feature acts as the main decision boundary for the model.

2. Secondary Fraud Signals

- Receiver balance change (num__diffdest) amplifies fraud risk when there is a high or abnormal balance increase.
- Moderate or normal receiver balance changes have little influence.
- This indicates the model checks whether money is actually accumulating at the receiver.

3.Transaction Amount Behavior

- Low transaction amounts slightly push predictions toward fraud.
- High transaction amounts push predictions toward non-fraud.

- Amount does not independently decide fraud; it only fine-tunes the confidence.

4.Transaction Type Influence

- CASH_IN transactions strongly push predictions toward non-fraud, acting as a corrective signal.
- DEBIT transactions generally indicate non-fraud behavior.
- TRANSFER and CASH_OUT types have weak or unreliable influence and rarely affect the final decision.

5. Time-Based Features

- Late-night transactions show a noticeable push toward fraud.
- Other time slots (morning, afternoon, evening) have minimal or no impact.

Overall Model Behavior :

The SHAP analysis shows that the model primarily focuses on abnormal money outflow from the sender as the strongest fraud signal. It then evaluates receiver balance changes to confirm whether funds are accumulating suspiciously. Transaction type and amount are used only to adjust confidence, not to make the core fraud decision. Overall, fraud detection in the model is driven by money flow behavior rather than transaction metadata.

Model Weakness Analysis

This analysis focuses on **why the model misses certain fraud cases**, even though those transactions are actually fraudulent. Instead of looking at metrics, the goal here is to understand the **behavioral gap between detected and missed fraud**.

When the Model Successfully Detects Fraud

The model performs well when fraud behavior is **clearly abnormal**.

In detected fraud cases, transactions usually show:

- **Noticeable money outflow from the sender** — the sender balance drops sharply.
- **Visible money accumulation at the receiver** — the receiver balance increases clearly.
- **Medium to high transaction amounts**, which stand out from regular activity.

Interpretation:
When money movement is aggressive and obvious, the model has no confusion.

A large sender loss combined with a clear receiver gain creates a **strong and unambiguous fraud signal**, which the model confidently captures.

When the Model Fails to Detect Fraud

Missed fraud cases tell a very different story.

In these cases:

- **Transaction amounts are small to moderate**, not extreme.
- **Sender balance reduction is mild**, without a sharp drop.
- **Receiver balance increase is weak**, appearing almost normal.

Interpretation:
These fraud transactions are **behaviorally subtle**.
They do not look dangerous in isolation and closely resemble legitimate user activity.
As a result, the model struggles to separate them from normal transactions and often classifies them as non-fraud.

What This Reveals About Model Behavior

The model is **heavily dependent on strong money-flow signals**:

- It expects **clear sender loss**.
- It expects **clear receiver gain**.

If both signals are weak at the same time, the model's confidence drops sharply — even if the transaction is fraudulent.

In simple terms:

The model is good at catching loud fraud, but weak at catching quiet fraud.

Core Weakness Pattern (Key Insight)

Fraud is most likely to be missed when:

- The amount is **not large enough to stand out**,
- The sender does **not lose a significant portion of balance**, and
- The receiver does **not show a strong balance jump**.

These cases represent **stealth fraud**, where attackers intentionally keep transactions small and gradual to avoid detection.

6. Model Improvement Trials

After deeply understanding the data and the model's weaknesses, I tried to overcome these limitations using two advanced approaches.

1. Rule-Based + Model System

In this approach, I combined the **LightGBM model** with a **rule-based layer** designed specifically for the model's weak zone.

The model was using a very **strict threshold (~0.99)**, which made it very precise but caused it to miss some frauds that looked normal.

So I created a **dynamic threshold system**:

- When **numerical anomalies are very high**, the model uses the **strict threshold**.
- When numerical signals are **moderate**, but **supporting signals** are present (such as **late-night**, **CASH-OUT**, or **TRANSFER**), the system uses a **lower threshold**.

This was designed to allow the model to detect **normal-looking frauds** without fully relying on numerical anomalies.

---

2. Recall Booster Model

In this approach, I trained a **second LightGBM model** on a special dataset:

- Fraud cases that were **missed by the main model**
- Non-fraud cases that were **similar to those missed frauds**

The idea was:

- The **main model** catches **abnormal frauds**
- The **recall booster** catches **normal-looking frauds**

Why These Approaches Did Not Work

Both approaches failed because of the **strong overlap between normal-looking fraud and genuine transactions**.

When the system becomes more aggressive to catch these frauds:

- It starts flagging many **genuine transactions as fraud**
- Precision drops sharply

When the system becomes more conservative:

- It misses **normal-looking fraud**
- Recall drops

So the system gets trapped in a **precision–recall trade-off**.

This confirmed that:

The limitation is not in the model, but in the **data itself**, where fraud and non-fraud look too similar in some regions.

The Solution :

After understanding the model's weaknesses and testing multiple improvement strategies, I designed a **practical and business-ready solution**.

Since some fraud transactions look **very similar to genuine transactions**, no automated model can detect them with high confidence without increasing false alarms.
So instead of forcing the model to make risky decisions, I introduced a **manual review system**.

The solution works as:

Transaction → Model → Review → Manual Check → Final Decision

In this system:

- When the model is **highly confident** that a transaction is fraud, it is **blocked**.
- When the model is **highly confident** that a transaction is normal, it is **allowed**.
- When the model is **not confident** (normal-looking but suspicious cases), the transaction is sent to a **manual review queue**.

Human analysts then review these cases using additional information and decide whether to approve or block them.

This approach allows the system to:

- Catch **more fraud**
- Avoid **blocking genuine users**
- Handle **normal-looking fraud** that machine learning alone cannot detect

This makes the overall fraud prevention system both **accurate and realistic for real-world use**.

Here is your **final section polished, clean, and professional**, keeping your numbers and logic exactly as they are:

---

7. Final Model Selection and Evaluation

After designing the review-based solution, I evaluated three systems:

- **Single LightGBM model**
- **LightGBM + Rule-Based System**
- **LightGBM + Recall Booster Model**

The two best-performing approaches were:

- **Single LightGBM**
- **Combined Model (Main + Booster)**

Their business-level performance is shown below.

Model-1: Combined Model (Main Model + Booster)

(Confusion summary: **ALLOW = 692,771 | BLOCK = 132 | REVIEW = 63,597**)

Numbers Used

- **True Fraud Blocked (TP)** = 1,144
- **False Blocks (FP)** = 132
- **Missed Frauds (FN)** = 17
- **Frauds Sent to Review** = 342
- **Non-Frauds Sent to Review** = 63,597
- **Total Transactions** = **756,500**

Metrics

- **Precision** = 1144 / (1144 + 132) = **0.8966 (~89.7%)**
- **Recall** = 1144 / (1144 + 17) = **0.9853 (~98.5%)**
- **Review Rate** = (63,597 + 342) / 756,500 = **0.0846 (~8.46%)**

Interpretation

- The system **almost never misses fraud**.
- When a transaction is blocked, it is **correct about 90% of the time**.
- However, about **8.5% of all transactions** must be manually reviewed.

Model-2: Single LightGBM Model

(Confusion summary: **ALLOW = 738455 | BLOCK = 29 | REVIEW = 18016**)

Numbers Used

- **True Fraud Blocked (TP)** = 1,102
- **False Blocks (FP)** = 29
- **Missed Frauds (FN)** = 74
- **Frauds Sent to Review** = 327

- **Non-Frauds Sent to Review** = 18016
- **Total Transactions** = **756,500**

Metrics

- **Precision** = 1102 / (1102 + 29) = 0.9743 (~97.4%)
- **Recall** = 1102 / (1102 + 74) = 0.9371 (~93.7%)
- **Review Rate** = (327 + 18,016) / 756,500 = 0.0242 (~2.42%)

Interpretation

- The model maintains **high precision**.
- It detects most fraud cases, though fewer than the combined system.
- Only **~2% of transactions** require manual review.

Final Decision: Loss–Cost Optimized Strategy

These results clearly show a **trade-off**:

- **Higher recall** → **more manual reviews**
- **Lower review volume** → **more missed fraud**

Since every action has a cost:

- **Missed fraud** → financial loss
- **Manual review** → operational cost
- **False block** → customer dissatisfaction

The goal is to **minimize total business cost**, not just maximize recall.

Final Model Choice

Based on this loss–cost perspective:

- The **single LightGBM model** provides the **best balance** between:
    - Fraud detection
    - Review workload
    - Customer impact
- It achieves **strong precision** while keeping **review volume low**
- The combined system catches slightly more fraud, but at a **much higher operational cost**

Therefore, the **Single LightGBM model** is selected as the **final production system**.
