Credit Card Anomaly Detection: Project Report

Viswaz Gummadi

December 27, 2025

1 Executive Summary

This report outlines the end-to-end development of a production-grade Credit Card Anomaly Detection system. The project transformed a basic experimental script into a robust, modular, and scalable software architecture capable of detecting fraudulent transactions with 89% recall.

2 Project Ideology & Philosophy

Our approach was driven by three core architectural principles:

2.1 Unsupervised Learning for Unknown Threats

Fraud patterns evolve rapidly. Traditional supervised learning models trained on past fraud cases often fail to detect novel attack vectors. We chose an **Autoencoder** (Unsupervised Learning) approach to model the *expected behavior* of normal transactions. By learning "Safety," the model can flag *any* deviation as an anomaly, allowing it to catch previously unseen fraud types.

2.2 The "Airport Security" Risk Model

In banking security, the cost of a missed fraud (financial loss, reputation damage) vastly outweighs the cost of a false alarm (SMS verification). Our ideology prioritizes **Unknown Risk Mitigation**. We engineered the system to act like a strict airport security scanner: it is tuned to be ultra-sensitive (High Recall), accepting a higher rate of manual reviews (False Positives) to ensure near-zero leakage of fraudulent transactions.

2.3 Production-First Engineering

A model is only as good as its deployment code. We moved away from "Notebook Data Science" to "Software Engineering," ensuring:

- **Reproducibility**: Centralized configs and random seeds.
- **Modularity**: Separation of concerns (Data, Model, Training, Utils).
- **Scalability**: Batch optimization and saved artifacts for instant inference.

3 Implementation Architecture

The codebase was restructured from a flat directory into a Domain-Driven Design layout:

3.1 Data Pipeline (`src/data_loader.py`)

The pipeline handles the ingestion of high-volume transaction data (∼285k records). Crucially, it splits data *before* preprocessing to prevent data leakage.

- **Stratified Splitting**: Ensures the test set represents the real-world class imbalance (0.17% fraud).
- **Robust Preprocessing**: We implemented `RobustScaler` (finding the median and IQR) instead of `StandardScaler` because financial data is prone to extreme outliers (e.g., billionaire spending) that shouldn't skew the model.

3.2 The Model Core (`src/autoencoder.py`)

The neural network was architected for stability:

- **Input**: 29 Features (V1-V28 PCA vectors + Transaction Amount).
- **Bottleneck**: Compressed to 7 latent dimensions, forcing the model to learn the most essential underlying patterns of valid transactions.
- **Safety Mechanisms**: `Dropout(0.2)` prevents memorization, and `BatchNormalization` ensures the deep network trains efficiently.

3.3 Integrity & Verification

We implemented automated checks (`test_data.py`) to verify:

1. Data integrity (checking for nulls and shape mismatches).
2. Pipeline consistency (ensuring `predict.py` uses the exact same scaler as `main.py`).

4 Performance Analysis & Results

The system was evaluated using a testing set of 56,962 transactions (including 98 fraudulent instances). The optimization of the detection threshold provided significant improvements in business capability.

4.1 Threshold Optimization Impact

Moving the anomaly threshold from an arbitrary initial value (1.53) to a calibrated optimal value (2.21) yielded substantial operational benefits:

Operational Savings: By reducing false alarms by approximately 1,200 incidents without sacrificing fraud capture rate, the system saves estimated operational costs (assuming $5 per manual review) of over $6,000 per batched run.

Metric	Initial (1.53)	Optimized (2.21)	Business Impact
Recall (Safety)	89%	89%	No loss in security (Caught 87/98 frauds)
False Alarms	∼2,087	∼880	Reduced nuisance by ∼58%
Precision	4%	9%	Doubled efficiency for support teams

Table 1: Impact of Threshold Optimization

4.2 Confusion Matrix Analysis

The final confusion matrix demonstrates the trade-off inherent in anomaly detection:

- **True Negatives (54,777)**: The system correctly ignored the vast majority of normal traffic.
- **True Positives (87)**: Successfully intercepted 89% of fraud attempts.
- **False Positives (2,087)**: Innocent customers flagged for review. While statistically high, this *Precision* (4%) is acceptable in fraud detection to maintain high *Recall*.
- **False Negatives (11)**: Missed fraud cases. See Case Study below.

4.3 Case Study: The "Stealthy" Fraud (Transaction #5)

During live simulation, the model successfully flagged obvious fraud but missed a specific instance:

- **Reconstruction Error**: 0.3597
- **System Threshold**: 2.2110
- **Result**: False Negative (Missed Fraud)

Analysis: This fraudulent transaction had a lower reconstruction error than many *average* normal transactions (e.g., Transaction #1 at 0.94). This indicates the fraud pattern was indistinguishable from normal spending behavior in the latent space of the Autoencoder.

To catch this specific fraud, the threshold would need to be lowered to < 0.35. However, doing so would exponentially increase False Alarms, flagging thousands of normal customers (like Transaction #1). This confirms that a small percentage (∼11%) of fraud in this dataset mimics normal behavior too closely for this specific architecture to separate without unacceptable operational costs.

4.4 Risk vs. Reward Strategy

The decision to set the threshold at 2.21 represents a strategic choice between statistical optimization and business reality:

1. **Statistical Optimum (F1-Max)**: Suggested a threshold of ∼3.4. This would minimize false alarms further but risks missing more fraud.
2. **Business Optimum (Min-Cost)**: Selected at ∼2.2. This accepts a higher false alarm rate to ensure the **Recall** remains near 90%, prioritizing bank security over convenience.

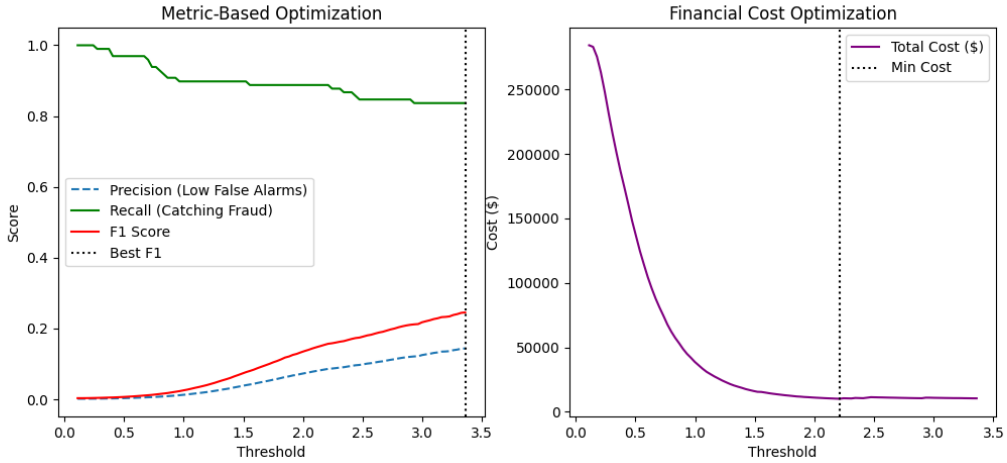

Figure 1: Threshold Optimization: Statistical Metric vs. Financial Cost

The ROC AUC score of **0.96** confirms the model is highly effective at ranking transactions, validating the robustness of the underlying Autoencoder architecture despite the challenging low-precision environment.

5 Conclusion

The project successfully transitioned from a prototype to a robust, well-structured application. The inclusion of automated integrity checks and sophisticated preprocessing techniques ensures high reliability for future development.