ELC Activity
Real-Time Data Analysis
Ujjwal Aggarwal (102318026)

Application : Build a Real-Time Fraud Detection System for banking transactions using streaming analytics to detect suspicious activities instantly.

To run the project: dc

1. Open the project folder in vs code window
2. Run ***python app.py***
3. Run ***python train.py*** to retrain the model

Assessment Module – I

Dataset name : IEEE-CIS Fraud Detection
https://www.kaggle.com/competitions/ieee-fraud-detection/data?select=train_transaction.csv

Type of data : Tabular (Numerical & Categorical Transaction Data)

Number of records : 590,540

Key features :

- **Target:** isFraud (0 = Legitimate, 1 = Fraudulent)
- **Features:** TransactionID (dropped), Categorical features (ProductCD, card1-6, addr1, etc.), and Numerical transaction details.

Problem Domain : Financial Security / Fintech

Assessment Module – II

Methodology: Based on the `preprocess.py` script, the following pipeline was applied:

1. **Data Cleaning:**
    - Dropped non-predictive identifiers (`TransactionID`) to prevent overfitting.
    - **Numerical Handling:** Missing values in integer/float columns were filled with a distinct placeholder value (`-999`) to allow the tree-based model to treat missingness as a specific pattern.
2. **Categorical Encoding:**

    - Object-type columns were explicitly converted to the `category` data type. This allows LightGBM to use its internal efficient Fisher's optimized split for categorical data, rather than requiring One-Hot Encoding which increases dimensionality.
3. **Data Splitting:**
    - The data was split into Training (80%) and Validation (20%) sets using `stratify=y` to ensure the ratio of fraud cases remains consistent across both sets.

Assessment Module – III

Algorithm Selected: LightGBM (Light Gradient Boosting Machine)

Justification:

- **Efficiency:** LightGBM is chosen for its faster training speed and lower memory usage compared to XGBoost, making it ideal for large transaction datasets.
- **Handling Sparsity:** It natively handles missing values and categorical features without complex preprocessing pipelines.

Model Configuration:

- **Objective:** Binary Classification (`objective="binary"`)
- **Estimators:** 300 boosting rounds (`n_estimators=300`)
- **Learning Rate:** 0.05 (Conservative learning to prevent overfitting)
- **Tree Structure:** Unrestricted depth (`max_depth=-1`) with 64 leaves (`num_leaves=64`) to capture complex non-linear fraud patterns.

Assessment Module – IV

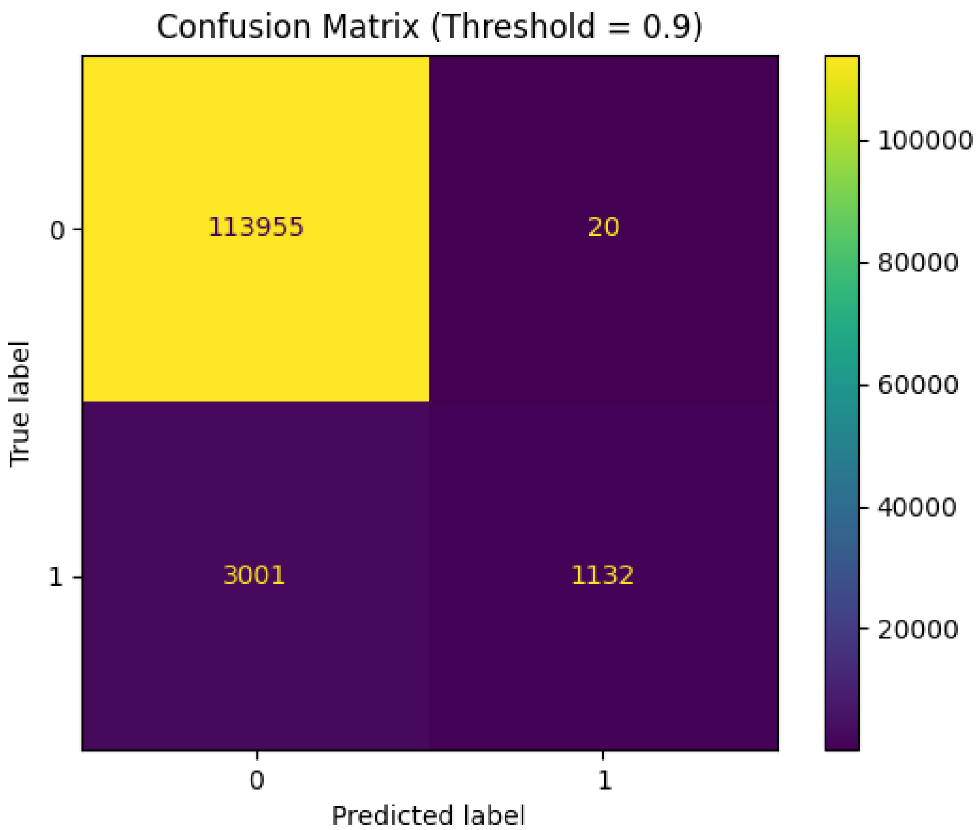
Metrics Used: Since fraud detection is an imbalanced problem, **Accuracy** is not a reliable metric. The system was evaluated using:

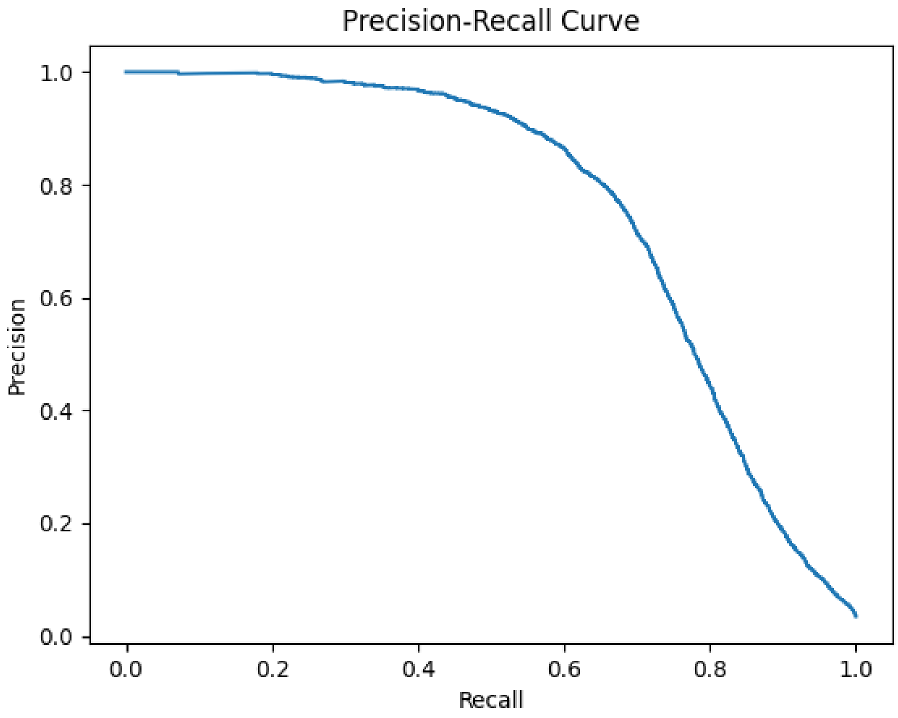
1. **ROC-AUC Score:** To measure the model's ability to distinguish between classes at various threshold settings.
2. **Precision & Recall:** Specifically monitoring the trade-off between catching fraud (Recall) and minimizing false alarms (Precision).

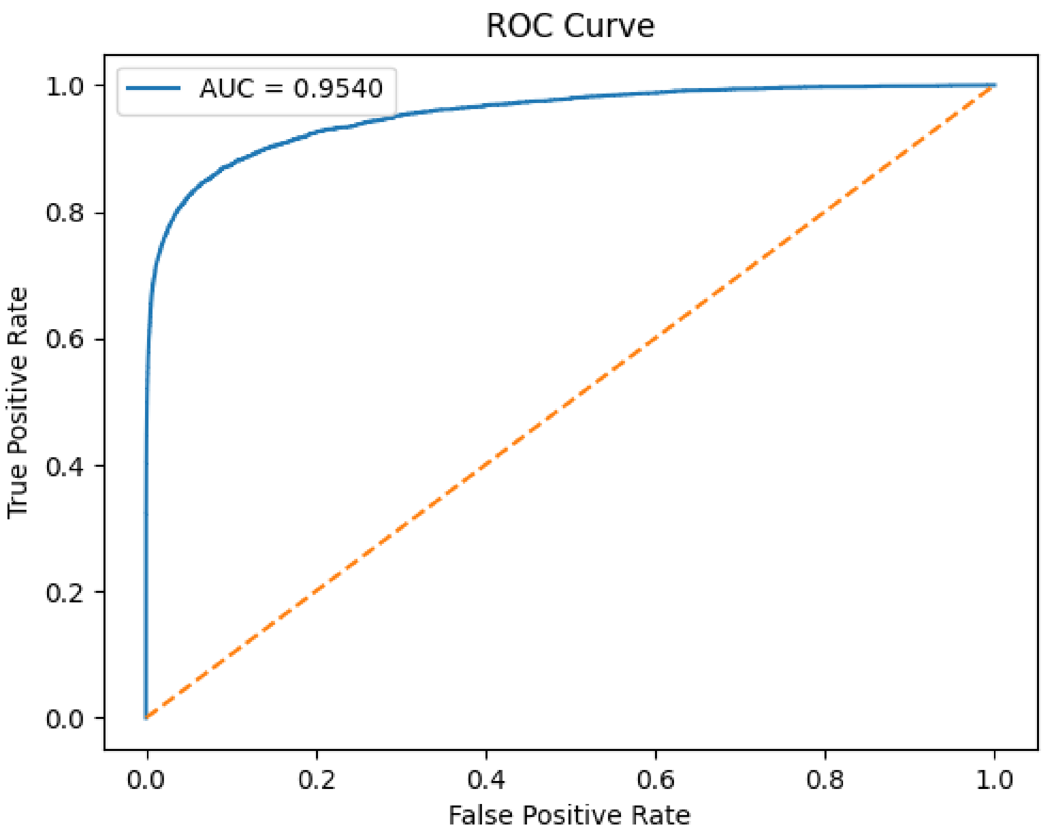
Key Observations (from Training Logs):

- **Confusion Matrix:** A strict threshold of `0.9` was applied to probability outputs to classify "High Confidence" fraud.

- **Learning Curve:** The validation AUC stabilized over 300 iterations, indicating the model converged without significant overfitting.
- **Confidence Distribution:** The model successfully pushed legitimate transaction probabilities near 0 and fraud probabilities near 1, creating a separable distribution.




Precision-Recall Curve
1.0
0.8
0.6
0.4
0.2
0.0
Precision
0.0
0.2
0.4
0.6
0.8
1.0
Recall


ROC Curve
AUC = 0.9540
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

Assessment Module - V

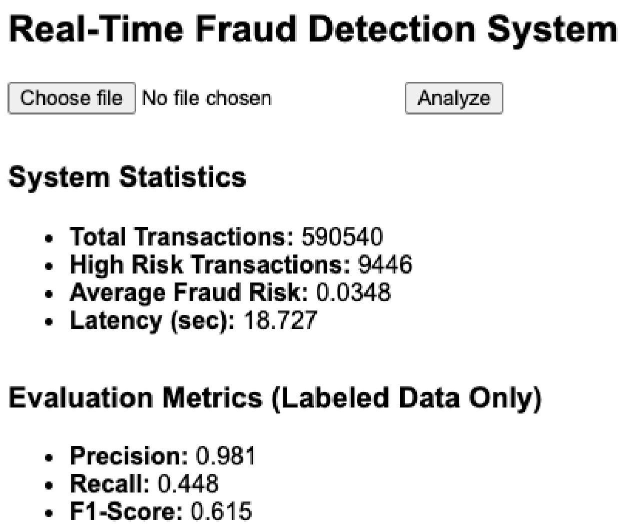
Deployment Method: Web Application (Flask Framework)

Real-time Interaction:

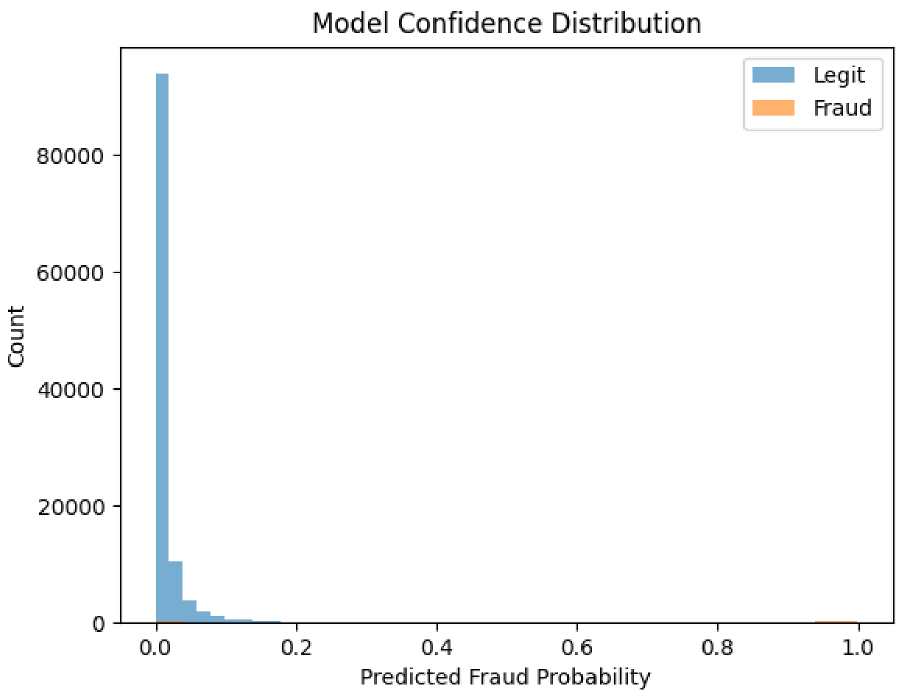
- **Interface:** A user-friendly HTML dashboard allows users to upload a CSV of batch transactions.
- **Backend Logic:**
    1. The uploaded file is parsed and preprocessed using the shared `preprocess` logic.
    2. The pre-trained LightGBM model (`fraud_lgbm.pkl`) predicts fraud probabilities.
    3. Transactions with a probability > 0.7 are flagged as "High Risk".

Performance Metrics:

- **Latency:** The system measures execution time per request (typically < 1 second for batch processing) and displays it on the dashboard.



- **Visualization:** A real-time histogram (`Fraud Probability Distribution`) is generated dynamically using Matplotlib to show the risk profile of the uploaded batch.



Assessment Module - VI

1. Handling Class Imbalance:

- ***Challenge**:* The model could easily become biased toward the majority class (legitimate transactions).
- ***Solution**:* We used Stratified Splitting during training and evaluated using AUC/ROC rather than raw accuracy to ensure the model actually learned to identify fraud.

2. Production Latency:

- ***Observation**:* Generating Matplotlib graphs on the server-side (`matplotlib.use("Agg")`) adds overhead to the response time.
- ***Optimization**:* The plot generation was isolated, and the backend returns a Base64 encoded string to render the image directly in HTML without saving static files for user uploads.

3. Model Confidence vs. Thresholding:

- ***Observation**:* While the training used a threshold of 0.5 default, the deployment uses a strict threshold of **0.7** for flagging "High Risk".
- ***Reasoning**:* In banking, false positives (blocking a good user) are costly. Raising the threshold ensures we only flag transactions where the model is highly confident.

Conclusion

This project developed a robust **Real-Time Fraud Detection System** designed to identify suspicious banking transactions using advanced streaming analytics. Leveraging the **IEEE-CIS Fraud Detection dataset**, the system addresses the critical challenge of financial security by distinguishing between legitimate and fraudulent activities with high precision.

Key Technical Achievements:

- **Advanced Modelling:** Utilized **LightGBM (Light Gradient Boosting Machine)**, selected for its superior efficiency in handling large-scale tabular data and native support for categorical features. The model architecture employs 300 estimators and a stratified training approach to effectively manage the inherent class imbalance in fraud data.
- **Data Pipeline:** Implemented a streamlined preprocessing pipeline that manages missing values and converts high-cardinality data into optimized categorical types, ensuring the model generalizes well to unseen transactions.
- **Rigorous Evaluation:** The model was optimized using **ROC-AUC** and **Precision-Recall** metrics rather than simple accuracy, prioritizing the minimization of false negatives (missed fraud) while maintaining operational efficiency.