Knight Foundation School of Computer & Information Sciences

Spring 2026 Senior Design Project

FraudShield

Students: Monisha Kummari, Gabriel Xirinachs, Chloe Lindsay, Maria Vega Gonzalez, Jabbar Morla
Instructor/Faculty: *Masoud Sadjadi*, Florida International University

Problem

Credit card fraud costs the global economy billions of dollars annually, with sophisticated attackers constantly finding ways to bypass standard security measures. Small-scale fraudulent transactions often go undetected because they blend in with normal consumer behavior. FraudShield addresses this by utilizing an amply detection algorithms that 'learn' what normal behavior looks like, allowing the system to flag suspicious outliers that traditional methods miss.

Current System

Existing Fraud Detection Landscape:
Traditional fraud detection relies on rule-based systems and manual reviews using fixed thresholds. These approaches lack adaptability, resulting in high false positives and difficulty detecting evolving fraud patterns.

The Kaggle Credit Card Fraud dataset (284,807 transactions, 0.17% fraud) is widely used for benchmarking, but accessible real-time prediction tools for end users are limited. Severe class imbalance further complicates accurate detection.

Dataset:
- Source: Kaggle Credit Card Fraud Detection
- 30 features (V1–V28 PCA-transformed, Amount, Time)
- Preprocessing: StandardScaler and SMOTE for class balancing

Verification

Evaluated end-to-end system performance by testing trained models on unseen data resulting in reliable classification of fraudulent and legitimate transactions.

Compared model effectiveness by analyzing precision, recall, and confusion matrices allowing identification of the best performing model for fraud detection.

Validated system stability and reliability by performing full pipeline testing from data input to prediction output confirming consistent performance across all stages.

REFERENCES

Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. Decision Support Systems, 50(3), 602-613. https://doi.org/10.1016/j.dss.2010.08.008

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.

Dal Pozzolo, A., Caelen, O., & Bontempi, G. (2015). Credit card fraud detection [Dataset]. Kaggle. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

FastAPI. (n.d.). FastAPI documentation. https://fastapi.tiangolo.com/

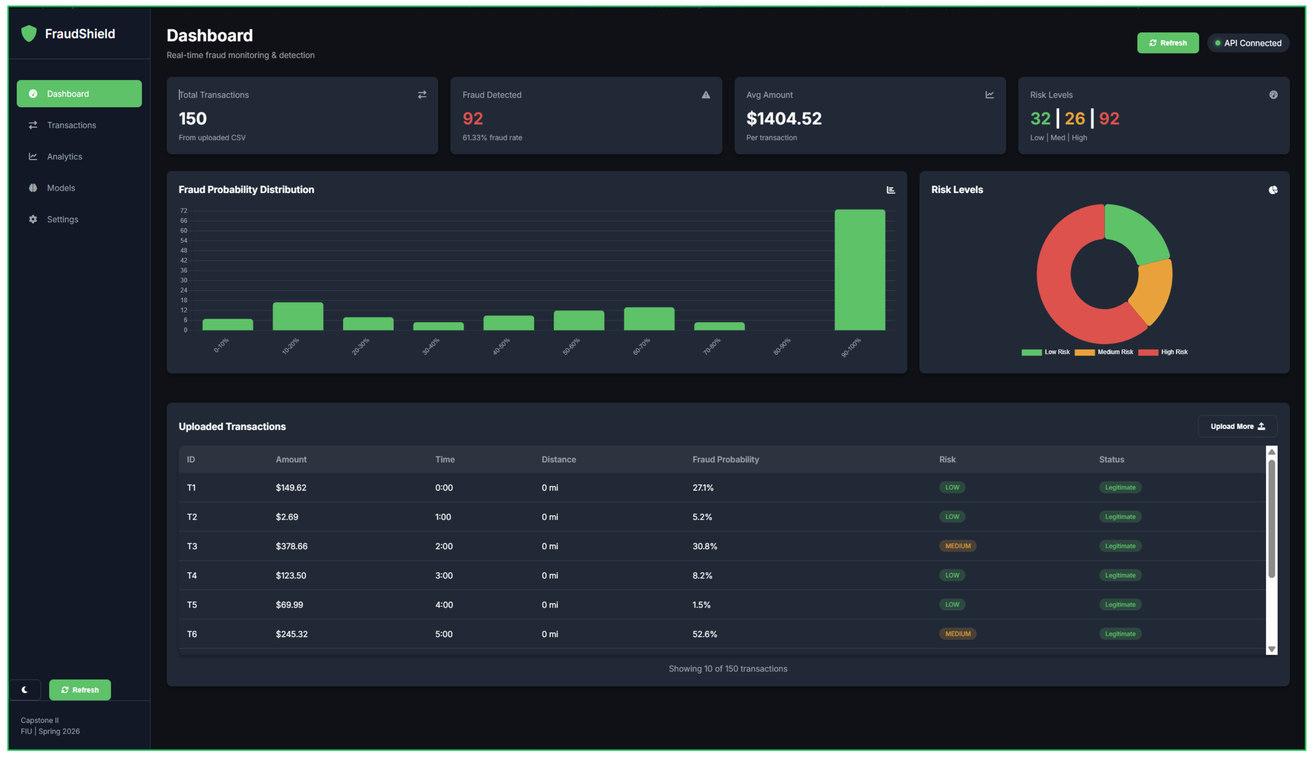
Fernández, A., García, S., Herrera, F., & Chawla, N. V. (2018). SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. Journal of Artificial Intelligence Research, 61, 863-905. https://doi.org/10.1613/jair.1.11192

Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). O'Reilly Media.

Chart.js contributors. (2025). Chart.js documentation. https://www.chartjs.org/docs/latest/

Object Design

Module	Responsibility
data_prep.ipynb	Load, clean, scale, split, SMOTE balance
baseline_models.ipynb	Train LR and RF, evaluate, compare ROC-AUC
advanced_models.ipynb	Train Isolation Forest, final model selection
backend/predict.py	Load models, build inference pipeline
backend/main.py	FastAPI routes, CORS, request validation
frontend/app.js	Form input, API calls, chart rendering



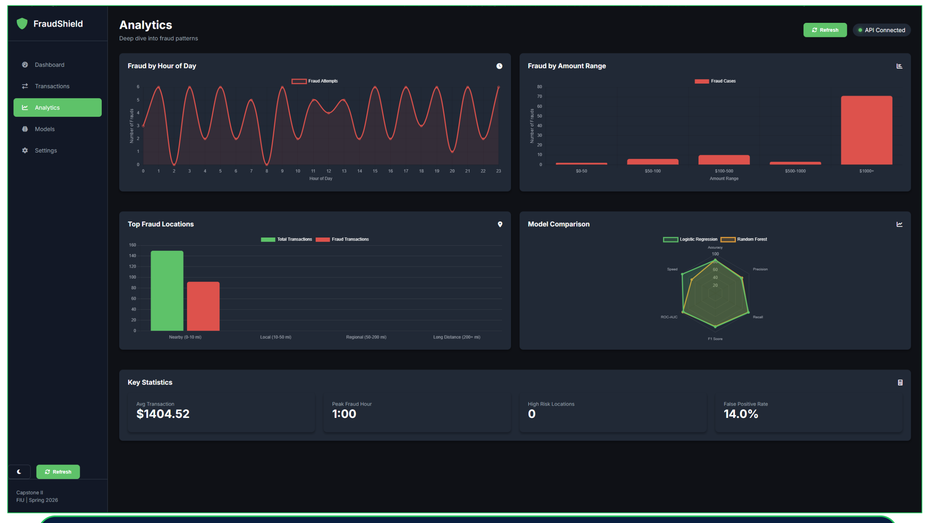
Summary

FraudShield is an open source, full-stack fraud detection system that analyzes credit card transactions using machine learning. It is tested based on the Kaggle credit card dataset CSV, such that the formatting required for testing is:

Time,V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,V11,V12,V13,V14,V15,V16,V17,V18,V19,V20,V21,V22,V23,V24,V25,V26,V27,V28,Amount,Class

The backend processes the 30 features above (V1-V28, Time, Amount) through trained models: logistic regression and random forest, along with anomaly detection, to detect fraud probability. Results are displayed on an interactive dashboard and risk classifications (Low, Medium, and High), probability distributions, and analytics charts.

The system handles class imbalance using SMOTE and includes a responsive frontend with the option to switch a light and dark theme and settings persistence.



System Design

Data layer: The Kaggle Credit Card Fraud dataset is loaded, cleaned, and scaled. Amount and Time are normalized using StandardScaler. SMOTE (Synthetic Minority Oversampling Technique) is applied to the training set to balance the 492 fraud cases against 284,315 legitimate transactions.

Model layer: Three models are trained and serialized to disk using joblib. The Random Forest model is designated as the primary classifier and is loaded at backend startup.
API layer: FastAPI exposes a /predict endpoint that accepts a 30-feature transaction vector and returns a structured JSON response with prediction, probability, anomaly score, and risk level.

Frontend layer: A single-page HTML/CSS/JS dashboard connects to the API and presents results with gauge charts, probability bars, and a transaction history table.

Implementation

Developed the fraud detection pipeline by loading and preprocessing the Kaggle Credit Card Fraud Detection, including cleaning data, scaling key features (Amount, Time), and applying SMOTE for class imbalance resulting in a clean dataset for model training.
Trained baseline machine learning models by implementing Logistic Regression and Random Forest using scikit-learn resulting in models capable of classifying fraudulent and legitimate transactions.
Built and stored model artifacts by saving processed datasets and trained models for reuse in later system components, allowing reproducibility and integration into the full application pipeline.

Requirements

Functional Requirements:
- Accept transaction input (V1–V28, Amount, Time) and return real-time fraud prediction
- Output fraud probability and risk classification (LOW / MEDIUM / HIGH)
- Support multiple models: Logistic Regression, Random Forest, Isolation Forest
- Provide RESTful API endpoints (FastAPI) for frontend integration
- Display results and anomaly indicators on an interactive dashboard

Non-Functional Requirements:
- Input validation and error handling using Pydantic schemas
- Low-latency inference with models loaded at startup
- CORS-enabled API for seamless frontend-backend communication
- Reproducible pipeline with versioned models and scalers (GitHub)

ACKNOWLEDGEMENT

The material presented in this poster is based upon the work supported by XXXXX (*name of the project sponsor: federal, state or local government, or corporate partners, where applicable*). We/I thank XXXX for his/her/their assistance and mentorship that I/we received throughout the senior design project.