Computer Science — FraudShield Documentation

Knight Foundation School of Computing and Information Sciences (KFSCIS) — Florida International University

Instructor: Masoud Sadjadi | Version: 2025 Fall | License: MIT

Poster: 36"×48" horizontal • Videos: Intro & Demo • Presentation: 10–15 min • Scrum: disciplined, evidence-based.

ACM Student Outcomes (O1–O6)

Outcome	Description
O1. Problem Analysis & Requirements	Elicit and analyze user/stakeholder needs; define constraints and success criteria.
O2. System Design & Implementation	Design solutions using appropriate abstractions, patterns, and tools; implement maintainable code.
O3. Experimentation, Testing & Evaluation	Devise evaluation methods; collect evidence; interpret results to improve the system.
O4. Communication & Collaboration	Communicate effectively with technical and non-technical stakeholders; collaborate in teams.
O5. Professional, Ethical, Legal & Societal	Recognize responsibilities; address ethics, security, privacy, accessibility, and sustainability.
O6. Algorithmic Thinking & Complexity	Discipline-specific depth outcome; addressed in dedicated sections below.

Outcome & Rubric Crosswalk (Checklist)

Use this to ensure your documentation and artifacts demonstrate each outcome and satisfy the rubric. Mark each ☐ when complete.

Outcome	Where It Appears in This Document	External Evidence (Links/Appendices)	Status (☐/☑)
O1	§2 Problem & Requirements; §3 System Overview	Appendix A (User Research), Backlog, User Stories	☑
O2	§3 System Overview & Architecture; §5 Implementation Notes	Repo structure, Code, CI status badge	☑
O3	§6 Testing & Evaluation	Test reports, coverage, performance dashboards	☑
O4	§1 Executive Summary; §9 Limitations & Future Work	Poster, Slides, Videos	☑
O5	§7 Security/Privacy/Compliance	Threat model / Model card / SBOM	☐
O6	§4 Discipline-Specific Depth	Artifacts noted in §4	☑

1. Executive Summary

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 anomaly detection algorithms that learn what normal behavior looks like, allowing the system to flag suspicious outliers that traditional methods miss.

FraudShield is an open source, full-stack fraud detection system that analyzes credit card transactions using machine learning. The backend processes 30 features (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 with risk classifications (Low, Medium, and High), probability distributions, and analytics charts.

Key results: Logistic Regression (97.6% accuracy, 91.8% recall), Random Forest (98.2% accuracy, 89.2% recall). The system handles class imbalance using SMOTE and includes a responsive frontend with light/dark theme and settings persistence.

2. Problem & Requirements (O1)

Problem: Credit card fraud costs businesses and consumers billions annually. Only 0.17% of transactions are fraudulent, creating extreme class imbalance. Traditional rule-based systems miss sophisticated fraud patterns, generate excessive false positives, and cannot adapt to new fraud tactics.

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 backend to frontend communication
- Reproducible pipeline with versioned models and scalers (GitHub)

Include a prioritized backlog (top 10–15 items) and a link to your full backlog board.

3. System Overview & Architecture (O2)

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.

Paste a current architecture diagram (C4 containers/components or equivalent).

4. Discipline-Specific Depth (O6)

Algorithms & Data Structures:

- 30 features: V1–V28 (PCA-transformed), Time, Amount
- SMOTE: Synthetic Minority Oversampling Technique using k-nearest neighbors (k=5) to address class imbalance
- Models: Logistic Regression (~1,800 parameters), Random Forest (100 trees, depth 10)

Architecture & Patterns:

- Data → Model → API → Frontend pipeline
- RESTful API with FastAPI
- Singleton pattern for model loading (loaded once at startup)
- MVC-like frontend with JavaScript class

Engineering Evidence:

- Logistic Regression: 97.6% accuracy, 91.8% recall, 6.3% precision
- Random Forest: 98.2% accuracy, 89.2% recall, 15.7% precision
- API response time: <200ms per transaction

5. Implementation Notes (O2)

Repository Structure:

```
fraudshield/
├─── data/
│    └─── creditcard.csv
├─── notebooks/
│    ├─── 01_data_prep.ipynb
│    ├─── 02_baseline_models.ipynb
│    └─── 03_advanced_models.ipynb
├─── models/
├─── backend/
│    ├─── main.py
│    ├─── model_loader.py
│    └─── schemas.py
├─── frontend/
│    ├─── index.html
│    ├─── style.css
│    └─── app.js
├─── requirements.txt
└─── .gitignore
```

Technologies:

- Python, Scikit-learn, FastAPI (backend)
- HTML5, CSS3, JavaScript, Chart.js (frontend)
- SMOTE for class imbalance handling

Key Decisions:

- 3,000 transaction limit to prevent browser freezing and long processing time
- Dark/Light theme with local Storage persistence
- Chart.js for data visualizations

6. Testing & Evaluation (O3)

Test Dataset: Kaggle Credit Card Fraud dataset (284,807 transactions, 0.17% fraud rate). Format required: Time, V1, V2,..., V28, Amount, Class

Model Performance:

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	97.6%	6.3%	91.8%	11.8%
Random Forest	98.2%	15.7%	89.2%	26.8%

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. Validated system stability and reliability by performing full pipeline testing from data input to prediction output.

Strategy: End-to-end testing was conducted using sample CSV files (simplecreditcard.csv, 150-transaction test file) to validate the full pipeline from CSV upload to prediction output.

Coverage: Core functionality including CSV upload, API prediction, and frontend rendering was tested across all implemented features.

Defect Summary: Defects resolved included chart rendering issues (fixed with setTimeout) and CSV format detection (fixed with auto-detection). Known limitation: 3,000 transaction upload limit.

7. Security, Privacy, Accessibility & Compliance (O5)

Summarize how the solution addresses ethical, legal, and societal impacts; include security, privacy, and accessibility considerations.

8. Deployment & Operations

The FraudShield system runs locally on a development machine. No cloud deployment is required for the Capstone demonstration.

Prerequisites:

- Python 3.12 or higher
- Git
- Modern web browser (Chrome recommended)
-

Backend Deployment (via IDE terminal recommended):

```
#Navigate to backend directory
cd backend
# Start the FastAPI server
uvicorn main:app –reload
```

The server runs at http://localhost:8000 with auto-reload enabled for development. The API documentation is available at http://localhost:8000/docs.

Frontend Deployment:

Simply open the frontend/index.html file in a web browser. No build step or server required.

Model Files:

Trained models (logistic_regression.pkl, random_forest.pkl) must be present in the models/ directory before starting the backend. These are generated by running the 02_baseline_models.ipynb notebook.

Verification:

- Backend is running when terminal shows Application startup complete
- Frontend status indicator shows API Connected (logistic_regression, random_forest)

link to detailed guides in the appendices.

9. Limitations & Future Work

Current Limitations:

Limitation	Impact
• 3,000 transaction upload limit	– Cannot process full Kaggle dataset (284,807 rows) in a single upload
• No batch API endpoint	– Each transaction requires an individual API call (~200ms each)
• Charts may not render on first load	– Requires page refresh in some cases
• Local storage only	– User settings do not sync across devices
• No user authentication	– Anyone with API access can make predictions

Technical Debt:

- Duplicate checkApiStatus method in frontend (non-breaking)
- Chart initialization relies on setTimeout hack
- CSV parsing uses manual logic instead of a library like Papa Parse

Future Work (Roadmap):

Priority	Item	Description
High	Batch API endpoint	/batch-predict endpoint to process 100 transactions per API call
Medium	WebSocket support	Real-time fraud alerts without page refresh
Medium	User authentication	Login system with transaction history
Low	Production deployment	Deploy to cloud platform (AWS/Heroku/Render)
Low	Mobile app	React Native version for mobile devices

10. 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.

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.

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. *Journal of Artificial Intelligence Research, 61*, 863-905.

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/

Appendices (Evidence & How-To)

A. Installation Guide (step-by-step with screenshots)

B. User Manual (task-oriented walkthroughs)

C. Admin/Operations Guide (runbook, on-call, backup/restore)

D. API Reference (OpenAPI/GraphQL schema)

E. Poster (36"×48" horizontal), Slides, and Video Links (Intro, Demo)

F. Scrum Evidence Index (links to Sprint Planning, Dailies, Reviews, Retros)