Smart Contract Vulnerability Detection Using Machine Learning

A Pattern-Based Approach to Label Generation and Classification

Lucas Kim

Course: CMSI 5350

Why Smart Contract Security Matters

- Smart contracts manage **billions of dollars** in decentralized applications.
- Code-level vulnerabilities are the **primary cause** of blockchain exploits.
- Manual security auditing is expensive ($10k–$100k per contract) and time-consuming.
- Critical need for **automated, scalable** vulnerability detection systems.

Research Question & Objectives

Research Question

Can machine learning models effectively detect vulnerabilities in smart contracts using features extracted directly from source code?

Primary Objectives

- F1-score ≥ 0.85
- False Positive Rate < 0.10
- Extract comprehensive features
- Evaluate multiple ML models

The Labeling Problem: The Core Blocker

The Core Issue

Dataset contains 42,908 real contracts but **no vulnerability labels**. Without labels, supervised learning is impossible.

Why It Matters

Existing tools fail: Slither fails on 99% of contracts due to compilation errors. Manual labeling would take weeks.

⚠ **Synthetic labels yield F1 < 0.30**

Unlabeled Data

42,000+ Files

0 Labels

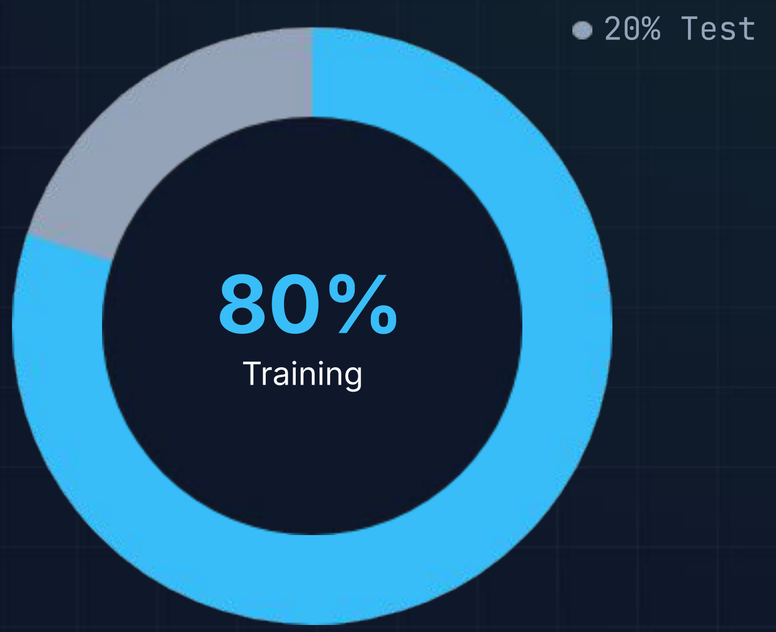
Dataset Description & Split

Ethereum Smart Contract Dataset

- **Total Available:** 42,908 contracts
- **Processed:** 2,000 contracts (subset)
- **Time Period:** 2020-2025

Stratified Split (80/20)

Random State 42, 'Balanced' Weighting



The Breakthrough: Hybrid Detection

Transforming an unsupervised problem into a supervised one by generating labels based on code patterns.

Tier 1: Slither Analysis

Attempts compilation and static analysis.

~1% Success

Most contracts fail to compile.

Tier 2: Pattern-Based

Works directly on source code without compilation.

100% Success

Ensures every contract is labeled.

9 Major Vulnerability Patterns

High Severity (+2 Score)

- **Reentrancy:** External calls followed by state updates.
- **Unchecked External Calls:** Calls without validation.
- **Delegatecall Usage:** Execution in calling context.

Medium Severity (+1 Score)

- tx.origin usage
- Selfdestruct
- Integer overflow
- Uninitialized storage
- Low-level calls
- No access control

```
// Reentrancy Example

function withdraw() public {
   // 1. External Call
    msg.sender.call.value(amount)("");

   // 2. State Update (Too Late)
    balances[msg.sender] = 0;
}
```

Danger: Attacker can re-enter before balance is set to 0.

Label Generation Implementation

1. File Collection & ID Gen

→

2. Dual Analysis

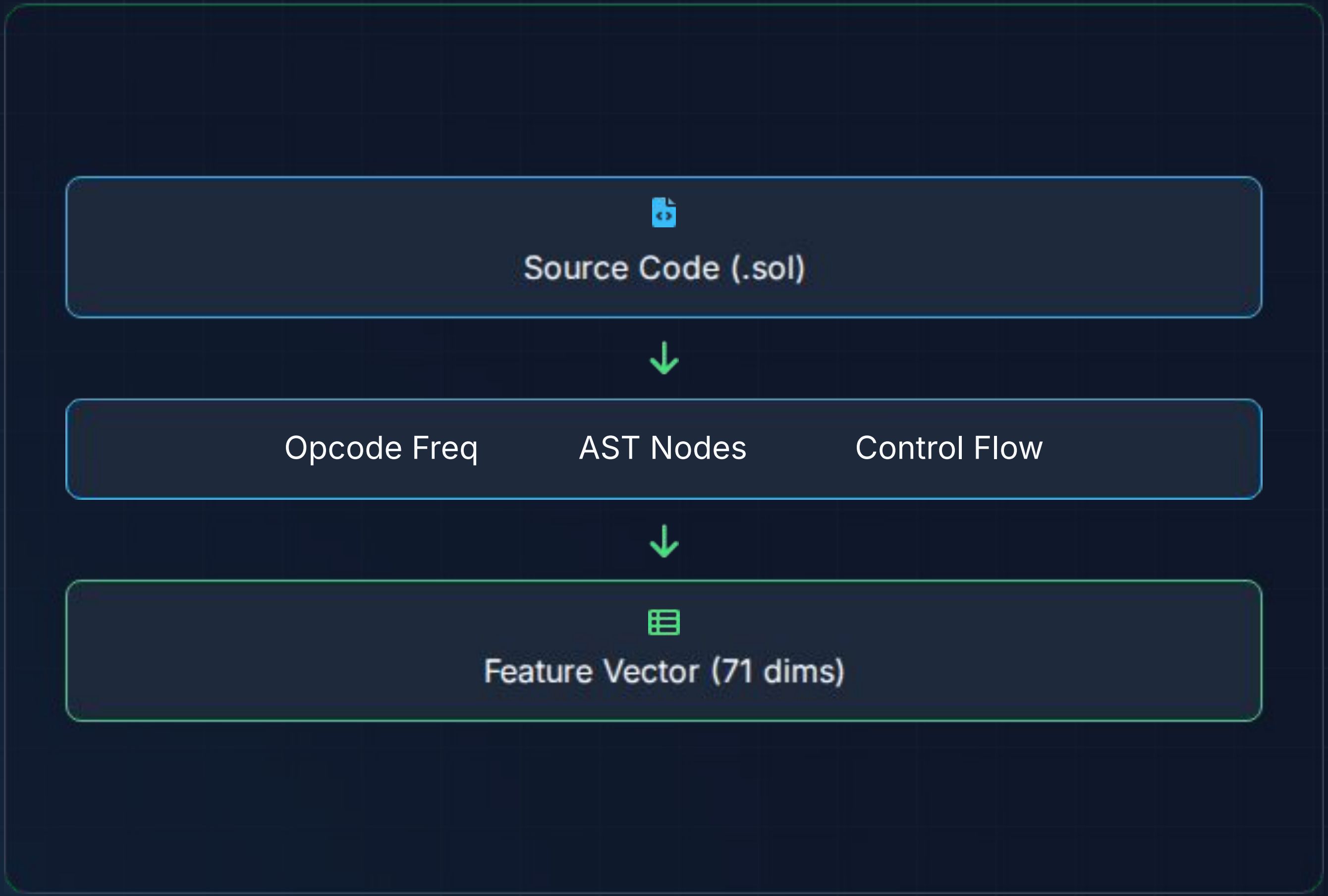
(Slither → Pattern)

→

3. Binary Label Assignment

Total Contracts	Vulnerable	Safe
2,000	71.8%	28.2%

71 Comprehensive Features



1. Opcode Frequency (35)

CALL, DELEGATECALL, SSTORE, SLOAD...

2. AST Node Features (21)

FunctionDefinition, ModifierDefinition, Control Structures...

3. Control Flow (5)

Nesting depth, Loop counts, Complexity...

4. Code Metrics (7)

LOC, Character count, Function count...

Model Training Configuration

Logistic Regression

Baseline with L1/L2 regularization.

Random Forest

Ensemble of decision trees. Tuned for depth and split.

XGBoost

Gradient boosting. Tuned for learning rate and subsample.

5-Fold CV GridSearchCV Class Weight: Balanced

Model Performance (Real Labels)

MODEL	F1-SCORE	PRECISION	RECALL	FPR
Logistic Regression	0.9607	0.9853	0.9373	0.0354
Random Forest (Best)	**0.9825**	0.9894	0.9756	**0.0265**
XGBoost	0.9789	0.9859	0.9721	0.0354

Target Exceeded
F1 Score: 0.98 vs Target 0.85

3x Better FPR
0.0265 vs Target 0.10

Best Model: Random Forest

Key Strengths

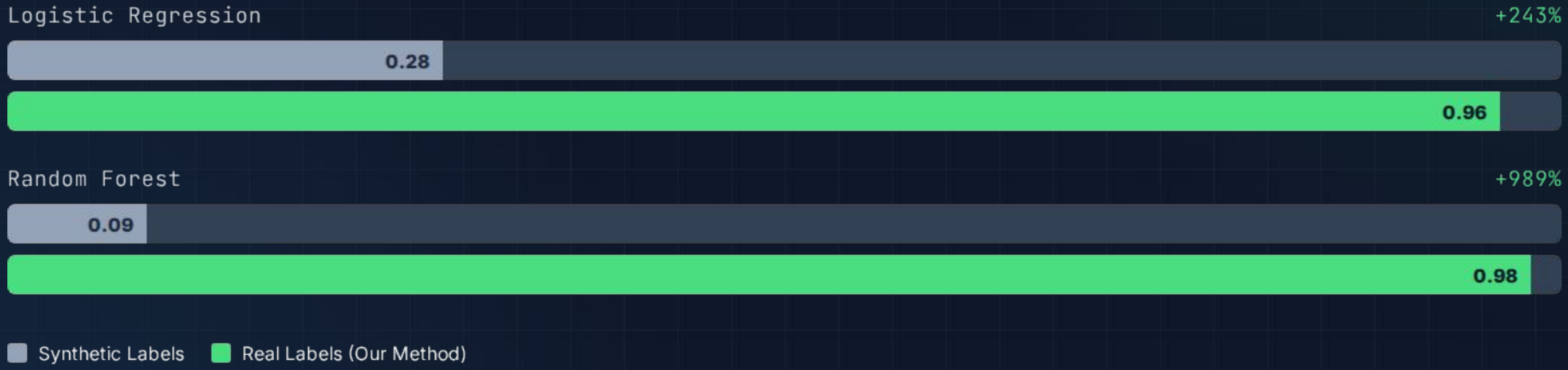
- **Highest F1-score:** 0.9825
- **Lowest FPR:** 0.0265
- Balanced Precision (0.989) and Recall (0.976).
- Robust and ideal for production deployment.

Confusion Matrix (Test Set: 400)

	Pred: 0	Pred: 1
Act: 0	110 TN	3 FP
Act: 1	7 FN	280 TP

Impact of Real Labels

Label quality is more critical than model sophistication.



Technical Challenges Overcome

Missing Labels

Solved via Pattern-based detection system.

Compilation Errors

Solved via Source-code analysis (100% coverage).

Class Imbalance

Solved via Stratified splits & balanced weighting.

ID Matching

Standardized formats (99.4% match rate).

Project Achievements

Targets Exceeded

- F1-score: **0.96-0.98** (Target: ≥0.85)
- FPR: **0.03** (Target: <0.10)

Contributions

- Pattern-based label generation system
- Hybrid Slither + pattern matching approach
- End-to-end production pipeline

Key Findings & Impact

"We transformed an **unsupervised problem**

into a **supervised one**."

By solving the labeling problem through pattern-based detection,

we enabled standard ML models to achieve state-of-the-art performance, proving that

data quality > model complexity.

Limitations & Future Work

Model Improvements

Deep Learning (LSTM, Transformers), Ensemble methods.

Feature Engineering

Graph-based features (CFG, DFG), Semantic embeddings.

Production

Real-time API, Model versioning, SHAP explainability.

Summary

The project's success hinged on solving the labeling problem.

Pattern-based detection enabled us to achieve:

F1 Score	FPR	Improvement
0.98	0.02	+989%

Thank you for your attention.

Lucas Kim | CMSI 5350