Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
Università Ca'Foscari Venezia

On the Role of ML and LLMs in Vulnerability Detection for Smart Contracts

Dalila Ressi
Postdoc researcher
Ca' Foscari University of Venice

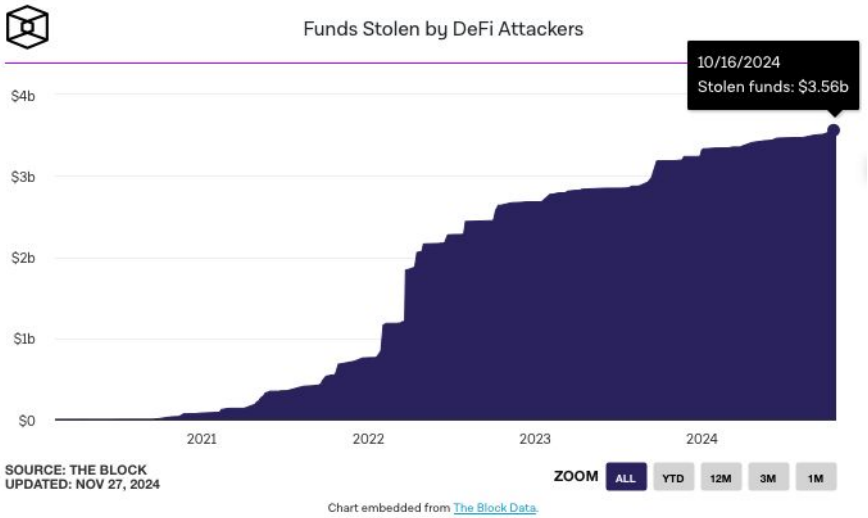
NiRvAna

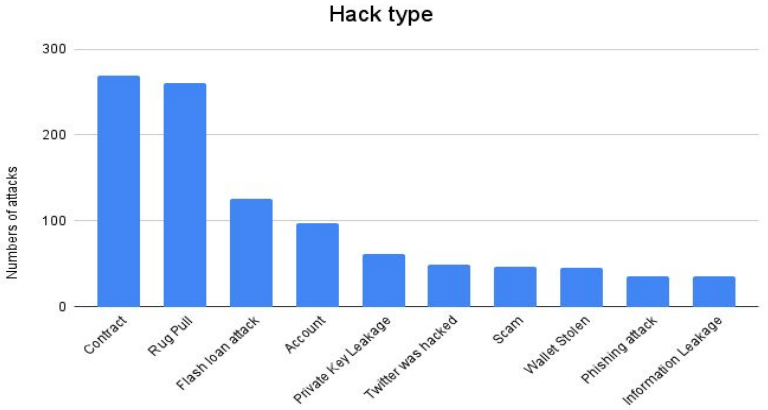
Noninterference and Reversibility Analysis in Private Blockchains

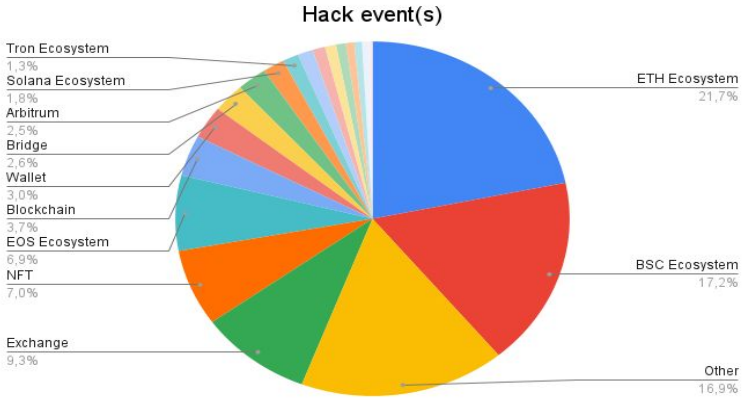
Final Meeting in Urbino (2025)

ethereum

Vulnerabilities in Ethereum Smart Contracts: Why?







sources:
https://hacked.slowmist.io/
https://www.theblock.co/

Ethereum Smart Contracts Representations

Not available most of the times

```
pragma solidity ^0.8.0;

contract Storage {
   uint256 data;

   constructor (uint256 _data) {
      data = _data;
   }

   function set(uint256 _data) public {
      data = _data;
   }
}
```

Contract Code

Easy to understand (for a human)

```
PUSH1 0x80 PUSH1 0x40 MSTORE
CALLVALUE DUP1 ISZERO PUSH2 0x10
JUMPI PUSH1 0x0 DUP1 REVERT
JUMPDEST POP PUSH1 0x40 MLOAD
PUSH2 0x188 CODESIZE SUB DUP1 PUSH2
0x188 DUP4 CODECOPY DUP2 DUP2 ADD
PUSH1 0x40 MSTORE DUP2 ADD SWAP1
PUSH2 0x32 SWAP2 SWAP1 PUSH2 0x54
JUMP JUMPDEST DUP1 PUSH1 0x0 DUP2
SWAP1 SSTORE POP POP PUSH2 0x9E
JUMP JUMPDEST PUSH1 0x0 DUP2
MLOAD SWAP1 POP PUSH2 0x4E DUP2
PUSH2 0x87 JUMP JUMPDEST SWAP3
SWAP2 POP POP JUMP JUMPDEST PUSH1
0x0 PUSH1 0x20 DUP3 DUP5 SUB SLT
ISZERO ....
```

OpCode

Hard to understand

Deployed on Ethereum

```
6080604052348015610010576000 80fd5
b506040516101883803806101888339 81
8101604052810190610032919061005 45
65b8060008190555050610 09e565b6000
8151905061004e81610087565b9291505
0565b600060208284031215610066576 0
0080fd5b6000610074848285016100 3f56
5b91505092915050565b6000819050919
050565b6100908161007d565b8114610 0
9b57600080fd5b50565b60dc806100ac6
000396000f3fe6080604052348015600f5
7600080fd5b5060043610602857600035
60e01c806360fe47b114602d575b60008
0fd5b6043600480360381019060 3f9190
6062565b6045565b005b8060008190....
```

ByteCode

Almost impossible to understand

Vulnerability Detectors: Formal Methods vs ML

Most static analysis tools (abstract interpretation, taint analysis, model checking, symbolic execution)

- Check the *semantics*
- Create an abstraction of the program
- Define security rules
- Check if the program satisfies the rules
- They can return ***True***, ***False***, or ***I don't know***

These techniques usually guarantee to not have false negatives

Most AI models take sequences: LSTM, GRU, LLMs (ChatGPT) or graphs: GNNs

- Check the *syntax (code patterns)*
- Require a large labeled datasets
- Might include complex feature extraction phases
- The detectors need to be trained
- They can return ***the most probable label: True or False***

These techniques do not provide any guarantees on the correctness

Many AI methods use formal methods to label the dataset used

Ressi D. et al. "Vulnerability Detection in Solidity Smart Contracts via Machine Learning: A Qualitative Analysis" (BCRA 2025)

AI-based Detectors: Pros

Advantages of a AI-based detector:

- does not require formalization
- can easily be re-trained to include new vulnerabilities
- detection of a very large number of different possible vulnerabilities
- might catch **variations** of a certain vulnerability that has not been taken into consideration by formal methods
- fast inference time wrt static analyzers
- lately, can come with an **explanation**

Ressi D. et al. "Vulnerability Detection in Solidity Smart Contracts via Machine Learning: A Qualitative Analysis" (BCRA 2025)

AI-based Detectors: Cons

- Machine Learning-related:
  - Scalability
  - Interpretability
  - Replicability
- Comparison-related:
  - Lack of comparison with related work
  - Missing results for single vulnerability
  - Using different metrics
  - Heterogeneity of used datasets
  - Vulnerabilities nomenclature
  - Results inconsistency
- Usability-related:
  - Missing or high inference time
  - Vulnerability location
  - Source code vs. bytecode
  - Risk of vulnerabilities
- Dataset-related:
  - Limited number of contracts
  - Untreated unbalanced classes
  - Unclear dataset creation process
  - Labeling methods

Ressi D. et al. “Vulnerability Detection in Solidity Smart Contracts via Machine Learning: A Qualitative Analysis” (BCRA 2025)

Vulnerabilities: Taxonomies

DASP TOP 10 → https://dasp.co/ (too general)

SWC → https://swcregistry.io/ (no longer maintained)

CWE → https://cwe.mitre.org/index.html (not specifically targeted for blockchain languages)

OPENSCV → Vidal, F.R. et al. “OpenSCV: an open hierarchical taxonomy for smart contract vulnerabilities”. Empirical Software Engineering 2024 (has not been adopted so far)

The same vulnerability is referred to as:

- Call Stack Depth
- Mishandled Exception
- Unchecked Return Value
- No Check After Contract Invocation
- Unchecked external call

Different names, same vulnerability!

Ressi D. et al. “Vulnerability Detection in Solidity Smart Contracts via Machine Learning: A Qualitative Analysis” (BCRA 2025)

Vulnerabilities: Complexity

Parity wallet is a popular Ethereum client that provides users with a secure way to store and manage their Ether (ETH) and other ERC-20 tokens

Parity Wallet Hack (2017)

- all wallets used a library (smart contract) stored at specific address
- the library contained vulnerability: if you called **initWallet()** method you could own the library
- the same person who hacked it and became the owner, called self-destruct
- as a result, all wallets became frozen

Which are the vulnerabilities?

- Denial of service
- Access control
- Unprotected self-destruct

Where? Are the contracts using the library vulnerable or the library itself

Many vulnerable contracts actually present multiple vulnerabilities

Open Problems

Formalization of vulnerabilities

Collection of representative smart contracts

Creation of benchmark dataset with different methods (generation, manual labelling, semi-supervised, …)

Evaluation of the best AI model is hindered by the heterogeneity of dataset used

Which model works the best? Which data representation?

Ressi D. et al. “Vulnerability Detection in Solidity Smart Contracts via Machine Learning: A Qualitative Analysis” (BCRA 2025)

Our first attempt

- Largest manually labelled smart contracts available (CodeSmells)
- source code, bytecode (‘creation code’) and runtime
- Add 3 modalities: opcodes, ASTs and CFGs

Consider the problem as multiclass, multilabel classification task:

- Classic ML methods: SVM, RF, LR, KNN, GB, and XGB
- Some DL methods: FNN, LSTM, CodeBERT, GNN

Rizzo, M. et al. “A Comparison of Machine Learning Techniques for Ethereum Smart Contract Vulnerability Detection” (OVERLAY 2024)

CodeSmells Dataset

Property	Definition	DASP	SWC	N. Contracts	Can be detected by
Unchecked External Calls	Do not check the return value of external call functions.	4	104	25	Oyente, Zeus, Contractfuzzer
Dos Under External Influence	Throwing exceptions inside a loop which can be influenced by external users.	5	113	6	Zeus
Strict Balance Equality	Using strict balance quality to determine the execute logic.	10	132	5	
Unmatched Type Assignment	Assigning unmatched type to a value, which can lead to integer overflow.	10	0	22	Zeus
Transaction State Dependency	Using tx.origin to check the permission.	2	115	5	Zeus
Reentrancy	The re-entrancy bugs.	1	107	11	Oyente, Zeus, Contractfuzzer
Hard Code Address	Using hard code address inside smart contracts.	10	0	84	
Block Info Dependency	Using block information-related APIs to determine the execute logic.	6	120	42	Oyente, Zeus, Contractfuzzer
Nested Call	Executing CALL instruction inside an unlimited-length loop.	5	128	13	
Deprecated APIs	Using discarded or unrecommended APIs or instructions.	10	0	217	
Unspecified Compiler Version	Do not fix the smart contract to a specific version	10	103	508	
Misleading Data Location	Do not clarify the reference types of local variables of struct, array or mapping.	10	0	1	
Unused Statement	Creating values which never be used.	10	135	10	
Unmatched ERC-20 standard	Do not follow the ERC-20 standard for ICO contracts.	10	0	45	
Missing Return Statement	A function denote the type of return values but do not return anything.	10	0	231	
Missing Interrupter	Missing backdoor mechanism in order to handle emergencies.	10	0	488	
Missing Reminder	Missing events to notify caller whether some functions are successfully executed.	10	0	27	
Greedy Contract	A contract can receive Ethers but can not withdraw Ethers.	10	997	6	Mayan
High Gas Consumption Function Type	Using inappropriate function type which can increase gas consumption.	10	0	352	
High Gas Consumption Data Type	Using inappropriate data type which can increase gas consumption.	10	0	0	

551 source codes, 551 bytecodes, and 504 runtimes -> 103 opcodes, 287 ASTs, and 470 CFGs

Preliminary Results

- Coarse SWC labels >> fine-grained "Property" labels
- Ensemble models (RF, GB, XGB) excelled in fine-grained classification; DL models performed best with coarse labels.
- GNNs had perfect recall on graph data but very low precision, limiting their effectiveness.
- Multimodal integration notably improved classification performance, especially for fine-grained tasks.

			Model								
Modality	**Definition**	**Metric**	GB	KNN	LR	RF	SVM	XGB	CodeBERT	LSTM	FNN
Bytecode	**Property**	**Precision**	0.82	0.83	0.82	**0.84**	0.82	0.83	0.81	0.82	0.61
		Recall	**0.80**	**0.80**	0.69	**0.80**	0.69	0.79	0.71	0.70	0.77
		F1-score	0.79	0.79	0.72	**0.80**	0.72	0.79	0.74	0.74	0.66
	SWC	**Precision**	0.87	0.91	**0.92**	0.91	**0.92**	0.90	0.91	0.91	0.78
		Recall	0.83	0.83	0.83	0.83	0.83	0.83	0.84	0.84	**0.86**
		F1-score	0.83	0.85	**0.86**	0.85	**0.86**	0.85	**0.86**	**0.86**	0.79
Opcode	**Property**	**Precision**	0.78	**0.86**	0.26	0.85	0.22	0.84	0.58	0.45	0.37
		Recall	0.76	0.79	0.99	0.81	**1.00**	0.81	0.72	0.72	0.70
		F1-score	0.75	0.80	0.40	**0.81**	0.36	**0.81**	0.61	0.54	0.48
	SWC	**Precision**	0.76	**0.88**	**0.88**	**0.88**	**0.88**	0.86	0.82	0.30	0.84
		Recall	0.74	0.80	0.80	0.80	0.80	0.78	0.82	0.86	**0.87**
		F1-score	0.73	0.82	0.82	0.82	0.82	0.80	0.82	0.45	**0.85**
Runtime	**Property**	**Precision**	0.82	**0.84**	0.82	**0.84**	0.82	**0.84**	0.80	0.82	0.60
		Recall	0.80	**0.81**	0.69	**0.81**	0.69	0.80	0.70	0.70	0.78
		F1-score	0.79	0.80	0.72	**0.81**	0.72	0.80	0.73	0.74	0.66
	SWC	**Precision**	0.88	0.91	**0.92**	0.92	**0.92**	0.91	0.91	0.91	0.79
		Recall	0.84	0.83	0.83	0.83	0.83	0.84	0.84	0.84	**0.86**
		F1-score	0.84	0.85	**0.86**	**0.86**	**0.86**	**0.86**	**0.86**	**0.86**	0.79
Source	**Property**	**Precision**	0.83	0.85	0.83	**0.86**	0.83	0.85	0.80	0.62	0.63
		Recall	0.82	**0.83**	0.77	**0.83**	0.77	0.82	0.70	0.70	**0.83**
		F1-score	0.81	**0.82**	0.77	**0.82**	0.78	0.81	0.73	0.64	0.70
	SWC	**Precision**	0.88	0.90	**0.92**	0.91	**0.92**	0.90	0.91	0.91	0.79
		Recall	0.83	0.83	0.83	0.83	0.83	0.83	**0.84**	**0.84**	**0.84**
		F1-score	0.83	0.85	**0.86**	**0.86**	**0.86**	0.85	**0.86**	**0.86**	0.79
Multimodal	**Property**	**Precision**	0.84	0.84	0.65	**0.86**	0.66	**0.86**	-	-	0.63
		Recall	0.87	0.82	**0.89**	0.86	**0.89**	0.87	-	-	0.81
		F1-score	0.84	0.81	0.73	0.84	0.74	**0.85**	-	-	0.69
	SWC	**Precision**	0.88	0.91	**0.92**	0.91	**0.92**	0.90	-	-	0.82
		Recall	0.82	0.84	0.83	0.83	0.83	0.83	-	-	**0.85**
		F1-score	0.83	**0.86**	**0.86**	**0.86**	**0.86**	0.85	-	-	0.81

			Model									
Modality	**Definition**	**Metric**	GB	KNN	LR	RF	SVM	XGB	CodeBERT	LSTM	FNN	GNN
AST	**Property**	**Precision**	0.74	0.72	0.69	**0.76**	0.71	0.75	-	-	-	0.21
		Recall	0.80	0.78	0.76	0.81	0.78	0.80	-	-	-	**1.00**
		F1-score	0.70	0.71	0.68	**0.73**	0.70	0.72	-	-	-	0.33
	SWC	**Precision**	0.74	0.76	0.74	**0.77**	0.74	**0.77**	-	-	-	0.19
		Recall	0.74	0.70	0.74	0.78	0.74	0.78	-	-	-	**1.00**
		F1-score	0.64	0.62	0.64	**0.66**	0.64	**0.66**	-	-	-	0.32
CFG	**Property**	**Precision**	0.79	0.81	0.78	**0.84**	0.77	0.81	-	-	-	0.23
		Recall	0.77	0.76	0.76	0.77	0.77	0.77	-	-	-	**1.00**
		F1-score	0.75	0.75	0.74	**0.78**	0.74	0.76	-	-	-	0.37
	SWC	**Precision**	0.93	0.93	0.96	**0.97**	0.95	0.97	-	-	-	0.23
		Recall	0.89	0.89	0.87	0.89	0.88	0.89	-	-	-	**1.00**
		F1-score	0.88	0.89	0.90	**0.91**	0.90	**0.91**	-	-	-	0.35

Rizzo, M. et al. “A Comparison of Machine Learning Techniques for Ethereum Smart Contract Vulnerability Detection” (OVERLAY 2024)

What about LLMs?

Differently from the techniques mentioned so far,

they can provide an ***explanation*** of the bug,

and also a ***possible fix***!

But far are we into code understanding?

Code understanding in LLMs: a case study

Semantic preserving code transformation on source code (Python)

- copy propagation
- constant folding

Dataset of couples optimized/perturbed

“Are these programs semantically equivalent?”

Contextless Preamble

```
Are the following functions semantically equivalent to the first one?
```

Contextual Preamble

```
You are a chatbot for comparing the semantics of small Python programs. I will
provide you with multiple implementations of the same Python function. The first
function is the reference version. The other functions are perturbed with copy
propagation, constant folding or a combination of the two. Tell me whether the
functions are semantically equivalent to the reference version or not.
```

Laneve, C. et al. “Assessing Code Understanding in LLMs” (FORTE 2025)

Code understanding in LLMs: a case study

only semantically equivalent

also not semantically equivalent

	Contextless Preamble	Contextual Preamble
Single-Class	Prompt 1	Prompt 2
Multi-Class	Prompt 3	Prompt 4

	VSCode Plugin			Web Interface				
	Copilot		Extension					
Prompt #	Claude	ChatGPT	Amazon Q	Gemini	ChatGPT	DeepSeek	Claude	Average
#1	63.20%	53.68%	56.99%	47.62%	67.53%	70.56%	76.77%	62.34%
#2	71.52%	79.39%	76.36%	41.82%	61.21%	82.42%	**84.85%**	**71.08%**
#3 / Correct Programs	43.03%	51.52%	46.06%	45.45%	46.06%	76.97%	71.72%	**54.40%**
#4 / Correct Programs	63.03%	72.73%	65.45%	**40.00%**	70.91%	77.58%	83.03%	67.53%
Average	60.19%	64.33%	61.22%	**43.72%**	61.43%	76.88%	**79.09%**	

	VSCode Plugin			Web Interface				
	Copilot		Extension					
Prompt #	Claude	ChatGPT	Amazon Q	Gemini	ChatGPT	DeepSeek	Claude	Average
#3 / Correct Programs	**43.03%**	51.52%	46.06%	45.45%	46.06%	76.97%	71.72%	**54.40%**
#3 / Incorrect Programs	94.55%	89.55%	95.00%	**95.91%**	90.00%	95.45%	93.94%	**93.48%**
#3 / Overall	72.47%	73.25%	74.03%	74.29%	**71.17%**	**87.53%**	76.19%	75.56%
#4 / Correct Programs	63.03%	72.73%	65.45%	**40.00%**	70.91%	77.58%	83.03%	**67.53%**
#4 / Incorrect Programs	**98.64%**	85.45%	96.82%	93.64%	90.45%	89.55%	96.36%	**92.99%**
#4 / Overall	83.38%	80.00%	83.38%	**70.65%**	82.08%	84.42%	**90.65%**	82.08%

Laneve, C. et al. "Assessing Code Understanding in LLMs" (FORTE 2025)

Code understanding in LLMs: a case study

Limitations:

- Web interface ≠ API
- Continuously evolving (i.e. ChatGPT answer format)
- Task-oriented performance

What about understanding vulnerabilities?

Laneve, C. et al. “Assessing Code Understanding in LLMs” (FORTE 2025)

Rizzo, M. et al. “A Comparison of Machine Learning Techniques for Ethereum Smart Contract Vulnerability Detection” (OVERLAY 2024)

Reentrancy: a complex vulnerability

A practical example:

Reentrancy

$900 million loss

```
contract C{
   mapping (address => uint256) public bal;

   function withdraw(uint256 amt) public {
     require(bal[msg.sender] >= amt, "No funds");
     (bool success, ) = msg.sender.call{value:amt}("");

     require(success, "Call failed");
     bal[msg.sender] -= amt;
   }
}
```

Reentrancy: a complex vulnerability

This is a call to an external account (another smart contract)

```
contract C{
   mapping (address => uint256) public bal;

   function withdraw(uint256 amt) public {
     require(bal[msg.sender] >= amt, "No funds");
     (bool success, ) = msg.sender.call{value:amt}("");

     require(success, "Call failed");
     bal[msg.sender] -= amt;
   }
}
```

Reentrancy: a complex vulnerability

This is a call to an external account (another smart contract)

Only after the function returns the balance is updated

```
contract C{
    mapping (address => uint256) public bal;

    function withdraw(uint256 amt) public {
      require(bal[msg.sender] >= amt, "No funds");
      (bool success, ) = msg.sender.call{value:amt}("");

      require(success, "Call failed");
      bal[msg.sender] -= amt;
    }
}
```

Reentrancy: a complex vulnerability

How to avoid reentrancy:

- CEI Pattern
- Modifiers
- External libraries
- **Mutex/mutex modifier**
- Use send/transfer

```
contract C{
   mapping (address => uint256) public bal;

   modifier nonReentrant() {
     require(!flag, "Locked");
     flag = true;
     _;   // placeholder for the function body
     flag = false;
    }

   function withdraw(uint256 amt) nonReentrant public {
     require(bal[msg.sender] >= amt, "No funds");
     (bool success, ) = msg.sender.call{value:amt}("");
     require(success, "Call failed");
     bal[msg.sender] -= amt; // effect after call is ok
    }
}
```

Reentrancy: a complex vulnerability

withdraw is protected
with a modifier/mutex,

but an attacker can reenter in
the transfer!

```
contract C{
   mapping (address => uint256) public bal;

   modifier nonReentrant() {
     require(!flag, "Locked");
     flag = true;
     _;   // placeholder for the function body
     flag = false;
    }

   function withdraw(uint256 amt) nonReentrant public {
     require(bal[msg.sender] >= amt, "No funds");
     (bool success, ) = msg.sender.call{value:amt}("");
     require(success, "Call failed");
     bal[msg.sender] -= amt; // effect after call is ok
    }
   function transfer(address to, uint256 amt) public {
     require(bal[msg.sender] >= amt, "No funds");
     bal[to] += amt; // reentering here alters state
     bal[msg.sender] -= amt;
    }
}
```

Reentrancy: a complex vulnerability

withdraw is protected
with a modifier/mutex,

but an attacker can reenter in
the deposit!

In this case, though, it is the same as doing consecutive calls

```
contract C{
   mapping (address => uint256) public bal;

   modifier nonReentrant() {
     require(!flag, "Locked");
     flag = true;
     _;   // placeholder for the function body
     flag = false;
    }

   function withdraw(uint256 amt) nonReentrant public {
     require(bal[msg.sender] >= amt, "No funds");
     (bool success, ) = msg.sender.call{value:amt}("");
     require(success, "Call failed");
     bal[msg.sender] -= amt; // effect after call is ok
    }

   function deposit() public payable {
     bal[msg.sender] += msg.value;
    }

}
```

Reentrancy: two datasets

Aggregated benchmark dataset

a collection of existing datasets, re-labeled

Category (Initial Labels)	Initial Raw Count	Unique Contracts	Compilable Contracts
Potentially Reentrant (total)	368	358	145
from CGT [12]	300	296	93
from HuagGai [6]	22	22	21
from Reentrancy Study [34]	46	40	31
Potentially Safe (total) (from Reentrancy Study [34])	123,169	118,422	73,434
Total Contracts	123,537	118,780	73,579

Taxonomic Reentrancy Dataset

a collection 150 MWE contract implementing complex scenarios hard to detect

- single function
- cross function
- modifiers
- mutex/modifiers
- isHuman modifier
- semantically safe/reentrant contracts
- stateless contracts
- ERC20
- safe contracts breaking the CEI
- other uncommon variants

Results

MODEL PERFORMANCE METRICS ON THE *Aggregated Benchmark* AND THE *Taxonomic Reentrancy Scenarios (TRS)*

Strategy	*Aggregated Benchmark*				*Taxonomic Reentrancy Scenario (TRS)*			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
confuzzius	0.88	0.94	0.63	0.75	0.59	0.58	0.70	0.64
conkas	0.82	0.66	0.75	0.70	-	-	-	-
mythril	0.75	0.61	0.31	0.41	0.61	0.58	**0.84**	0.69
oyente	0.80	0.88	0.31	0.46	-	-	-	-
security	0.81	0.93	0.33	0.49	-	-	-	-
sfuzz	0.76	0.68	0.27	0.38	-	-	-	-
slither	0.86	0.82	0.64	0.72	0.68	0.63	0.85	0.72
gradient_boosting	0.90 ± 0.04	0.87 ± 0.04	0.76 ± 0.11	0.81 ± 0.05	0.62 ± 0.02	0.63 ± 0.03	0.62 ± 0.02	0.61 ± 0.02
gnb	0.82 ± 0.05	0.70 ± 0.06	0.65 ± 0.12	0.67 ± 0.07	0.58 ± 0.03	0.71 ± 0.05	0.58 ± 0.03	0.51 ± 0.07
knn	0.88 ± 0.01	0.85 ± 0.13	0.74 ± 0.13	0.78 ± 0.01	0.62 ± 0.01	0.62 ± 0.01	0.62 ± 0.01	0.62 ± 0.01
logistic_regression	0.80 ± 0.06	0.93 ± 0.12	0.33 ± 0.12	0.47 ± 0.12	0.49 ± 0.01	0.33 ± 0.12	0.49 ± 0.01	0.35 ± 0.03
random_forest	0.90 ± 0.04	0.90 ± 0.07	0.73 ± 0.13	0.80 ± 0.07	0.63 ± 0.03	0.64 ± 0.03	0.63 ± 0.03	0.62 ± 0.04
svm	0.84 ± 0.04	0.93 ± 0.12	0.51 ± 0.16	0.64 ± 0.09	0.50 ± 0.00	0.33 ± 0.01	0.50 ± 0.00	0.34 ± 0.02
xgboost	0.91 ± 0.02	0.88 ± 0.08	0.79 ± 0.10	0.83 ± 0.02	0.72 ± 0.03	0.72 ± 0.03	0.72 ± 0.03	0.72 ± 0.03
codebert	0.90 ± 0.13	0.82 ± 0.24	$\mathbf{0.96} \pm 0.03$	0.87 ± 0.14	0.65 ± 0.06	0.67 ± 0.07	0.65 ± 0.06	0.64 ± 0.06
lstm	0.86 ± 0.12	0.76 ± 0.17	$\mathbf{0.96} \pm 0.03$	0.83 ± 0.11	0.60 ± 0.05	0.62 ± 0.06	0.60 ± 0.05	0.59 ± 0.05
ffnn	$\mathbf{0.96} \pm 0.01$	0.85 ± 0.03	0.89 ± 0.03	0.86 ± 0.02	0.61 ± 0.03	0.62 ± 0.03	0.61 ± 0.03	0.61 ± 0.03
gemini-1.5-flash	0.81 ± 0.02	0.83 ± 0.03	0.81 ± 0.02	0.78 ± 0.03	0.52 ± 0.03	0.52 ± 0.03	0.52 ± 0.03	0.51 ± 0.04
gemini-2.0-flash	0.87 ± 0.02	0.87 ± 0.02	0.84 ± 0.02	0.85 ± 0.02	0.51 ± 0.02	0.52 ± 0.02	0.52 ± 0.02	0.49 ± 0.03
gemini-2.5-flash	0.83 ± 0.04	0.85 ± 0.03	0.82 ± 0.04	0.83 ± 0.04	0.58 ± 0.02	0.66 ± 0.02	0.58 ± 0.01	0.53 ± 0.01
gpt-4o	0.85 ± 0.02	0.89 ± 0.02	0.85 ± 0.02	0.86 ± 0.02	0.53 ± 0.04	0.53 ± 0.01	0.53 ± 0.01	0.51 ± 0.02
gpt-4.1	0.87 ± 0.02	0.90 ± 0.01	0.87 ± 0.02	0.88 ± 0.02	0.63 ± 0.06	0.63 ± 0.00	0.63 ± 0.06	0.63 ± 0.05
o3-mini	$\mathbf{0.96} \pm 0.07$	$\mathbf{0.96} \pm 0.06$	$\mathbf{0.96} \pm 0.07$	$\mathbf{0.96} \pm 0.07$	0.79 ± 0.07	0.80 ± 0.07	0.79 ± 0.07	0.79 ± 0.04
o4-mini	0.94 ± 0.02	0.94 ± 0.01	0.94 ± 0.02	0.93 ± 0.02	$\mathbf{0.82} \pm 0.05$	$\mathbf{0.83} \pm 0.05$	0.82 ± 0.05	$\mathbf{0.82} \pm 0.03$

Thank you!

Dalila Ressi
Postdoc researcher
Ca’ Foscari University of Venice
dalila.ressi@unive.it

- *Rizzo, M. et al.* Towards Explainable AI for Smart Contract Security: A RAG-Based Approach with Human Validation. *(Work in progress)*
- Ressi, D., Spanò, A., Benetollo, L., Piazza, C., Bugliesi, M., & Rossi, S. Vulnerability Detection in Ethereum Smart Contracts via Machine Learning: A Qualitative Analysis. *BCRA 2025*
- Laneve, C., Spanò, A., Ressi, D., Rossi, S., & Bugliesi, M. Assessing Code Understanding in LLMs. *FORTE 2025*
- Rizzo, M., Ressi, D., Gasparetto, A., & Rossi, S. A Comparison of Machine Learning Techniques for Ethereum Smart Contract Vulnerability Detection. *OVERLAY 2024*
- Ressi, D., Romanello, R., Piazza, C., & Rossi, S. AI-enhanced blockchain technology: A review of advancements and opportunities. *Journal of Network and Computer Applications 2024*