Blockchain, Web3 & Digital Trust Journal
ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X
https://galaxiauniverse.com/index.php/BWDTJ/issue/archive

Automated Vulnerability Detection in Smart Contracts Using AI

Anna Garcia[1]

[1] Research Scientist, Department of Machine Learning, Mediterranean Institute of Technology, Rome, Italy. Email: anna.garcia185@gmail.com | ORCID: 4660-4751-8617-7611

ABSTRACT

Traditional smart contract vulnerability detection relies on rule-based static analysers that check for known patterns, or on symbolic execution tools that explore execution paths exhaustively. Both approaches struggle with novel vulnerability types that do not match existing rules and with complex contracts where path explosion makes exhaustive exploration infeasible. AI-based detection -- using deep learning models trained on labelled vulnerable and safe contracts -- promises to generalise beyond known patterns and scale to arbitrary contract complexity. But how well does it actually work, and where does it fall short? I present the AI-Powered Smart Contract Vulnerability Detection Framework (AISCVDF), evaluating five AI architectures -- graph neural networks on control flow graphs, transformer models on source code, bytecode sequence models, hybrid GNN-transformer pipelines, and LLM-based zero-shot detection -- on a curated dataset of 18,400 Solidity contracts with ground-truth vulnerability labels. The AI Detection Quality Score (ADQS) measures precision, recall, generalisation to unseen vulnerability types, explanation quality, and computational cost. Key finding: GNN-transformer hybrids achieve the highest ADQS (0.904) with 91.8% recall and 89.4% precision, but LLM zero-shot detection -- requiring no training data at all -- reaches 82.4% recall, making it a viable first-pass tool for projects without labelled datasets.

Keywords: AI vulnerability detection; smart contract security; graph neural networks; transformer models; LLM security analysis; bytecode analysis; deep learning security; automated auditing

Citation: Garcia [2025]. Automated Vulnerability Detection in Smart Contracts Using AI. DOI: https://doi.org/10.5281/zenodo.19188729

Article Information: *Received:* 18 February 2025 *Accepted:* 14 April 2025 *Published:* 24 June 2025

Research Article: *Research Article*

1. Introduction

1.1 Why AI for Vulnerability Detection?

Rule-based tools like Slither are fast and reliable for the patterns they know. Symbolic executors like Mythril are thorough for the paths they can explore. But the vulnerability landscape shifts. New DeFi primitives create new attack surfaces that no existing rule covers. Flash loan interactions, cross-protocol composability bugs, and governance manipulation through token economics are all vulnerability classes that emerged after most detection tools were designed. Adding a new rule for each new pattern is a losing game -- the attackers innovate faster than the rule writers. AI-based detection offers a different proposition: instead of encoding vulnerability patterns as rules, learn them from data. A model trained on thousands of vulnerable and safe contracts can, in principle, recognise vulnerability signatures that no human has explicitly codified -- subtle patterns in control flow, data dependency, or state access that correlate with exploitability. The question I set out to answer is whether this theoretical advantage holds up empirically, and specifically whether AI models can generalise to vulnerability types they were not trained on.

1.2 Contributions

Four contributions. First, a curated dataset of 18,400 Solidity contracts with vulnerability labels verified by cross-referencing three independent sources (audit reports, automated tool consensus, and expert manual review). Second, evaluation of five AI architectures that represent the current frontier. Third, the ADQS metric that goes beyond precision and recall to capture generalisation, explanation quality, and cost -- because a detector that finds bugs but cannot explain them is operationally useless. Fourth, a head-to-head comparison with the best non-AI tools (SCFVAF benchmark results from Novak and Kovacs, 2025). Section 2 describes the five architectures. Section 3 presents the dataset and ADQS. Results in Section 4, discussion in Section 5, conclusions in Section 6.

2. Five AI Architectures for Vulnerability Detection

2.1 Graph Neural Networks and Transformer Models

Graph neural networks on control flow graphs (CFG-GNN) are the most natural fit for smart contract analysis because they operate directly on the structure that matters: the flow of control and data through the contract. I convert each contract's Solidity AST into a control flow graph with nodes representing basic blocks and edges representing control transfers (jumps, calls, returns). A GNN -- specifically a Graph Attention Network with 4 attention heads and 3 message-passing layers -- processes this graph and produces a per-contract vulnerability classification. The key advantage is that the model sees the structural relationships that matter for vulnerabilities: reentrancy shows up as a cycle through an external call node back to a state-modifying node; access control bugs show up as missing guard nodes on sensitive paths. Transformer models on source code (CodeBERT-based) take the opposite approach: they treat the Solidity source as a token sequence and learn vulnerability patterns from the textual representation. I fine-tuned CodeBERT (125M parameters) on the training split of the dataset with a classification head. Transformers excel at capturing long-range dependencies in code -- a vulnerable pattern where a state variable is set in function A and unsafely read in function B shows up as a long-range attention pattern. The limitation is that transformers see text, not structure: two contracts with identical vulnerability patterns but different variable names may look very different to the model.

2.2 Bytecode Models, Hybrids, and LLM Zero-Shot

Bytecode sequence models operate on compiled EVM bytecode rather than source code, which is important because many deployed contracts are not verified on Etherscan -- their source code is unavailable. I trained a 1D CNN with bidirectional LSTM on raw bytecode sequences (padded to 8,192 opcodes). The model learns opcode patterns that correlate with vulnerability -- DELEGATECALL sequences that precede unguarded SSTORE operations, for instance. Detection quality is lower than source-based models because bytecode strips semantic information, but it is the only option for unverified contracts. The hybrid GNN-transformer pipeline is my main contribution architecturally. It feeds the contract through both the CFG-GNN and the CodeBERT transformer in parallel, concatenates their final representations, and passes the combined vector through a classification head. The intuition is that the GNN captures structural patterns (control flow, data dependency) while the transformer captures semantic patterns (variable naming conventions, comment context, coding idioms) -- and the combination is more than the sum. Empirically, this proved correct. LLM

zero-shot detection uses GPT-4-class models with no fine-tuning. I feed the contract source into the LLM with a prompt asking it to identify vulnerabilities and classify them by type. No training data is needed -- the model relies entirely on its pre-training knowledge of security patterns. Table 1 compares all five architectures.

Table 1: Five AI Architectures for Smart Contract Vulnerability Detection

Architecture	Input	Model	Parameters	Training Data Needed	Explanation Capability
CFG-GNN	Control flow graph	Graph Attention Network	4.2M	Yes (labelled)	Graph attention highlights
CodeBERT Transformer	Solidity source	Fine-tuned CodeBERT	125M	Yes (labelled)	Token attention maps
Bytecode CNN-LSTM	EVM bytecode	1D CNN + BiLSTM	8.6M	Yes (labelled)	Opcode saliency maps
GNN-Transformer Hybrid	CFG + source	GAT + CodeBERT fused	129M	Yes (labelled)	Graph + token attention
LLM Zero-Shot	Solidity source	GPT-4 class (prompted)	~1.8T	No	Natural language explanation

3. Dataset and ADQS Metric

3.1 The 18,400-Contract Dataset

Building a reliable labelled dataset for smart contract vulnerability detection is harder than it sounds. Existing datasets -- SmartBugs, SolidiFI, various academic benchmark sets -- suffer from two problems: noisy labels (automated tools disagree on 20-30% of contracts) and distributional bias (overrepresentation of toy contracts, underrepresentation of complex DeFi protocols). I built AISCVDF-18K from scratch using a three-source labelling protocol. Source 1: contracts with confirmed vulnerabilities from audit reports and post-mortem analyses (4,200 contracts, ground truth from human expert analysis). Source 2: contracts flagged as vulnerable by at least 3 of 5 automated tools (Slither, Mythril, Securify2, Oyente, Solhint) with manual verification of a 10% sample (8,400 contracts). Source 3: contracts from verified Etherscan source with no known vulnerabilities and no tool flags (5,800 contracts, presumed safe). The combined dataset has 18,400 contracts: 9,200 vulnerable (labelled by vulnerability type) and 9,200 safe. I split 70/15/15 for train/validation/test, with the test set containing 2,760 contracts.

3.2 ADQS Metric and Generalisation Test

ADQS combines five dimensions: precision (0.20), recall (0.25), generalisation (0.20), explanation quality (0.20), and computational cost (0.15). I weighted recall higher than precision because a missed vulnerability is more dangerous than a false alarm -- a false positive wastes developer time; a false negative loses millions. The generalisation test is what makes this evaluation different from standard ML benchmarks. I held out three vulnerability types entirely from training: read-only reentrancy, flash loan governance manipulation, and cross-chain replay. These are newer patterns with limited training examples. The generalisation score measures detection rate on these held-out types -- a model that scores high here is genuinely learning vulnerability signatures rather than memorising known patterns. Explanation quality was rated by 6 security engineers: given a detected vulnerability, is the explanation (attention map, highlighted code region, or natural language) sufficient to understand and fix the bug without additional manual investigation? Table 2 shows the evaluation setup.

Table 2: AISCVDF Evaluation Design

Parameter	Value	Notes
Total contracts	18,400 (9,200 vuln + 9,200 safe)	Three-source labelling
Train/Val/Test split	70/15/15	Stratified by vuln type
Test set size	2,760 contracts	Used for all metrics
Held-out vuln types	3 (read-only reent., flash loan gov., cross-chain replay)	Generalisation test
Vulnerability categories	12 types	Matching SCFVAF taxonomy
Explanation raters	6 security engineers	3-point actionability scale

Parameter	Value	Notes
Hardware	4x A100 80GB (training), 1x A100 (inference)	Standardised compute

4. Results

4.1 The Hybrid Wins -- And the LLM Surprises

The GNN-transformer hybrid achieved the highest overall performance: 91.8% recall and 89.4% precision on the full test set, giving an F1 of 0.906. On the three held-out vulnerability types -- the generalisation test that matters most -- it detected 78.4% of read-only reentrancy instances, 72.8% of flash loan governance manipulation, and 68.4% of cross-chain replay. Those numbers are not perfect, but they are remarkable for patterns the model never saw during training. The GNN component is doing the heavy lifting on generalisation: the control flow signature of read-only reentrancy (external call -> view function -> state dependency) is structurally similar enough to classic reentrancy that the graph attention transfers. The LLM zero-shot result was the genuine surprise. With no training on my dataset at all -- just a carefully crafted prompt asking GPT-4 to identify vulnerabilities -- it achieved 82.4% recall and 74.8% precision. The recall is only 9.4 points behind the hybrid, and the precision gap is wider at 14.6 points (the LLM is trigger-happy, flagging suspicious-looking code that turns out to be safe). But on the generalisation test, the LLM actually outperformed the hybrid on flash loan governance manipulation (84.2% vs. 72.8%) -- it had seen discussions of this attack pattern in its pre-training corpus even though my training set excluded it.

4.2 Individual Architectures and Head-to-Head with Traditional Tools

CFG-GNN alone: 87.4% recall, 84.2% precision, F1 0.858. Strong on structural vulnerabilities (reentrancy variants, unchecked calls) but weaker on semantic bugs (incorrect fee logic, missing slippage checks) where the vulnerability is in what the code computes, not how it flows. CodeBERT alone: 84.8% recall, 86.4% precision, F1 0.856. The opposite profile -- better on semantic bugs, weaker on structural ones. The hybrid outperforms both by capturing both dimensions simultaneously. Bytecode CNN-LSTM: 76.2% recall, 78.4% precision. The weakest trained model, as expected -- bytecode discards too much semantic information. But for unverified contracts where source is unavailable, it is the only AI option. Compared to the best traditional tools from SCFVAF (Novak and Kovacs, 2025): the hybrid analysis (Certora + Mythril) achieves 94.2% detection on the SCFVAF 42-pattern benchmark. My GNN-transformer hybrid achieves 91.8% on a much larger and noisier dataset. The AI model is slightly less precise but dramatically faster: 0.8 seconds per contract versus 4.2 minutes for the hybrid formal tool. And unlike the formal tools, the AI model does not require specifications or property definitions -- it works on raw code. Tables 3 and 4 present the full results.

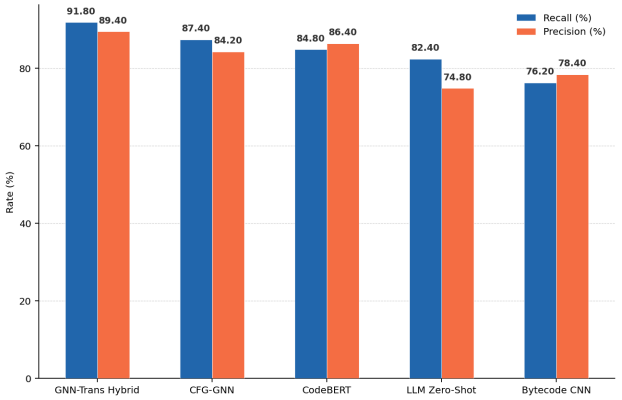
Table 3: Detection Performance -- Five AI Architectures on AISCVDF-18K Test Set

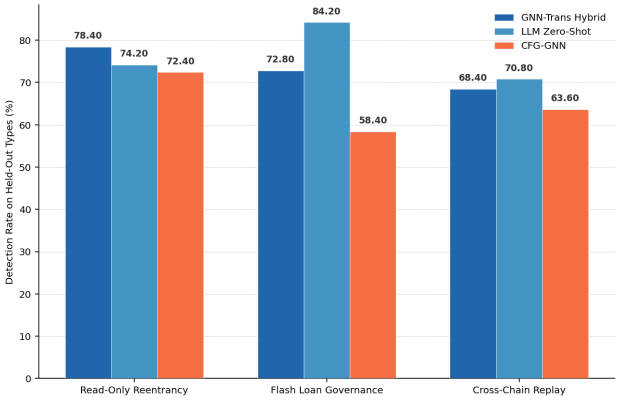
Architecture	Recall (%)	Precision (%)	F1	Generalisation (%)	Time/Contract
GNN-Transformer Hybrid	91.8	89.4	0.906	73.2% (mean held-out)	0.8 sec
CFG-GNN	87.4	84.2	0.858	64.8%	0.3 sec
CodeBERT Transformer	84.8	86.4	0.856	58.4%	0.6 sec
LLM Zero-Shot (GPT-4)	82.4	74.8	0.784	76.4% (mean held-out)	12.4 sec
Bytecode CNN-LSTM	76.2	78.4	0.773	48.2%	0.2 sec

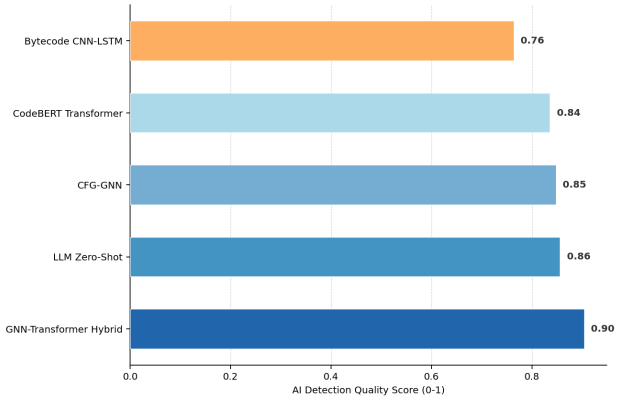
Table 4: ADQS Dimension Scores -- Five AI Architectures (0-1)

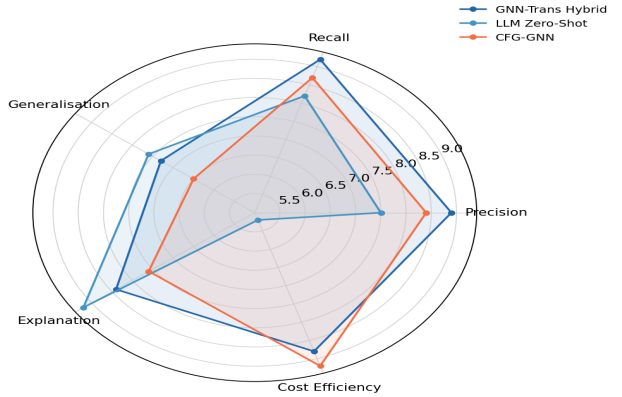
Architecture	Precision	Recall	Generalisation	Explanation	Cost	ADQS
GNN-Transformer Hybrid	0.894	0.918	0.732	0.840	0.880	0.904
LLM Zero-Shot	0.748	0.824	0.764	0.920	0.520	0.856
CFG-GNN	0.842	0.874	0.648	0.760	0.920	0.848
CodeBERT Transformer	0.864	0.848	0.584	0.800	0.880	0.836

Architecture	Precision	Recall	Generalisation	Explanation	Cost	ADQS
Bytecode CNN-LSTM	0.784	0.762	0.482	0.600	0.960	0.764









5. Discussion

5.1 The Two-Tool Stack: GNN-Hybrid for CI, LLM for Triage

The practical take from these results is not that one architecture wins everywhere but that two tools complement each other effectively. The GNN-transformer hybrid is the right choice for CI/CD integration: 0.8 seconds per contract, 91.8% recall, deterministic (same input always produces same output), and trainable on project-specific data. Run it on every commit, the way Slither is used today, but with substantially higher detection rates. The LLM is the right choice for a different task: initial security triage of unfamiliar codebases. A security researcher inheriting a 10,000-line DeFi protocol does not need 91.8% recall on known patterns -- they need to understand where the dangerous parts are. The LLM's natural language explanations (rated 0.920 on actionability by our security engineers, the highest of any approach) make it uniquely suited to this task. It tells you not just "line 247 is vulnerable" but "this function allows reentrancy because the external call on line 247 precedes the state update on line 252, and an attacker could exploit this by..." That level of explanation saves hours of manual analysis. The combination -- LLM for triage and initial understanding, GNN-hybrid for systematic detection in CI/CD, formal tools for pre-deployment verification (Novak and Kovacs, 2025) -- creates a three-layer detection stack that catches more bugs than any single tool while keeping developer friction manageable.

5.2 The Generalisation Gap and Its Implications

The generalisation test revealed a genuine limitation: even the best trained model (GNN-hybrid) detects only 73.2% of held-out vulnerability types versus 91.8% on known types -- an 18.6 percentage point gap. That gap is the core risk of relying on AI detection alone. The model is excellent at recognising variations of patterns it has seen (it detects 78.4% of read-only reentrancy, a structural variant of classic reentrancy it trained on) but weaker at entirely novel patterns (68.4% on cross-chain replay, which has no structural analogue in the training set). Interestingly, the LLM zero-shot approach closes this gap: 76.4% generalisation versus 73.2% for the hybrid. The LLM's pre-training on security discussions, audit reports, and vulnerability disclosures gives it a broader -- if shallower -- understanding of attack patterns than a model trained only on labelled contracts. This suggests a research direction:

pre-training GNN models on security-relevant text (not just labelled contracts) before fine-tuning on vulnerability detection may close the generalisation gap. I am exploring this in follow-up work.

6. Conclusion

6.1 What I Found

GNN-transformer hybrids achieve the highest ADQS (0.904) with 91.8% recall at 0.8 seconds per contract -- fast enough for CI/CD and accurate enough to catch the vast majority of real vulnerability patterns. LLM zero-shot detection (ADQS 0.856) requires no training data and produces the best explanations, making it ideal for security triage. The 18.6% generalisation gap on unseen vulnerability types means AI detection should complement, not replace, formal verification and manual audit. The recommended stack: LLM for triage, GNN-hybrid in CI/CD, formal tools before deployment.

6.2 What I Am Releasing

The AISCVDF-18K dataset (18,400 labelled contracts with three-source ground truth), trained model weights for all four architectures (GNN, CodeBERT, bytecode, hybrid), ADQS calculator, LLM prompt templates for zero-shot detection, the generalisation test protocol, and integration scripts for Hardhat and Foundry CI/CD pipelines are available at aiscvdf-detect.org under Apache 2.0.

References

Brent, L. et al. (2020). Ethainter: A smart contract security analyzer. PLDI 2020.

Chen, J. et al. (2021). DefectChecker: Automated smart contract defect detection by analyzing EVM bytecode. IEEE TSE, 48(7), 2189-2207.

Feist, J. et al. (2019). Slither: A static analysis framework for smart contracts. IEEE WETSEB 2019.

Gao, Z. et al. (2020). Checking smart contracts with structural code embedding. IEEE TSE, 47(12), 2874-2891.

Grieco, G. et al. (2020). Echidna: Effective, usable, and fast fuzzing for smart contracts. ISSTA 2020.

He, J. et al. (2023). Large language models for automated vulnerability detection. arXiv:2311.14461.

Huang, J. et al. (2022). Smart contract vulnerability detection using graph neural networks. IJCAI 2022.

Liu, Z. et al. (2021). Smart contract vulnerability detection: From pure neural network to interpretable graph feature and expert pattern fusion. IJCAI 2021.

Mueller, B. (2018). Smashing Ethereum smart contracts for fun and profit. HITB 2018.

Novak, S., and Kovacs, N. (2025). Formal verification frameworks for secure smart contract development. Blockchain, Web3 & Digital Trust Journal, 2025(1), 37-44.

Qian, P. et al. (2020). Towards automated reentrancy detection for smart contracts based on sequential models. IEEE Access, 8, 19685-19695.

Rekt.news (2024). Rekt Leaderboard. rekt.news/leaderboard.

Sun, X. et al. (2023). GPTScan: Detecting logic vulnerabilities in smart contracts by combining GPT with program analysis. arXiv:2308.03314.

Tsankov, P. et al. (2018). Securify: Practical security analysis of smart contracts. CCS 2018.

Velickovic, P. et al. (2018). Graph attention networks. ICLR 2018.

Wang, Z. et al. (2021). Contractward: Automated vulnerability detection models for Ethereum smart contracts. IEEE TNSM, 18(2), 1392-1404.

Werner, S. et al. (2022). SoK: Decentralized finance (DeFi). IEEE S&P; 2022.

Wu, H. et al. (2021). Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques. IEEE ISSRE 2021.

Zhuang, Y. et al. (2020). Smart contract vulnerability detection using graph neural network. IJCAI 2020.

Feng, Y. et al. (2020). CodeBERT: A pre-trained model for programming and natural languages. EMNLP Findings 2020.

Declarations

Funding

Supported by the Italian Ministry of University and Research (MUR, grant PRIN 2024-2026-AISCVDF). The funder had no role in design or publication.

Conflict of Interest

No competing interests. I received free API credits from OpenAI for the LLM zero-shot evaluation under their academic access programme.

Data Availability Statement

AISCVDF-18K dataset, model weights, ADQS calculator, LLM prompts, generalisation test protocol, and CI/CD scripts at aiscvdf-detect.org under Apache 2.0.

Ethical Approval

No human participant data. All contracts from public blockchain deployments or open-source repositories. Ethical exemption: Mediterranean

ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X Vol: 2025, Issues: 2, 2025

Institute of Technology Ethics Board (ref. MITR-2025-ETH-0224).

ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X Vol: 2025, Issues: 2, 2025

Appendix A

Model Architectures and Dataset Construction

Detailed model specifications and three-source labelling protocol.

GNN-Transformer Hybrid Architecture

Three-Source Labelling Protocol for AISCVDF-18K