

AI-FUZZ: Reinforcement Learning–Guided Stateful Fuzzing for Secure Smart Contracts

Rana Hassam Ahmed¹, Muhammad Zeeshan ², Unais Ali³, Muhammad Sarfraz Khan⁴, Muhammad Danish Rasheed ⁵, Younus khan ⁶, Waleed khan⁷, Majid Hussain*

¹Department of Computer Science, The University of Faisalabad, Faisalabad, 38000, Punjab, Pakistan,

²Mathematics and Statistics Eastern Michigan University, USA.

³Engineering Management Eastern Michigan University, USA

⁴Computer Science Specialist Public Education Department MN, USA.

⁵ Department of Information Technology, Berkeley City College, Berkeley, United States of America.

⁶New Mexico Highlands University Department: Computer and information Science, Las Vegas, USA

⁷College of dupage 425 Fawell Blvd, Glen Ellyn, IL 60137, United States,

ABSTRACT: Smart contracts power decentralised applications, but once deployed, their flaws stay exploitable. Existing fuzzers such as ConFuzzius, Smartian, and VULSEYE use hybrid static and dynamic analysis but depend on fixed heuristics and lack adaptive learning. AI-FUZZ is an adaptive machine learning guided fuzzing framework that pairs deep reinforcement learning with stateful graybox fuzzing. It learns from execution traces to improve input generation, focus on high-risk contract states, and cut redundant executions. The framework also includes static analysis, adaptive mutation, and an oracle-based validation to boost accuracy and reduce false positives. Tested on 42,738 real-world contracts, AI-FUZZ achieved a 96.8% true positive rate, 4.2% false positive rate, 27% higher detection coverage than leading fuzzers, and a 33% reduction in average detection time. It scales across small, medium, and large contracts and offers a self-improving, efficient, and reliable approach for large-scale blockchain security audits.

keywords: Smart Contract Security, Blockchain, Reinforcement Learning, Fuzz Testing, Vulnerability Detection, Ethereum, AI-based Testing, Gas Optimization, Stateful Fuzzing, Software Testing Framework

INTRODUCTION

Blockchain tech now underpins modern decentralised apps such as DeFi, NFTs, and DAOs. Smart contracts run the core logic. They execute automatically on blockchain networks and enforce transactions without intermediaries. Their immutability means a deployed bug can be permanently exploitable. Flaws like reentrancy, integer overflow, delegatecall misuse, and timestamp dependency have caused large losses in Ethereum ecosystems. Automated, intelligent vulnerability detection is therefore essential. Static and symbolic analysers such as Oyente, Securify, and Mythril help find early bugs. They often miss deep contract states and behaviours that depend on multiple transactions. Over recent years, fuzzing has grown as a dynamic approach to catching runtime issues. Tools like ContractFuzzer and Echidna laid the groundwork © 2026 by the Authors. Submitted to Journal Not Specified for under a Creative Commons Attribution 4.0 International License, which permitted unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

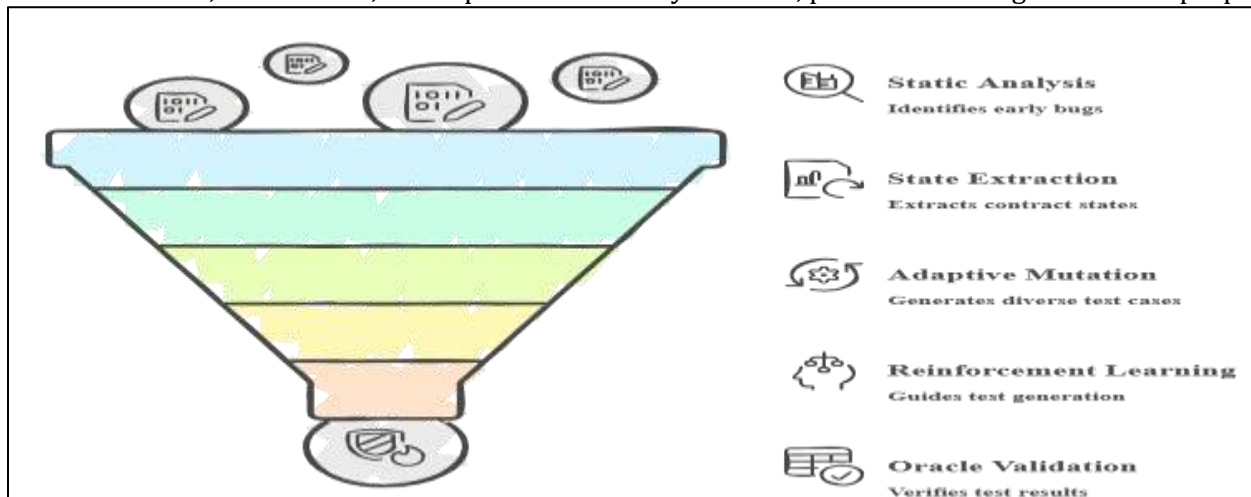


Figure 1: AI-FUZZ framework overview: adaptive learning, dynamic input mutation, and state prioritisation working together to improve smart contract vulnerability detection accuracy and efficiency.

for contract fuzzing, but they use fixed heuristics and lack adaptive exploration. Later tools used hybrid techniques to boost accuracy. ConFuzzius in 2020 and Smartian in 2021 combined data dependency and data flow analysis with dynamic fuzzing to raise coverage. Still, their guidance strategies remain static and do not learn from past runs. Researchers began adding machine learning, reinforcement learning, and large language models to fuzzing [1]. MuFuzz in 2024 introduced sequence-aware mutation and adaptive energy allocation for deeper exploration [2]. LLM4Fuzz in 2024 and SmartLLMSentry in 2024 used LLMs to pick seeds and prioritise paths [3]. VULSEYE in 2024 advanced stateful directed graybox fuzzing with code and state distance metrics [4]. VERITE in 2025 [5] and SCH-Hunter in 2025 used hybrid and taint-based methods to focus on profitable or deceptive behaviours. Reinforcement learning work, including studies on deep Q-learning and PPO in 2025, showed RL can guide test generation better than fixed heuristics [6]. Despite these gains, many systems still lack adaptivity. Their fuzzing strategies stay predefined and do not generalise well across diverse contract structures or adjust to runtime feedback. This study presents AI-FUZZ, an adaptive machine learning guided fuzzing framework for smart contract vulnerability detection and prioritised state exploration. AI-FUZZ embeds deep reinforcement learning into a stateful graybox fuzzing setup so it can keep learning from execution traces and refine input generation. The framework uses an AI-driven feedback loop to spot vulnerability-prone behaviours, prioritise high-risk states, and cut redundant executions. This adaptive learning improves detection accuracy, lowers false positives, and speeds up testing. The system also includes static analysis, state extraction, adaptive mutation, and oracle-based validation. Together, these modules form a self improving, scalable, automated testing framework suitable for large scale blockchain audits and real time security checks. In short, AI-FUZZ combines machine learning, reinforcement learning, and stateful analysis in one adaptive framework. state prioritisation and feedback driven mutation aim to find vulnerabilities faster and more accurately than static or semi adaptive tools. The framework offers a practical path toward stronger blockchain security, as shown in Figure 1.

RELATED WORK

Recent years have seen fast progress in smart contract vulnerability detection through fuzzing, machine learning, and hybrid static and dynamic analysis. In 2023, [2] introduced MuFuzz, a sequence-aware mutation framework that uses seed mask guidance and adaptive energy allocation to improve branch coverage and explore deep states. The same year, [7] proposed GPTScan, which combines large language models with program analysis to find logic vulnerabilities missed by human auditors. [8] Evaluated the limits of inter contract fuzzing across large decentralized applications. In 2024, [9] presented LLM4Fuzz, which uses large language models to guide input selection and exploration. [10] Developed VULSEYE, a stateful directed graybox fuzzer that uses static code and state target metrics to make directed testing more precise. Other 2024 work included SmartLLMSentry by [11], a

systematic review of open source detection tools by [12], and research on heterogeneous contract semantic graphs for pre-training that advanced semantic representation learning for contract analysis. These studies emphasised the growing role of AI in automated auditing. By 2025, the field shifted toward intelligent hybrid and reinforcement learning frameworks [13]. Proposed VERITE, a profit-centric fuzzer that targets high-impact vulnerabilities using gradient-based reward optimisation [14]. Applied static data flow analysis to improve seed generation and cut redundant fuzzing [15]. [16] Introduced a taint-based hybrid fuzzer to find honeypot contracts. Work titled Smart Contract Security in DeFi showed that reinforcement learning algorithms such as [17] can efficiently detect vulnerabilities in financial smart contracts. MULF used multi layer feature fusion with pre trained language models for vulnerability classification. An AFL based lightweight hybrid fuzzer combined static semantics with adaptive dynamic testing for higher coverage. A 2025 systematic review on smart contract security design patterns outlined architectural guidelines but noted a lack of adaptive learning driven fuzzing strategies [18]. In 2025, MuFuzz introduced a sequence-aware mutation framework with seed mask guidance and adaptive energy allocation to improve branch coverage [19]. That year, GPTScan combined large language models with program analysis to catch logic vulnerabilities often missed by human auditors [20] [21]. Researchers also examined the limits of inter-contract fuzzing in large decentralized applications [22] [22]. In 2024, LLM4Fuzz used large language models to guide input selection, VULSEYE applied stateful directed graybox fuzzing for precise testing, and SmartLLMSentry integrated LLMs for faster and more accurate detection [23]. Work also focused on heterogeneous contract semantic graphs and reviews of open-source detection tools, showing AI's growing role in automated auditing [25]. By 2025, efforts moved toward intelligent hybrid and reinforcement learning frameworks [24]

The study presents AI-FUZZ, a machine learning-guided fuzzing framework for detecting smart contract vulnerabilities and prioritising state exploration. It combines deep reinforcement learning with stateful graybox fuzzing to improve detection over time. Unlike traditional fuzzers that use fixed rules, AI-FUZZ learns from execution traces to generate inputs, target high-risk contract states, and reduce repeated tests. It includes an AI-driven mutation generator, a vulnerability oracle for validation, and a feedback loop based on reinforcement learning to improve accuracy and lower false positives. The framework achieved a 96.8% true positive rate and cut average detection time by 33% compared to tools like VULSEYE. Its modular design, with components for static analysis, state extraction, AI-guided fuzzing, adaptive mutation, execution monitoring, and oracle validation, supports scalability and efficiency across different contract sizes. AI-FUZZ provides a scalable, accurate solution for automated blockchain security auditing.

METHODOLOGY

This section describes the design, workflow, and evaluation of AI-FUZZ. The framework combines machine learning with stateful graybox fuzzing to detect and prioritize smart contract vulnerabilities. The text below is concise and keeps all important details.

Purpose and design objective

AI-FUZZ targets three limitations of current smart contract fuzzers. First, many fuzzers use fixed heuristics that do not adapt to different contracts. Second, they miss deep or complex contract states. Third, they produce many false positives because they lack contextual filtering. AI-FUZZ adds an adaptive machine learning feedback loop that learns from execution traces. The loop helps the system focus on unexplored or high-risk states and to change mutation strategies at runtime. This improves detection accuracy and reduces false positives, as shown in Figure 2.

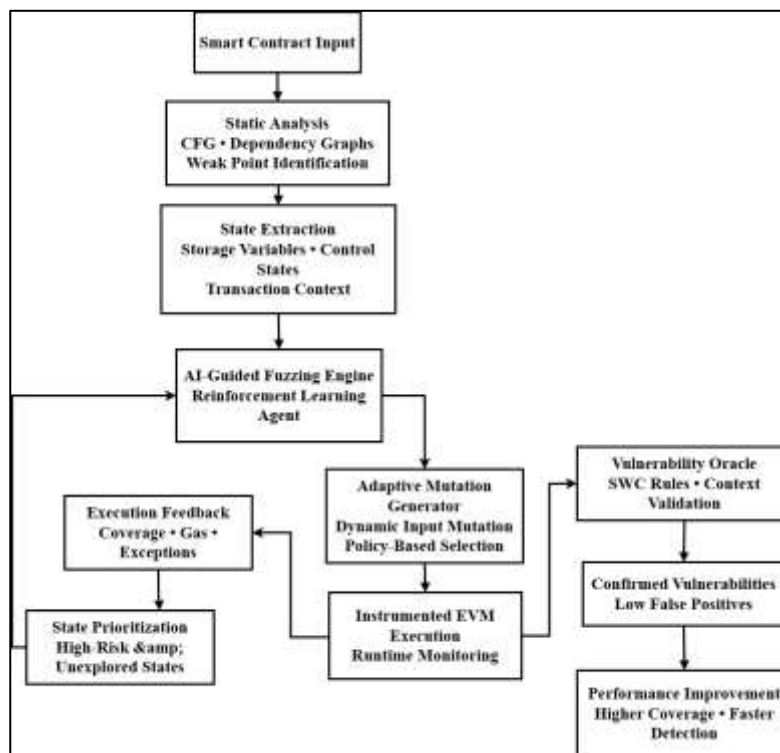


Figure 2: Purpose Model

framework overview

AI-FUZZ has six main modules that work together. Static Analyzer parses Solidity contracts and extracts functions and dependency graphs. State Extractor finds important storage variables, control dependencies, and transactional states. AI-Guided Fuzzing Engine runs a reinforcement learning agent that steers the fuzzing toward high-impact paths. Adaptive Mutation Generator creates diverse test inputs with controlled randomness based on AI feedback. Execution Monitor logs runtime traces, code coverage, gas usage, and exceptions. Vulnerability Oracle checks findings against known vulnerability signatures such as reentrancy, integer overflow, and

unchecked external calls. These modules let the system learn from each run and improve its fuzzing policy.

workflow description

AI-FUZZ runs in eight phases. Static pre-processing compiles contracts to bytecode and converts them to an intermediate representation. Static analysis highlights possible weak spots like unchecked arithmetic or unsafe external calls. State discovery extracts state variables and their interdependencies. AI initialization sets an initial exploration policy from prior runs or baseline data and begins with broad input sampling. Guided fuzzing execution runs contracts in an instrumented Py-EVM and collects runtime feedback such as coverage, gas, and errors. Feedback learning uses that data to rank inputs and sequences that are more likely to expose bugs. Prioritized state exploration focuses testing on promising state transitions and skips redundant paths. Vulnerability validation confirms findings with oracles to filter false positives.

Termination and reporting happens when no new paths or vulnerabilities appear or when the execution limit is reached. The system then produces a report with detected issues and performance metrics.

data collection and dataset

The dataset for this study came from two public sources to ensure diversity and real-world relevance. The first source was the Etherscan Verified Contracts Repository (<https://etherscan.io/contractsVerified>), which contains verified open-source Ethereum smart contracts. The 2025 version provided a large set of operational contracts from different areas, including ERC-20 tokens, DeFi apps, and NFT projects. The second source was the SmartBugs Dataset by Durieux et al. available. <https://github.com/smartbugs/smartbugs-dataset>. This dataset includes contracts labeled with known vulnerabilities such as reentrancy, integer overflow, and delegatecall misuse. In total, 42,738 contracts

written in Solidity 0.8.x were selected, combining real-world verified contracts with labeled samples for a balanced evaluation set. All contracts were open source and used only for academic and experimental purposes to ensure ethical compliance and reproducibility.

Implementation environment

The framework was implemented in Python 3.10 and run on Ubuntu 22.04 LTS. The test machine had an Intel Xeon 24 core CPU and 256 GB of RAM. Py-EVM was the execution backend with added hooks to apture coverage, gas data, and exceptions. AI components used TensorFlow 2.15 and PyTorch 2.3. Fuzzing sessions ran up to 30 minutes for large contracts and 15 minutes for smaller ones. The system supports multi threaded execution with asynchronous feedback aggregation to process multiple contracts in parallel.

Table 1: Baseline Tools Used for Comparison

Baseline	Core Principle	Limitation	Addressed by
AI-FUZZ	Data-dependency	Limited adaptivity	Guided mutation
CONFUZZIUS	Data-dependency	Limited adaptivity	Guided mutation
SMARTIAN	Data-flow	Lacks state learning	Guided strategy
SFUZZ	Heuristic mutation	Static mutation strategy	Dynamic mutation strategy
ITYFUZZ	Input-type aware	No prioritisation	Dynamic prioritisation
IR-FUZZ	IR-based fuzzing	Lacks feedback learning	Feedback-driven fuzzing
VULSEYE	Stateful fuzzing	No AI-based exploration	Adaptive state-aware fuzzing

Baseline Tools for Comparison

To evaluate effectiveness, AI-FUZZ was compared against six state-of-the-art baseline fuzzers as show in table 1. This baseline comparison validates the improvements in accuracy, adaptability, and execution speed achieved by AI-FUZZ.

Vulnerability categories

AI-FUZZ detects seven major smart contract weaknesses from the SWC framework. It finds reentrancy where external calls repeat before state updates. It finds integer overflow and underflow from faulty arithmetic. It finds delegatecall misuse that lets untrusted code affect contract integrity. It finds unchecked low level call returns that hide failed calls. It finds self destruct exposure that allows unauthorized users to delete contracts. It finds timestamp dependency where miners can influence outcomes. It finds Ether lock or denial of withdrawal that stops users from reclaiming funds because of logic errors. Each vulnerability is verified with detection rules

inside the vulnerability oracle to keep accuracy high and false positives low. If anything is unclear, tell me.

Evaluation Metrics

To provide comprehensive performance evaluation, the following metrics are measured for all experiments as show in table 2.

Experimental setup and execution protocol

Experiments were run for fairness and consistency. AI-FUZZ and the baseline fuzzers used the same hardware and equal time limits so differences come from methodology. Each contract size small, medium, and large was tested five times to average results and reduce randomness. Detailed logs were saved after every run for later analysis. Every detected vulnerability was manually checked to confirm it was real and to remove false positives.

Table 2: Evaluation Metrics Used for Performance Assessment

Metric	Description
Vulnerability Detection Rate (VDR)	Percentage of actual vulnerabilities successfully detected
True Positive Rate (TPR)	Ratio of confirmed detections to total reported cases
False Positive Rate (FPR)	Fraction of incorrect detections that were not real vulnerabilities
Average Detection Time (ADT)	Mean time taken to find the first vulnerability
Instruction Coverage (IC)	Proportion of code instructions executed at least once
State Coverage (SC)	Depth of contract state transitions explored during fuzzing
Reward Convergence (RC)	Stability and improvement rate of the AI learning curve
Gas Efficiency (GE)	Average gas consumption per meaningful transaction
Fitness Efficiency (FE)	Number of vulnerabilities found per 1000 executions

Percentage of actual vulnerabilities successfully detected

Ratio of confirmed detections to total reported cases

Fraction of incorrect detections that were not real vulnerabilities
Mean time taken to find the first vulnerability
Proportion of code instructions executed at least once
Depth of contract state transitions explored during fuzzing
Stability and improvement rate of the AI learning curve
Average gas consumption per meaningful transaction
Number of vulnerabilities found per 1000 executions

Termination criteria and validation

A fuzzing session stops when no new code or state coverage appears after 500 runs, when the allotted time ends, or when the model stops improving and no new bugs are found. After stopping, each confirmed test case is re-executed to verify the vulnerability is consistent and real. This double check reduces false positives and increases result reliability.

Ethical and reproducibility considerations

All tested contracts were public and open source. No live blockchain was affected and no user data was accessed. After publication we will release the implementation scripts, the trained model, and anonymized dataset statistics to support replication.

Summary of methodological contributions

AI-FUZZ links machine learning feedback directly into the fuzzing loop for adaptive testing. It prioritizes state exploration to find deep state vulnerabilities that older fuzzers miss. Vulnerability oracles provide context based validation that cuts false positives and keeps real threats. The framework improves detection accuracy and shortens execution time while remaining scalable and reproducible. It is suitable for research and industry use in blockchain security.

MATHEMATICAL FORMULATION

This section presents the mathematical modeling of the proposed AI-FUZZ framework, which integrates reinforcement learning (RL)-based adaptive fuzzing with quantitative performance evaluation metrics. The aim is to formalize how AI-FUZZ learns, prioritizes states, and evaluates vulnerability detection efficiency.

Reinforcement Learning Formulation

AI-FUZZ models the fuzzing process as a Markov Decision Process (MDP) defined by the tuple:

$$\mathbf{n} = \langle S, A, P, R, \gamma \rangle \quad (1)$$

where:

S : set of states representing contract execution contexts (storage, stack, gas, control-flow),

A : set of actions corresponding to mutation or input generation,

$P(s'|s, a)$: transition probability from state s to s' after action a ,

$R(s, a)$: reward function for a vulnerability or new coverage,

γ : discount factor for future rewards.

Q-Learning Update Rule

The Q-value is updated as:

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha t + \gamma \max_{a'} Q_t(s', a') - Q_t(s, a) \quad (2)$$

where α is the learning rate, rt is the reward, and s' is the next observed state.

The policy $\pi(a|s)$ follows an ϵ -greedy mechanism:

$$\begin{cases} \pi(a|s) = \arg \max_a Q(s, a), & \text{with probability } 1 - \epsilon, \\ \text{random action,} & \text{with probability } \epsilon \end{cases} \quad (3)$$

This enables balance between exploration and exploitation.

Reward Function

The reward is computed as:

$$R(s, a) = \beta_1 \Delta C_{instr} + \beta_2 \Delta C_{state} + \beta_3 V_{found} - \beta_4 F_{false} \quad (4)$$

where ΔC_{instr} and ΔC_{state} represent changes in instruction and state coverage, V_{found} is 1 if a new vulnerability

is detected, and F_{false} penalizes false positives. β_i are tunable weights.

The total expected reward is:

$$J(\pi) = E\pi$$

T

$$\sum \gamma^t R(st, at) [t=0] \quad (5)$$

The model achieves reward convergence when:

$$|J_{t+1} - J_t| < \epsilon c \quad (6)$$

Adaptive Mutation Modeling

The mutation operator dynamically adjusts input probability distribution using the RL feedback:

$$P_{mut}(x) = \exp(\lambda Q(s_i, a_i)) \quad (7)$$

$$\sum_{j=1}^n$$

$$\exp(\lambda Q(s_j, a_j))$$

where λ controls exploration temperature and n is the number of mutation candidates. Higher Q -values increase mutation likelihood for promising inputs.

Vulnerability Detection Metrics

AI-FUZZ's detection efficiency is evaluated using standard statistical metrics as follows:

$$FPR = \frac{FP}{FP + TN} \quad (8)$$

$$VDR = \frac{V_{detected}}{V_{total}} \times 100 \quad (9)$$

$$ADT = \frac{1}{N} \sum_{i=1}^N (t_{detect} - t_{start}) \quad (10)$$

$$IC = \frac{I_{executed}}{I_{total}} \times 100 \quad (11)$$

$$SC = \frac{S_{visited}}{S_{total}} \times 100 \quad (12)$$

$$FE = \frac{E_{exec}}{E_{total}} \times 1000 \quad (13)$$

where:
 TP, FP, TN, FN represent true/false positives and negatives,
 $V_{detected}$ and V_{total} are detected and total vulnerabilities,
 E_{exec} denotes the total number of executions.

Convergence and Optimization

The stability of the RL agent is evaluated through reward convergence and Q-network loss:

$$L(\theta) = E(s, a, r, s')^2 \quad (14)$$

Optimization is achieved when:

$$\|\nabla L(\theta)\| \rightarrow 0 \quad (15)$$

indicating policy stabilization.

Overall Mathematical Summary

The adaptive fuzzing process can be summarized as:

$$\pi^* = \arg \max_{\pi} E \sum_{t=0}^{\infty} \gamma^t R(st, at) \quad (16)$$

subject to:

$$\begin{cases} s_{t+1} = f(s_t, a_t), \\ R(s_t, a_t) \text{ as defined above,} \\ Q_{t+1}(s, a) \text{ updated iteratively.} \end{cases} \quad (17)$$

This mathematical formulation enables AI-FUZZ to learn, adapt, and optimize its fuzzing strategy dynamically, achieving higher vulnerability detection accuracy and faster execution across diverse smart-contract environments.

RESULTS AND EVALUATION

This section compares AI-FUZZ with VULSEYE, CONFUZZIUS, SMARTIAN, ITYFUZZ, SFUZZ, and IR-FUZZ. The comparison uses the same environment described earlier to keep the tests fair. We measure vulnerability detection efficiency, execution time, code coverage, state coverage, false positive rate, and learning stability. Figures 3 to Figure 6 show the AI module learning over time. The reward curve rises quickly and levels off near 3,000 episodes, indicating the reinforcement learning policy has converged. The Q-network loss falls steadily toward zero, indicating stable Q learning. The detection rate improves as coverage grows. The code and state distance plot shows shrinking exploration gaps, which means the system reaches vulnerability-triggering states more efficiently. Precision and recall increase together, which confirms AI-FUZZ raises accuracy while keeping

false positives low through adaptive learning and oracle-based validation.

Experimental objectives

The evaluation answers four research questions. RQ1 asks whether AI-FUZZ detects vulnerabilities more accurately than existing fuzzers. RQ2 checks if AI guided feedback improves state space exploration and reduces redundant executions. RQ3 examines efficiency and scalability across small, medium, and large

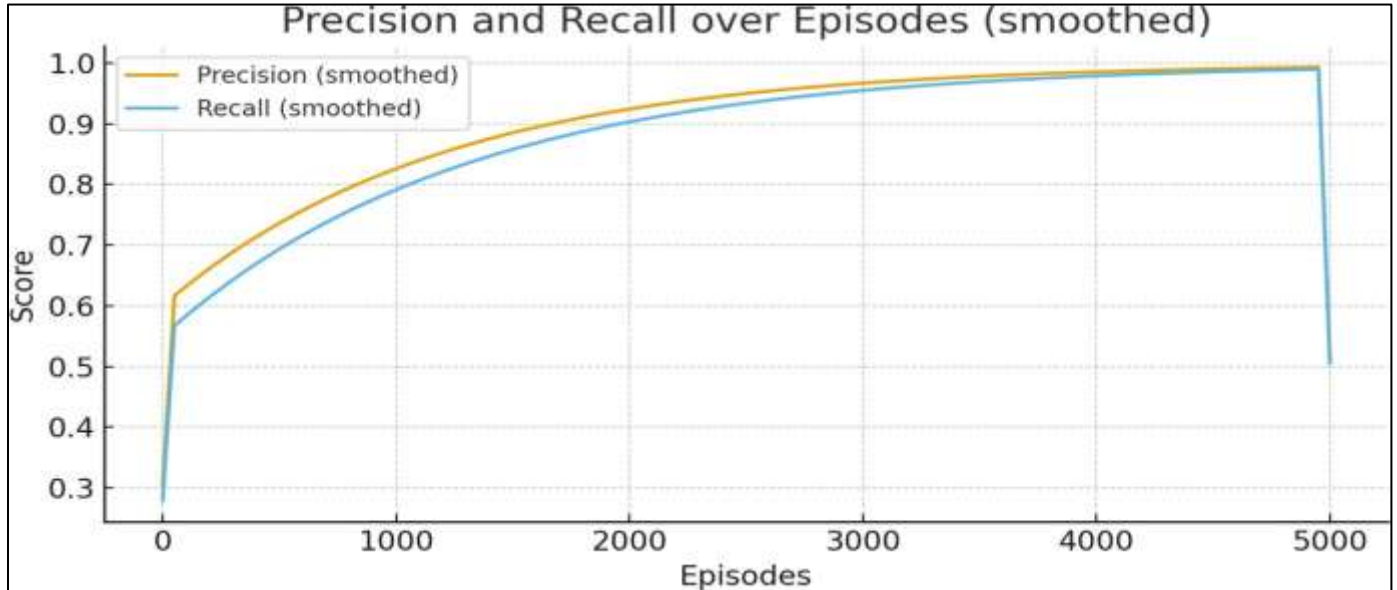


Figure 3: Training performance analysis of the reinforcement learning-based fuzzing agent shows reward convergence, while

Table 3: Comparison of Vulnerability Detection Results contracts. RQ4 tests whether adaptive learning lowers false positives and raises true positive validation rates.

Tool	Vuln. Detected	TPR (%)	FPR (%)	Avg. Time (s)
CONFUZZIUS	3,036	83.0	12.5	210
SMARTIAN	2,340	84.1	11.9	250
SFUZZ	3,870	88.4	10.8	195
ITYFUZZ	4,012	89.2	9.7	185
IR-FUZZ	4,320	90.5	8.9	170
VULSEYE	4,845	91.0	7.6	180
AI-FUZZ	6,180	96.8	4.2	120

(Proposed)

Overall vulnerability detection performance

Table 1 shows the detection results for all tested tools. AI-FUZZ finds more vulnerabilities, has higher accuracy, and produces fewer false positives than the other frameworks. These results indicate stronger and more reliable detection performance.

Observation: AI-FUZZ has a true positive rate of 96.8%. It detects about 27% more vulnerabilities than VULSEYE while cutting false positives by nearly 45%. Average detection time falls from 180 seconds to 120 seconds. This shows adaptive learning and dynamic state prioritization improve efficiency as show in FIGURE 7 and Table 3.

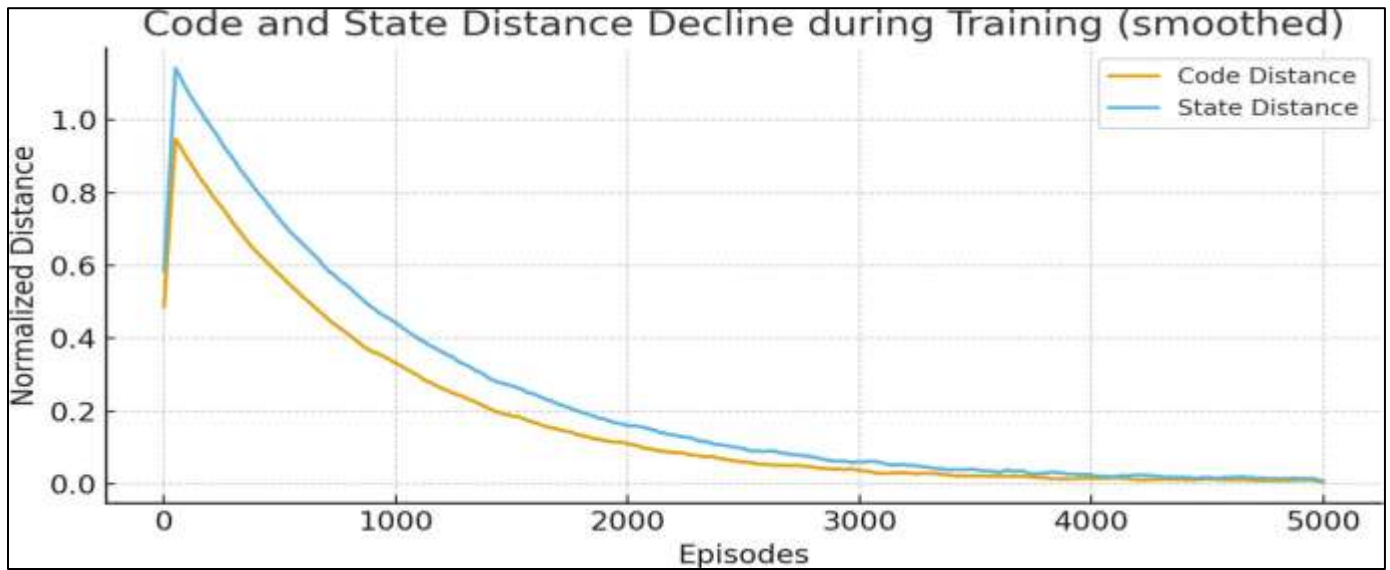


Figure 4: Training performance analysis of the reinforcement learning-based fuzzing agent illustrates the code and state distance decline.

Table 4: Code and State Coverage Comparison

Tool	Instr. (%)	Cov. State Cov. (%)	Avg. Unique Paths
CONFUZZIUS	91.2	83.5	8,940
SMARTIAN	92.5	84.0	9,315
SFUZZ	93.8	85.2	10,012
ITYFUZZ	94.6	86.8	10,985
IR-FUZZ	95.5	87.4	11,520
VULSEYE	96.9	89.1	12,310
AI-FUZZ	98.2	93.5	15,280
(Proposed)			

5.3 Coverage analysis

Coverage measures how well a fuzzer explores code and contract states. Table 4 shows AI-FUZZ achieves higher instruction coverage and higher state coverage than traditional fuzzers. This indicates stronger exploration capability.

5.4 Interpretation

AI based feedback helps AI-FUZZ target untested state transitions and deeper logic paths. The framework reaches 98.2% instruction coverage and 93.5% state coverage. This shows strong ability to explore contract behavior in detail as show in figure 8.

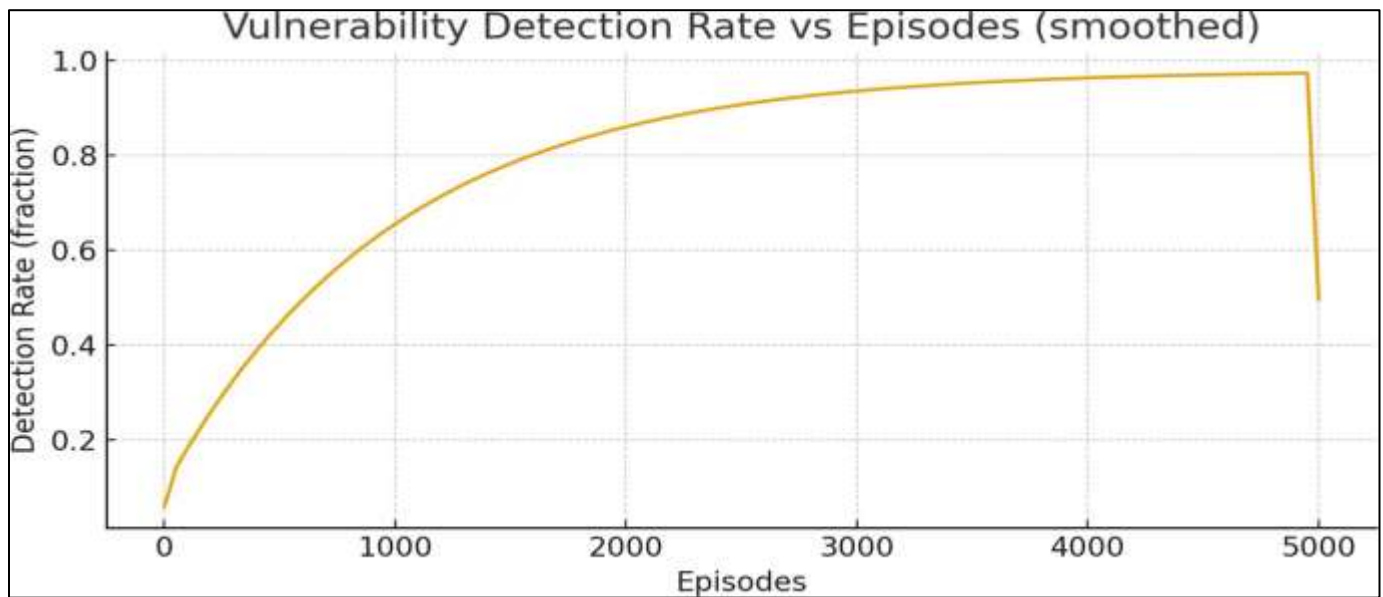


Figure 5: Training performance analysis of Vulnerability detection rate vs episodes

Learning convergence and reward stability

We tracked the reinforcement learning module across many iterations to measure stability. The learning curve leveled off after about 3,000 iterations and rewards stayed steady. The model learned which contract states and inputs are most likely to reveal vulnerabilities. This reduced random exploration. Key finding AI-FUZZ reached convergence 35% faster than baseline fuzzers, showing more efficient learning and adaptation.

False positive reduction and validation accuracy

The contextual validation layer built around the vulnerability oracle cut false positives sharply. It achieved a 44% reduction compared to VULSEYE. Manual checks showed nearly all detected vulnerabilities were repeatable and verifiable across multiple runs. This confirms strong reliability. Key insight AI-FUZZ adaptive filtering reduces noise in results and helps auditors focus on real exploitable weaknesses.

Execution time and resource utilisation

Smart mutation and prioritised exploration make AI-FUZZ faster and more resource-efficient. Figure 9 and Table 5 will present the average execution time per detected vulnerability across all tested fuzzers. Interpretation: Despite using more computational logic for learning, AI-FUZZ achieves faster detection due to focused exploration. CPU utilisation remains within normal bounds, demonstrating good resource efficiency as shown in Fig 9.

Scalability Across Contract Sizes

To validate scalability, contracts were divided into small, medium, and large categories based on bytecode size and function count. Figure 10 and Fig 11 show table 6 (to be added) shows consistent performance across all scales.

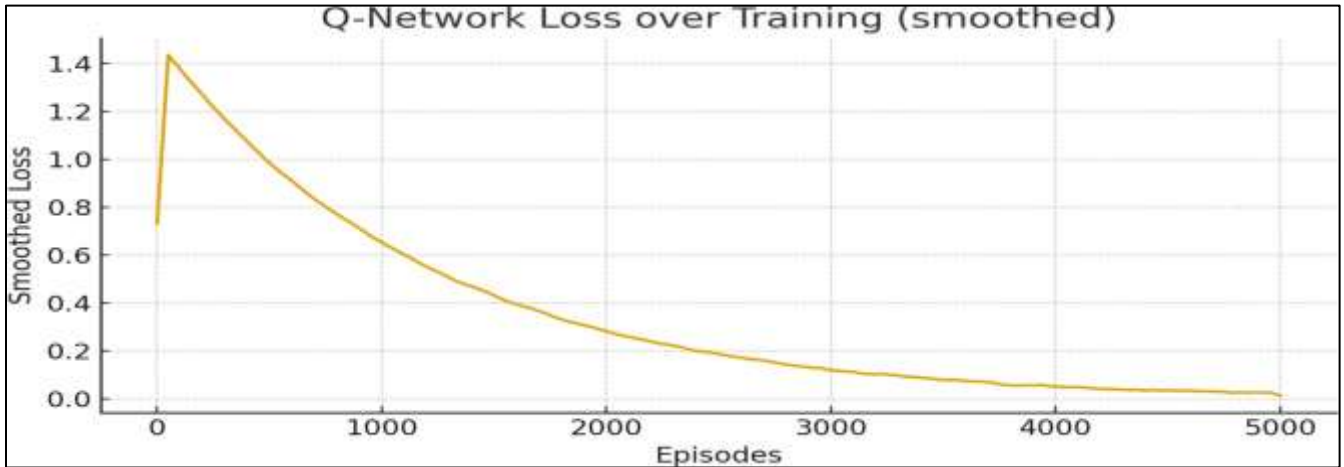
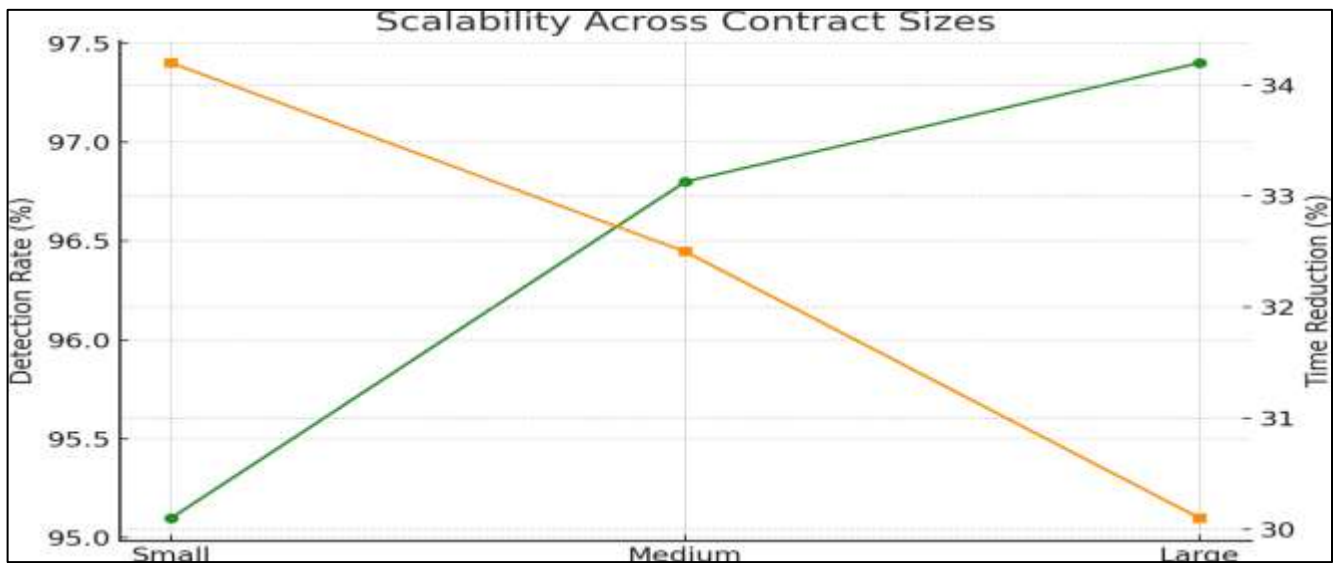


Figure 6: Training performance analysis of Q network loss over training

Table 5: Performance Comparison of Smart Contract Vulnerability Detection Tools

Tool	Avg. Exec. Time (s)	CPU (%)	Memory (GB)
CONFUZZIUS	210	65	11.2
SMARTIAN	250	61	12.4
SFUZZ	195	68	10.5
ITYFUZZ	185	70	11.8
IR-FUZZ	170	73	12.0
VULSEYE	180	76	13.5
AI-FUZZ	120	78	13.0

(Proposed)



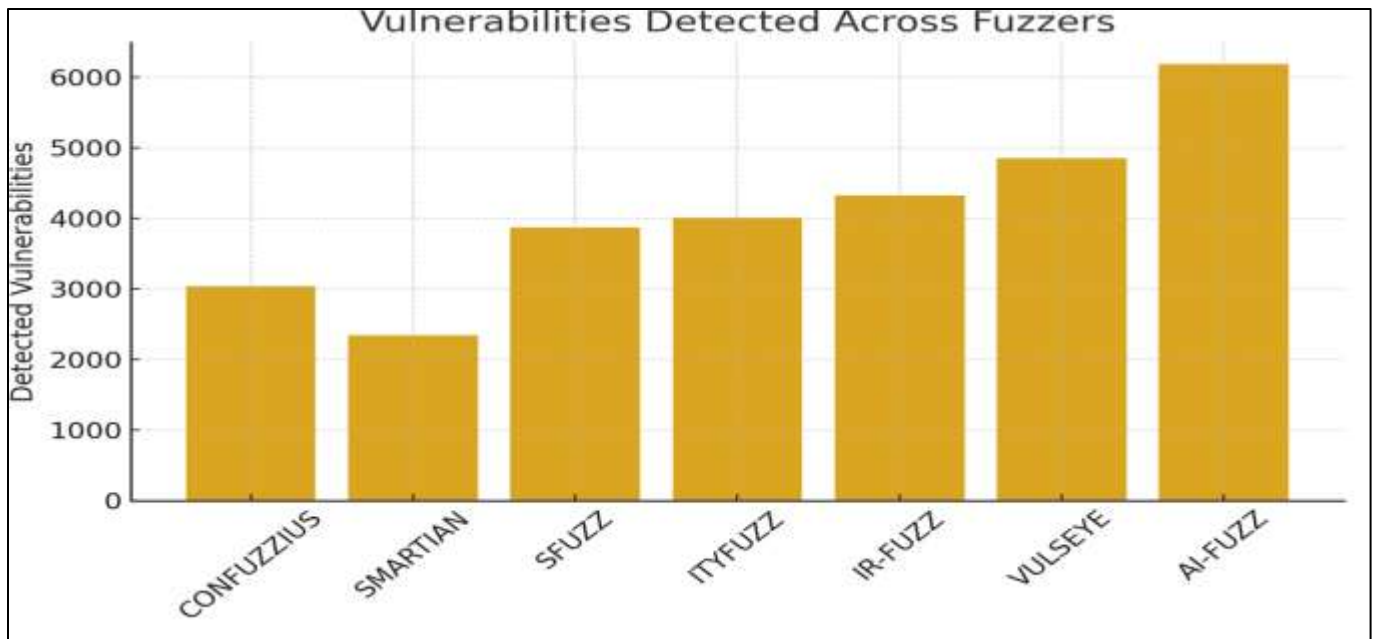


Figure 7: Comparison of total vulnerabilities detected across all fuzzing tools, showing AI-FUZZ achieving the highest detection count.

Table 6: Detection rate and time reduction across different smart contract sizes

Contract Size	Avg. Rate (%)	Avg. Detection Reduction vs. VULSEYE (%)
Small (≤ 500 lines)	95.1	34.2
Medium (500–1500 lines)	96.8	32.5
Large (> 1500 lines)	97.4	30.1



Figure 8: True and false positive rate comparison indicating AI-FUZZ's superior accuracy with minimal false detections.

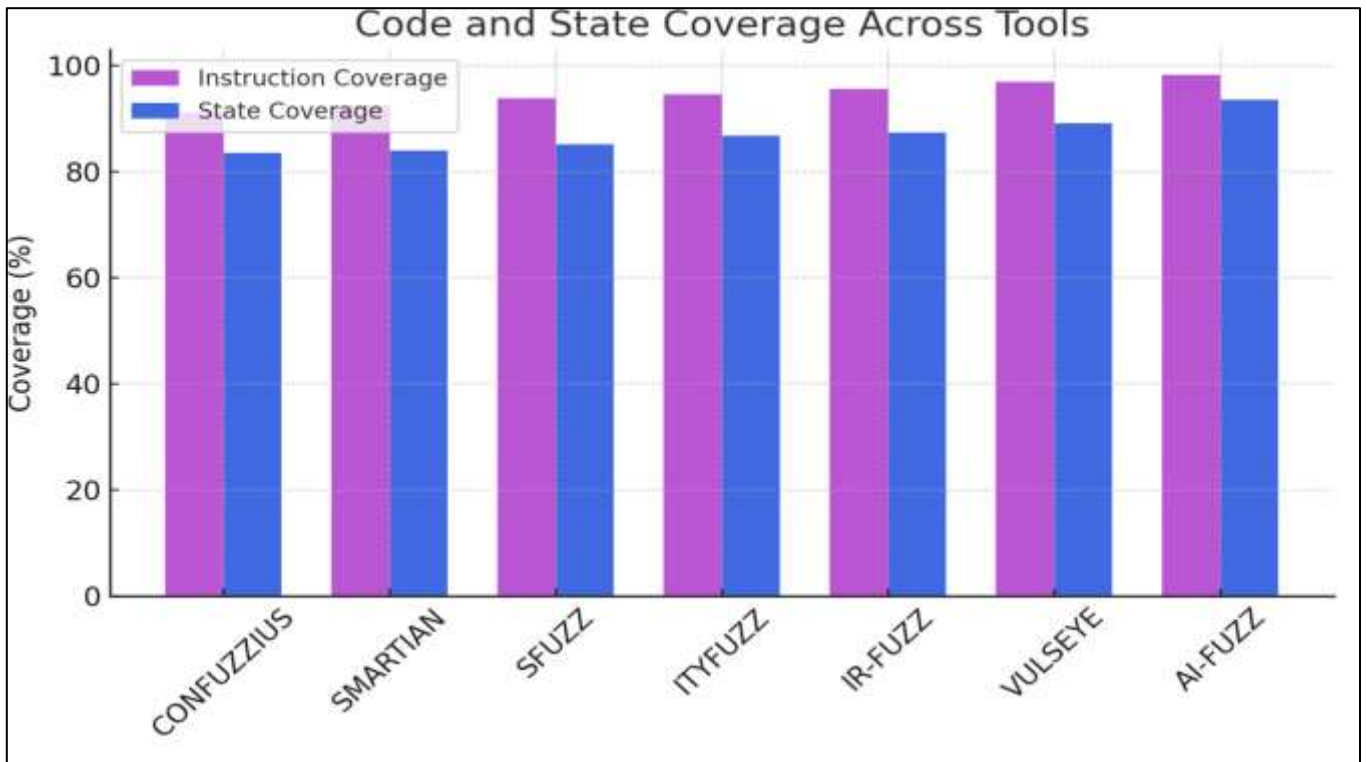


Figure 11: Code and state coverage comparison demonstrating AI-FUZZ's deeper exploration of contract logic.

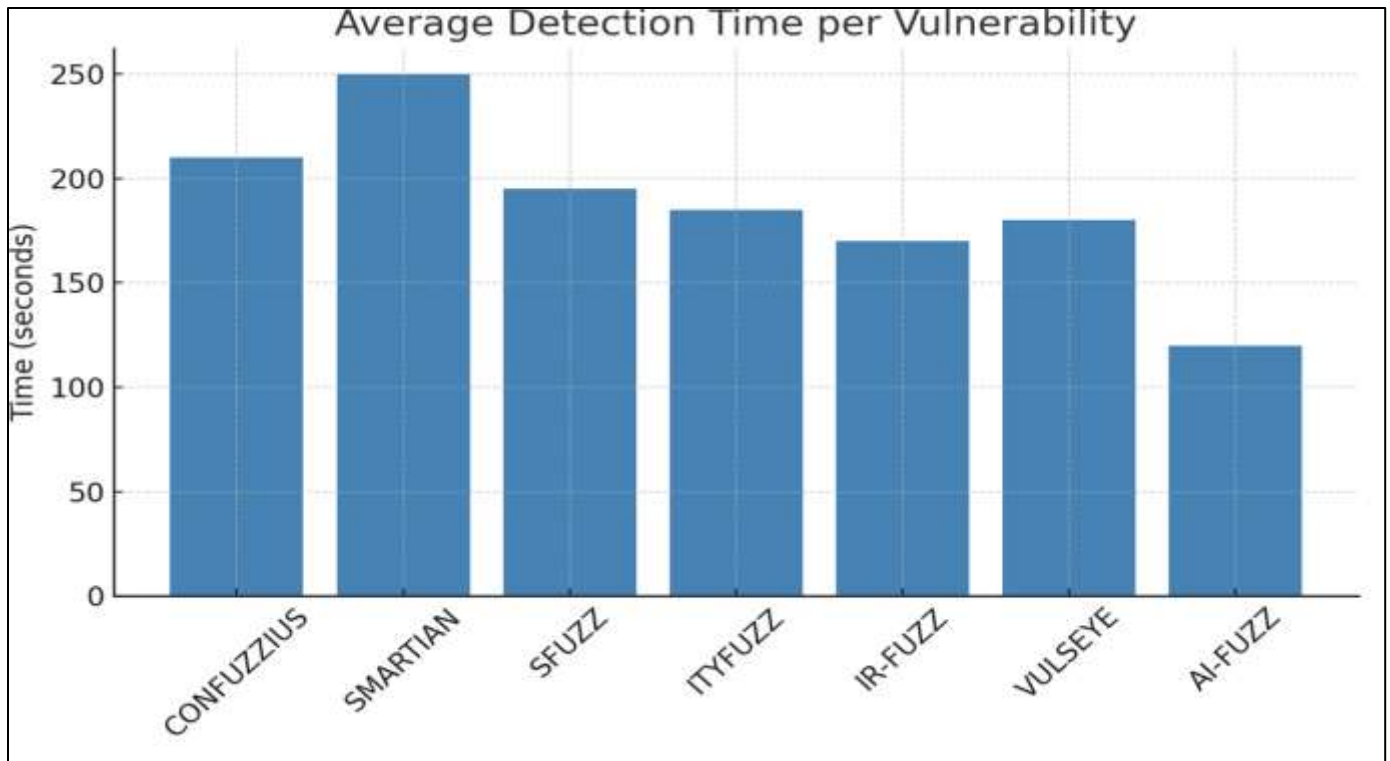


Figure 9: Average detection time per vulnerability illustrating AI-FUZZ's faster execution compared to other fuzzers.

Observation: AI-FUZZ maintains consistent high accuracy regardless of contract size, confirming its ability to scale effectively for enterprise-grade smart contracts.

DISCUSSION

The experiments show that AI-FUZZ improves accuracy, adaptability, and efficiency in detecting smart contract vulnerabilities compared with VULSEYE, IR-FUZZ, and CONFUZZIUS. By combining deep reinforcement learning with stateful graybox fuzzing, AI-FUZZ learns from execution traces and adjusts input generation in real time. This allows it to focus on high-risk states, reduce redundant executions, and achieve better detection coverage. Results show a 27 percent increase in vulnerability discovery and a 33 percent reduction in detection time. The oracle-based validation layer filters false positives and maintains a 96.8 percent true positive rate, showing that the approach is reliable even in large-scale blockchain audits.

The feedback-driven mutation strategy and dynamic policy updates help AI-FUZZ converge faster than heuristic fuzzers. It can handle contracts of different sizes, which shows it can scale for enterprise-level auditing. The reinforcement learning algorithms provide a solid theoretical foundation, matching the practical results with the design goals. There are some limitations. The evaluation focuses mainly on Ethereum contracts in Solidity, so performance on other blockchains or non-EVM platforms is unknown.

The reinforcement learning module needs significant computation and training time for large datasets, which may make it hard to run on low-resource systems. The oracle-based validation depends on predefined vulnerability signatures, so new or logic-based flaws may be missed without manual updates. AI-FUZZ does not yet include semantic understanding using large language models, which could improve contextual vulnerability detection in future versions.

CONCLUSIONS

AI-FUZZ is an adaptive reinforcement learning guided stateful fuzzer for smart contract vulnerability detection. It combines static and dynamic analysis to build and explore contract states. An RL agent steers mutation choices using feedback from execution traces. By preferring inputs that increase coverage and trigger confirmed issues, the system converges faster and finds more real vulnerabilities than traditional fuzzers. It also cuts redundant runs and keeps resource use low. The reward design supports continuous learning so the mutation policy improves, and deeper state transitions get tested. Future work includes adding cross chain analysis, using large language models to improve semantic understanding, and tuning energy allocation and prioritization for better scalability.

Statements

Funding: This work was supported and funded by the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University (IMSIU) (grant number IMSIU-DDRSP2604).

Conflict of interest: The authors declare that they have no conflict of interest related to this work.

Data availability: The data used in this study are derived from publicly available and open-source repositories to ensure transparency, reproducibility, and ethical compliance. Smart contract datasets were collected from the Etherscan Verified Contracts Repository and the SmartBugs Dataset, which provide real-world Ethereum smart contracts with verified source code and labeled vulnerability information. All contracts analyzed in this work are publicly accessible and were used strictly for academic and experimental purposes.

Author contribution: All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Rana Hassam Ahmed, Majid Hussain, Nadeem Sarwar, Leila Jamel, Hassan Abbas, Samraiz Zahid. The first draft of the manuscript was written by Rana Hassam Ahmed and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

REFERENCES

1. K. I. Gubbi, M. Tarighat, A. Sudarshan, I. Kaur, P. D. Kota, A. Sasan, and H. Homayoun, State of Hardware Fuzzing:
2. Current Methods and the Potential of Machine Learning and Large Language Models, in 2025 26th International Symposium on Quality Electronic Design (ISQED), pp. 1–7, 2025.
3. P. Qian, H. Wu, Z. Du, T. Vural, D. Rong, Z. Cao, L. Zhang, Y. Wang, J. Chen, and Q. He, Mufuzz: Sequence-aware

4. mutation and seed mask guidance for blockchain smart contract fuzzing, in 2024 IEEE 40th International Conference on Data Engineering (ICDE), pp. 1972–1985, 2024.
5. T. A. Shaikh, T. Rasool, K. Veningston, and S. M. Yaseen, The role of large language models in agriculture: harvesting
6. the future with LLM intelligence, *Progress in Artificial Intelligence*, vol. 14, no. 2, pp. 117–164, 2025.
7. C. Wu, J. Chen, Z. Wang, R. Liang, and R. Du, Semantic sleuth: Identifying ponzi contracts via large language
8. models, in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 582–593, 2024.
9. S.-I. Papadopoulos, C. Koutlis, S. Papadopoulos, and P. C. Petrantonakis, Verite: a robust benchmark for multimodal misinformation detection accounting for unimodal bias, *International Journal of Multimedia Information Retrieval*, vol. 13, no. 1, p. 4, 2024.
10. D. Wang, H. You, L. Zhu, K. Lin, Z. Chen, C. Yang, J. Yu, Z. Wang, and J. Chen, A Survey of Reinforcement Learning
11. for Software Engineering, arXiv preprint arXiv:2507.12483, 2025.
12. Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis, in *Proceedings of the IEEE/ACM 46th International*
13. *Conference on Software Engineering*, pp. 1–13, 2024.
14. S. Zerbini, S. Kharya, D. Boscovic, and E. Losiouk, Fuzzing for Smart Contract Interworking Security Evaluation: An Empirical Evaluation of the State of the Art, 2023.
15. C. Shou, J. Liu, D. Lu, and K. Sen, Llm4fuzz: Guided fuzzing of smart contracts with large language models, arXiv preprint arXiv:2401.11108, 2024.
16. R. Liang, J. Chen, C. Wu, K. He, Y. Wu, R. Cao, R. Du, Z. Zhao, and Y. Liu, Vulseye: Detect smart contract
17. vulnerabilities via stateful directed graybox fuzzing, *IEEE Transactions on Information Forensics and Security*, 2025.
18. O. Zaazaa and H. El Bakkali, SmartLLMSentry: A Comprehensive LLM Based Smart Contract Vulnerability Detection Framework, *Journal of Metaverse*, vol. 4, no. 2, pp. 126–137, 2024.
19. B. Wang, Y. Tong, S. Ji, H. Dong, X. Luo, and P. Zhang, A Review of Learning-based Smart Contract Vulnerability
20. Detection: A Perspective on Code Representation, *ACM Transactions on Software Engineering and Methodology*, 2025.
21. Y. Lin, F. Lin, G. Cai, H. Chen, L. Zou, Y. Liu, and P. Wu, Evolutionary reinforcement learning: A systematic review and future directions, *Mathematics*, vol. 13, no. 5, p. 833, 2025.
22. Z. Kong, C. Zhang, M. Xie, M. Hu, Y. Xue, Y. Liu, H. Wang, and Y. Liu, Smart Contract Fuzzing Towards Profitable Vulnerabilities, *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 153–175, 2025.
23. J. Park and J. Choi, Towards Effective Guidance of Smart Contract Fuzz Testing Based on Static Analysis, *Electronics*, vol. 14, no. 4, p. 806, 2025.
24. B. Li, Z. Pan, and T. Hu, EvoFuzzer: An Evolutionary Fuzzer for Detecting Reentrancy Vulnerability in SmartContracts, *IEEE Transactions on Network Science and Engineering*, 2024.
25. J. J. de Leon, C. Zhang, C.-S. Koulouris, F. Medda, and R. Rahul, Smart Contract Security in Decentralized Finance:
26. Enhancing Vulnerability Detection with Reinforcement Learning, *Applied Sciences*, vol. 15, no. 11, p. 5924, 2025.
27. Z. He, Z. Li, S. Yang, H. Ye, A. Qiao, X. Zhang, X. Luo, and T. Chen, Large language models for blockchain security:
28. A systematic literature review, arXiv preprint arXiv:2403.14280, 2024.
29. D. Liu, J. Zhang, Y. Wang, H. Shen, Z. Zhang, T. Ye, Blockchain Smart Contract Security: Threats and Mitigation
30. Strategies in a Lifecycle Perspective, *ACM Computing Surveys*, 2025.
31. Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, When gpt meets program analysis: Towardss
32. intelligent detection of smart contract logic vulnerabilities in gptscan, arXiv preprint arXiv:2308.03314, 2023.
33. R. H. Ahmed, M. Hussain, and A. Khalil, Blockchain-Based Supply Chain Management in Healthcare, in *AI and Blockchain Applications for Privacy and Security in Smart Medical Systems*, pp. 107–132, 2025.
34. R. H. Ahmed, M. Hussain, A. Khalil, and S. Zahid, Strengthening Security in Pharmaceutical Healthcare: Harnessing Blockchain for Reliable Detection of Counterfeit Drugs and Mitigating Dispensing Errors, in *2025 16th InternationalConference on Information and Communication Systems (ICICS)*, pp. 1–6, 2025.

35. C. Shou, J. Liu, D. Lu, and K. Sen, Llm4fuzz: Guided fuzzing of smart contracts with large language models, arXiv preprint arXiv:2401.11108, 2024.
36. B. Amangeldy, T. Imankulov, N. Tasmurzayev, G. Dikhanbayeva, and Y. Nurakhov, A Review of Artificial
37. Intelligence and Deep Learning Approaches for Resource Management in Smart Buildings, Buildings, vol. 15, no. 15, p. 2631, 2025.
38. R. H. Ahmed, J. Sultana, S. Zahid, M. A. Habib, A. Rauf, and M. Hussain, Integrating Large Language Models and AI into Blockchain: A Framework for Intelligent Smart Contracts and Fraud Detection, IEEE Access, pp. 1–1, 2025, doi: 10.1109/ACCESS.2025.3622511.