Large Language Models for Synthetic Dataset Generation: A Case Study on Ethereum Smart Contract DoS Vulnerabilities

Giacomo Ibba[1], Gavina Baralla[1], Giuseppe Destefanis[2]
[1]*University of Cagliari, Italy*
[2]*Brunel University London, UK*
{*giacomof.ibba,gavina.baralla*}*@unica.it*
giuseppe.destefanis@brunel.ac.uk

***Abstract*—The use of Ethereum smart contracts has significantly influenced sectors that depend on decentralized control and automated financial transactions. However, ensuring their security and reliable execution remains a complex task. Among the most serious challenges is the Denial of Service (DoS) attack, which can make a contract nonfunctional. The broad range of vulnerabilities that enable these attacks complicates prevention efforts. While dynamic security tools exist, they require substantial computational resources, and machine learning-based approaches face limitations due to a lack of training data. To address this issue, we propose a methodology using Large Language Models (LLMs), specifically Antropic's Claude and OpenAI's GPT-4, to generate synthetic examples of Ethereum smart contracts exposed to DoS attacks. Our results show that, with properly designed prompts, these models can produce high-quality synthetic examples, enabling the development of classification and anomaly detection models.**

***Keywords*—Large Language Models, Smart Contracts, Denial of service, Synthetic Data, OpenAI.**

I. Introduction

Ethereum smart contracts (SCs) offer a new approach for applications requiring decentralized execution and control. These contracts form the foundation of decentralized applications (DApps), driving innovation in IoT, user-generated content, and decentralized finance (DeFi) systems [2]. However, like traditional software, SCs are susceptible to vulnerabilities and software breaches [13], which can disrupt their expected behavior and execution. Security breaches in smart contracts can result in irreversible financial losses, as seen in the infamous DAO attack [12], or cause complete system unavailability.

Among the various vulnerabilities affecting smart contracts [8], Denial of Service (DoS) attacks are particularly severe, as they can permanently disable a contract. Detecting these vulnerabilities often requires significant computational effort, as they manifest during contract execution. Static analysis tools struggle to reliably identify these issues, necessitating the use of dynamic analysis tools or machine learning-based models such as classification and anomaly detection systems. However, dynamic tools demand extensive computational resources to analyze entire DApps, which can include thousands of contracts and dependencies. Meanwhile, machine learning-based approaches require large training datasets, which are currently scarce.

To address this gap, this paper introduces a methodology using Large Language Models (LLMs), specifically Antropic's Claude and OpenAI's GPT-4, to generate synthetic examples of smart contracts with DoS vulnerabilities. By employing a prompt engineering process, we generate realistic synthetic contracts and test cases to support the development of classification and anomaly detection models. Our results demonstrate that Claude outperforms GPT-4 in both the quality of generated contracts and the number of iterations required to produce functional examples.

The remainder of the paper is organized as follows: Section II reviews existing research on detecting DoS vulnerabilities in Ethereum smart contracts and generating synthetic datasets. Section III outlines the methodology for generating and testing synthetic contracts. Section IV presents the results of our experiments. Section V discusses potential threats to the validity of our findings. Finally, Section VI summarizes our contributions and highlights future directions.

II. Related Work

The Denial of Service (DoS) attack may lead a smart contract to a permanent inoperable state and for this very reason, it gathered the attention of the Ethereum community. Several tools have been proposed during the Ethereum lifecycle, such as SmartScan [15], a tool that statically scans smart contracts under test to identify potential exposed patterns and confirms the exploitability through dynamic analysis. Kumar et al. [10] propose a security framework for SC deployed in blockchain-IoT systems leveraging Random Forest and XGBoost and fog computing, giving full autonomy in decision making capabilities, and allowing to detect distributed DoS (DDoS) attacks. GASPER [4], is a framework based on symbolic execution on bytecode and Control Flow Graph allowing to detect gas-costly patterns in Ethereum smart contracts, which may lead to intentional or unintentional gas-exhaustion issues that may lead a smart contract to an inoperable state. Mythril [6] is

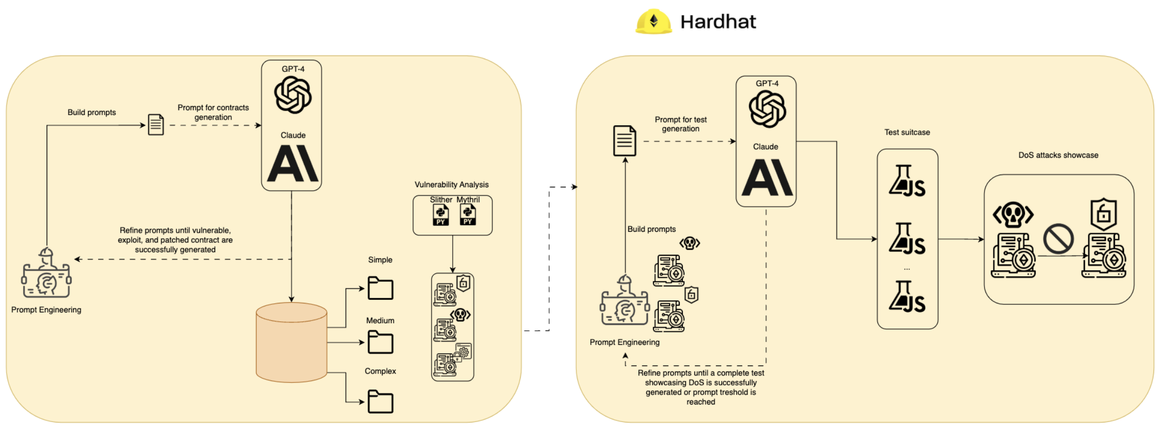
Hardhat
Build prompts
Prompt for contracts generation
GPT-4
Claude
Refine prompts until vulnerable, exploit, and patched contract are successfully generated
Prompt Engineering
Vulnerability Analysis
Slither
Mythril
Simple
Medium
Complex
Prompt for test generation
GPT-4
Claude
Test suitcase
DoS attacks showcase
Build prompts
Prompt Engineering
Refine prompts until a complete test showcasing DoS is successfully generated or prompt treshold is reached

Fig. 1: Design of the research methodology

a dynamic analysis based tool, that uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities. It represents a valuable tool to detect DoS, since it detects several related issues such as gas-destruction, unprotected self-destruct calls, reentrancies, and access control vulnerabilities. Although current literature provides a plethora of research exploring the quality, cost, and effectiveness of synthetic data generated by LLMs [11], [14], [3], [5], and works employing LLMs to detect SC vulnerabilities [1], [16], [9], to the best of our knowledge, this work would be the first to propose a methodology for the generation of a synthetic dataset tailored to encompass SC exposed to specific issues such as denial of service (DoS).

III. Methodology

Our methodology focuses on generating a synthetic dataset of SCs specifically designed to exhibit various DoS patterns, as presented in Figure 1. The primary goal was to ensure these SCs closely resemble real-world scenarios while avoiding artificial patterns with limited practical relevance. We selected Claude and GPT-4 due to their advanced natural language processing capabilities, which are essential for interpreting and structuring complex security scenarios within prompts.

Prompt Engineering. We began by designing prompts intended to generate SCs that demonstrated specific DoS vulnerabilities. To achieve this, we adopted a structured prompt-engineering strategy, ensuring that the outputs generated by Claude and GPT-4 adhered to predefined requirements. Initial experiments revealed that both models often produced contracts with limited diversity and overly simplistic representations of DoS vulnerabilities. To address this, we explicitly defined natural DoS entry points within the prompts to improve the variety and complexity of the vulnerabilities represented.

The subset of DoS attacks targeted in this study included block gas limit manipulation, external contract calls, misuse of revert statements, unexpected ETH transfers, and ownership manipulation. Prompts were tailored to introduce one or more of these attack types in the generated smart contracts, with varying levels of complexity in terms of functionality, code structure, and sophistication of DoS attacks. To further enhance the representativeness and heterogeneity of the data, we incorporated detailed descriptions of core functionalities, technical requirements, contextual information, and specific conditions for each use case into the prompt designs.

Contract Generation and Iteration. For each use case detailed in Table I, we designed a tailored prompt and provided it to both Claude and GPT-4. When the generated contract failed to meet the specified requirements or contained compilation errors, we refined the prompts iteratively. This process incorporated feedback from standard error messages generated by the Solidity compiler to address issues and ensure that the outputs conformed to the desired specifications.

Exploit and Patch Development. In addition to generating vulnerable contracts, we instructed the LLMs to produce exploit contracts designed to demonstrate the identified vulnerabilities, as well as patched versions of the original contracts. This process also relied on iterative prompt refinement to address any missing requirements or compilation issues, ensuring that all outputs were functional and met the outlined objectives.

Testing and Validation. Once the synthetic dataset was complete, encompassing vulnerable, exploit, and patched contracts, we conducted an initial analysis using open-source tools such as Slither and Mythril to identify security flaws. This analysis was a critical step in verifying that the generated vulnerable contracts reflected realistic vulnerability scenarios.

To further validate the dataset, we established a secondary testing environment in Hardhat[1], which provides a deterministic execution setup via its Network Helpers API. For this phase, we selected a subset of use cases from the synthetic dataset, covering simple, medium, and complex scenarios. We then instructed the LLMs to generate tests demonstrating the DoS vulnerabilities, including both the vulnerable and exploit contracts. This procedure was refined iteratively until the tests

[1]https://hardhat.org/

Use Case	Complexity	Compact Description
Membership System	Simple	Manages members via arrays, vulnerable to gas exhaustion as membership grows. Patch uses mappings and pagination for efficiency.
Auction System	Simple	Single-item auction with refund, bid history, and time limit. DoS risks from transfer, storage, and loop-based operations.
Token Vesting	Simple	Linear vesting with ERC20 tokens. Vulnerable to DoS via batch claims and token distribution. Patch simplifies operations.
Reward Distribution	Simple	Distributes ERC20 rewards based on points. Vulnerable to loop-based and storage-related DoS attacks.
Voting System	Simple	Binary voting for DAOs with real-time counting. DoS from vote tally loops and gas-heavy calculations.
Staking with Delegation	Medium	ERC20 staking with delegation. Vulnerable to cross-contract synchronization issues and cooldown abuse.
Multi-signature Wallet	Medium	Manages ETH/token transfers with owner confirmations. DoS risks from cross-contract synchronization and access control issues.
NFT Marketplace	Medium	Facilitates NFT trading with royalties. Vulnerable to token interaction, cross-contract state issues, and access control.
Lending Protocol	Medium	Multi-asset loans with collateral and liquidation. DoS risks from oracles, cross-contract dependencies, and network congestion.
Token Bridge	Medium	Cross-chain transfers with validator confirmation. DoS via validator desynchronization and timing mechanisms.
Asset Management Fund	Complex	Dynamically rebalances portfolios. Vulnerable to governance flooding, rebalancing delays, and asset withdrawal congestion.
Decentralized Insurance	Complex	Processes crypto-asset insurance claims. DoS risks from oracle delays, claim overloads, and liquidity lockups.
Prediction Market	Complex	Allows wagers on event outcomes. Vulnerable to escrow congestion, oracle issues, and dispute overload.
Supply Chain Tracking	Complex	Tracks products across organizations. DoS risks from verification congestion and encryption overhead.
Real Estate Marketplace	Complex	Facilitates property listings and mortgages. Vulnerable to escrow congestion, tokenization bottlenecks, and compliance delays.

TABLE I: Use Cases, Complexity, and Descriptions

accurately showcased all DoS vulnerabilities present in the selected contracts.

IV. Results

We observed differences in the generation processes of Claude and GPT-4. Both models were initially provided with a prompt emphasizing the research and educational purposes of the task and clarifying that the results would not be used for malicious purposes.

Behavior and Performance of GPT-4. GPT-4 showed inconsistent performance in generating synthetic smart contracts with DoS vulnerabilities, often requiring iterative refinements. It frequently resolved vulnerabilities instead of introducing them, necessitating repeated clarification. Defining tasks separately—vulnerable contracts, Proof of Vulnerability (PoV), and patched contracts—and addressing them sequentially improved results, though medium- and high-complexity contracts often remained incomplete or oversimplified.

To address these limitations, a dual-chat approach was used: one chat identified missing functionalities, while the other iteratively refined the contracts. Complex use cases required over 30 iterations, with half involving error corrections. Despite these challenges, GPT-4 produced high-quality outputs when guided by structured prompts.

ContractName	LLM	#Prompt	#Err	#Slither	#Mythril
Membership	Claude	1	0	0	0
Membership	GPT-4	4	1	1 (Medium)	0
Vesting	Claude	1	0	1 (High)	0
Vesting	GPT-4	3	2	1 (High)	7 (Low)
Staking	Claude	1	0	2 (1 High, 1 Medium)	2 (Low)
Staking	GPT-4	12	4	1 (Medium)	0
MultisigWallet	Claude	1	0	1 (Medium)	0
MultisigWallet	GPT-4	17	5	1 (High)	0
LendingProt	Claude	3	7	1 (High)	0
LendingProt	GPT-4	30	15	0	3 (Low)
Marketplace	Claude	3	4	2 (1 High, 1 Medium)	0
Marketplace	GPT-4	30	10	2 (1 High, 1 Medium)	0

TABLE II: LLM Smart Contracts Evaluation Summary. Contracts Highlighted in Green Represent Simple Use Cases, While Yellow and Red denote Medium and Complex Use Cases Respectively.

Behavior and Performance of Claude. Claude consistently outperformed GPT-4, requiring fewer prompts and producing fewer errors. It efficiently embedded specific vulnerabilities and handled simple and medium-complexity use cases within a single prompt. For complex cases, Claude maintained strong performance without needing repeated context reintroduction, unlike GPT-4.

A. *Vulnerability Detection Challenges*

Despite the deliberate introduction of vulnerabilities, tools like Slither[2] [7] and Mythril[3] often failed to detect them or flagged only minor issues. This underscores the limitations of current tools in identifying complex vulnerabilities and highlights the need for improved methods and integrated evaluations to address critical security flaws in blockchain applications.

Table II summarises the performance and vulnerability detection outcomes for Claude and GPT-4 across a selected sub-sample of simple, medium, and complex scenarios. To further validate the vulnerabilities, LLM-generated attack tests were conducted in a Hardhat environment. Simple cases were validated with ease, while medium-complexity cases required around 20 prompts per test, occasionally failing to produce functional attacks. Complex scenarios achieved satisfactory results within 7-8 prompts per iteration, confirming the accuracy of the generated synthetic contracts and their intended vulnerabilities.

V. Threats to Validity

When evaluating the quality and applicability of our approach and the generated data, several threats to validity must be considered.

[2]https://github.com/crytic/slither/
[3]https://github.com/Consensys/mythril

Model pattern generation limitations: The model, being trained on historical data, may recombine existing vulnerability patterns instead of creating novel ones. For instance, the DoS vulnerabilities generated for the lending protocol largely mirror known patterns such as state bloat and unbounded operations. This limitation may result in the omission of emerging vulnerability types, such as those involving MEV, cross-chain bridges, or new DeFi mechanisms that real-world attackers could exploit.

Controlled environment versus real-world deployment: The tests were conducted in a Hardhat simulation environment that differs significantly from mainnet conditions. Key limitations include instantaneous block timing versus real network latency, static gas prices instead of dynamic fluctuations, and memory/storage costs that do not perfectly align with mainnet values. Our isolated environment lacks critical real-world factors such as network congestion, front-running attacks, complex token economics, MEV considerations, mempool dynamics, and liquidity patterns, which may impact the validity of our vulnerability demonstrations.

Evolution of exploit vectors: Smart contract security is constantly advancing, with new attack methods emerging frequently, which may not yet be represented in our dataset. Furthermore, future vulnerabilities could arise from new Ethereum Improvement Proposals (EIPs), protocol updates, cross-chain interactions, or even advancements in quantum computing, potentially limiting the long-term relevance of the vulnerabilities generated.

VI. Conclusion and Future Work

This study contributes to Ethereum smart contract security by employing LLMs such as Antropic's Claude and OpenAI's GPT-4 to create a synthetic dataset of contracts with DoS vulnerabilities. Through prompt engineering, the models demonstrated their capability to simulate a range of vulnerability scenarios, marking an initial step toward improving the resilience of smart contracts against DoS attacks. Our findings highlight Claude's superior performance over GPT-4 in generating synthetic contracts and test cases, emphasizing the need for ongoing optimization of these technologies.

The dataset's ability to effectively model DoS vulnerabilities, which current tools like Slither and Mythril often fail to detect, underscores its fidelity in replicating subtle and intricate security flaws. Moreover, the study highlights the importance of continuously expanding synthetic datasets to address emerging vulnerabilities from new Ethereum Improvement Proposals (EIPs), protocol updates, and innovative DeFi mechanisms, ensuring preparedness against the evolving threat landscape in blockchain technology.

Future work will focus on expanding the dataset to include a wider range of DeFi protocols and smart contract architectures, and we will explore machine learning techniques, including traditional classifiers and deep learning models, for detecting DoS vulnerabilities through static and dynamic analysis. Anomaly detection methods will also be investigated to uncover novel DoS patterns from complex contract interactions, with an emphasis on identifying irregular gas consumption, abnormal state transitions, and unusual contract behaviors. These enhancements aim to develop more resilient smart contract architectures and strengthen security frameworks.

The synthetic data and the test suite employed to showcase the generated vulnerabilities are openly available in Zenodo [4]

References

[1] Biagio Boi, Christian Esposito, and Sokjoon Lee. Smart contract vulnerability detection: The role of large language model (llm). *ACM SIGAPP Applied Computing Review*, 24(2):19–29, 2024.

[2] Wei Cai, Zehua Wang, Jason B Ernst, Zhen Hong, Chen Feng, and Victor CM Leung. Decentralized applications: The blockchain-empowered software system. *IEEE access*, 6:53019–53033, 2018.

[3] Yung-Chieh Chan, George Pu, Apaar Shanker, Parth Suresh, Penn Jenks, John Heyer, and Sam Denton. Balancing cost and effectiveness of synthetic data generation strategies for llms. *arXiv preprint arXiv:2409.19759*, 2024.

[4] Ting Chen, Xiaoqi Li, Xiapu Luo, and Xiaosong Zhang. Under-optimized smart contracts devour your money. In *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*, pages 442–446. IEEE, 2017.

[5] Giuseppe Destefanis, Silvia Bartolucci, and Marco Ortu. A preliminary analysis on the code generation capabilities of gpt-3.5 and bard ai models for java functions. *arXiv preprint arXiv:2305.09402*, 2023.

[6] Thomas Durieux, João F Ferreira, Rui Abreu, and Pedro Cruz. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International conference on software engineering*, pages 530–541, 2020.

[7] Josselin Feist, Gustavo Grieco, and Alex Groce. Slither: a static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 8–15. IEEE, 2019.

[8] Giacomo Ibba and Marco Ortu. Analysis of the relationship between smart contracts' categories and vulnerabilities. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1212–1218. IEEE, 2022.

[9] Jaehyun Kim, Sangmyeong Lee, Howon Kim, et al. Robust vulnerability detection in solidity-based ethereum smart contracts using fine-tuned transformer encoder models. *IEEE Access*, 2024.

[10] Prabhat Kumar, Randhir Kumar, Govind P Gupta, and Rakesh Tripathi. A distributed framework for detecting ddos attacks in smart contract-based blockchain-iot systems by leveraging fog computing. *Transactions on Emerging Telecommunications Technologies*, 32(6):e4112, 2021.

[11] Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. On llms-driven synthetic data generation, curation, and evaluation: A survey. *arXiv preprint arXiv:2406.15126*, 2024.

[12] Muhammad Izhar Mehar, Charles Louis Shier, Alana Giambattista, Elgar Gong, Gabrielle Fletcher, Ryan Sanayhie, Henry M Kim, and Marek Laskowski. Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack. *Journal of Cases on Information Technology (JCIT)*, 21(1):19–32, 2019.

[13] Marco Ortu, Giacomo Ibba, Giuseppe Destefanis, Claudio Conversano, and Roberto Tonelli. Taxonomic insights into ethereum smart contracts by linking application categories to security vulnerabilities. *Scientific Reports*, 14(1):23433, 2024.

[14] Ajay Patel, Colin Raffel, and Chris Callison-Burch. Datadreamer: A tool for synthetic data generation and reproducible llm workflows. *arXiv preprint arXiv:2402.10379*, 2024.

[15] Noama Fatima Samreen and Manar H Alalfi. Smartscan: an approach to detect denial of service vulnerability in ethereum smart contracts. In *2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 17–26. IEEE, 2021.

[16] Junjie Zhao, Xiang Chen, Guang Yang, and Yiheng Shen. Automatic smart contract comment generation via large language models and in-context learning. *Information and Software Technology*, 168:107405, 2024.

[4]https://zenodo.org/records/14262663