# Evaluating Zero-shot Reasoning with Agentic LLMs for Smart Contract Vulnerability Detection

Eduardo Sardenberg Tavares
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Brazil
sardenberg@telemidia.puc-rio.br

Antonio José. G. Busson
BTG Pactual
São Paulo, Brazil
antonio.busson@btgpactual.com

Sérgio Colcher
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Brazil
colcher@inf.puc-rio.br

## ABSTRACT

Smart contracts are fundamental to blockchain ecosystems, but remain susceptible to security vulnerabilities that can lead to severe financial losses. Recent advances in agentic AI systems, powered by large language models (LLMs), enable autonomous code analysis and decision-making without explicit task-specific supervision. These systems leverage prompt engineering and zero-shot reasoning to detect vulnerabilities in smart contracts without prior fine-tuning. In this work, we evaluate the effectiveness of agentic LLM-based approaches in identifying vulnerabilities using prompt engineering and zero-shot reasoning across a curated dataset of Solidity smart contracts. Our findings highlight the limitations of current LLMs in automated vulnerability detection, providing insights into their practical applicability for securing decentralized applications. Our best-performing configuration, which integrates zero-shot reasoning with the Tree of Thoughts framework, achieved an F1-score of 73.66%.

## KEYWORDS

Smart contracts, Large Language Models, Vulnerability detection, Solidity, Prompt engineering, Zero-Shot

## 1 INTRODUCTION

Smart contracts are self-executing programs deployed on blockchain networks that autonomously enforce the terms of an agreement when predefined conditions are met [3]. They eliminate the need for intermediaries, offering increased efficiency, transparency, and trust in decentralized applications [16]. Vulnerabilities in smart contract code can lead to irreversible financial losses and systemic risks across decentralized ecosystems [1]. As such, ensuring the correctness and robustness of smart contracts through formal verification, rigorous testing, and code audits is essential for maintaining trust and resilience in blockchain-based systems.

Recent work has explored the potential of Large Language Models (LLMs) in assisting vulnerability detection. For instance, Chen et al. [4] examined how prompting strategies influence LLMs' performance in detecting common smart contract bugs, highlighting significant improvements with carefully engineered prompts. Similarly, Xiao et al. [18] conducted a comprehensive evaluation of LLMs across modern Solidity versions (e.g., v0.8), revealing that while some prompts can reduce false positive rates by more than 60%, the recall for particular vulnerabilities (such as reentrancy) drops drastically often due to the models' reliance on detecting outdated patterns or shared libraries.

In: I Workshop Brasileiro de Sistemas Web3 (BrWeb3 2025). Anais Estendidos do XXXI Simpósio Brasileiro de Sistemas Multimídia e Web (BrWeb3'2025). Rio de Janeiro/RJ, Brasil. Porto Alegre: Brazilian Computer Society, 2025.

ISSN 2596-1683

Although previous work has focused primarily on prompt engineering techniques, especially those based on knowledge engineering and few-shot prompting, important gaps remain regarding the potential of zero-shot reasoning strategies for detecting vulnerabilities in Solidity smart contracts. In this paper, we investigate different zero-shot prompting [9, 15] techniques, particularly reasoning-based approaches, such as chain-of-thought (CoT) [2, 7] and tree-of-thought (ToT) [8]; our goal is to assess the ability of LLMs to carry out effective analysis without relying on specific examples provided in the prompt, reflecting realistic scenarios of automated usage.

Accordingly, we propose the following research questions.

- **RQ1**: How do different zero-shot prompting strategies (e.g., zero-shot, chain-of-thought, tree-of-thought) affect accuracy and false-positive rates in vulnerability detection for Solidity smart contracts?
- **RQ2**: What are the main limitations of LLMs in identifying vulnerabilities in contracts written in Solidity v0.8, under zero-shot scenarios?

The remainder of this paper is structured as follows. Section 2 presents the related work. Next, in Section 3, we describe our proposed method. In Section 4, we present our empirical findings. Finally, in Section 5, we present our conclusions and future work.

## 2 RELATED WORK

Vulnerability detection in smart contracts has attracted growing attention due to the increasing adoption of blockchain applications and the severe financial impact of exploitable bugs. Several surveys have provided comprehensive overviews of techniques and challenges in this domain. Qian et al. [11] categorize existing detection methods into static, dynamic, and formal approaches, emphasizing the trade-off between precision and scalability across these tools. Similarly, Sürücü et al. [12] focus on the role of machine learning, discussing supervised and unsupervised models and their dependence on high-quality labeled datasets, which are often scarce in the smart contract ecosystem.

With the emergence of large language models (LLMs), recent work has explored their potential in detecting vulnerabilities without the need for domain-specific training. Chen et al. [4] investigate the use of ChatGPT for vulnerability detection, demonstrating that prompting techniques can significantly affect model performance.

However, their study is limited to a single LLM and relies primarily on manually curated prompts, without evaluating generalization across model versions and reasoning techniques.

Xiao et al. [18] extend this line of research by evaluating multiple state-of-the-art LLMs in Solidity smart contracts, including those written in modern versions. Their findings highlight critical limitations: models show a high false-positive rate, and their performance varies substantially across vulnerability types. Notably, the study demonstrates that simple prompt adjustments can reduce false positives but also reveals a sharp decline in recall for particular vulnerabilities, such as arithmetic and reentrancy issues.

## 3 METHOD

To detect vulnerabilities in smart contracts, we evaluated several large language models (LLMs), including GPT-4o, GPT-4.1 (and their mini and nano variants), as well as the O1, O3-mini, and O4-mini models. To ensure a robust evaluation, we utilized curated datasets comprising both bug-free and buggy contracts with annotated vulnerability locations, allowing for a comprehensive analysis of each model's detection capabilities. To investigate the impact of reasoning techniques on LLM performance, we applied three distinct approaches: Zero-Shot, Zero-Shot Chain-of-Thought (CoT), and Zero-Shot Tree-of-Thought (ToT). These techniques were applied consistently across all models and datasets, enabling a controlled comparison of how each approach affects the effectiveness of vulnerability detection. The source code and dataset used in our evaluation are publicly available in our Git repository[1].

Section 3.1 presents the zero-shot reasoning techniques used in our experiments. Then, Section 3.2 introduces the dataset used in our evaluation.

### 3.1 Reasoning

We used these three strategies to understand the impact of explicit reasoning on detection quality.

- **Zero-Shot:** In this baseline setup, the model is asked a direct question without any intermediate reasoning steps or examples. For instance, the model receives a prompt such as: *"Is the following smart contract vulnerable? Please answer yes or no."* This approach assesses the model's ability to perform binary classification based on raw understanding, without guidance.
- **Zero-Shot Chain-of-Thought (CoT):** This strategy encourages the model to reason step by step before producing a final answer. The prompt is extended with instructions like: *"Explain your reasoning step by step, then conclude whether the contract is vulnerable."* By eliciting intermediate reasoning, CoT helps the model better evaluate logic and control flow in smart contracts, often reducing spurious classifications.
- **Zero-Shot Tree-of-Thought (ToT):** ToT prompts encourage the model to consider multiple reasoning paths in parallel before concluding. In this format, the model explores different possible scenarios (e.g., whether a contract has particular vulnerabilities) and then selects the most plausible path. This can be manually emulated by prompting the model to "list multiple possible vulnerabilities and evaluate their likelihood."

[1]https://github.com/eduardosardenberg/Smart-Contracts-Vulnerability-Analysis

While Zero-Shot provides a baseline, CoT enables us to test whether intermediate reasoning enhances precision, and ToT examines whether structured exploration can reduce false positives or aid in uncovering subtle bugs. These prompting methods are inspired by recent studies on the planning abilities of LLM agents [5], which show that structured reasoning pathways can improve task reliability and decision-making, particularly in complex environments.

### 3.2 Dataset Construction

We constructed a balanced dataset containing 200 buggy and 200 non-buggy contracts. This dataset was built by combining samples from five public sources: JiuZhou [17], DAppSCAN [6], UniswapV2 [13], UniswapV3 [14], and OpenZeppelin [10]. These repositories provide real-world Solidity contracts, some of which are annotated with known vulnerabilities and corresponding fixed versions. Table 1 presents the detailed distribution of the dataset among projects and categories.

To compose the non-buggy set, we selected all available contracts from UniswapV2 (12 files), UniswapV3 (62 files), and JiuZhou (102 files), and added 24 additional clean contracts from OpenZeppelin to reach a total of 200. For the buggy set, we used all 66 buggy contracts from JiuZhou and selected 134 vulnerable contracts from DAppSCAN to complete the remaining examples.

We then created a unified CSV file to facilitate evaluation. Each row contains: the dataset source (`dataset_name`), the full path to the Solidity file (`file_path`), a semantic category label (`category`), the file name (`file_name`), and a binary label indicating whether the contract contains a vulnerability (`has_error`). This structured format enables automatic input generation for the LLMs, which are prompted to analyze each contract and assess its vulnerability status.

**Table 1: Dataset composition and vulnerability distribution**

| Dataset | Total Files | Error Files | Error Rate (%) |
|---|---|---|---|
| DAppSCAN | 134 | 134 | 100.0 |
| JiuZhou | 168 | 66 | 39.3 |
| UniswapV2 | 12 | 0 | 0.0 |
| UniswapV3 | 62 | 0 | 0.0 |
| openZeppelin | 24 | 0 | 0.0 |

## 4 RESULTS

In this section, we present the results of our empirical evaluation across different model and reasoning technique combinations. For each configuration, we report standard classification metrics, including precision, recall, and F1-score. All models were evaluated under consistent temperature and context window settings to ensure comparability. In Section 4.1, we present a comparative analysis of the different LLM models. Then, in Section 4.2, we discuss the results.

**Table 2: Performance of LLMs on vulnerability detection (sorted by F1-score)**

| LLM Name | Reasoning | Precision | Recall | F1-Score |
|---|---|---|---|---|
| gpt-4o | zero-shot-tot | 60.98 | 93.00 | 73.66 |
| gpt-4o | zero-shot-cot | 59.68 | 92.50 | 72.55 |
| gpt-4.1-mini | zero-shot-tot | 62.96 | 85.00 | 72.34 |
| gpt-4.1 | zero-shot-tot | 57.14 | 96.00 | 71.64 |
| gpt-4.1 | zero-shot | 61.82 | 85.00 | 71.58 |
| gpt-4.1 | zero-shot-cot | 57.49 | 94.00 | 71.35 |
| gpt-4.1-mini | zero-shot-cot | 61.07 | 85.50 | 71.25 |
| o4-mini | zero-shot-tot | 74.54 | 67.00 | 70.57 |
| gpt-4o-mini | zero-shot-tot | 59.26 | 88.00 | 70.82 |
| gpt-4o-mini | zero-shot-cot | 58.55 | 89.00 | 70.63 |
| gpt-4.1-mini | zero-shot | 61.39 | 79.50 | 69.28 |
| o4-mini | zero-shot-cot | 76.83 | 63.00 | 69.23 |
| gpt-4o | zero-shot | 60.35 | 68.50 | 64.17 |
| o1 | zero-shot-tot | 71.54 | 60.00 | 65.26 |
| o1 | zero-shot-cot | 69.34 | 56.00 | 61.96 |
| gpt-4.1-nano | zero-shot-tot | 56.13 | 59.77 | 57.77 |
| gpt-4o-mini | zero-shot | 58.78 | 82.00 | 58.48 |
| gpt-4.1-nano | zero-shot-cot | 54.13 | 59.30 | 56.59 |
| o1 | zero-shot | 60.54 | 51.00 | 55.36 |
| o4-mini | zero-shot | 61.54 | 48.00 | 53.93 |
| o3-mini | zero-shot-tot | 80.43 | 39.00 | 52.53 |
| o3-mini | zero-shot-cot | 81.40 | 35.00 | 48.95 |
| o3-mini | zero-shot | 90.91 | 30.00 | 45.11 |
| gpt-4.1-nano | zero-shot | 50.65 | 39.00 | 44.07 |

## 4.1 Model Comparison

Table 2 presents the performance results sorted by F1-score. The GPT-4o model combined with the zero-shot-tot technique achieved the best overall performance, with an F1-score of 73.66, closely followed by other variants of the same model using zero-shot-cot (72.55) and GPT-4.1-mini with zero-shot-tot (72.34). These results indicate that the Tree-of-Thought (ToT) technique tends to provide consistent performance gains, primarily when used with more advanced models, such as GPT-4.

The GPT-4.1 model also demonstrated strong results across all evaluated techniques, with highlights including zero-shot-ToT (71.64) and simple zero-shot (71.58), showing stable performance regardless of the adopted reasoning method. Interestingly, even more compact models, such as gpt-4.1-mini, achieved competitive F1-scores (e.g., 71.25 with zero-shot-CoT), demonstrating a good balance between performance and computational cost.

In contrast, smaller models, such as GPT-4.1-Nano and o3-Mini, delivered significantly lower performance, with F1-scores below 60 in all configurations, revealing substantial limitations in handling more complex tasks, like vulnerability detection, without fine-tuning. Notably, the gpt-4.1-nano model achieved the lowest result, with an F1-score of only 45.11 under the zero-shot configuration.

Beyond model selection, it is also evident that the reasoning technique has a significant impact on performance. The zero-shot-tot strategy appears more frequently among the top-performing results, suggesting that breaking down the task into multiple steps via Tree-of-Thought contributes to more effective reasoning, even without supervised training. The Chain-of-Thought (zero-shot-cot) approach showed slightly lower performance than ToT but still outperformed direct reasoning (zero-shot), reinforcing the benefits of structured reasoning approaches.

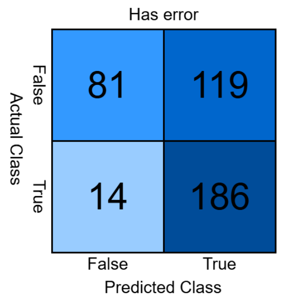

**Figure 1: Confusion matrix of our best config (GPT-4o – ToT).**

Figure 1 presents the confusion matrix obtained for our best configuration (GPT-4o + ToT). The model correctly classified 81 contracts with errors (true positives) and 186 contracts without errors (true negatives). There were 14 false negatives, where the model failed to detect existing errors, and 119 false positives, where it incorrectly classified error-free contracts as defective.

## 4.2 Discussion

Figure 2 shows the same results grouped by LLM, making it easier to compare the relative effectiveness of the reasoning strategies within each model. A clear pattern emerges: in most cases, the Tree-of-Thought (ToT) technique ranks first, followed by the Chain-of-Thought (CoT) technique, with direct zero-shot reasoning generally yielding the lowest performance. This suggests that structured reasoning approaches are consistently more effective across different LLM architectures, especially in complex tasks such as vulnerability detection.

Although LLMs using zero-shot approaches demonstrate remarkable capabilities for error detection without specific training, the example above highlights their limitations. Our best configuration correctly classified the majority of cases; however, the significant number of false positives (119 error-free contracts classified as defective) could lead to excessive alerts and wasted review efforts. Moreover, the 14 false negatives underscore a critical risk: contracts with actual vulnerabilities could go undetected, making it into production and potentially causing financial losses. These limitations underscore the need for more specialized approaches, complemented by fine-tuning, to enhance the reliability and efficiency of LLMs in critical scenarios, such as smart contract auditing.

The following false positive example was generated from the analysis of the file [2]. The output shown corresponds to the LLM's reasoning about this specific contract, which was incorrectly classified as erroneous despite being free of actual vulnerabilities.

[2]Datasets/openzeppelin-contracts-master/ contracts/mocks/InitializableMock.sol

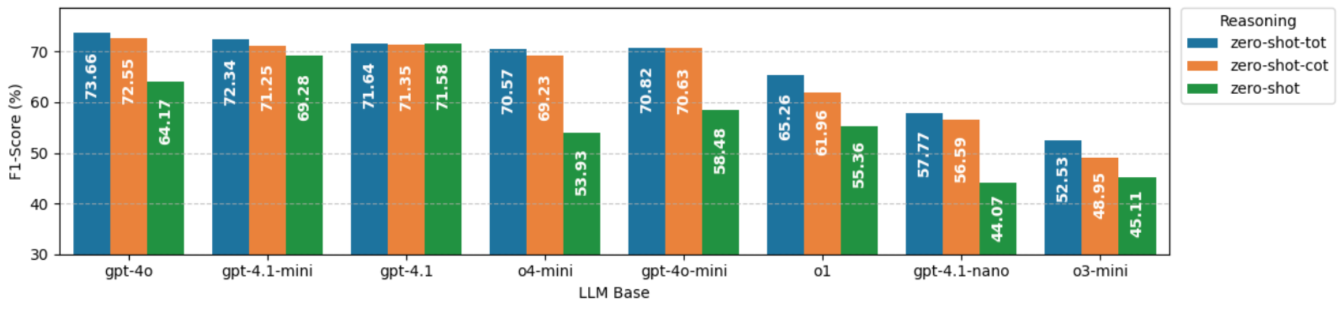

**Figure 2: F1-Score Performance of LLMs with Zero-Shot Reasoning Methods.**

```
{
    "has_error": true,
    "reasoning_paths": [
        {
            "path": "Initialization logic and use of Initializable",
            "analysis": "The contracts use the Initializable pattern to control
            ↪  initialization. The InitializableMock contract correctly uses
            ↪  the initializer and onlyInitializing modifiers. However, the
            ↪  ConstructorInitializableMock contract uses the initializer
            ↪  modifier in the constructor, which is not typical and can lead to
            ↪  issues if the contract is deployed as a proxy. The
            ↪  ChildConstructorInitializableMock contract inherits this pattern,
            ↪  which could propagate the issue. The DisableBad2 contract also
            ↪  uses the initializer modifier in the constructor and then calls
            ↪  _disableInitializers(), which is contradictory and can lead to
            ↪  unexpected behavior."
        },
        {
            "path": "Reinitialization logic",
            "analysis": "The ReinitializerMock contract uses the reinitializer
            ↪  modifier to allow multiple initializations with different
            ↪  versions. This is a valid use case, but the chainReinitialize
            ↪  function does not use the reinitializer modifier, which could
            ↪  lead to unexpected behavior if called multiple times. This
            ↪  function should be carefully managed to ensure it does not
            ↪  violate the intended initialization logic."
        },
        {
            "path": "Disable initializers logic",
           "analysis": "The DisableNew and DisableOld contracts correctly disable
            ↪  initializers in their constructors. However, the DisableBad1
            ↪  contract inherits from both DisableNew and DisableOld, which
            ↪  could lead to conflicts in initialization logic. The DisableOk
            ↪  contract inherits from DisableOld and DisableNew in a different
            ↪  order, which might resolve the conflict, but this depends on the
            ↪  specific implementation of the Initializable contract."
        }
    ],
    "final_reasoning": "The code contains several issues related to the use of
    ↪  the Initializable pattern. The use of the initializer modifier in
    ↪  constructors, as seen in ConstructorInitializableMock and DisableBad2,
    ↪  is problematic because it can lead to issues when deploying contracts as
    ↪  proxies. Additionally, the DisableBad1 contract may have conflicts due
    ↪  to multiple inheritance from contracts that disable initializers. These
    ↪  issues suggest that the code has errors related to initialization logic
    ↪  and the use of the Initializable pattern."
}
```

## 5 CONCLUSION

To address **RQ1**, our results confirm that the prompting strategy has a substantial impact on the performance of LLMs in detecting vulnerabilities in Solidity smart contracts. Using a balanced benchmark of 400 real-world contracts, we found that structured reasoning strategies - particularly zero-shot Chain-of-Thought (CoT) and Tree-of-Thought (ToT) - consistently outperform plain zero-shot reasoning across models. The best performing configuration was `gpt-4o` with zero-shot ToT, achieving an F1-Score of 73.66.

Regarding **RQ2**, the evaluation highlights significant limitations in smaller models. Despite the use of advanced prompting strategies, models such as `gpt-4.1-nano` failed to surpass an F1-score of 57.77, with plain zero-shot prompting yielding even worse results (down to 44.07). These findings underscore that lightweight LLMs struggle to generalize in complex reasoning tasks like vulnerability detection under zero-shot conditions. Such models may require further fine-tuning or supervised training to be viable in real-world security applications.

Regarding limitations, our evaluation is constrained by the scope of the benchmark and the task definition. First, although the dataset comprises real-world contracts, it covers a finite set of vulnerability types and may not reflect the full diversity of security issues encountered in practice. Second, the prompting strategies were tested under zero-shot conditions, which favor models with strong pretraining but may underestimate the capabilities of models that benefit more from fine-tuning. Third, our analysis is limited to binary classification (vulnerable vs. non-vulnerable), which, while helpful in benchmarking, does not capture the depth of the model's understanding, such as its ability to identify the type of vulnerability correctly or to localize and justify its reasoning. These limitations affect both the interpretability of the results and the generalizability of the findings to real-world auditing contexts.

As future work, we plan to extend the evaluation in two directions. First, we intend to incorporate other large language models, such as Google's Gemini, to assess whether architectural differences influence vulnerability detection. Second, and more importantly, we aim to move beyond binary metrics by conducting a more fine-grained analysis. This includes evaluating whether models can correctly identify the type of vulnerability, accurately localize it in the code, and provide coherent reasoning for their classification. Such qualitative insights are essential for understanding the true potential and limitations of LLMs in smart contract auditing, particularly in practical scenarios where actionable explanations matter as much as correct predictions.

## REFERENCES

[1] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust*. Springer, 164–186.
[2] Paulo Victor Borges, Adeoye Sunday Ladele, Yan MBG Cunha, Daniel de S Moraes, Polyana B da Costa, Pedro TC dos Santos, Rafael Rocha, Antonio JG Busson, Julio Cesar Duarte, and Sérgio Colcher. [n. d.]. Multimodal Prompt Engineering for Multimedia Applications using the GPT Model. ([n. d.]).
[3] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014), 2–1.

[4] Chong Chen, Jianzhong Su, Jiachi Chen, Yanlin Wang, Tingting Bi, Jianxing Yu, Yanli Wang, Xingwei Lin, Ting Chen, and Zibin Zheng. 2025. When chatgpt meets smart contract vulnerability detection: How far are we? *ACM Transactions on Software Engineering and Methodology* 34, 4 (2025), 1–30.

[5] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716* (2024).

[6] InPlusLab. 2023. DAppSCAN: Vulnerability Dataset for Smart Contracts. https://github.com/InPlusLab/DAppSCAN.

[7] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[8] Jieyi Long. 2023. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291* (2023).

[9] Daniel Moraes, Polyana Costa, Pedro Santos, Ivan Pinto, Sérgio Colcher, Antonio Busson, Matheus Pinto, Rafael Rocha, Rennan Gaio, Gabriela Tourinho, Marcos Rabaioli, and David Favaro. 2024. Tagging Enriched Bank Transactions Using LLM-Generated Topic Taxonomies. In *Proceedings of the 30th Brazilian Symposium on Multimedia and the Web* (Juiz de Fora/MG). SBC, Porto Alegre, RS, Brasil, 267–274. https://doi.org/10.5753/webmedia.2024.243267

[10] OpenZeppelin. 2023. OpenZeppelin Contracts: A library for secure smart contract development. https://github.com/OpenZeppelin/openzeppelin-contracts.

[11] Peng Qian, Zhenguang Liu, Qinming He, Butian Huang, Duanzheng Tian, and Xun Wang. 2022. Smart contract vulnerability detection technique: A survey. *arXiv preprint arXiv:2209.05872* (2022).

[12] Onur Sürücü, Uygar Yeprem, Connor Wilkinson, Waleed Hilal, S Andrew Gadsden, John Yawney, Naseem Alsadi, and Alessandro Giuliano. 2022. A survey on ethereum smart contract vulnerability detection using machine learning. *Disruptive Technologies in Information Sciences VI* 12117 (2022), 110–121.

[13] Uniswap. 2020. Uniswap V2: Non-Bug Contracts Dataset. https://github.com/Uniswap/v2-core/tree/master/contracts.

[14] Uniswap. 2021. Uniswap V3: Non-Bug Contracts Dataset. https://github.com/Uniswap/v3-core/tree/main/contracts.

[15] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. [n. d.]. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations*.

[16] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[17] Xf97. 2020. JiuZhou: A Dataset for Smart Contract Bug Localization. https://github.com/xf97/JiuZhou.

[18] ZeKe Xiao, Qin Wang, Hammond Pearce, and Shiping Chen. 2025. Logic meets magic: Llms cracking smart contract vulnerabilities. *arXiv preprint arXiv:2501.07058* (2025).