Comparative Analysis of Smart Contract Generation Using Large Language Models

Hiago Vinícius Benedito dos Santos[1], **Raissa Rosa dos Santos Januario**[1], **Ravelly Carvalho Zanatta**[1], **Saulo Neves Matos**[1,2], **Jó Ueyama**[1]

[1]Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo (USP) – São Carlos, SP – Brasil

[2]School of Computing and Communications, Lancaster University,
Lancaster, United Kingdom

{hiagovini, raissa.rosa, ravellyzanatta, saulo.matos}@usp.br

joueyama@icmc.usp.br

Abstract. *In recent years, blockchain technology has established itself as an effective, secure, and transparent data storage solution. In this context, smart contracts play a fundamental role by enabling the automated execution of agreements without intermediaries. With the advancement of language models, the opportunity to automatically generate these contracts has emerged, raising concerns about their reliability and potential vulnerabilities. This article proposes a comparative analysis of the available language models for developing smart contracts using Ethereum Virtual Machine's contracts as a case study. Experiments were made using various Large language models using different metrics to evaluate the susceptibility to vulnerabilities and computational cost. After comparing various models, ChatGPT appears to be the most suitable for generating smart contracts due to its higher compilation rate and, consequently, a larger sample size, despite detecting more vulnerabilities.*

1. Introduction

In recent years, advancements in Natural Language Processing (NLP) driven by Large Language Models (LLMs) have revolutionized human-computer interaction. These models enable the generation and understanding of complex texts with unprecedented accuracy and fluency, opening new possibilities across various domains [Jayakody and Dias 2024, Lewis et al. 2024]. One particularly promising application is the automated generation of smart contracts—self-executing programs running on blockchain platforms that facilitate secure and transparent transactions based on predefined conditions [Szabo 1997, Zhao 2023].

Smart contracts, introduced by Szabo 1997, are decentralized, immutable, and self-executing, ensuring security and transparency without the need for intermediaries [Sujeetha and Deiva Preetha 2021]. Since the advent of Ethereum [Buterin 2014], their utility has extended to diverse fields, including the Internet of Things (IoT) and decentralized autonomous organizations (DAOs) [Muneeb et al. 2022]. Despite their advantages, ensuring the security and efficiency of smart contracts is paramount, as vulnerabilities can lead to severe financial losses [Yang et al. 2022, Wang et al. 2024].

In the Ethereum Virtual Machine (EVM), gas serves as the unit of measurement for computational effort, determining the cost of executing operations and transactions. Each operation within the EVM has a predefined gas cost documented in the Ethereum protocol, functioning as a fee system for resource usage. Consequently, gas consumption directly impacts the execution costs of smart contracts, necessitating optimization to avoid excessive fees [Li 2021, Farokhnia and Goharshady 2023].

While previous studies have explored the use of LLMs for generating smart contracts, they have not systematically evaluated different types of LLMs or analyzed their limitations in terms of gas consumption and security. This gap motivates our investigation into whether LLMs can generate secure and efficient smart contracts. Specifically, we assess their susceptibility to vulnerabilities and their deployment gas cost, aiming to identify the most suitable models for this task. From this, the following research questions (RQs) emerge:

- **RQ1:** Are there vulnerabilities present in smart contracts generated by language models?
- **RQ2:** Which language model generates smart contracts with the lowest deployment gas cost?
- **RQ3:** Which language model is most effective and reliable for generating secure and efficient smart contracts?

To address these questions, this work conducts a comparative analysis of widely used LLMs, contributing to the state of the art in smart contract generation. Our findings aim to assist developers in selecting the most efficient and secure models for generating smart contracts. A sample of 30 contracts from Jusbrasil[1], a Brazilian platform that provides legal information, was used to generate the smart contracts.

2. Background

This section presents the fundamental concepts for the proposed work. Section 2.1 discusses Blockchain and smart contracts, while Section 2.2 addresses LLMs and Prompt Engineering, highlighting techniques in NLP and strategies for optimized interaction with language models.

2.1. Blockchain and Smart Contracts

Blockchain is a decentralized Peer-to-Peer (P2P) network that utilizes a ledger structured in an immutable and shared chain of blocks, ensuring the reliable recording of transactions and the control of assets. This technology provides real-time information in a completely transparent manner [Nakamoto 2008]. One of its greatest advantages is decentralization, where transactions are verified by multiple nodes in the network, using cryptographic techniques (hash) to ensure the authenticity and security of the stored data [Bashir 2020]. Each block contains transaction data, a timestamp, and the hash of the previous block.

[1]https://www.jusbrasil.com.br/

To validate transactions, Blockchain uses consensus mechanisms such as Proof-of-Work, Proof-of-Stake, and Proof-of-Authority, ensuring agreement among nodes about the validity of transactions before adding them to the chain [Islam et al. 2023].

With the emergence of Ethereum in 2014, Blockchain was expanded to include smart contracts. Smart contracts are computational protocols that automate the execution of contractual agreements, eliminating intermediaries and ensuring transparency and security [Buterin 2014]. Implemented in languages such as Solidity[2], the contracts contain encoded clauses that perform operations on data stored in the Blockchain, ensuring the precise and immutable execution of the agreed terms [Zheng et al. 2020].

The execution of smart contracts occurs in the Ethereum Virtual Machine (EVM), an autonomous environment that isolates contract operations from the main blockchain. For each execution, a resource called gas is charged, which measures the computational effort required and regulates resource consumption during execution. If the gas runs out, state changes are reverted, ensuring the integrity of the system [Ma et al. 2019, Wu et al. 2024].

EVM is widely adopted by other blockchains such as Binance Smart Chain and Avalanche, has established itself as the leading platform for smart contracts. Among the languages[3] compatible with the EVM, Solidity leads due to its comprehensive documentation and extensive community support, which motivated its choice in this work.

To measure the deployment gas cost of the smart contracts developed in Solidity, we used Remix, an IDE widely recognized for optimizing contract analysis by allowing cost calculations for deployment and function execution.

2.2. Large Language Models and Prompt Engineering

LLMs have emerged as artificial intelligence systems capable of processing and generating coherent text, generalizing for various NLP tasks [Naveed et al. 2024]. Based on Transformers architectures [Vaswani et al. 2017], they utilize deep neural networks trained on large datasets, allowing for the identification of complex patterns and dependencies in word sequences, which enables a precise understanding of linguistic and semantic nuances [Jayakody and Dias 2024, Lewis et al. 2024].

In addition to traditional capabilities such as text generation, translation, and summarization, large language models (LLMs) demonstrate emerging skills such as reasoning, contextual learning, and decision-making, even without explicit training for such tasks [Wei et al. 2022a, Boiko et al. 2023]. They also perform zero-shot tasks, responding to new scenarios without additional training [Naveed et al. 2024].

The success of using LLMs depends on the quality of the prompts, which directly influence the relevance and coherence of the responses. Prompt Engineering emerges as an essential discipline aimed at optimizing interaction with the models by designing inputs that guide the responses toward specific tasks [Chen et al. 2024]. Strategies such as Chain of Thought, zero-shot, and few-shot enhance the models' capabilities, allowing for more precise and adaptable responses in various scenarios [Wei et al. 2022b, Reynolds and McDonell 2021].

[2] 2https://docs.soliditylang.org/en/v0.8.25/

[3] https://ethereum.org/en/developers/docs/smart-contracts/languages/

The combination of LLMs and Prompt Engineering represents a significant advancement in using artificial intelligence to solve complex NLP problems. It offers scalable, high-performance solutions. In the context of Prompt Engineering, the framework CO-STAR is designed to structure and guide the creation of effective prompts. It has gained traction in various studies, with promising results observed in various applications [Peng et al. 2024, Fadi et al. 2024, Napoli et al. 2024, Shen 2024].

2.3. Smart Contract Vulnerability Detection Tools

For a robust analysis of vulnerabilities in smart contracts, complementary tools are used to enhance coverage and depth, as discussed in Wei et al. 2023. The combination of tools with different techniques allows for the detection of a broader range of vulnerabilities, leveraging the specific advantages of each method.

Based on this principle, exists several tools such as Mythril[4] and Slither[5]. Mythril uses symbolic execution and flow analysis to detect known vulnerabilities and simulate complex contract states. Conversely, Slither applies static analysis using pattern matching, providing rapid detection of logical errors and ease of integration.

Since smart contracts are self-executing and immutable, ensuring they are free of vulnerabilities is essential. Once deployed, their code cannot be altered, making any flaws permanent and exploitable. While immutability is key to blockchain's trustless design, it also highlights the importance of preventing vulnerabilities during development.

3. Related Works

In this section, we explore the development of smart contracts, focusing on identifying current gaps and the methods already implemented in this process. For instance, Jurgelaitis et al. 2022 presents a methodology that follows the principles of Model Driven Architecture (MDA), using Unified Modeling Language (UML) models to enhance the understanding and reusability of smart contracts, as well as to generate code in Solidity. The process involves modeling contracts as Platform Independent Models (PIM), which are transformed into Platform Specific Models (PSM) through algorithms in the Eclipse ATL and Acceleo tools. The approach was evaluated by comparing code metrics, similarity, and execution costs between the original code and the generated code, achieving a similarity rate greater than 90%.

The study conducted by Shynkarenko and Kopp 2022 explores the use of NLP techniques for converting business rules into smart contracts. The methodology combines intelligent text processing with software development techniques in Python, using libraries such as Spacy for experimental implementation. As a result, the work demonstrates the automated generation of a smart contract for the creation of a token based on a text containing the business rules.

Fan et al. 2023 introduces an innovative approach to converting contracts written in natural language into smart contracts using the Contract Text Markup Language (CTML). The methodology employs semantic markup to structure contractual elements in hierarchical levels – factors, properties, and legal components – which facilitates the

[4]https://mythril-classic.readthedocs.io/en/master/index.html
[5]https://crytic.github.io/slither/slither.html

extraction and organization of relevant information. A Metadata Data Model (EMD) is also established, allowing for the control of access permissions and customization, and through this, the methodology enables the conversion of contracts into automated formats.

The research made by Ahmed et al. 2024 explores the generation of smart contracts aimed at rental agreements. The study begins with the extraction of contractual models, facing the challenge of the scarcity of well-labeled data, which necessitates manual collection and labeling. The information extraction process considers the dynamics of payments and collects key attributes of the agreement, such as type and values, using a customized and trained Named Entity Recognition (NER) model. Finally, the user prepares a markup with customized tags, ensuring compliance with the legal context for the generation of the smart contract.

The article developed by Napoli et al. 2024 proposes a pipeline that utilizes LLMs to automate the creation of smart contracts, facilitating access for individuals without programming experience. The pipeline adopts the CO-STAR methodology [Science and Division 2023] to optimize prompt generation and Slither [Feist et al. 2019] to identify vulnerabilities in the generated contracts. The results indicate that 98.1% of the contracts are compilable. However, the evaluation is conducted exclusively using ChatGPT, focusing on metrics such as compilability, vulnerabilities, and the presence of comments.

The study conducted by Zhao et al. 2024 uses ChatGPT3.5 Turbo as the representative LLM for generating comments on smart contracts. The Smart Contract Comment generation via Large Language Models approach (SCCLLM) combines contextual learning and the selection of relevant code demonstrations from a historical corpus, enhancing the quality of the generated comments. Tests with data from the blocks explorer Etherscan.io show that SCCLLM outperforms previous approaches, highlighting the effectiveness of ChatGPT for creating comments in an existing smart contract.

Fadi et al. 2024 also investigated the use of GPT-4 for the generation of smart contracts based on textual descriptions, again utilizing the CO-STAR method to structure the prompts effectively. The evaluation was conducted through automated analyses and manual reviews, identifying limitations of the model, such as code failures and inconsistencies between the prompts and the generated contracts, highlighting that GPT-4 is still not suitable for producing ready-to-use contracts.

Over the years, the use of LLMs for generating smart contracts has intensified, especially in more recent studies. Previous research, such as that of Jurgelaitis et al. 2022 and Shynkarenko and Kopp 2022, focused on methodologies like Model Driven Architecture and the use of Natural Language Processing techniques in combination with traditional software development. However, more contemporary approaches, like those of Napoli et al. 2024, Zhao et al. 2024, and Fadi et al. 2024, highlight the adoption of LLMs as central tools, reflecting a natural evolution in the field towards automation through generative artificial intelligence.

Different from previous works [Napoli et al. 2024, Zhao et al. 2024, Fadi et al. 2024], this work stands out by expanding the analysis beyond ChatGPT, also incorporating LLMs such as Gemini, LLAMA, and Gemma, which allows for a broader evaluation of smart contract generation. Moreover, it uses multiple tools for vulnerability analysis, a feature that was not explored by previous studies. This aims to obtain a more accurate average of the quality of contracts generated by each LLM. Furthermore, the costs in terms of gas consumption on the Ethereum Virtual Machine are analyzed, providing a new perspective on the efficiency and cost-effectiveness of the generated contracts. Table 1 summarizes related works, highlighting the objectives and methodologies employed. This approach provides a solid foundation for the selection of the most suitable tools for developers, thus contributing to the advancement of practices in smart contract development.

Table 1. Related Works Focusing on LLMs to generate smart contracts.

Work	Evaluation of LLMs	Evaluation of Multiple Vulnerability Tools	CO-STAR	Deployment Gas Cost Evaluation
Jurgelaitis et al. 2022	○	○	○	○
Shynkarenko and Kopp 2022	○	○	○	○
Fan et al. 2023	○	○	○	○
Ahmed et al. 2024	○	○	○	○
Zhao et al. 2024	●	○	○	○
Fadi et al. 2024	●	○	●	○
Napoli et al. 2024	●	○	●	○
Our work	●	●	●	●

4. Methodology

The analysis focuses on three central aspects: the risk of generating vulnerable contracts, the cost of the generated contracts, and the ability of the models to produce contracts with a greater number of correct functions.

4.1. Language Models for Comparison

The selection of LLMs was based on criteria of performance, accessibility, and recognition in the field of natural language processing. ChatGPT[6], from OpenAI, was chosen for its pioneering role and widespread adoption, standing out for its versatility and ability to understand various linguistic tasks.

In addition, Llama3 was chosen, recognized for technical advancements and optimization for intensive workloads, along with Gemma, which integrates with the Groq[7], a platform that facilitates the use of APIs for practical implementation of models.

Finally, Gemini[8] was included for its broad applicability, reliability, and ease of access via API. This selection reflects a balance between efficiency, ease of implementation, and robustness, aligning with the goals of effective and scalable results.

4.2. Prompt Formulation

The chosen approach for formulating the prompt was the CO-STAR framework, which stood out as the champion in the inaugural prompt engineering competition with GPT-4

[6]https://chatgpt.com
[7]https://groq.com/
[8]https://gemini.google.com

organized by the Government Technology Agency of Singapore (GovTech) [Science and Division 2023], thereby distinguishing itself from others due to its ability to provide more comprehensive information to the language model.

The choice of CO-STAR is based on its effectiveness in establishing a robust structural context, which fosters the production of more precise and informative responses. This advantage makes CO-STAR a preferred framework for projects that require clarity, adaptability, and depth in prompt construction.

Table 2. Prompt Formulation - CO-STAR Framework

Element	Description
Context	Imagine a company that uses textual contracts but wants to migrate to a transparent and automated solution using smart contracts.
Objective	Develop a smart contract based on the provided textual contract.
Style	Clean code, gas-efficient execution, and security against vulnerabilities.
Tone	Not applicable
Audience	Companies that execute commercial agreements through contracts and wish to automate this process.
Response	The generated smart contract should include only the Solidity code, without any introductions or additional explanations.

The Table 2 presents the prompt used based on the CO-STAR model, detailing the key elements for formulating a structured request for the generation of smart contracts. Each element of the framework has been adapted to the context of a company that wishes to automate its contractual processes, ensuring that the generated contract is efficient, secure, and tailored to the specific needs of the target audience.

In this case, the Tone element will not be used, as the goal is to generate a smart contract directly, focusing only on the code. This eliminates the need for a specific tone, as the expected result is technical and objective, without the presence of introductory or explanatory texts.

4.3. Evaluation Metrics for LLM's Comparison

To measure susceptibility to vulnerabilities in generated contracts, we have formulated metrics that calculate, for each contract, the average number of vulnerabilities and the average percentage of vulnerabilities at each impact level (high, medium, and low) based on the total number of analyzed contracts. These metrics provide a detailed and comparative assessment of contracts generated by different LLMs.

The average number of vulnerabilities detected by tool X in the contracts generated by LLM y is given by:

$$M = \frac{\sum_{i=1}^{n} \text{quantity}_i}{n}, \tag{1}$$

where: M is the overall average of detected vulnerabilities, quantity_i is the number of vulnerabilities in the i-th contract, and n is the overall count of analyzed contracts.

In addition, the average percentage of impact vulnerabilities x (high, medium, low) is calculated as follows:

$$\bar{P}_x = \frac{\sum_{i=1}^{n} \text{percentage}_{x,i}}{n}, \quad \text{where,} \quad \text{percentage}_{x,i} = \frac{\text{quantity}_{x,i} \times 100}{\text{totalQuantity}_i}, \tag{2}$$

in which: $\bar{P}_x$ is the average percentage of vulnerabilities of impact x, $\text{quantity}_{x,i}$ is the number of impact vulnerabilities x in the i-th contract, and totalQuantity_i is the total vulnerabilities detected in the i-th contract.

Figure 1 illustrates the flow executed for the comparative analysis, which begins at Step 1 with the selection of a sample of real contracts, including lease agreements, purchase and sale of real estate, among others, written in Portuguese. These contracts can be found on Jusbrasil[9].

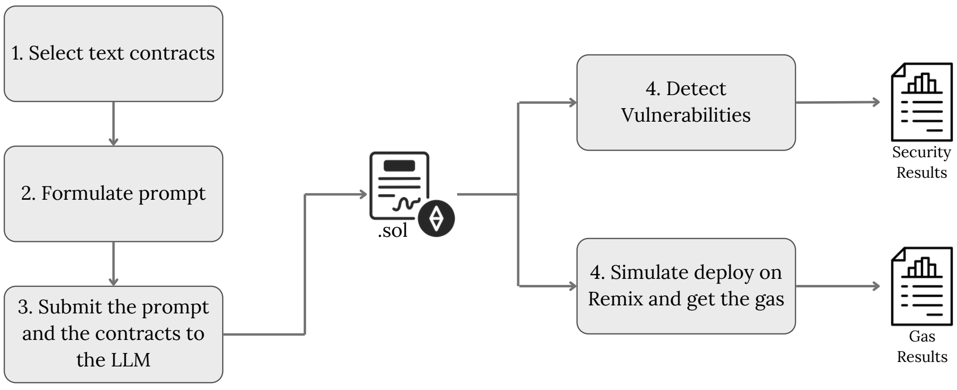
1. Select text contracts
2. Formulate prompt
3. Submit the prompt and the contracts to the LLM
.sol
4. Detect Vulnerabilities
4. Simulate deploy on Remix and get the gas
Security Results
Gas Results

Figure 1. Flow chart of the adopted methodology.

In Stage 2, the prompt follows a predefined framework to ensure consistency and clarity in the input provided to the LLM. In the next stage, both the prompt and the contract text are submitted to the LLM, which then generates the corresponding smart contract. Subsequently, the generated contract is analyzed by the selected vulnerability detection tools, and mathematical formulas are applied to calculate the security score of the contract. At the same time, a deployment simulation is performed using a web IDE, which provides a detailed estimate of the gas cost.

The Table 3 provides a summarized view of the tools applied in the analysis, highlighting each category and its related tools. The following sections will provide a detailed analysis and justification for selecting each of these tools.

5. Results

Figure 2 shows the percentage of compiled contracts by LLM and tool. The Gemini model achieved a compilation rate of 34% in Slither and 30% in Mythril, while the LLama3 model exhibited rates of 10% in Slither and 4% in Mythril. The Gemma model had no compiled contracts, consequently, it does not appear in the figure.

[9]https://www.jusbrasil.com.br/

Table 3. Summary of the evaluation tools utilized in this work.

Category	**Related Tools**
LLM	ChatGPT, LLama, Gemini, Gemma
Vulnerability Detection Tools	Slither, Mythril
Blockchain	Remix IDE, Solidity, EVM
Prompt Formulation	CO-STAR

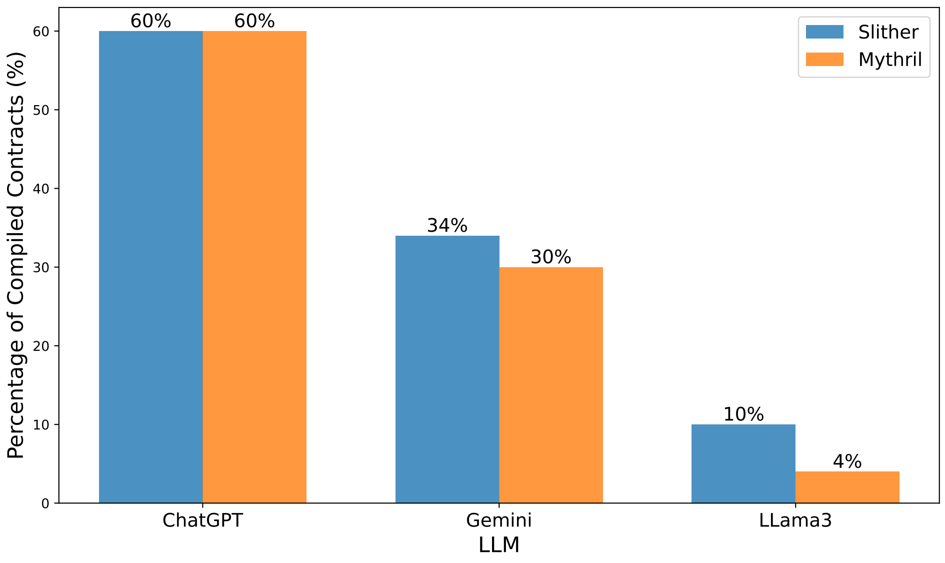

Figure 2. Percentage of compiled contracts by LLM and tool.

The Table 4 displays the average number of vulnerabilities per contract (M) for each LLM. The Gemini model had 2.5 vulnerabilities in Slither and 0.22 in Mythril. The LLama3 had one vulnerability in Slither and 0 in Mythril. The Gemma model showed no vulnerabilities, as no contracts were compiled.

Table 4. Average vulnerabilities per contract (M) for the analyzed LLMs.

LLM	**Slither**	**Mythril**
ChatGPT	4.16	0
Gemini	2.5	0.22
LLama3	1	0

Tables 5 and 6 present the impact of vulnerabilities by level (high, medium, low), and The Gemma model had no impact due to a compilation failure. In Slither (Table 5), the Gemini model had an average impact of 18.5% and a low impact of 71.5%. The LLama3 had a low impact of 33.3%.

Without Mythril (Table 6), the Gemini model showed 11.1% high impact, 11.1% medium impact, and 22.2% low impact. The LLama3 did not show a significant impact.

Table 5. Percentage of impact by vulnerability level (Slither) for each LLM.

LLM	High (%)	Medium (%)	Low (%)
ChatGPT	4.47	11.48	84.04
Gemini	0	18.5	71.5
LLama3	0	0	33.3

Table 6. Percentage of impact by vulnerability level (Mythril) for each LLM.

LLM	High (%)	Medium (%)	Low (%)
ChatGPT	0	0	0
Gemini	11.1	11.1	22.2
LLama3	0	0	0

Figure 3 presents the average deployment gas cost, the equivalent value in Ether, and the corresponding value in USD for the compiled contracts generated by each LLM. The gas price is denominated in Gwei, and its conversion to Ether follows by $\text{ETH} = \frac{\text{Gas Units} \times \text{Gas Price}}{1{,}000{,}000{,}000}$.

Contracts generated by ChatGPT consumed an average of 1,652,282.3 gas units. For Gemini, the consumption was 1,340,313.7 gas units, while for Llama, it was 1,324,725 gas units. Gemma did not show cost since none of the contracts generated by it were compiled. Table 7 presents the average gas consumption of each LLM in gas units, Ether, and U.S. dollars. All calculations related to the currency conversion were made on January 16, 2025. On this date, the price of gas was 2.228 Gwei, and the value of one Ether was US$3,373.80 USD.

Table 7. Average gas consumption values in gas units, Ether, and U.S. dollars.

LLM	Gas Units	ETH	US$
ChatGPT	1,652,282.3	0.003681284915	US$12.42
Gemini	1,340,313.7	0.002986218924	US$10.07
LLama3	1,324,725	0.0029514873	US$9.96

6. Discussion

A response can be given to the Research Questions on the basis of results shown in Section 5.

RQ1: Are there vulnerabilities present in smart contracts that are generated by language models?

Yes, the generated contracts exhibit vulnerabilities. The ChatGPT model achieved the highest average number of vulnerabilities, with a value of 4.16 in Slither and 0 in Mythril. The Gemini model, in turn, showed an average of 2.5 vulnerabilities in Slither and 0.22 in Mythril. In contrast, the LLama3 model recorded an average of 1 vulnerability in Slither and 0 in Mythril, while the Gemma model did not generate compilable contracts, resulting

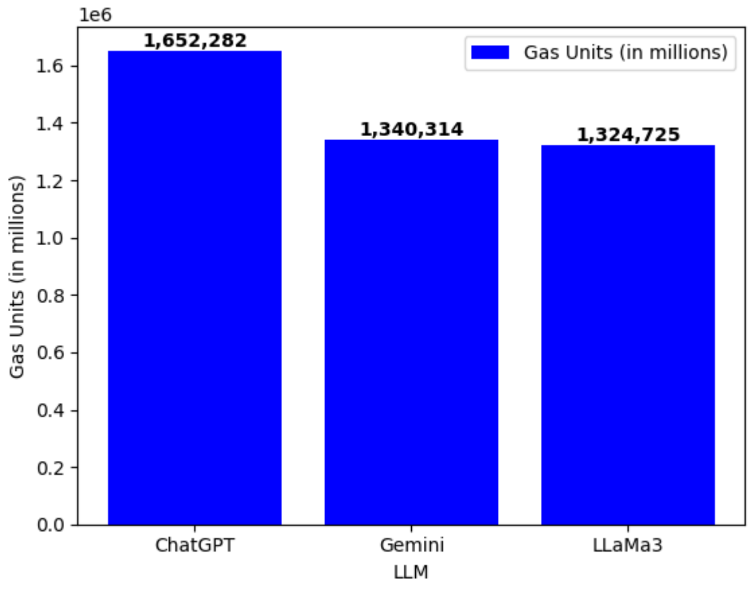

Figure 3. Average gas consumption of each LLM.

in a 100% failure rate. However, the high failure rate observed in most models limits the comparative analysis. As a result, it is not possible to accurately conclude that ChatGPT is the most prone to generate vulnerable contracts. The discrepancy in the number of samples considered between ChatGPT and the other models makes the analysis biased.

RQ2: Which language model generates smart contracts with the lowest deployment gas cost?

Among the models that presented compiled contracts, excluding Gemma, the LLama3 model demonstrated, on average, the lowest deployment gas cost. This was followed by Gemini, which recorded a Gas cost 15,589 units higher. ChatGPT, in turn, showed the highest average deployment gas cost. However, it is important to note that the number of samples considered for the first two models (LLama3 and Gemini) was significantly low, which may introduce bias into the results. In contrast, ChatGPT had a larger number of samples, allowing for greater reliability in the analysis of its performance.

RQ3: Which language model is most effective and reliable for generating secure and efficient smart contracts?

ChatGPT is the most suitable for generating smart contracts. The central metric chosen for this evaluation was the rate of compilable contracts, as it represents a more robust and consistent estimate of each LLM's practical ability to meet the primary objective of generating functional smart contracts. The results indicated that ChatGPT achieved significantly superior performance, with 60% of the contracts generated being compilable. In contrast, Gemini presented only 34% compilable contracts, LLama3 achieved 10%, and Gemma did not produce any compilable contracts.

Although vulnerability analysis and deployment gas cost are relevant for assessing the quality and security of the generated contracts, these metrics were calculated based on limited samples for models with lower compilation rates. This factor compromises the reliability of these analyses. In the case of ChatGPT, however, the higher rate of compilable contracts allowed for a broader sample base, granting greater credibility to the metrics obtained.

7. Conclusion

This study examined the generation of smart contracts by LLMs, assessing the compilation failure rate, average vulnerabilities, and deployment gas cost. The findings highlighted that while vulnerabilities persist, ChatGPT proved the most effective model for generating smart contracts, due to its higher compilation rate and, consequently, a larger sample despite having more detected vulnerabilities.

However, no LLM can yet independently generate a smart contract from a real-world scenario suitable for production, as even the best-performing model failed to compile all requested contracts.

Future work could expand the sample size, explore diverse contract types, and incorporate additional LLMs to provide a broader perspective on their potential. Additionally, investigating the causes of compilation failures could help address model limitations and support the creation of more robust and secure smart contracts.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and Fundação de Amparo à Pesquisa do Estado de São Paulo - Brasil (FAPESP) - Process Number 2024/07478-8.

References

Ahmed, S. U., Danish, A., Ahmad, N., and Ahmad, T. (2024). Smart contract generation through NLP and blockchain for legal documents. *Procedia Computer Science*, 235:2529–2537.

Bashir, I. (2020). *Mastering Blockchain: A Deep Dive Into Distributed Ledgers, Consensus Protocols, Smart Contracts, DApps, Cryptocurrencies, Ethereum, and More*. Expert insight. Packt Publishing.

Boiko, D. A., MacKnight, R., and Gomes, G. (2023). Emergent autonomous scientific research capabilities of large language models.

Buterin, V. (2014). Ethereum white paper: A next-generation smart contract and decentralized application platform. Accessed: 2024-11-02.

Chen, B., Zhang, Z., Langrené, N., and Zhu, S. (2024). Unleashing the potential of prompt engineering in large language models: a comprehensive review.

Fadi, B., Napoli, E. A., Gatteschi, V., Schifanella, C., et al. (2024). Automatic smart contract generation through llms: When the stochastic parrot fails. In *Proceedings of DLT 2024*. CEUR.

Fan, Y., Chen, E., Zhu, Y., He, X., Yau, S. S., and Pandya, K. (2023). Automatic generation of smart contracts from real-world contracts in natural language. In *2023 IEEE Smart World Congress (SWC)*, pages 1–8. DOI: 10.1109/SWC57546.2023.10448870.

Farokhnia, S. and Goharshady, A. K. (2023). Reducing the gas usage of ethereum smart contracts without a sidechain. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–3.

Feist, J., Grieco, G., and Groce, A. (2019). 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.

Islam, S., Islam, M. J., Hossain, M., Noor, S., Kwak, K.-S., and Islam, S. M. R. (2023). A survey on consensus algorithms in blockchain-based applications: Architecture, taxonomy, and operational issues. *IEEE Access*, 11:39066–39082.

Jayakody, R. and Dias, G. (2024). Performance of recent large language models for a low-resourced language. In *2024 International Conference on Asian Language Processing (IALP)*, pages 162–167. DOI: 10.1109/IALP63756.2024.10661169.

Jurgelaitis, M., čeponienė, L., and Butkienė, R. (2022). Solidity code generation from UML state machines in model-driven smart contract development. *IEEE Access*, 10:33465–33481.

Lewis, D.-M., DeRenzi, B., Misomali, A., Nyirenda, T., Phiri, E., Chifisi, L., Makwenda, C., and Lesh, N. (2024). Human review for post-training improvement of low-resource language performance in large language models. In *2024 IEEE 12th International Conference on Healthcare Informatics (ICHI)*, pages 592–597.

Li, C. (2021). Gas estimation and optimization for smart contracts on ethereum. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1082–1086.

Ma, F., Fu, Y., Ren, M., Wang, M., Jiang, Y., Zhang, K., Li, H., and Shi, X. (2019). EVM: From offline detection to online reinforcement for ethereum virtual machine. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 554–558. IEEE.

Muneeb, M., Raza, Z., Haq, I. U., and Shafiq, O. (2022). Smartcon: A blockchain-based framework for smart contracts and transaction management. *IEEE Access*, 10:23687–23699. DOI: 10.1109/ACCESS.2021.3135562.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*. Available at: https://bitcoin.org/bitcoin.pdf.

Napoli, E. A., Barbàra, F., Gatteschi, V., and Schifanella, C. (2024). Leveraging large language models for automatic smart contract generation. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 701–710.

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., and Mian, A. (2024). A comprehensive overview of large language models.

Peng, J., Han, Z., Zhang, H., Ye, J., Liu, C., Liu, B., Guo, M., Chen, H., Lin, Z., and Tang, Y. (2024). A multilingual text detoxification method based on few-shot learning and co-star framework. *Working Notes of CLEF*.

Reynolds, L. and McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pages 1–7.

Science, G. D. and Division, S. G. A. (2023). Prompt engineering playbook. Online. Available at: https://www.developer.tech.gov.sg/products/collections/data-science-and-artificial-intelligence/playbooks/prompt-engineering-playbook-beta-v3.pdf.

Shen, Y. (2024). Enhancing english language education with chatgpt. In *Proceedings of the 2024 International Conference on Artificial Intelligence and Communication (ICAIC 2024)*, pages 512–521. Atlantis Press.

Shynkarenko, D. and Kopp, A. (2022). Towards the approach to building smart contracts based on business rules using natural language processing. *Grail of Science*, (22):144–150.

Sujeetha, R. and Deiva Preetha, C. A. S. (2021). A literature survey on smart contract testing and analysis for smart contract based blockchain application development. In *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, pages 378–385. DOI: 10.1109/ICOSEC51865.2021.9591750.

Szabo, N. (1997). The idea of smart contracts. Accessed: 2024-10-22.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Wang, D., Shan, M., and Tong, N. (2024). Smart contract vulnerability detection based on machine learning. In *2024 6th International Conference on Electronic Engineering and Informatics (EEI)*, pages 1038–1042. DOI: 10.1109/EEI63073.2024.10696331.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. (2022a). Emergent abilities of large language models.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022b). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Wei, Z., Sun, J., Zhang, Z., Zhang, X., Li, M., and Zhu, L. (2023). A comparative evaluation of automated analysis tools for solidity smart contracts.

Wu, G., Wang, H., Lai, X., Wang, M., He, D., and Chan, S. (2024). A comprehensive survey of smart contract security: State of the art and research directions. *Journal of Network and Computer Applications*, 226:103882.

Yang, H., Zhang, J., Gu, X., and Cui, Z. (2022). Smart contract vulnerability detection based on abstract syntax tree. In *2022 8th International Symposium on System Security, Safety, and Reliability (ISSSR)*, pages 169–170. DOI: 10.1109/ISSSR56778.2022.00032.

Zhao, B. (2023). Reverse real-time model detection of chain smart contract security based on association method. In *2023 International Conference on Telecommunications, Electronics and Informatics (ICTEI)*, pages 759–762.

Zhao, J., Chen, X., Yang, G., and Shen, Y. (2024). Automatic smart contract comment generation via large language models and in-context learning. *Information and Software Technology*, 168:107405.

Zheng, Z., Xie, S., Dai, H.-N., Chen, W., Chen, X., Weng, J., and Imran, M. (2020). An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491.