Automatic Smart Contract Generation Through LLMs: When The Stochastic Parrot Fails

Automatic Smart Contract Generation Through LLMs: When The Stochastic Parrot Fails*

Fadi Barbàra[1,*], Emanuele A. Napoli[2], Valentina Gatteschi[2] and Claudio Schifanella[1]

[1]Department of Computer Science, University of Turin, Turin, Italy Via Pessinetto 12, 10149, Turin, Italy
[2]Department of Computer and Control Engineering, Politecnico di Torino, Turin, Italy, Corso Duca degli Abruzzi 24, 10129, Turin, Italy

Abstract

Blockchain and Artificial Intelligence (AI) have revolutionized the technology landscape, evolving predominantly along parallel trajectories. This research investigates the feasibility of integrating these two domains, specifically examining the potential of the latest advancements in Large Language Models (LLMs) to assist non-experts in the development of production-ready Solidity smart contracts. Utilizing a lease agreement as a case study, we employ GPT-4 to translate the document into smart contract code. To ensure consistency, we design several distinct prompts with analogous objectives, each employed multiple times.

Our evaluation methodology encompasses both automated analysis and expert manual review. The findings indicate a clear limitation: the current iteration of GPT-4 is *incapable* of generating production-ready smart contracts, primarily due to undetected coding flaws and discrepancies between the prompts and the generated code. This study underscores the challenges and limitations inherent in leveraging LLMs for the autonomous generation of complex, real-world applicable smart contracts.

Keywords

Blockchain, Smart Contract, AI, LLM, GPT-4, propt engineering,

1. Introduction

In recent years, remarkable advancements in artificial intelligence (AI) have paralleled the evolution of blockchain technology, with Large Language Models (LLMs) emerging as a significant development. These models, including LLaMa 1 & 2 by Meta [1, 2], PaLM by Google/Alphabet [3], or Mistral by MistralAI [4], have revolutionized problem-solving across diverse domains. Through processing and generating human-like text, they have fundamentally altered our approach to everyday challenges, demonstrating an advanced capacity to understand context, produce coherent responses, and adapt to various tasks. Their versatility has rendered them indispensable across industries such as healthcare, finance, and software engineering [5, 6, 7], signaling a new era of AI-driven innovation and problem-solving.

Another disruptive technology that has taken hold in recent years is blockchain. In summary, blockchain represents a decentralized database, shared by all peers in the network, in which so-called transactions are kept track of. Each group of transactions, or "block", is linked to the previous one cryptographically to form a blockchain. Blockchain technology eliminates the need for intermediaries while still guaranteeing the validity and integrity of the data, ensuring that it cannot be modified. The fact that the database is distributed ensures transparency among participants and reduces the risk of fraud or tampering. This type of technology lends itself well to areas such as supply chain or notarization of documents where it is essential to maintain a copy of data that is chronologically ordered and cannot be modified.

The power of blockchain extends to the concept of smart contracts (SC), which are self-executing programs with terms written directly into code. Smart contracts serve as digital counterparts to legal

6th Distributed Ledger Technology Workshop, May 24-25, 2024, Turin, Italy
*Corresponding author.
fadi.barbara@unito.it (F. Barbàra); emanuele.napoli@polito.it (E. A. Napoli); valentina.gatteschi@polito.it (V. Gatteschi); claudio.schifanella@unito.it (C. Schifanella)
https://fadibarbara.it (F. Barbàra); insert (E. A. Napoli); https://staff.polito.it/valentina.gatteschi (V. Gatteschi); insert (C. Schifanella)

agreements, leveraging properties of non-repudiability and trustless execution. By embodying these characteristics, smart contracts offer a secure and transparent mechanism for automating and enforcing contractual obligations across various domains. However, their development traditionally requires a deep understanding of blockchain technology and development expertise, posing challenges for non-technical individuals.

Over the years, interest in blockchain has grown because of the above-mentioned properties of transparency, reliability, non-tampering and programmability, attracting the attention not only of people with programming skills but also of people with no special knowledge in this area. In this sense, LLMs lend themselves well to narrowing the knowledge gap of non-experts who aim to harness the potential of blockchain. In this paper, as a preliminary step for developing a comprehensive tool for automatic smart contract generation, we try to establish which is the best prompt to feed to LLMs to produce the best smart contract. In order to do so, we developed a pipeline that automatically generates s smart contracts starting from a legal contract or textual description. To properly assess the effectiveness of the pipeline, we introduce a benchmarking suite tailored to our specific objectives. This suite incorporates multifaceted metrics, such as compilability, vulnerability detection, and the presence of comments, among others. To the best of our knowledge, this is the first work proposing a framework for prompt engineering applied to the generation of smart contracts. We also present an expert evaluation of the produced smart contract. By contrasting these two evaluation methods we see how major flaws in the code produced by the LLM go undetected by automatic vulnerability detection tools. In the interest of promoting transparency and facilitating reproducibility, we have made all pertinent data publicly available.

Specifically, the corpus of 68 smart contracts under examination is accessible via our dedicated GitHub repository[1]. This repository also contains the scripts employed in our analysis pipeline and the procedures for automatic testing. Additionally, the outcomes of our automatic testing process are meticulously documented in a spreadsheet[2].

Finally, the lease agreement we used as case study can be found in Appendix A while the prompts used for the LLM are in Appendix B.

In summary, the following are the contributions of our current work:

1. we propose a set of metrics to evaluate the quality of the generated smart contracts in order to evaluate the effectiveness of proposed tool;
2. we perform a double smart contract review (automatic and manual) to evaluate different prompts, adhering to the CO-STAR [8](Context, Objective, Style, Tone, Audience, Response) methodology in order to determine the most effective approach to producing high-quality smart contracts;
3. we assess the consistency, stability and flaws of the LLM's response across multiple calls using the same prompt;
4. we freely release a usable dataset of LLM-produced code that can be used for other studies.

The paper is so composed. In Section 2 we present the related works. We explain how our testing has been done in Section 3 and we analyze the results in Section 4. We perform a discussion of the results in Section 5. Finally in Section 6 we present conclusions and future works.

2. Related Works

The advent of Large Language Models creates the need to get the best response from them by asking the right question. This art is called prompt engineering and includes all those practices that aim to advise how to communicate clearly with the LLM and get the best quality response in terms of precision, context, and data. Prompt engineering has already been used in a wide range of context such as education [9, 10], academic writing [11], healthcare [12, 13], automotive [14], and insolvency

[1]GitHub link: https://github.com/BChain4all/pipeline/tree/main/results-feb-24

[2]Spreadsheet file containing the outcomes of the proposed research: https://docs.google.com/spreadsheets/d/1zPk6Ucb8n2UAaExGa1F18ODV_s2CYQV6/edit?usp=sharing&ouid=101965872120892214476&rtpof=true&sd=true

and bankruptcy law [15]. In the domain of Natural Language Processing, a spectrum of strategies has been employed [16]. Nevertheless, any papers investigated the best strategies and framework in terms of prompt engineering so that the LLM response is the most accurate possible. Wang et al. [10] investigated the best strategies used by undergraduates students in prompting ChatGPT to effective complete tasks by evaluating the information quality obtained by the LLM. Ratnayake et al. [16] propose the PERFECT prompting framework, which entails specifying the Purpose (the primary goal or objective of the prompt), Element (an additional element to consider), Role (the role or perspective the model should adopt), Format (the desired response format), Examples (including examples or case studies), Conditions (conditions or constraints the answer should adhere to), and Timeframe (historical context or future predictions). To address the complexities inherent in constructing a prompt another method adopted is the so-called CO-STAR [8]. This method involves providing COntext, defining the Style for consistency, articulating the Target for focus, defining the Tone for sentiment alignment, identifying the Audience for personalization, and specifying the Response for format clarity. Through this structured approach, the aim is to improve the effectiveness and relevance of the language model output within the defined constraints. In their recent work, Bao et al. [17] introduce a novel prompt technique tailored for multi-document summarization. The Chain-of-Event (CoE) technique comprises four components: task description, process summary reasoning, an exemplar of summary reasoning, and multi-document input. The effectiveness of the proposed method is evaluated against both zero-shot and few-shot prompting methods, assessing its comparative efficacy. On the other hand, Arawjo et al. [18] introduce ChainForge, an open-source visual programming environment tailored for prompt engineering and LLM sensemaking, thereby facilitating users in navigating and comprehending LLM outputs. Chen et al. [19] present a novel approach called AutoPrompt-based Prompt Tuning (APT), which enhances continuous prompts with a gradient-guided automatic search mechanism aimed at creating optimal discrete templates and identifying trigger tokens.

Several papers evaluate LLMs regarding the generation of code from a text description, also called text-to-code task. Nguyen et al. [20] present a framework aimed at generating API testing scripts employing ChatGPT. Their approach involves designing a prompt template that incorporates the test case scenario, data generation rules, pertinent information extracted from API documentation, the expected result format, and, if available, anticipated execution feedback. Leveraging the few-shot technique, they utilize previous logs and the model's responses. Kwon et al. [21] explore the potential of ChatGPT to enhance the quality of Ansible scripts within the domain of edge cloud systems. Through the evaluation of ChatGPT's code recommendation proficiency on 48 code revision instances extracted from 25 Ansible project GitHub repositories by three independent raters, the study verifies the model's capability to detect and comprehend Ansible scripts. However, the authors highlight that ChatGPT's efficacy is substantially influenced by the query formulation. In their investigation, Liu et at. [22] conduct experiments utilizing the CodeXGlue dataset to assess the text-to-code and code-to-code generation capabilities of ChatGPT. Employing the chain-of-thought strategy, the authors implement multi-step optimization techniques guided by feedback from ChatGPT. This approach results in a notable enhancement in performance levels, underscoring the efficacy of their methodology.

The "one-shot", "few-shot", and the PERFECT framework proposed by [16] are unsuitable for our goal since they involves providing the LLM with illustrative examples. Indeed, in our case, the availability of an existing smart contract example would eliminate the need to generate the smart contract from the provided textual description or legal agreement unnecessary. However, the techniques mentioned above are more suitable for enhancing or debugging pre-existing contracts rather than generate new ones from legal documents. Therefore, we decided to use the CO-START method which does not require providing a code example in its formulation.

As far as text-to-code is concerned, there is a lack of prompt engineering methods and frameworks, or at least the frameworks proposed in the literature are not tested on code generation.

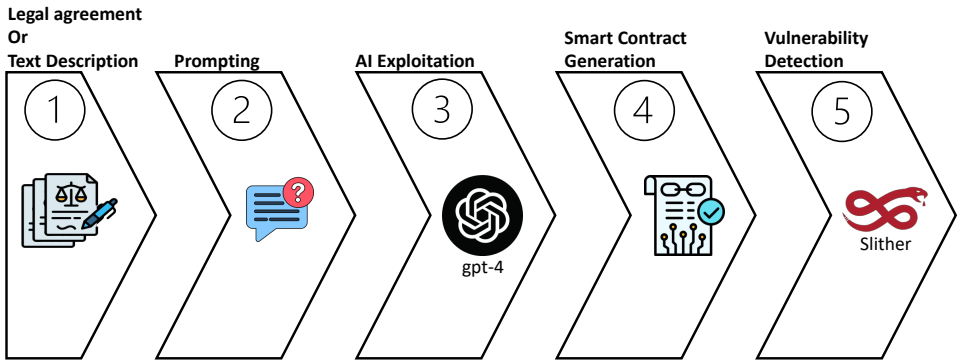

Figure 1: The adopted pipeline

3. Experimental Methodology and Evaluation

Our research experiment integrates the generation and evaluation of smart contracts from prompts composed of a legal agreement or textual descriptions using Large Language Models, specifically OpenAI's GPT-4 model, with a focus on security and performance metrics. As depicted in Figure 1, our experiment consists of a pipeline split into three key steps:

1. Selection of a legal agreement or textual description as input (Section 3.1)
2. Embedding the selected text into a set of prompts, tailored for the LLM to generate preliminary smart contract code (Section 3.2)
3. Analysis of the resulting source code with a vulnerability detection tool to assess security integrity (Section 3.3)

In this section we present these steps and in Section 4 we present an evaluation of the results gathered.

3.1. Legal Agreement Analysis for Smart Contract Conversion

We specifically focus on a legal agreement [3] detailing the lease terms between a landlord and a potential tenant. This document is featured in Appendix A and has been chosen not only for its relevance but also because its length falls within the optimal context window of the LLM. This characteristic ensures that the model can fully comprehend and process the entire agreement without the need for segmenting or summarizing the document, which is crucial for maintaining the legal integrity of the converted smart contract.

This Standard Lease Agreement outlines the rental terms between David Miller (Landlord) and Richard Garcia (Tenant) for a residential property located at 7000 NW 27th Ave, Miami, Florida. The lease runs for a fixed term from March 1, 2022, to March 1, 2023, with the Tenant agreeing to pay 850 USD monthly. A security deposit of 1,700 USD is required, which is refundable under certain conditions post-lease termination. A late fee of 200 USD for rent payments made after the 10th day past due are expected.

3.2. Prompt Formulation and LLM Selection

Based on the findings from Chang et al. [23] and our analysis, we have selected OpenAI's `gpt-4-0125-preview` model for its superior performance in tasks related to natural sciences, engineering, and semantic understanding. This selection is critical for accurately translating complex legal agreements into secure smart contracts.

The employment of "one-shot" or "few-shot" (see Section 2) prompting methods did not align with our project goals, as these techniques hinge on providing the model with specific examples to direct

[3] https://www.lawdepot.com/

Metric	Evaluation Criteria/Weight
Compilability	1 if compiles, 0 if does not
Vulnerability Assessment	-1 (high risk), -0.5, -0.25 (low risk)
Solidity Version	1 if $\geq$ 0.8.0, 0 if $\leq$ 0.8.0
Number of Contracts Generated	0.25 per contract
Number of Functions	0.15 per function
Presence of Comments	1 if present, 0 if absent
Number of Initialized Parameters	0.5 per parameter

Table 1
Summary of Smart Contract Evaluation Metrics

its outputs toward a desired result. For this reason, the strategy advocated by Ratnayake et al. [16], which also involves supplying the LLM with illustrative instances, was considered unsuitable for our needs. In our particular case, the presence of an existing smart contract example would eliminate the necessity for initiating prompts to the model, implying that such prompting techniques are better suited for enhancing or debugging pre-existing contracts rather than creating new ones from legal documents.

Given the inadequacy of "few-shot" or "one-shot" prompting for our objectives, we adopted the CO-STAR framework for prompt construction, which includes Context, Objective, Style, Tone, Audience, and Response components. This approach ensures that the prompts are comprehensively designed to elicit high-quality smart contract code from the LLM [8].

3.3. Security Analysis Tools and Evaluation Metrics

For ensuring the security of the generated smart contracts, we incorporated Slither [24], a static analysis framework for Solidity contracts. This tool is selected for its efficiency in identifying a broad spectrum of security vulnerabilities and its fast execution time, which is paramount for processing multiple iterations within our pipeline.

The reliability, security, and effectiveness of the generated smart contracts are assessed starting from the metrics defined by Tonelli et al. [25]. These metrics include compilability, vulnerability assessment, Solidity version compatibility, and structural aspects such as the number of contracts, functions, and the presence of comments and initialized parameters, see Table 1 for the exact values. A Weighted Total Score, aggregating these metrics by multiplying the compilability result with the sum of all the other values, serves as a comprehensive measure to evaluate the smart contracts' quality.

These methodologies and metrics ensure a thorough evaluation of the smart contracts generated through our pipeline, providing insights into both their technical robustness and security posture. On the other hand, the tools can not assess functional correctness. for this reason we also performed a exper evaluation as explained in Section 4

4. Comprehensive Evaluation

In the following we present the results of the automatic analysis (Section 4.1 and the expert evaluations (Sections 4.2 to 4.4)

4.1. Weighted Total Score Evaluation

As delineated in Section 3.3, we carried out an assessment of the results derived from the autonomous generation of smart contracts, utilizing appropriate metrics (Table 1). The results are available as a spreadsheet file[4]. Of the entire 68 smart contracts under examination, merely four of them could not be successfully compiled. The aforementioned derivative is rather promising as it indicates the proficient

[4]The spreadsheet file containing our results can be found here: https://docs.google.com/spreadsheets/d/1zPk6Ucb8n2UAaExGa1F18ODV_s2CYQV6/edit?usp=sharing&ouid=101965872120892214476&rtpof=true&sd=true

capabilities of the LLM in generating a majority of the code that could be compiled with no difficulties (94.1%).

Additionally, the vulnerability evaluation carried out employing the automated tool, Slither, proved to be promising. In essence, a majority of the smart contracts revealed only low-impact vulnerabilities. A minor segment of the smart contracts displayed moderate vulnerabilities, on the other hand, only 2 of the smart contracts exhibited any critical vulnerability. These results can be seen in the `vuln_count` column of the spreadsheet file.

A Weighted Total Score (`total_score` column of the spreadsheet file) was formulated for the purpose of this evaluation (Section 3.3 to see how it is computed). A worrying observation made in this regard was the apparent lack of correlation between the prompt and the consequential value of the total score. Interestingly, all the resultant values tend to cluster around 4 to 6, yet there are sporadic peaks at values of 10 and 12, predominantly in prompt $PR3$. Based on the metrics employed, this scenario signifies a confluence of a very low presence of vulnerabilities, combined with a high function count, and elevated comment and variable initialization and assignment rates. This outcome does not align with the anticipate results as per the CO-STAR techniques, whereby a higher role accompanied by a specific constraint in the generation is expected to yield optimized and superior code, a finding we were unable to corroborate.

4.2. Functions and Functionalities

In this section we explore the integration of smart contract functionalities within generated smart contracts. From a purview of the functional elements present in the lease agreement (Appendix A) and encoded within the smart contracts generated by the LLM, we expected that the generated smart contract included four key functionalities. The preliminary functionality we perceived as indispensable relates to ensuring the efficient execution of lease payments. This would entail the establishment of a function capable of facilitating the seamless transition of payment from the tenant to the landlord in an automatic fashion.

Moreover, an effective smart contract must be equipped with a feature conducive to efficient security deposit management. This illustrates a functionality analogous to the rent payment operation; however, in stark contrast, the designated funds are anticipated to be retained within the confines of the tenant's account rather than facilitated towards the landlord. Thus, effectively enabling the smart contract to function in the capacity of a trustless escrow.

Additionally, any robust leasing agreement would necessitate the presence of a meticulously crafted lease management system. We anticipated two distinct functions to serve the purpose: the initial being for lease termination, and the latter concerning activation. In our comprehensive analysis, we assumed that these simplistic yet efficient functions would be optimally suited to undertake checks on the status of the smart contract. In doing so, it would render the decision, whether to engage or disengage the lease, by modifying a select variable accordingly.

Another anticipated functionality pertained to the comprehensive management of late fees. We envisioned a single function that systematically checks for the projected payment timeline for the rent. If a delay on the tenant's behalf is ascertained, the function would promptly generate the updated rent payment and corresponding amount reflective of the late fee incurred. The function geared towards managing late fees could be integrated within the ambit of rent management or it could potentially function independently.

Lastly, we conceived the potential inclusion of a function dedicated to coin management. Given that the agreed rental payment is 850 USD as per the lease agreement specifications, we expected the smart contract to comprehend if an equivalent payment of the said amount was due in Ethers. The function would then process the conversion of 850 USD into Ethers or alternatively, manage a scenario wherein said payment should be made in a stablecoin. In the first instance, the coin management feature would arrange potential swaps between Ether and a landlord-specified stablecoin. In the second instance, it would facilitate potential transfers between different stablecoins under certain conditions. This feature could potentially extend to manage late fee and security deposits in the same way.

In the following we present the analysis of the actual functions we obtained by the smart contracts coding the functionalities we just described. All smart contracts are available in our GitHub[5].

4.2.1. The `payRent` Function

The `payRent` function, found ubiquitously across all 68 smart contracts analyzed, is primarily responsible for controlling the rent payment operation. These contracts present two main versions of this function. The first variant receives the month as an input parameter, predominantly exploited to verify whether the payment for the current month has been executed. If not, it proceeds to update the corresponding variable to indicate the receipt of payment.

The alternative version of this function notably lacks any parameters. Consequently, the sole validity check conducted within the function is to ensure that a rent payment arrives within the designated `leaseStart` and `leaseEnd` timeframe.

Both versions verify that the rent amount corresponds with the incoming message value (`msg.value` in solidity). This function, as is explained in Section 5.5, confirms this message value in *Ethers*, even if it is highly implausible for that amount (850 Ethers) to be made in that currency.

An anticipated feature within the `payRent` function would be the automated forwarding of the payment to the landlord. However, we observed that the bulk of the `payRent` functions do not contain the code lines necessary for this operation. This may lead to instances where the coins either remain unspent in the smart contract or require manual collection by the landlord via another function trigger.

It's crucial to note that the latter scenario is rarely encountered, while the majority of `payRent` function implementations fail to forward any coins to the landlord. We discerned this to be a form of logic error undetectable by current vulnerability detection systems (see Section 5.2).

The behavior mentioned above could potentially be attributable to a comment found within the smart contract, procured using prompt $PR1$ during the second test, which alluded to the fact that the entire smart contract was an imitation of the landlord's balance for simplification purposes. This comment suggested that the smart contract may be the landlord himself. However, given that actual landlords are real individuals, inhibiting their ability to withdraw coins is tantamount to the coins being immobilized (*freezed*) within the smart contract.

The `payRent` functions across different smart contracts exhibited minor variations concerning their modifiers. All of these functions are designated as `external` and `payable`. Some versions bear the `onlyTenant` modifier, which limits the function trigger to the tenant exclusively. They also apply the `isLeaseActive` modifier, which verifies an active lease status.

Contrarily, the alternative `payRent` function variant integrates a `require` directive within the function to check the message sender or the current lease status. Nonetheless, a significant percentage of smart contracts opt for the modifier version, which was deemed the preferred option in our analysis owing to the adaptability of these modifiers for application in other contract functions.

4.2.2. The `paySecurityDeposit` Function

The `paySecurityDeposit` function, which facilitates the process of tenants paying the security deposit, serves as the focal point of this exposition.

Surprisingly, this function is absent in the majority of Solidity contracts we examined. This particular omission signifies a critical error by the LLM, as the function's absence renders it unfeasible for the tenant to make payments towards their respective security deposit. This particular anomaly is particularly baffling, considering all smart contracts ubiquitously include the `securityDeposit` variable.

The `paySecurityDeposit` function, when implemented, exhibits uniformity across all smart contracts. Noteworthy is that the function does not incorporate any variables and is characterized as `external` and `payable` and can only be activated by the tenant. It commonly requires the message value to align with the pre-established `securityDeposit` variable.

[5]GitHub link: https://github.com/BChain4all/pipeline/tree/main/results-feb-24

Despite the expectation that this function should be executed prior to the initiation of the lease term, as explicitly stated within the lease agreement, several iterations of the `paySecurityDeposit` function conspicuously lack embedded controls for this timeline regulation.

Additionally, multiple versions of the `paySecurityDeposit` function include a comment acknowledging the distinct security requirements that should be managed by the smart contract developer. This recognition implies a crucial consideration for potential code enhancement, but implementation remains sorely lacking.

Another point of contention is the common misinterpretation of the security deposit fund flow dynamics. Ideally, the security deposit funds should be retained within the smart contract as a form of escrow, but it is alarming to note that certain `paySecurityDeposit` functions facilitate an automatic transfer of these funds to the landlord, a feature we categorize as programmatically erroneous. This necessary escrow provision is designed to ensure that the security deposit is appropriately sheltered from premature landlord access until the conclusion of the lease tenure.

4.2.3. The `refundSecurityDeposit` Function

The `refundSecurityDeposit` function, a ubiquitous feature in all smart contracts, introduces a nuanced perspective compared to the `paySecurityDeposit` function.

The function exists in two distinct versions. The first one does not necessitate any parameter and is generally executed by the landlord. Additionally, it mandates that the lease is no longer operative. Both requisite conditions are substantiated through assigned modifiers. Pertinent checks concerning the admission of the initial security deposit are incorporated within the function provided the `paySecurityDeposit` function is a present aspect of the smart contract. However, in its absence, no substantial controls are executed beyond the ones established by the modifiers.

The alternate version of the `refundSecurityDeposit` function requires a parameter, typically referred to as `deductions`. These deductions are consequently deducted from the `securityDeposit`, establishing a refund proportional to the tenant.

In-depth analysis reveals how almost all iterations of the `refundSecurityDeposit` function autonomously transfer the refund amount, either equivalent to the security deposit itself or the resultant difference between the security deposit and the deduction, to the tenant. These versions invariably circumvent any transfers to the landlord, hence the funds associated with the security deposit typically remain freezed within the smart contract given the absence of auxiliary functions to extricate them.

4.2.4. Management of Late Fees

Pertaining to the lease agreement, referenced in Appendix A, a penalty fee of 200 USD is levied against the tenant who fails to timely settle rent payments, specifically beyond fifteen days after the fifth of the month. The handling of said late fee necessitates the involvement of a designated function or incorporation within the `payRent` function.

The examination of 68 smart contract revealed that, in the majority of them, this late fee provision has been neglected. In the few instances where late fees have been administrated within the smart contract, we have identified two primary methods of implementation.

One approach relies on the security deposit (refer to Sections 4.2.2 and 4.2.3 for an explanation of what it is and how it is managed) and offers a `lateFee` function that deducts the late fee from the security deposit. This function has been adapted such that only the landlord can collect the late fee. Generally, it overlooks checking the sufficiency of funds within the security deposit. Evidently, this method of late fee management falls short of expectations. Theoretically, a tenant could accrue a total of USD $1,800.00$ or more in late fees (for nine or more occurrences). This equivalency is more than USD $1,700.00$, exceeding the value that should be withdrawn from the security deposit.

We propose that a superior method would be to internalize the management of late fees within the `payRent` function. Consequently, rent totals would increase relative to a designated date via the block timestamp. A limited number of `payRent` functions were discovered to effectively manage late fees in

this manner. It is noteworthy that these functions lack the automatic transfer of rent to the landlord and therefore, even when late fees are duly administered, the respective amount fails to reach the landlord. This situation is elaborated on further in Section 4.2.1.

4.2.5. Lease start management

The deployment of a smart contract on a blockchain does not necessarily implicate that the lease agreement commences immediately upon deployment. A more intuitive approach is to deploy the smart contract before the lease's start date, principally considering that preliminary security deposits stipulated in the lease agreement need to be addressed before the lease's activation. As discussed in Section 4.2.2, these payments are essential components of the smart contract's function. Consequently, two predominant strategies can be identified to manage this aspect.

In the first approach, the LLM integrates the variables `leaseStart` (indicating the start time of the lease) and `leaseActive` (representing the activation status of the lease) within the contract's `constructor`. In such instances, the commencement of the lease is considered instantaneous and automatic upon deployment, as the `leaseActive` variable is set to true inside the constructor. Resultantly, the entire lease contract is activated. It is worth noting that in this scenario, those contracts featuring a `paySecurityDeposit` function should not be obliged to verify `leaseActive`, as it is inherently set to `true` at the contract's start and deployment phase. These contracts should solely verify `leaseStart`. This autonomous lease commencement management is the most prevalent method.

Conversely, the second approach offers a more manual control over lease commencement. In these cases, smart contracts typically feature a specialised function, `startLease`, that can solely be triggered by the landlord after verifying the `leaseStart` variable. This customarily sets `leaseActive` to true, thereby activating the lease agreement. In such instances, the `paySecurityDeposit` function can be triggered prior to the lease activation, thus ensuring adherence to the terms stipulated in the lease agreement.

In cases where the `startLease` function can assign the `active` status to the current lease state, the `paySecurityDeposit` function may verify the lease activity status insted of a the configuration derived from the current timestamp set by the block, merely referencing the defined `leaseStart` variable. This ensures a more meticulous verification procedure.

4.2.6. Management of Lease Termination

Unlike lease commencement, lease termination within smart contracts incorporates an assigned function. However, this function often exhibits notable deficiencies. Predominantly, it can only be activated by the landlord (`onlyLandlord` modifier) during the period of active lease (`isLeaseActive` modifier), but it doesn't account for specifics such as lease termination dates or other relevant variables. As such, to the landlord is given the capacity to terminate the lease at will, in accordance with the stipulations of smart contracts generated by the LLM. This makes it challenging to forestall unwarranted terminations initiated by landlords. We note that, while it is relatively easy for experts to introduce legal prerequisites, translating these requirements into practical components of a smart contract proves more complex.

An apt solution may involve invoking a specific oracle prior to lease termination. Although it has been observed that the GPT-4 LLM probably lacks currently the capacity to devise separate contracts to manage such scenarios (since it never created them nor suggested to the developer to do so in any of the 68 calls, see Section 4.1), it is pertinent to note that it has not been explicitly tasked to do so. Equally, numerous smart contracts' comments have highlighted the need for completion of such functions. This is yet another manifestation of the flaw observed earlier in the `payRent` function (Section 4.2.1), which resists detection through automated tools.

In closing, we discerned that the lease termination function demonstrates remarkable consistency across different smart contracts, with negligible variation witnessed in its formulation.

4.3. Parameters and Initialization

In the formulation of our analysis, we elected to scrutinize the initialization and assignment of variables. The initialization of a variable is ascribed when a variable obtains a value beyond basic initialization. For instance, the `uint256 public constant rentAmount` variable is denoted as initialized, while `uint256 public constant rent amount = 850` is identified as both initialized and assigned.

We explicitly directed the language model to assign values to the variables in prompts $PR4$, $PR5$, $PR9$, $PR10$, and $PR14$ to $PR17$. Our expectation stipulated that, within these prompts, every conceivable variable must be assigned. The practical implication of this assertion ensures that these parameters are not passed as input during deployment of the smart contract.

Upon interpreting the lease agreement, we identified multiple variables that could be assigned. We postulated these must be set prior to deploying the smart contract:

- The names and addresses of both the landlord and tenant
- The address of the property for lease
- The lease commencement and conclusion dates
- The monthly rental charge
- The security deposit
- The late fee

There typically exist two distinct mechanisms for assigning variables in a smart contract in Solidity. The primary method simply assigns a value to an initialized variable within the contract. Alternatively, the value of the variable can be assigned within the `constructor`. Consequently, even if variable assignment transpires during contract deployment, user input is not necessary during the deployment stage.

In our evaluation of the smart contracts, we discovered an unexpectedly weak correlation between our expressly requested variable assignment and the actual resultant assignment. For instance, within prompt $PR4$, only one of the four calls includes the assignment of the landlord and tenant names and addresses. In contrast, prompt $PR5$ never assigns the landlord and tenant's names and addresses in any contract, notwithstanding our explicit request.

An intriguing variable assignment feature is the consistent use of Ether as a currency, despite our lease agreement exclusively utilizing USD currency. This utilization occurs throughout all 68 smart contracts, evidenced by a 200 USD late fee listed in the agreement always being equated as 200 Ether in the smart contract. This detail is explored further in Section 5.5, which notes the absence of a function to manage currency conversion in any smart contract.

Finally, any lease start and end times that are assigned adopt UTC times in seconds. The lease commencement date is March 1st, 2022, while the lease conclusion date is March 1st, 2023. The UTC time used is always accurate[6].

4.4. Smart contract comments

Our examination extended to the comments embedded within the smart contracts, notwithstanding that our prompts did not explicitly mandate their inclusion. The variability in comments within the smart contracts under review was not significantly compelling.

The comments, ostensibly terse, offered a correspondingly precise explanation of the functionality being executed. They were employed strategically to demarcate distinct sections of the smart contracts, distinguishing among function and variable definitions, events, and modifiers.

However, an argument for the substantive relevance of these comments raises some doubt. Despite their accurate representation of the function's purpose, the comments, for the most part, revert to an

[6]The `bash` command `date --date='@1646092800'` (the set time in the smart contracts for the starting date) resulted in `Wed Mar 1 01:00:00 AM CET 2022`. This time corresponds to the midnight in CET, the timezone in which the LLM was called. We had anticipated the midnight according to a USA timezone, considering the lease agreement clearly pertains to an apartment located in Florida, USA, and the OpenAI servers are also located in the USA. Same result has been obtained for the termination date.

oversimplified explanation that could be deemed inconsequential. The core reasoning behind this style of commentary remains uncertain.

One interpretation is that this reflects an inherent simplicity of the functions under scrutiny, such that any developer, irrespective of their expertise level, might comprehend the functionality readily. An alternative supposition posits a restriction in the capability of the large language model in fabricating comprehensive comments.

It is important to note that the LLM's training draws from Solidity code available publicly on GitHub, thus implying a possibility that the observed comment trend might echo a broader developer trend. This trend may suggest a prevalent underutilization of comments as a tool for meaningful and expressive communication within the developer community.

5. Discussion

The following discussion pertains to the revelations gathered from the automated and expert evaluations carried out during this study and presented in Section 4.

5.1. Absence of correlation between prompts and results

We applied the CO-STAR technique, anticipating an observable connection between the ranking of the software developer (referring to varying roles) used in the prompt, and the quality of the resulting smart contract. We expected that the code engendered by an imaginary master's student would be inferior to the contract generated by the junior developer, and subsequently the senior Solidity developer (see Appendix B to read all the prompts used throughout the study). Additionally, we formulated different objectives manifested in the prompts, envisioning that objectives with explicit precision (for instance, the version of Solidity to use, the demand for fully articulated function logic, the overt call for variable assignment), would mirror the directive more than prompts lacking these attributes. These requests and expectations are in line with our ultimate goal to fashion a prompt, or a series of prompts, which can be employed by non-professionals in the field to autonomously generate production-ready smart contracts.

As highlighted before, our principal conclusion is the absence of any correlation between the assigned ranking and the quality of the resultant smart contract. This is discernible both in the automated evaluation deploying quantitative parameters and metrics and in the manual expert evaluation. Similarly, we found no correlation between the unambiguous objectives introduced in the prompts and the actual realization of said objectives. Refer e.g. to Sections 4.2.2 and 4.3 where we note multiple instances of missing variable assignments and a critical insufficiency of requisite functions.

5.2. Non-detectable Flaws

Another alarming and prevalent issue we discovered during the smart contract analysis was the production of logical bugs by the LLM. These bugs are effectively undetectable through automated vulnerability detection tools. For instance, as presented in Section 4.2.1, from the perspective of the LLM, the smart contract itself personifies the landlord. Consequently, funds are stuck inside the smart contract and the (real) landlord can not withdraw them. This conclusion was reached only after a thorough manual review of all contracts. As a result, non-experts in the field would be unlikely to discover such a flaw.

The subsequent issue of concern pertains to the absence of a genuine comprehension of the inherent complexities of a legal contract. For instance, despite the fact that the termination of a contract can only be effected by the landlord and only while the contract is active, a plain variable alteration bereft of any additional control can potentially engender vulnerabilities which include unwarranted interference by the landlord himself, who can then terminate the contract prematurely.

5.3. Incomplete Functions and Variable Assignment

Another observation was the tendency for the LLM to neglect to complete functions. Despite the clarity of instructions from the legal agreement, the generated smart contracts seldom possess fully developed functions. Strikingly, this pattern does not correlate with whether or not the prompt explicitly requests fully defined function logic.

A related issue pertains to the assignment of variables: only a few variables are assigned. Furthermore, numerical variables (e.g. the amount of the rent) are generally perceived as fixed parameters and therefore assigned, whereas string parameters (e.g. the physical address of the rental property) are seldom utilized as a whole and lack assignment.

5.4. Limited Variations of Function Representation

Another commonality we discerned across all functions in every smart contract was the limited variability for each function. Most functions manifested in one of two versions, with a third version accounting for the absence of that function altogether. It is challenging to predict which function will be included or excluded. This further underscores the lack of correlation between the LLM prompts and the ensuing results.

5.5. Failures in Currency Understanding

A peculiar error we observed in the LLM pertained to the management of currency. All values in the lease agreement were expressed in USD, while each smart contract expressed every variable in Ethers. The currency discrepancy clearly affects the sustainability of the rent and further exemplifies how unsuitable the smart contracts generated by the LLM are for non-professional users in terms of production readiness.

6. Conclusions

This research embarked on an exploratory journey to examine the potential integration between two disruptive technologies, Blockchain and AI, with a focus on the application of LLMs for aiding non-experts in crafting production-ready Solidity smart contracts. Through a methodical investigation involving the translation of a lease agreement into smart contract code by GPT-4, supported by 17 meticulously designed prompts that generated 68 smart contracts (4 per prompt), our study provided a thorough assessment of the capabilities of current LLM technology in this context.

The dual-faceted evaluation methodology, integrating automated analysis and expert manual review, has revealed significant insights into the practical utility of GPT-4 within the blockchain development sphere. The conclusions drawn from this investigation are unequivocal: the present version of GPT-4 falls short of the necessary standards for generating production-ready smart contracts. This shortfall is attributed predominantly to the presence of undetected logic flaws and the misalignment between the provided prompts and the resulting code.

These findings not only highlight the existing gaps within the capabilities of LLMs like GPT-4 but also accentuate the critical challenges associated with the autonomous creation of smart contracts that are applicable in real-world scenarios.

Building upon our findings, our future endeavors aim to incorporate alternative prompting techniques. Our exploration thus far suggests that zero-shot prompting offers suboptimal results; consequently, we intend to incorporate more promising strategies such as few-shot and chain of thought prompting techniques in our methodology.

In theory, we could consider training or fine-tuning the LLM specifically to enhance its comprehension of smart contracts. This potential adaptation can potentially serve to augment its overall performance in this context.

Acknowledgments

The work discussed in this paper has been supported by the B4A - Blockchain for All project (https://www.blockchain4all.it/), ref. no. 20225MN5K3. This project has been funded with support from the Ministry of Education, University and Research. This document reflects the views only of the authors, and the Ministry of Education, University and Research cannot be held responsible for any use which may be made of the information contained therein.

References

[1] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, 2023.

[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al., Llama 2: Open foundation and fine-tuned chat models, 2023.

[3] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al., Palm: Scaling language modeling with pathways, Journal of Machine Learning Research 24 (2023) 1–113.

[4] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al., Mistral 7b, 2023.

[5] Y. Li, S. Wang, H. Ding, H. Chen, Large language models in finance: A survey, 2023, p. 374 – 382. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85179853539&doi=10.1145%2f3604237.3626869&partnerID=40&md5=ee2b728c8b9fe0013974da302d3afc8a. doi:10.1145/3604237.3626869, cited by: 0; All Open Access, Green Open Access, Hybrid Gold Open Access.

[6] J. Li, A. Dada, B. Puladi, J. Kleesiek, J. Egger, Chatgpt in healthcare: A taxonomy and systematic review, Computer Methods and Programs in Biomedicine 245 (2024). doi:10.1016/j.cmpb.2024.108013.

[7] L. Belzner, T. Gabor, M. Wirsing, Large language model assisted software engineering: Prospects, challenges, and a case study, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 14380 LNCS (2024) 355 – 374. doi:10.1007/978-3-031-46002-9_23, cited by: 2.

[8] G. D. Science, S. G. AI Division, Prompt engineering playbook, ???? URL: https://www.developer.tech.gov.sg/products/collections/data-science-and-artificial-intelligence/playbooks/prompt-engineering-playbook-beta-v3.pdf.

[9] U. Lee, H. Jung, Y. Jeon, Y. Sohn, W. Hwang, J. Moon, H. Kim, Few-shot is enough: exploring chatgpt prompt engineering method for automatic question generation in english education, Education and Information Technologies (2023) 1–33.

[10] M. Wang, M. Wang, X. Xu, L. Yang, D. Cai, M. Yin, Unleashing chatgpt's power: A case study on optimizing information retrieval in flipped classrooms via prompt engineering, IEEE Transactions on Learning Technologies 17 (2024) 629–641. doi:10.1109/TLT.2023.3324714.

[11] L. Giray, Prompt engineering with chatgpt: A guide for academic writers, Annals of Biomedical Engineering (2023) 1–5.

[12] D. Grabb, The impact of prompt engineering in large language model performance: a psychiatric example, Journal of Medical Artificial Intelligence 6 (2023).

[13] L. Wang, X. Chen, X. Deng, H. Wen, M. You, W. Liu, Q. Li, J. Li, Prompt engineering in consistency and reliability with the evidence-based guideline for llms, npj Digital Medicine 7 (2024) 41.

[14] X. Li, E. Liu, T. Shen, J. Huang, F.-Y. Wang, Chatgpt-based scenario engineer: A new framework on scenario generation for trajectory prediction, IEEE Transactions on Intelligent Vehicles (2024) 1–10. doi:10.1109/TIV.2024.3363232.

[15] M. Ribary, P. Krause, M. Orban, E. Vaccari, T. Wood, Prompt engineering and provision of context in domain specific use of gpt, in: Legal Knowledge and Information Systems, IOS Press, 2023, pp. 305–310.

[16] H. Ratnayake, C. Wang, A prompting framework to enhance language model output, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2023, pp. 66–81.
[17] S. Bao, T. Li, B. Cao, Chain-of-event prompting for multi-document summarization by large language models, International Journal of Web Information Systems (2024).
[18] I. Arawjo, P. Vaithilingam, M. Wattenberg, E. Glassman, Chainforge: An open-source visual programming environment for prompt engineering, in: Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST '23 Adjunct, Association for Computing Machinery, New York, NY, USA, 2023. URL: https://doi.org/10.1145/3586182.3616660. doi:10.1145/3586182.3616660.
[19] Y. Chen, G. Yang, D. Wang, D. Li, Eliciting knowledge from language models with automatically generated continuous prompts, Expert Systems with Applications 239 (2024) 122327. URL: https://www.sciencedirect.com/science/article/pii/S0957417423028294. doi:https://doi.org/10.1016/j.eswa.2023.122327.
[20] C. Nguyen, H. Bui, V. Nguyen, T. Nguyen, An approach to generating api test scripts using gpt, in: Proceedings of the 12th International Symposium on Information and Communication Technology, SOICT '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 501–509. URL: https://doi.org/10.1145/3628797.3628947. doi:10.1145/3628797.3628947.
[21] S. Kwon, S. Lee, T. Kim, D. Ryu, J. Baik, Exploring the feasibility of chatgpt for improving the quality of ansible scripts in edge-cloud infrastructures through code recommendation, in: International Conference on Web Engineering, Springer, 2023, pp. 75–83.
[22] C. Liu, X. Bao, H. Zhang, N. Zhang, H. Hu, X. Zhang, M. Yan, Improving chatgpt prompt for code generation, arXiv preprint arXiv:2305.08360 (2023).
[23] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al., A survey on evaluation of large language models, ACM Transactions on Intelligent Systems and Technology (2023).
[24] J. Feist, G. Grieco, A. 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), IEEE, 2019. URL: http://dx.doi.org/10.1109/WETSEB.2019.00008. doi:10.1109/wetseb.2019.00008.
[25] R. Tonelli, G. A. Pierro, M. Ortu, G. Destefanis, Smart contracts software metrics: A first study, PLOS ONE 18 (2023) 1–31. URL: https://doi.org/10.1371/journal.pone.0281043.

A. Lease Agreement

STANDARD LEASE AGREEMENT

This Agreement, dated , 20. , by and between an individual known as David Miller of 324 W Gore St, Orlando, Florida, 32806, hereinafter known as the "Landlord",

AND

An individual known as Richard Garcia, hereinafter known as the "Tenant(s)", agree to the following:

OCCUPANT(S): The Premises is to be occupied strictly as a residential dwelling with only the Tenant(s) mentioned above as the Occupant(s).

OFFER TO RENT: The Landlord hereby rents to the Tenant(s), subject to the following terms and conditions of this Agreement, an apartment with the address of 7000 NW 27th Ave, Miami, Florida, 33147 consisting of 1 bathroom(s) and 2 bedroom(s) hereinafter known as the "Premises". The Landlord may also use the address for notices sent to the Tenant(s).

PURPOSE: The Tenant(s) and any Occupant(s) may only use the Premises as a residential dwelling. It may not be used for storage, manufacturing of any type of food or product, professional service(s), or for any commercial use unless otherwise stated in this Agreement.

FURNISHINGS: The Premises is furnished with the following:

Bedroom Set(s), Dining Room Set(s), Kitchen Set (Including Pots, Pans, Glasses, Mugs, Dishes, etc.), and all other furnishings to be provided by the Tenant(s). Any damage to the Landlord's furnishings shall be the liability of the Tenant(s), reasonable wear-and-tear excepted, to be billed directly or less the Security Deposit.

APPLIANCES: The Landlord shall not provide any appliances in the Premises.

LEASE TERM: This Agreement shall be a fixed-period arrangement beginning on March 1 2022 and ending on March 1 2023 with the Tenant(s) being required to move-out at the end of the Lease Term if a new Lease Agreement is not authorized. Hereinafter known as the "Lease Term".

RENT: Tenant(s) shall pay the Landlord in equal monthly installments of $850.00 (USD) hereinafter known as the "Rent". The Rent will be due on the Fifth (5th) of every month and be paid by sending payment to the Landlord's aforementioned mailing address.

NON-SUFFICIENT FUNDS (NSF CHECKS): If the Tenant(s) attempts to pay the rent with a check that is not honored or an electronic transaction (ACH) due to insufficient funds (NSF) there

shall be no fee (USD).

LATE FEE: If rent is not paid on the due date, there shall be a late fee assessed by the Landlord in the amount of $200.00 (USD) per occurrence for each month payment that is late after the 10th Day rent is due.

FIRST (1ST) MONTH'S RENT: First (1st) month's rent shall be due by the Tenant(s) upon the execution of this Agreement.

PRE-PAY MENT: The Landlord shall not require any pre-payment of rent by the Tenant(s).

PRORATION PERIOD: The Tenant(s) will not move into the Premises before the start of the Lease Term.

SECURITY DEPOSIT: A Security Deposit in the amount of $1,700.00 (USD) shall be paid by the Tenant(s) to the Landlord, at the execution of this Agreement for the faithful performance of all the terms and conditions. The Security Deposit is to be returned by the Landlord to the Tenant(s) within 15 days after this Agreement has terminated if there are no deductions. If there are deductions to the Security Deposit the amount shall be returned within 30 days from termination of this Agreement. This Security Deposit shall not be credited towards rent unless the Landlord gives their written consent.

POSSESSION: Tenant(s) has examined the condition of the Premises and by taking possession acknowledges that they have accepted the Premises in good order and in its current condition except as herein otherwise stated. Failure of the Landlord to deliver possession of the Premises at the start of the Lease Term to the Tenant(s) shall terminate this Agreement at the option of the Tenant(s). Furthermore, under such failure to deliver possession by the Landlord, and if the Tenant(s) cancels this Agreement, the Security Deposit (if any) shall be returned to the Tenant(s) along with any other pre-paid rent, fees, including if the Tenant(s) paid a fee during the application process before the execution of this Agreement.

ACCESS: Upon the beginning of the Proration Period or the start of the Lease Term, whichever is earlier, the Landlord agrees to give access to the Tenant(s) in the form of keys, fobs, cards, or any type of keyless security entry as needed to enter the common areas and the Premises. Duplicate copies of the access provided may only be authorized under the consent of the Landlord and, if any replacements are needed, the Landlord may provide them for a fee. At the end of this Agreement all access provided to the Tenant(s) shall be returned to the Landlord or a fee will be charged to the Tenant(s) or the fee will be subtracted from the Security Deposit.

MOVE-IN INSPECTION: Before, at the time of the Tenant(s) accepting possession, or shortly thereafter, the Landlord and Tenant(s) shall perform an inspection documenting the present condition of all appliances, fixtures, furniture, and any existing damage within the Premises.

SUBLETTING: The Tenant(s) shall not have the right to sub-let the Premises or any part thereof without the prior written consent of the Landlord. If consent is granted by the Landlord, the Tenant(s) will be responsible for all actions and liabilities of the Sublessee including but not limited to: damage to the Premises, non-payment of rent, and any eviction process (In the event of an eviction the Tenant(s) shall be responsible for all court filing fee(s), representation, and any other fee(s) associated with removing the Sublessee). The consent by the Landlord to one sub-let shall not be deemed to be consent to any subsequent subletting.

ABANDONMENT: If the Tenant(s) vacates or abandons the property for a time-period that is the minimum set by State law or seven (7) days, whichever is less, the Landlord shall have the right to terminate this Agreement immediately and remove all belongings including any personal property off of the Premises. If the Tenant(s) vacates or abandons the property, the Landlord shall immediately have the right to terminate this Agreement.

ASSIGNMENT: Tenant(s) shall not assign this Lease without the prior written consent of the Landlord. The consent by the Landlord to one assignment shall not be deemed to be consent to any subsequent assignment.

PARKING: The Landlord shall not provide parking to the Tenant(s).

RIGHT OF ENTRY: The Landlord shall have the right to enter the Premises during normal working hours by providing notice in accordance with the minimum State requirement in order for inspection, make necessary repairs, alterations or improvements, to supply services as agreed or for any reasonable purpose. The Landlord may exhibit the Premises to prospective purchasers, mortgagees, or lessees upon reasonable notice.

SALE OF PROPERTY: If the Premises is sold, the Tenant(s) is to be notified of the new Owner, and if there is anew Manager, their contact details for repairs and maintenance shall be forwarded. If the Premises is conveyed to another party, the new owner shall not have the right to terminate this Agreement and it shall continue under the terms and conditions agreed upon by the Landlord and Tenant(s).

UTILITIES: The Landlord agrees to pay for the following utilities and services:

Electricity, Internet, Natural Gas, Water, with all other utilities and services to be the responsibility of the Tenant(s).

MAINTENANCE, REPAIRS, OR ALTERATIONS: The Tenant(s) shall, at their own expense and at all times, maintain premises in a clean and sanitary manner, and shall surrender the same at termination hereof, in as good condition as received, normal wear and tear excepted. The Tenant(s) may not make any alterations to the leased premises without the consent in writing of the Landlord. The Landlord shall be responsible for repairs to the interior and exterior of the building. If the Premises includes a washer, dryer, freezer, dehumidifier unit and/or air conditioning unit, the Landlord makes no warranty as to the repair or replacement of units if one or all shall fail to operate. The Landlord will place fresh batteries in all battery-operated smoke detectors when the Tenant(s) moves into the premises. After the initial placement of the fresh batteries it is the responsibility of the Tenant(s) to replace batteries when needed. A monthly "cursory" inspection may be required for all fire extinguishers to make sure they are fully charged.

EARLY TERMINATION: The Tenant(s) may not be able to cancel this Agreement unless the Tenant is a victim of Domestic Violence, in such case, the Tenant may be able to cancel in accordance with any local, state, or federal laws.

PETS: The Tenant(s) shall not be allowed to have pets on the Premises or common areas except those that are necessary for individuals with disabilities.

NOISE/WASTE: The Tenant(s) agrees not to commit waste on the premises, maintain, or permit to be maintained, a nuisance thereon, or use, or permit the premises to be used, in an unlawful manner. The Tenant(s) further agrees to abide by any and all local, county, and State noise ordinances.

GUESTS: There shall be no other persons living on the Premises other than the Tenant(s) and any Occupant(s). Guests of the Tenant(s) are allowed for periods not lasting for more than forty-eight hours unless otherwise approved by the Landlord.

SMOKING POLICY: Smoking on the Premises is prohibited on the entire property, including individual units, common areas, every building and adjoining properties.

COMPLIANCE WITH LAW: The Tenant(s) agrees that during the term of the Agreement, to promptly comply with any present and future laws, ordinances, orders, rules, regulations, and requirements of the Federal, State, County, City, and Municipal government or any of their departments, bureaus, boards, commissions and officials thereof with respect to the premises, or the use or occupancy thereof, whether said compliance shall be ordered or directed to or against the Tenant(s), the Landlord, or both.

DEFAULT: If the Tenant(s) fails to comply with any of the financial or material provisions of this Agreement, or of any present rules and regulations or any that may be hereafter prescribed by the Landlord, or materially fails to comply with any duties imposed on the Tenant(s) by statute or State laws, within the time period after delivery of written notice by the Landlord specifying the non- compliance and indicating the intention of the Landlord to terminate the Agreement by reason thereof, the Landlord may terminate this Agreement. If the Tenant(s) fails to pay rent when due and the default continues for the time-period specified in the written notice thereafter, the Landlord may, at their option, declare the entire balance (compiling all months applicable to this Agreement) of rent payable hereunder to be immediately due and payable and may exercise any and all rights and remedies available to the Landlord at law or in equity and may immediately terminate this Agreement.

The Tenant(s) will be in default if: (a) Tenant(s) does not pay rent or other amounts that are owed in accordance with respective State laws; (b) Tenant(s), their guests, or the Occupant(s) violate this Agreement, rules, or fire, safety, health, or criminal laws, regardless of whether arrest or conviction occurs; (c) Tenant(s) abandons the Premises; (d) Tenant(s) gives incorrect or false information in the rental application; (e) Tenant(s), or any Occupant(s) is arrested, convicted, or given deferred adjudication for a criminal offense involving actual or potential physical harm to a person, or involving possession, manufacture, or delivery of a controlled substance, marijuana, or drug paraphernalia under state statute; (f) any illegal drugs or paraphernalia are found in the Premises or on the person of the Tenant(s), guests, or Occupant(s) while on the Premises and/or; (g) as otherwise allowed by law.

MULTIPLE TENANT(S) OR OCCUPANT(S): Each individual that is considered a Tenant(s) is jointly and individually liable for all of this Agreement's obligations, including but not limited to rent monies. If any Tenant(s), guest, or Occupant(s) violates this Agreement, the Tenant(s) is considered to have violated this Agreement. Landlord's requests and notices to the Tenant(s) or any of the Occupant(s) of legal age constitutes notice to the Tenant(s). Notices and requests from the Tenant(s) or any one of the Occupant(s) (including repair requests and entry permissions) constitutes notice from the Tenant(s). In eviction suits, the Tenant(s) is considered the agent of the Premise for the service of process.

DISPUTES: If a dispute arises during or after the term of this Agreement between the Landlord and Tenant(s), they shall agree to hold negotiations amongst themselves, in "good

faith", before any litigation.

SEVERABILITY: If any provision of this Agreement or the application thereof shall, for any reason and to any extent, be invalid or unenforceable, neither the remainder of this Agreement nor the application of the provision to other persons, entities or circumstances shall be affected thereby, but instead shall be enforced to the maximum extent permitted by law.

SURRENDER OF PREMISES: The Tenant(s) has surrendered the Premises when (a) the move-out date has passed and no one is living in the Premise within the Landlord's reasonable judgment; or (b) Access to the Premise have been turned in to Landlord — whichever comes first. Upon the expiration of the term hereof, the Tenant(s) shall surrender the Premise in better or equal condition as it were at the commencement of this Agreement, reasonable use, wear and tear thereof, and damages by the elements excepted.

RETALIATION: The Landlord is prohibited from making any type of retaliatory acts against the Tenant(s) including but not limited to restricting access to the Premises, decreasing or cancelling services or utilities, failure to repair appliances or fixtures, or any other type of act that could be considered unjustified.

WAIVER: A Waiver by the Landlord for a breach of any covenant or duty by the Tenant(s), under this Agreement is not a waiver for a breach of any other covenant or duty by the Tenant(s), or of any subsequent breach of the same covenant or duty. No provision of this Agreement shall be considered waived unless such a waiver shall be expressed in writing as a formal amendment to this Agreement and executed by the Tenant(s) and Landlord.

EQUAL HOUSING: If the Tenant(s) possess(es) any mental or physical impairment, the Landlord shall provide reasonable modifications to the Premises unless the modifications would be too difficult or expensive for the Landlord to provide. Any impairment of the Tenant(s) is/are encouraged to be provided and presented to the Landlord in writing in order to seek the most appropriate route for providing the modifications to the Premises.

HAZARDOUS MATERIALS: The Tenant(s) agrees to not possess any type of personal property that could be considered a fire hazard such as a substance having flammable or explosive characteristics on the Premises. Items that are prohibited to be brought into the Premises, other than for everyday cooking or the need of an appliance, includes but is not limited to gas (compressed), gasoline, fuel, propane, kerosene, motor oil, fireworks, or any other related content in the form of a liquid, solid, or gas.

WATERBEDS: The Tenant(s) is not permitted to furnish the Premises with waterbeds.
INDEMNIFICATION: The Landlord shall not be liable for any damage or injury to the Tenant(s), or any other person, or to any property, occurring on the Premises, or any part thereof, or in common areas thereof, and the Tenant(s) agrees to hold the Landlord harmless from any claims or damages unless caused solely by the Landlord's negligence. It is recommended that renter's insurance be purchased at the Tenant(s)'s expense.

COVENANTS: The covenants and conditions herein contained shall apply to and bind the heirs, legal representatives, and assigns of the parties hereto, and all covenants are to be construed as

conditions of this Agreement.

NOTICES: Any notice to be sent by the Landlord or the Tenant(s) to each other shall use the following mailing addresses:

Landlord's/Agent's Mailing Address

David Miller 324 W Gore St, Orlando, Florida, 32806

Tenant(s)'s Mailing Address

Richard Garcia 7000 NW 27th Ave, Miami, Florida, 33147

AGENT/MANAGER: The Landlord does not have an Agent or Manager and all contact in regards to any repair, maintenance, or complaint must go through the Landlord through the following contact information:

Landlord's Phone Number: (185) 546-3277 Email: lease.email@tmp.com.

PREMISES DEEMED UNINHABITABLE: If the Property is deemed uninhabitable due to damage beyond reasonable repair the Tenant(s) will be able to terminate this Agreement by written notice to the Landlord. If said damage was due to the negligence of the Tenant(s), the Tenant(s) shall be liable to the Landlord for all repairs and for the loss of income due to restoring the Premises back to a livable condition in addition to any other losses that can be proved by the Landlord.

ACCESS BY LANDLORD: The Landlord must provide at least twelve (12) hours notice to the Tenant(s) before entering the Premises for any non-emergency reason.

RADON GAS: Radon is a naturally occurring radioactive gas that, when it has accumulated in a building in sufficient quantities, may present health risks to persons who are exposed to it over time. Levels of radon that exceed federal and state guidelines have been found in buildings in Florida. Additional information regarding radon and radon testing may be obtained from your county health department.

SECURITY DEPOSIT DISCLOSURE: YOUR LEASE REQUIRES PAYMENT OF CERTAIN DEPOSITS. THE LANDLORD MAY TRANSFER ADVANCE RENTS TO THE LANDLORD'S ACCOUNT AS THEY ARE DUE AND WITHOUT NOTICE. WHEN YOU MOVE OUT, YOU MUST GIVE THE LANDLORD YOUR NEW ADDRESS SO THAT THE LANDLORD CAN SEND YOU NOTICES REGARDING YOUR DEPOSIT. THE LANDLORD MUST MAIL YOU NOTICE, WITHIN 30 DAYS AFTER YOU MOVE OUT, OF THE LANDLORD'S INTENT TO IMPOSE A CLAIM AGAINST THE DEPOSIT.

IF YOU DO NOT REPLY TO THE LANDLORD STATING YOUR OBJECTION TO THE CLAIM WITHIN 15 DAYS AFTER RECEIPT OF THE LANDLORD'S NOTICE, THE LANDLORD WILL COLLECT THE CLAIM AND MUST MAIL YOU THE REMAINING DEPOSIT, IF ANY.

IF THE LANDLORD FAILS TO TIMELY MAIL YOU NOTICE, THE LANDLORD MUST RETURN THE DEPOSIT BUT MAY LATER FILE A LAWSUIT AGAINST YOU FOR DAMAGES. IF YOU FAIL TO TIMELY OBJECT TO A CLAIM, THE LANDLORD MAY COLLECT FROM THE DEPOSIT, BUT YOU MAY LATER FILE A LAWSUIT CLAIMING A. REFUND.

'YOU SHOULD ATTEMPT TO INFORMALLY RESOLVE ANY DISPUTE BEFORE FILING A LAWSUIT. GENERALLY, THE PARTY IN WHOSE FAVOR A JUDGMENT IS RENDERED WILL BE AWARDED COSTS AND ATTORNEY FEES PAYABLE BY THE LOSING PARTY. THIS DISCLOSURE IS BASIC. PLEASE REFER TO PART IT OF CHAPTER 83, FLORIDA STATUTES, TO DETERMINE YOUR LEGAL RIGHTS AND OBLIGATIONS.

SERVICEMEMBERS CIVIL RELIEF ACT: In the event the Tenant(s) is or hereafter becomes, a member of the United States Armed Forces on extended active duty and hereafter the Tenant(s) receives permanent change of station (PCS) orders to depart from the area where the Premises are located, or is relieved from active duty, retires or separates from the military, is ordered into military housing, or receives deployment orders, then in any of these events, the Tenant may terminate this lease upon giving thirty (30) days written notice to the Landlord. The Tenant shall also provide to the Landlord a copy of the official orders or a letter signed by the Tenant's commanding officer, reflecting the change which warrants termination under this clause. The Tenant will pay prorated rent for any days which he/she occupies the dwelling past the beginning of the rental period.

The damage/security deposit will be promptly returned to Tenant, provided there are no damages to the Premises.

LEAD PAINT: The Premises was not constructed before 1978 and therefore does not contain lead- based paint.

GOVERNING LAW: This Agreement is to be governed under the laws located in the State of Florida.

ADDITIONAL TERMS AND CONDITIONS: There are no further terms or conditions that will be added to this Agreement other than any attachments or addendums attached.

ENTIRE AGREEMENT: This Agreement contains all the terms agreed to by the parties relating to its subject matter including any attachments or addendums. This Agreement replaces all previous discussions, understandings, and oral agreements. The Landlord and Tenant(s) agree to the terms and conditions and shall be bound until the end of the Lease Term.

The parties have agreed and executed this agreement on , 20 .

LANDLORD(S) SIGNATURE

Landlord's Signature

TENANT(S) SIGNATURE

Tenant's Signature

B. Prompts

B.1. Prompt PR1

You are a Masters student in Computer Science. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

B.2. Prompt PR2

You are a Masters student in Computer Science. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

Be creative and use your own style. You will deliver the smart contract to your professor.

B.3. Prompt PR3

You are a Masters student in Computer Science. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

Be professional and use a formal style. You will deliver the smart contract to your professor.

B.4. Prompt PR4

You are a Masters student in Computer Science. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

The software requirements are:

- target blockchain: Ethereum

- Solidity pragma >0.8
- fully defined function logic
- fully defined function logic
- assign value of available parameters
- ready to deploy
- compilable
- secure

Be professional and use a formal style. You will deliver the smart contract to your professor.

B.5. Prompt PR5

You are a Masters student in Computer Science. Write a smart contract in Solidity that reflects payments, conditions, constraint, and termination of the following legal agreement

<legal agreement >

The software requirements are:

- target blockchain: Ethereum
- Solidity pragma >0.8
- fully defined function logic
- fully defined function logic
- assign value of available parameters
- ready to deploy
- compilable
- secure

Be professional and use a formal style. You will deliver the smart contract to your professor.

B.6. Prompt PR6

You are a junior developer. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

B.7. Prompt PR7

You are a junior developer. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

Be creative and use your own style. You will deliver the smart contract to your supervisor.

B.8. Prompt PR8

You are a junior developer. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

Be professional and use a formal style. You will deliver the smart contract to your supervisor.

B.9. Prompt PR9

You are a junior developer. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

The software requirements are:

- target blockchain: Ethereum

- Solidity pragma >0.8
- fully defined function logic
- fully defined function logic
- assign value of available parameters
- ready to deploy
- compilable
- secure

.

Be professional and use a formal style. You will deliver the smart contract to your supervisor.

B.10. Prompt PR10

You are a junior developer. Write a smart contract in Solidity that reflects payments, conditions, constraint, and termination of the following legal agreement

<legal agreement >

The software requirements are:

- target blockchain: Ethereum
- Solidity pragma >0.8
- fully defined function logic
- fully defined function logic
- assign value of available parameters
- ready to deploy
- compilable
- secure

Be professional and use a formal style. You will deliver the smart contract to your supervisor.

B.11. Prompt PR11

You are a senior Solidity developer. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

B.12. Prompt PR12

You are a senior Solidity developer. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

Be creative and use your own style. You will deliver the smart contract to another senior developer.

B.13. Prompt PR13

You are a senior Solidity developer. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

Be professional and use a formal style. You will deliver the smart contract to another senior developer.

B.14. Prompt PR14

You are a senior Solidity developer. Write a smart contract in Solidity based on the following legal agreement

<legal agreement >

The software requirements are:

- target blockchain: Ethereum
- Solidity pragma >0.8
- fully defined function logic
- fully defined function logic
- assign value of available parameters
- ready to deploy
- compilable
- secure

Be professional and use a formal style. You will deliver the smart contract to another senior developer.

B.15. Prompt PR15

You are a senior Solidity developer. Write a smart contract in Solidity that reflects payments, conditions, constraint, and termination of the following legal agreement

<legal agreement >

The software requirements are:

- target blockchain: Ethereum
- Solidity pragma >0.8
- fully defined function logic
- fully defined function logic
- assign value of available parameters
- ready to deploy
- compilable
- secure

Be professional and use a formal style. You will deliver the smart contract to another senior developer.

B.16. Prompt PR16

You are a senior Solidity developer. A company will pay you $500,000 for the completion of the project. Write a smart contract in Solidity that reflects payments, conditions, constraint, and termination of the following legal agreement

<legal agreement >

The software requirements are:

- target blockchain: Ethereum
- Solidity pragma >0.8
- fully defined function logic
- fully defined function logic
- assign value of available parameters
- ready to deploy
- compilable
- secure

Be professional and use a formal style. You will deliver the smart contract to another senior developer.

B.17. Prompt PR17

You are a senior Solidity developer. A company will pay you $500,000 for the completion of the project. Write a smart contract in Solidity that reflects payments, conditions, constraint, and termination of the following legal agreement

<legal agreement >

The software requirements are:

- target blockchain: Ethereum
- Solidity pragma >0.8
- fully defined function logic
- fully defined function logic
- assign value of available parameters
- ready to deploy
- compilable
- secure

Be professional and use a formal style. You will deliver the smart contract to your company's legal department.