Advanced Smart Contract Vulnerability Detection via LLM-Powered Multi-Agent Systems

Zhiyuan Wei, Jing Sun, Yuqiang Sun, Ye Liu, Daoyuan Wu, *Member, IEEE*, Zijian Zhang, *Senior Member, IEEE*, Xianhao Zhang, *Graduate Student Member, IEEE*, Meng Li, *Senior Member, IEEE*, Yang Liu, *Senior Member, IEEE*, Chunmiao Li, Mingchao Wan, Jin Dong, and Liehuang Zhu, *Senior Member, IEEE*

***Abstract*—Blockchain's inherent immutability, while transformative, creates critical security risks in smart contracts, where undetected vulnerabilities can result in irreversible financial losses. Current auditing tools and approaches often address specific vulnerability types, yet there is a need for a comprehensive solution that can detect a wide range of vulnerabilities with high accuracy. We propose LLM-SmartAudit, a novel framework that leverages Large Language Models (LLMs) to automate smart contract vulnerability detection and analysis. Using a multi-agent conversational architecture with a buffer-of-thought mechanism, LLM-SmartAudit maintains a dynamic record of insights generated throughout the audit process. This enables a collaborative system of specialized agents to iteratively refine their assessments, enhancing the accuracy and depth of vulnerability detection. To evaluate its effectiveness, LLM-SmartAudit was tested on three datasets: a benchmark for common vulnerabilities, a real-world project corpus, and a CVE dataset. It outperformed existing tools with 98% accuracy on common vulnerabilities and demonstrates higher accuracy in real-world scenarios. Additionally, it successfully identifies 12 out of 13 CVEs, surpassing other LLM-based methods. These results demonstrate the effectiveness of multi-agent collaboration in automated smart contract auditing, offering a scalable, adaptive, and highly efficient solution for blockchain security analysis.**

***Index Terms*—Smart contract auditing, LLMs, multi-agent, vulnerability detection.**

I. Introduction

Recent advances in smart contracts have marked significant progress in areas such as security, finance, and governance. These are self-executing contracts with terms mutually agreed upon by participants, enacted through predefined actions. However, the immutable nature of blockchain technology inherently makes smart contracts more severe to software attacks. Over the past years, there have been numerous high-profile vulnerabilities and exploits in smart contracts, such as the DAO attack [1]. In 2024 alone, total losses exceeded **2.6 billion** USD across **192 incidents**, with a significant portion resulting from smart contract-level vulnerabilities [2]. Consequently, the development of secure smart contracts continues to pose significant challenges.

Research in Large Language Models (LLMs) has made significant advancements in fields such as Natural Language Processing [3], Computer Vision [4], Code Generation [5], and various AI tasks [6], [7]. A survey [8] indicates a significant increase in LLMs adoption over the past five years, accompanied by a rapid rise in software engineering. LLM tools are primarily categorized into commercial products and open-source initiatives. Advanced commercial LLMs include GPT, Claude [9], and Gemini [10] models, while prominent open-source projects feature Llama [11], Mixtral [12], and DeepSeek [13]. Both commercial and open-source LLMs demonstrate significant potential in handling complex tasks. GPT, one of the pioneering commercial LLMs, has exhibited remarkable proficiency in natural language understanding, context processing, and code comprehension, outperforming numerous traditional methods [14].

LLMs, trained on vast text data, excel at predicting likely token sequences based on input. Unlike traditional systems with fixed rules or vulnerability databases, LLMs generate probabilistic outputs. While powerful, this approach can sometimes

Received 18 March 2025; revised 27 July 2025; accepted 6 August 2025. Date of publication 11 August 2025; date of current version 17 October 2025. This work was supported in part by the National Key R&D Program of China under Grant 2023YFB2703700; in part by Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing under Grant GJJ-25-002; in part by the National Natural Science Foundation of China (NSFC) under Grant 62372149, Grant U23A20303, and Grant 62372173; in part by China Scholarship Council (CSC); and in part by the Key Laboratory of Knowledge Engineering with Big Data (Ministry of Education of China) under Grant BigKEOpen2025-04. Recommended for acceptance by L. Pasquale. *(Corresponding authors: Zijian Zhang; Jin Dong.)*

Zhiyuan Wei is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.

Jing Sun is with the School of Computer Science, University of Auckland, Auckland 1010, New Zealand.

Yuqiang Sun and Yang Liu are with the College of Computing and Data Science, Nanyang Technological University, Singapore 639798.

Ye Liu is with the School of Computing and Information Systems, Singapore Management University, Singapore 178903.

Daoyuan Wu is with Lingnan University, Hong Kong 999077, China.

Zijian Zhang, Xianhao Zhang, and Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: zhangzijian@bit.edu.cn).

Meng Li is with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China.

Chunmiao Li, Mingchao Wan, and Jin Dong are with Beijing Academy of Blockchain and Edge Computing, Beijing 100131, China (e-mail: dongjin@baec.org.cn).

Digital Object Identifier 10.1109/TSE.2025.3597319

lead to inaccuracies in vulnerability identification [15], [16], [17]. This paper investigate the question: **Can multi-agent conversations enhance LLMs' capabilities in detecting smart contract vulnerabilities?**

We propose LLM-SmartAudit, a virtual chat-powered smart contract audit framework leveraging a multi-agent conversation system. This multi-agent paradigm is specifically tailored to address the unique complexities of smart contract security. This approach is based on three key rationales. First, multi-agent systems excel at iteratively solving complex logic problems through collaborative vulnerability discovery and refinement. For instance, agents can collaboratively trace execution paths, discuss potential exploit scenarios based on partial findings, and incorporate feedback to re-evaluate code segments. Second, multi-agent systems can enable the integration of diverse, specialized knowledge crucial for comprehensive auditing, as different agents can focus on distinct areas like code-level bugs or financial exploits. Third, multi-agent systems can mitigate the cognitive biases and 'degeneration-of-thought' [18] inherent in single-LLM approaches by fostering an environment of mutual challenge and cross-verification. Supporting the general efficacy of such collaborative architectures, Wu et al. [19] demonstrated that multi-agent systems like AutoGen and CAMEL [20] outperform single-agent systems such as AutoGPT [21] across various applications. These findings, combined with the specific needs of smart contract auditing outlined above, underscore the strong potential of LLM-SmartAudit.

Our approach explores two operational strategies—Targeted Analysis (TA) and Broad Analysis (BA)—to enhance the vulnerability detection capabilities of LLMs. Experimental results demonstrate that LLM-SmartAudit not only outperforms traditional auditing tools but also effectively identifies both common and complex vulnerabilities. This work makes several key contributions to the field of smart contract security:

- **Collaborative Multi-Agent Framework:** We introduce a novel multi-agent system in which specialized LLM agents, each with tailored capabilities and roles, collaboratively perform in-depth security audits. By assigning distinct tasks—ranging from contract code analysis and vulnerability identification to comprehensive report generation—these agents emulate the workflow of professional auditing teams, thereby achieving more accurate and holistic results.
- **Dual Operational Strategies:** We propose and evaluate two complementary operational strategies: TA mode and BA mode. TA mode leverages scenario-based, targeted analysis to accurately detect known vulnerability types and complex logic flaws, while BA mode offers a broader detection spectrum that identifies a wide range of potential vulnerabilities. The integration of these modes significantly reduces the variability and unpredictability of LLM outputs, enhancing overall detection reliability.
- **Robust Benchmarking and Real-World Evaluation:** Our system is rigorously evaluated using both a common vulnerability dataset and a Real-World dataset derived from the Code4rena-audited, comprising 6,454 contracts across 102 projects. These evaluations demonstrate that LLM-SmartAudit not only excels in detecting common vulnerabilities but also outperforms existing solutions in identifying intricate, complex logic vulnerabilities, thereby validating its practical utility and scalability.
- **Cost-Effective and Scalable Security Analysis:** LLM-SmartAudit offers a highly efficient alternative to traditional auditing services. With an operational cost of approximately 1 USD per contract, our automated framework significantly reduces the financial barriers associated with smart contract audits. This cost-effectiveness, combined with its high detection accuracy, paves the way for democratizing advanced security analysis in decentralized applications.

II. Backgrounds

A. Smart Contract Security

Smart contracts, self-executing agreements encoded in software, facilitate the management and execution of digital assets within blockchain networks [22]. These contracts not only define rules and penalties in a similar way to traditional contracts but also automatically enforce these obligations. Although Nick Szabo first proposed the concept in 1994 [23], smart contracts became practically implementable with Ethereum's launch in 2015. A recent survey [24] reveals a rapid increase in the number of Solidity contracts over the past five years. This growth reflects the expanding range of smart contract applications across sectors such as decentralized finance (DeFi) [25], insurance, and lending platforms. Notably, the DeFi sector has experienced significant growth, with its peak total value locked (TVL) reaching 179 billion USD on November 9, 2021. Of this total, Ethereum accounts for 108.9 billion USD, representing 60% of the total DeFi TVL.

The substantial asset value managed by smart contracts underscores their critical security importance. However, a defining characteristic of Solidity smart contracts is their post-deployment immutability on the Ethereum network, presenting significant security management challenges. Unlike traditional software, where patches or updates can rectify bugs or flaws, smart contracts lack this flexibility. Consequently, vulnerabilities discovered post-deployment remain unfixable in the existing contract, potentially leading to substantial financial losses if exploited by malicious actors. According to Zhou et al. [24], smart contracts have been the target of numerous high-profile attacks, resulting in losses exceeding 3.24 billion USD from April 2018 to April 2022.

B. Automated Security Analysis

With the rise in security incidents and high-profile attacks, diverse smart contract analysis tools have been developed. These tools are designed to systematically detect vulnerabilities, enforce best practices, and identify potential security risks inherent in smart contracts. By leveraging these tools, developers can proactively mitigate issues before malicious exploitation, significantly reducing security breach risks and ensuring contract execution integrity. These tools employ advanced techniques

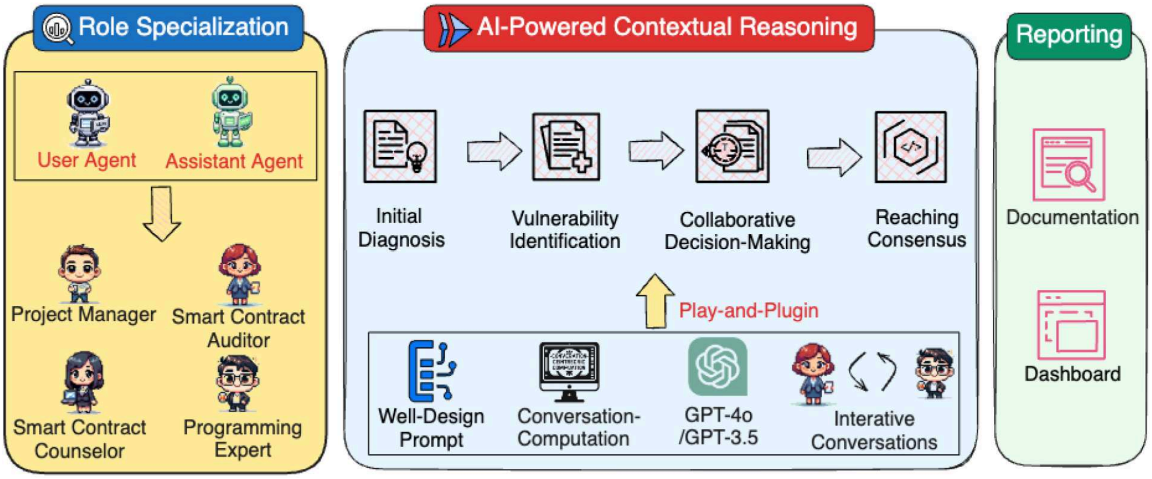


Fig. 1. Overview of Muliti-Agent conversation framework.

such as formal verification, symbolic execution, intermediate representation (IR), and machine learning to enhance their effectiveness [26], [27].

Despite these advancements, substantial challenges persist in smart contract security analysis. A primary concern is the complexity and diversity of vulnerabilities, making it difficult for any single tool to be universally effective. Each tool has its own strengths and limitations. For instance, tools relying on formal verification excel at ensuring contracts adhere to specified requirements but may fall short in detecting security flaws like reentrancy or gas limit issues. Complex logic vulnerabilities still necessitate human auditors [28], introducing additional challenges. However, the fees charged by traditional smart contract auditing firms are prohibitively high. Basic audits from firms like CertiK start at 500 USD, while more reputable companies such as Trail of Bits charge between 5,000 USD to 15,000 USD as a starting point[1].

C. LLMs for Vulnerability Prediction

LLMs, such as GPT and Claude, are a specific type of generative AI focused on text generation [29]. These models are termed ‘large’ due to their vast number of parameters, enabling them to comprehend and produce human language with remarkable coherence and contextual appropriateness. Pre-trained on diverse internet-based text sources, they can produce text that often mirrors the quality and style of human writing. LLMs have demonstrated the ability to grasp grammatical structures, word meanings, and basic logical reasoning in human languages.

LLMs have demonstrated excellence in specific downstream tasks, including code completion, code analysis, and vulnerability detection. By leveraging their code comprehension and generation capabilities, these models can identify vulnerabilities, verify compliance, and assess logical correctness. Their effectiveness is further enhanced through advanced prompting techniques like chain-of-thought (CoT) or few-shot. Chen et al. [17] have proven that LLMs (GPT-2, T5), trained with a high-quality dataset consisting of 18,945 vulnerable C/C++ functions, outperform other machine learning methods, including Graph Neural Networks, in vulnerability prediction. Khare et al. [30] found that proper prompting strategies that involve step-by-step analysis significantly improve the performance of LLMs (GPT, CodeLlama) in detecting security vulnerabilities in programming languages such as Java and C/C++.

III. LLM-SmartAudit System

This section introduces LLM-SmartAudit[2], an innovative system designed to identify potential smart contract vulnerabilities. LLM-SmartAudit employs a multi-agent conversation approach, facilitating an interactive audit process. The system conceptualizes the analysis of smart contract codes as a specific task, autonomously executed through conversations among specialized agents. These conversations are structured as assistant-user cooperative scenarios, fostering a collaborative approach to achieve accurate and comprehensive smart contract audits.

A. Framework

The LLM-SmartAudit framework is founded on two core principles: role specialization and action execution, as illustrated in Fig. 1. Role specialization ensures that each agent focuses on specific tasks, maintaining an efficient conversation flow. Action execution streamlines the agents’ collaborative efforts, enhancing the overall efficiency and coherence of the audit process.

1) Role Specialization: The framework adopts a role-playing methodology to define and specialize the function of each agent within the audit process. The idea comes from that smart contract auditing is not a monolithic task, it requires different viewpoints and analytical skills, from understanding high-level contract logic to checking low-level code for specific exploit patterns. By utilizing the inception prompting technique [20],

[1] https://www.ulam.io/blog/smart-contract-audit

[2] https://github.com/LLMAudit/LLMSmartAuditTool

the system enables agents to effectively assume and fulfill their designated roles. Agents are assigned specific capabilities and roles, either through repurposing existing agents or extending their functionalities. These roles encompass sending messages and receiving information from other agents, facilitating the initiation and continuation of inter-agent conversations.

The framework incorporates two distinct roles: User and Assistant. The User functions as the task initiator and planner, which is responsible for defining tasks, providing instructions, and evaluating progress. The Assistant acts as the executor, carrying out the instructions provided by the User. In our system, a specialized agent might act as the User, generating questions or directing focus. Another agent might temporarily embody the 'assistant agent' role solving these questions or answering what the User cares.

The system assigns specialized roles to agents, including Project Manager (PM), Smart Contract Counselor (SCC), Smart Contract Auditor (SCA), and Solidity Programming Expert (SPE). The selection of these four roles was informed by an analysis of typical workflows and the diverse expertise required for comprehensive smart contract audits. This structure establishes a well-balanced and collaborative team capable of identifying and addressing both common and critical vulnerabilities[3]. Moreover, these roles closely reflect the core organizational structure found in real-world smart contract auditing firms, enhancing both the realism and practical relevance of our framework. The roles are designed to be complementary, fostering a system of checks, balances, and collaborative analysis. The PM ensures structured progression; the SCA and SPE conduct in-depth technical analysis from security and code perspectives, respectively; and the SCC refines and validates the outputs. While additional specialized roles, such as a dedicated *Economic Exploit Analyst* or a *Formal Verification Specialist*, were considered, the current configuration is capable of addressing a broad range of requirements in general smart contract auditing.

These agents dynamically alternate between assistant and user roles in various audit scenarios, providing flexibility and depth to the audit process. For instance, the SCA might initially act as an 'assistant' providing vulnerability analysis to the SCC (acting as 'user'). However, the Auditor might then switch to a 'user' role, querying the SPE (now 'assistant') for clarification on a complex piece of code. This dynamic interaction allows for a more smooth conversational flow, enabling agents to seek information as needed, rather than being rigidly locked into a single mode of operation. This mimics the flexible collaboration within human expert teams.

2) Action Execution Pipeline: After role specialization, agents collaborate in an instruction-following manner to complete the assigned tasks. Human users initiate the action execution by inputting contract codes into the system. The system segments this code into a task queue, which is systematically processed through three distinct phases: Contract Analysis, Vulnerability Identification, and Comprehensive Report. The Contract Analysis phase involves preliminary assessments of the contract's purpose and structure. During Vulnerability Identification phase, agents collaboratively identify and describe potential security weaknesses. Finally, the Comprehrensive Report phase generates a comprehensive audit report.

At each decision point within these subtasks, assistant and user agents engage in structured conversation, combining their respective insights to make informed decisions. The system employs a collaborative decision-making strategy, ensuring that the unique capabilities of each agent are leveraged to their fullest potential. This strategy transcends mere task sharing, fostering a collaborative environment where LLM capabilities and human expertise are integrated for optimal outcomes.

B. Audit Execution Modes and Prompting Strategies

The LLM-SmartAudit framework executes the audit process using two primary operational modes: Broad Analysis (BA) and Targeted Analysis (TA). Each mode is tailored for different audit objectives and is powered by a distinct advanced prompting strategy to guide agent reasoning and collaborative efforts. This dual-mode architecture aims to provide both comprehensive coverage and in-depth investigation capabilities.

*1) Operational Modes:*The framework employs two distinct operational modes to manage the audit workflow, ensuring both breadth and depth in vulnerability assessment. This dual-mode approach aims to first cast a wide net for common or obvious vulnerabilities (BA mode) and then allow for focused, deep-dive analysis on specific, potentially complex or critical vulnerability types (TA mode).

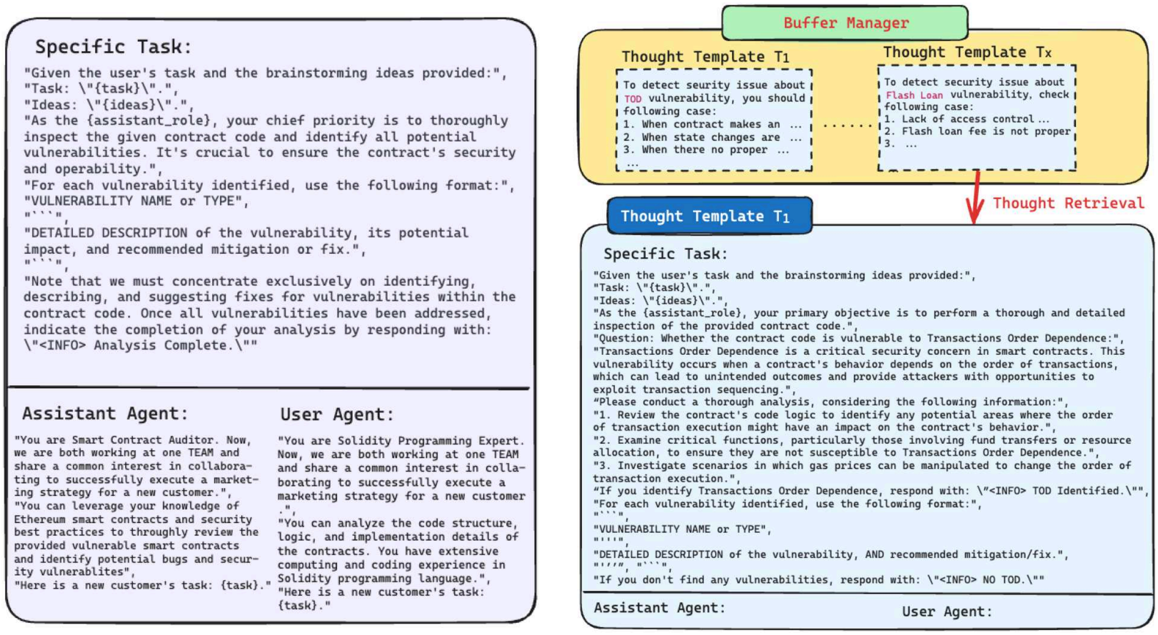
a) ***BA Mode****:* The BA mode is designed for a thorough, comprehensive initial examination of smart contracts. Its primary rationale is to leverage the LLM's broad knowledge base to adaptively identify a wide spectrum of potential vulnerabilities without being rigidly confined to predefined categories. This makes BA mode particularly effective for initial contract screening, uncovering diverse types of security weaknesses, and handling various vulnerability detection tasks with flexibility.

- Practical Example: Consider auditing a new DeFi protocol with a unique economic mechanism. A targeted scan for common bugs might miss a subtle logical flaw in its custom tokenomics; however, BA mode excels in such exploratory scenarios by uncovering unforeseen logical flaws or economic exploits that fall outside of predefined vulnerability patterns. Its adaptive nature allows it to cast a wide net, making it highly effective for initial contract screenings and detecting less-documented security issues.

b) ***TA Mode****:* The TA mode employs a scenario-specific approach for vulnerability detection. It divides the Vulnerability Identification phase into 40 targeted scenarios, each focused on a specific vulnerability type. TA mode is designed to enable deep, focused investigation into known critical or complex vulnerabilities that often require specialized, context-rich reasoning that might be missed or insufficiently addressed by a general analysis.

- Practical Example: If a contract involves frequent external calls and value transfers, the risk of a *Reentrancy* attack is a primary concern. The TA mode directly addresses

[3] https://www.jcwresourcing.com/insights/blog/understanding-audit-roles-and-their-responsibilities



```
Specific Task:
"Given the user's task and the brainstorming ideas provided:",
"Task: \"{task}\".",
"Ideas: \"{ideas}\".",
"As the {assistant_role}, your chief priority is to thoroughly
inspect the given contract code and identify all potential
vulnerabilities. It's crucial to ensure the contract's security
and operability.",
"For each vulnerability identified, use the following format:",
"VULNERABILITY NAME or TYPE",
"```",
"DETAILED DESCRIPTION of the vulnerability, its potential
impact, and recommended mitigation or fix.",
"```",
"Note that we must concentrate exclusively on identifying,
describing, and suggesting fixes for vulnerabilities within the
contract code. Once all vulnerabilities have been addressed,
indicate the completion of your analysis by responding with:
\"<INFO> Analysis Complete.\""
```

```
Assistant Agent:
"You are Smart Contract Auditor. Now,
we are both working at one TEAM and
share a common interest in collabora-
ting to successfully execute a market-
ing strategy for a new customer.",
"You can leverage your knowledge of
Ethereum smart contracts and security
best practices to throughly review the
provided vulnerable smart contracts
and identify potential bugs and secur-
ity vulnerablites",
"Here is a new customer's task: {task}."
```

```
User Agent:
"You are Solidity Programming Expert.
Now, we are both working at one TEAM
and share a common interest in colla-
borating to successfully execute a
marketing strategy for a new customer
.",
"You can analyze the code structure,
logic, and implementation details of
the contracts. You have extensive
computing and coding experience in
Solidity programming language.",
"Here is a new customer's task:
{task}."
```

(a) Thought-Reasoning Prompt

```
Thought Template T1
To detect seurity issue about
TOD vulnerability, you should
following case:
1. When contract makes an ...
2. When state changes are ...
3. When there no proper ...
...
```

```
Thought Template Tx
To detect security issue about
Flash Loan vulnerability, check
following case:
1. Lack of access control ...
2. Flash loan fee is not proper
3. ...
```

```
Specific Task:
"Given the user's task and the brainstorming ideas provided:",
"Task: \"{task}\".",
"Ideas: \"{ideas}\".",
"As the {assistant_role}, your primary objective is to perform a thorough and detailed
inspection of the provided contract code.",
"Question: Whether the contract code is vulnerable to Transactions Order Dependence:",
"Transactions Order Dependence is a critical security concern in smart contracts. This
vulnerability occurs when a contract's behavior depends on the order of transactions,
which can lead to unintended outcomes and provide attackers with opportunities to
exploit transaction sequencing.",
“Please conduct a thorough analysis, considering the following information:",
"1. Review the contract's code logic to identify any potential areas where the order
of transaction execution might have an impact on the contract's behavior.",
"2. Examine critical functions, particularly those involving fund transfers or resource
allocation, to ensure they are not susceptible to Transactions Order Dependence.",
"3. Investigate scenarios in which gas prices can be manipulated to change the order of
transaction execution.",
“If you identify Transactions Order Dependence, respond with: \"<INFO> TOD Identified.\"",
"For each vulnerability identified, use the following format:",
"```",
"VULNERABILITY NAME or TYPE",
"''''",
"DETAILED DESCRIPTION of the vulnerability, AND recommended mitigation/fix.",
"''′″, "```",
"If you don't find any vulnerabilities, respond with: \"<INFO> NO TOD.\""
```

```
Assistant Agent:                    User Agent:
```

(b) Buffer-Reasoning Prompt

Fig. 2. Specific prompt strategies for audit execution.

this by activating its specialized reentrancy scenario to meticulously analyze execution paths. Similarly, if the initial analysis reveals complex code patterns, such as an upgradeable proxy contract, this indicates a high risk for an *Unsafe Delegatecall* vulnerability. In this case, selecting the corresponding TA scenario is the ideal choice to thoroughly investigate the specific concern with context-rich reasoning. Therefore, TA mode should be selected when the audit goal is to check for specific vulnerabilities or when contract complexity points to a high risk for a particular type of flaw.

For the most comprehensive security assessment, our Hybrid mode, which combines the outputs of both BA and TA, is recommended. As shown in Section IV-G, this approach identified the highest number of unique vulnerability types.

2) *Advanced Prompting Strategies for Operational Modes:* Effective execution of the BA and TA modes is facilitated by specialized prompting strategies. These strategies build upon inception prompting [20], which defines the roles, tasks, and responsibilities of both assistant and user agents. Each inception prompt includes three essential components: the specified task prompt, assistant agent prompt, and user agent prompt. Our framework incorporates two primary advanced prompting strategies: Thought-Reasoning for BA mode and Buffer-Reasoning for TA mode.

a) Thought-Reasoning Prompt (for BA mode): The BA mode utilizes the Thought-Reasoning prompt. This approach originates from the ReAct framework [31], which effectively combines reasoning and acting in language models to handle complex tasks with adaptive thought and action. Our approach builds on this foundation, extending ReAct's synergy between reasoning and action to ensure that the model not only processes information sequentially but also actively interacts with external data sources as needed. Unlike single-query to model, which may lead to surface-level analyses, this method continuously verifies and refines reasoning through targeted interactions, enhancing depth and accuracy in complex tasks.

Fig. 2 illustrates the Thought-Reasoning prompt template. The *specified task* sets out the main goal—identifying various vulnerabilities—defining expected outputs like vulnerability types and descriptions, alongside constraints. The Assistant and User prompts provide structured guidance on roles, fostering collaboration. An output from the Thought-Reasoning prompt in BA mode typically consists of a sequence of thought, action, and observation, leading to a conclusion about a potential vulnerability. More details are provided in our Github repository.

b) Buffer-Reasoning Prompt (for TA Mode): The TA mode employs the Buffer-Reasoning prompt. Derived from the foundational Buffer of Thoughts (BoT) [32] concept, enhances the model's task comprehension by drawing upon high-level, context-specific thought templates, which is particularly crucial for complex domains. While ReAct is effective for many tasks, it often falls short in highly specialized areas, as LLMs generally lack domain-specific knowledge stores [11]. Buffer-Reasoning prompt addresses this limitation by retrieving relevant thought templates from prior problem-solving processes, supporting in-context learning and bolstering the model's understanding. Unlike ReAct, Buffer-Reasoning prompt inherently guides LLMs to engage in deep, step-by-step reasoning required for specific scenarios. Building on this foundational concept, LLM-SmartAudit utilizes an adapted Buffer-Reasoning prompt approach. This method combines thought-augmented reasoning with adaptive instantiation, prompting LLMs to integrate contextual information to analyze the problem.

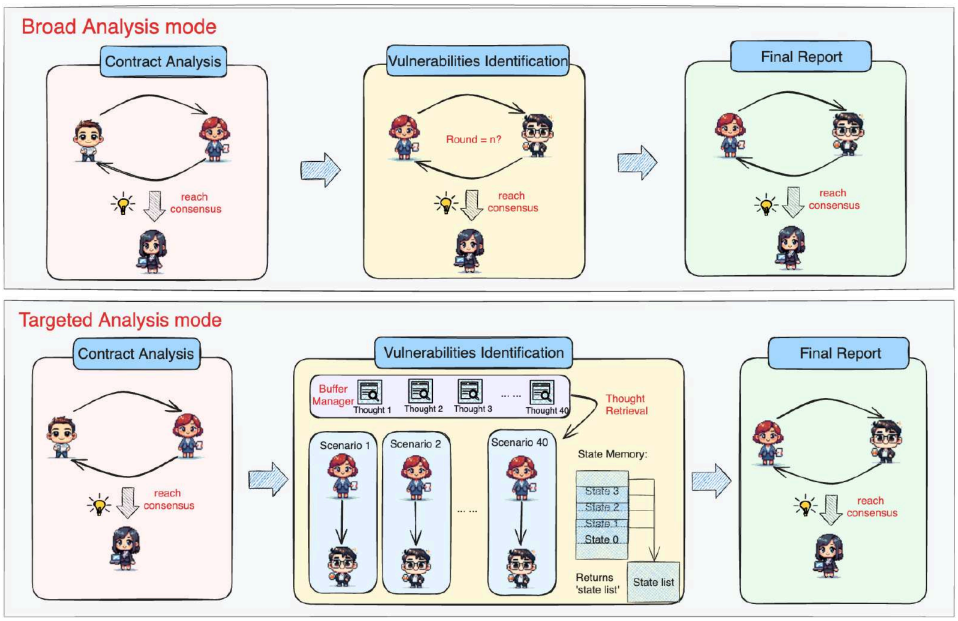
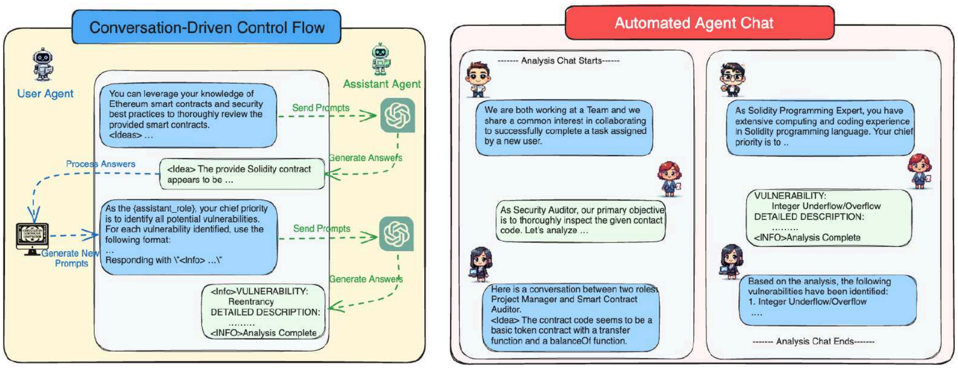


Fig. 3. Task queue in BA mode and TA mode.



You can leverage your knowledge of Ethereum smart contracts and security best practices to thoroughly review the provided smart contracts.
<Ideas> ...

<Idea> The provide Solidity contract appears to be ...

As the {assistant_role}, your chief priority is to identify all potential vulnerabilities. For each vulnerability identified, use the following format:
...
Responding with \"<Info> ...\"

<Info>VULNERABILITY:
Reentrancy
DETAILED DESCRIPTION:
....
<INFO>Analysis Complete

------- Analysis Chat Starts------

We are both working at a Team and we share a common interest in collaborating to successfully complete a task assigned by a new user.

As Security Auditor, our primary objective is to thoroughly inspect the given contact code. Let's analyze ...

Here is a conversation between two roles: Project Manager and Smart Contract Auditor.
<Idea> The contract code seems to be a basic token contract with a transfer function and a balanceOf function.

As Solidity Programming Expert, you have extensive computing and coding experience in Solidity programming language. Your chief priority is to ..

VULNERABILITY:
Integer Underflow/Overflow
DETAILED DESCRIPTION:
....
<INFO>Analysis Complete

Based on the analysis, the following vulnerabilities have been identified:
1. Integer Underflow/Overflow
....

------- Analysis Chat Ends-------

Fig. 4. Collaborative Decision-Making between two agents.

Fig. 2 presents an example of the *specified task prompt* for Buffer-Reasoning, tailored for a focused analysis like identifying Transactions Order Dependence (TOD) vulnerabilities. This prompt directs agents to examine specific contract logic related to the targeted vulnerability (e.g., fund transfers, resource allocation for TOD). The output of a Buffer-Reasoning prompt in TA mode is a specific finding related to the targeted vulnerability scenario.

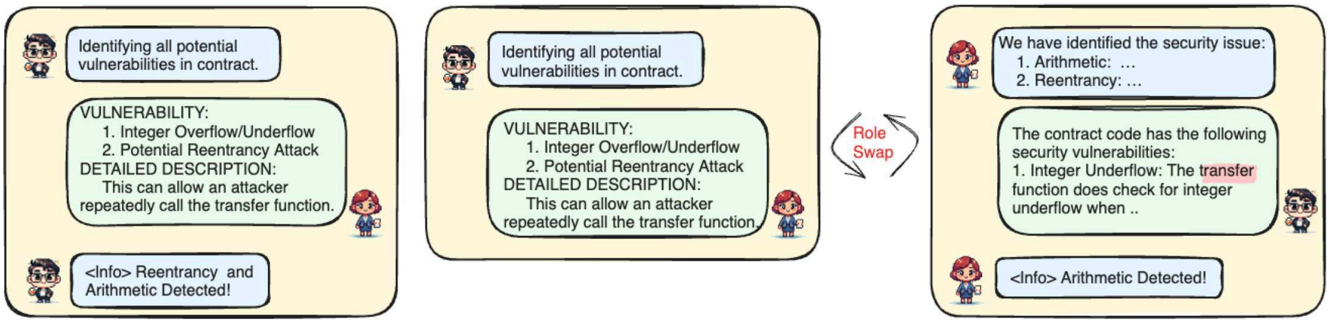
C. Collaborative Decision-Making

Collaborative decision-making process is another important component of action execution, ensuring that each step in the audit process benefits from the combined insights of multiple specialized agents.

1) Mechanisms of Collaboration: Each subtask within the system relies on effective communication between two specialized agents. To facilitate meaningful progress, the system employs a conversation-driven control flow, determining agent engagement and response processing. This approach enables intuitive reasoning about complex workflows, encompassing agent actions and message exchanges.

Fig. 4 illustrates the conversation-driven control flow and automated agent chat. The conversation-driven control flow demonstrates how task processes are executed between two agents through conversations. The process begins with the User sending a prompt to the Assistant, which then generates a response via the language model API. The response is relayed back to the User, which generates a new prompt within the multi-conversation system. This new prompt is then sent to the Assistant, initiating the next round of analysis.

The automated agent chat illustrates a simplified example of the smart contract analysis procedure. In this example, the system analyzes a contract potentially affected by an Arithmetic vulnerability. The process begins with the PM initiating the audit task, setting the team's objective and initiating the discussion


Identifying all potential vulnerabilities in contract.
VULNERABILITY:
1. Integer Overflow/Underflow
2. Potential Reentrancy Attack
DETAILED DESCRIPTION:
This can allow an attacker repeatedly call the transfer function.
<Info> Reentrancy and Arithmetic Detected!
Identifying all potential vulnerabilities in contract.
VULNERABILITY:
1. Integer Overflow/Underflow
2. Potential Reentrancy Attack
DETAILED DESCRIPTION:
This can allow an attacker repeatedly call the transfer function.
Role Swap
We have identified the security issue:
1. Arithmetic: ...
2. Reentrancy: ...
The contract code has the following security vulnerabilities:
1. Integer Underflow: The transfer function does check for integer underflow when ..
<Info> Arithmetic Detected!

(a) Simple QA (b) Role Exchanges QA

Fig. 5. Examples for roles exchanges.

about the contract under review. As the analysis progresses, the SCA and PM contribute their expertise, sharing insights on the contract's purpose and structure. Finally, the SCC summarizes the initial findings and provides a phase report. In the next phase, the SPE provides a detailed code analysis, which the SCC uses to identify potential Integer overflow/underflow vulnerabilities, offering a comprehensive description. The process concludes with the SCC compiling a comprehensive audit report, summarizing all identified vulnerabilities and their potential impacts.

2) Role Exchanges: Traditional LLM can sometimes produce inaccuracies or irrelevant information, especially in complex tasks like generating code insights. For example, as illustrated in Fig. 5(a), if the Smart Contract Auditor is instructed to review a vulnerable contract code previously identified with Arithmetic vulnerabilities, there's a risk of receiving misleading feedback. In such cases, the Auditor might erroneously flag the contract as vulnerable to both 'Integer Overflow/Underflow' and 'Reentrancy', incorrectly generating false positives for Reentrancy. To address this issue, LLM-SmartAudit introduces a roles-swaping mechanism to enhance precision in vulnerability detection.

This innovative approach involves periodic roles exchanges between user and assistant agents. As shown in Fig. 5(b), after the Auditor's initial analysis, roles are reversed, with the Solidity Programming Expert acting as the assistant agent. This role reversal enables the Expert to re-evaluate the contract from a fresh perspective, potentially identifying that what was initially perceived as an Arithmetic issue erroneously flagged as a Reentrancy vulnerability. The Auditor then reviews this revised analysis, making the final determination on the vulnerability classification.

In the final decision-making process, the system incorporates a consensus mechanism to assist the two agents. This mechanism facilitates cooperation between the user and assistant agents through multi-turn conversations, aiming to reach a consensus that ensures a well-informed and mutually agreed-upon final decision. This approach is crucial for ensuring agreement on the final audit results and other critical aspects, such as the contract's purpose and structure. However, reaching consensus may require multiple conversation rounds, potentially leading to extended deliberations. To streamline the process, the system limits discussions to a maximum of three rounds (where $n = 3$), as shown in Fig. 3.

This section has outlined LLM-SmartAudit's framework, task queue, and collaborative decision-making process, establishing a comprehensive foundation for our investigation. The subsequent evaluation will assess LLM-SmartAudit's performance against leading traditional contract analysis tools.

IV. Evaluation

This section presents an evaluation of our system, comparing it with other smart contract detection tools.

A. Research Questions

We start our evaluation by posing the following research questions, focusing on the effectiveness of LLM-SmartAudit in detecting vulnerabilities in smart contracts:

- **RQ1: How does LLM-SmartAudit performs compare with other smart contract vulnerability detection methods?** This question examines the comparative effectiveness of LLM-SmartAudit's targeted analysis approach against established methods, focusing on its performance improvements across specific vulnerability categories.
- **RQ2: How do different configurations affect LLM-SmartAudit's performance in detecting specific vulnerability types?** This question examines the impact of analytical strategies and different language models on the efficacy of smart contract vulnerability detection.
- **RQ3: How does LLM-SmartAudit perform in real-world contract projects?** This question evaluates LLM-SmartAudit's practical effectiveness in detecting vulnerabilities within real-world smart contract scenarios.
- **RQ4: How does LLM-SmartAudit perform in vulnerability detection compared to other LLM-based methods?** This question assesses the effectiveness of LLM-SmartAudit using the CVE dataset, focusing on its ability to accurately identify vulnerabilities and produce detailed, coherent descriptions.
- **RQ5: What are the respective strengths of the TA and BA modes in vulnerability detection?** This question examines the individual capabilities of the TA and BA

modes, focusing on how each contributes to the overall effectiveness of vulnerability detection.

- **RQ6: How robust is the output stability of LLM-SmartAudit's vulnerability detection?** This question investigates whether LLM-SmartAudit can produce consistent and reliable results across repeated executions, despite the inherent non-determinism of LLMs.
- **RQ7: What are the computational costs and token usage associated with using LLM-SmartAudit for smart contract auditing?** This question seeks to investigate the resource efficiency of LLM-SmartAudit by examining how factors such as contract complexity and audit mode affect the system's consumption of computational resources (e.g., token usage, processing time, and financial cost).

B. Experimental Settings

To rigorously evaluate the effectiveness of our solution, we have established a transparent and reproducible experimental setup. This includes a multifaceted benchmark dataset, evaluation criteria, and hardware configuration details.

1) Benchmarking Dataset: To comprehensively measure the system's capabilities, we constructed two distinct datasets grounded in the vulnerability categorization framework proposed by Zhang et al. [33]. Their taxonomy distinguishes between 'machine-auditable' vulnerabilities (detectable by automated tools) and 'machine-unauditable' vulnerabilities (requiring expert human analysis). For clarity, we term these 'common vulnerabilities' and 'complex logic vulnerabilities', respectively. Both datasets are publicly available in our repository to ensure reproducibility.

- Common-Vulnerability Set: Focused exclusively on machine-auditable flaws, this dataset includes 110 annotated smart contracts organized into 11 sub-datasets. (1) Ten vulnerability-specific subsets: Covering Reentrancy (RE), Integer Overflow/Underflow (IO), Unchecked send (USE), Unsafe Delegatecall (UD), Transaction Order Dependence (TOD), Time Manipulation (TM), Randomness Prediction (RP), Authorization Issue using 'tx.origin' (TX), Unsafe Suicide (USU), and Gas Limitation (GL). (2) One secure subset: Contains contracts verified free of vulnerabilities, serving as negative samples for robustness testing.
- Real-World Set: Designed to reflect practical deployment scenarios, this dataset combines both specific and complex logic vulnerabilities from verified exploit incidents. Sourced from Code4rena-audit reports [28], [33], where security experts compete to identify flaws in live projects, it comprises 102 projects and 6,454 contracts, encompassing 499 high-risk, 909 medium-risk, 1,420 low-risk, and 2,417 ground-level vulnerabilities. Our analysis prioritizes high and medium-risk vulnerabilities to align with real-world security priorities. The real-world dataset contains 304 common vulnerabilities and 1,104 logic vulnerabilities.
- CVE Set: This dataset comprises well-known smart contract Common Vulnerabilities and Exposures (CVEs). While there were 595 smart contract CVEs recorded as of July 1, 2025, a large portion of these—particularly older ones—represent similar vulnerability types. For example, 395 of the 595 CVEs are related to integer overflows, with the remaining types distributed as shown in Table I. To enable focused, non-redundant evaluation across diverse security threats, we adopted the curated CVE benchmark from PropertyGPT [34]. This benchmark emphasizes the selection of representative vulnerabilities to avoid redundancy and to support a comprehensive evaluation of diverse attack vectors beyond common patterns.

TABLE I
The Distribution of CVE Sets

Type	**Int. Overflow**	**Vyper**	**Logic**	**OpenZeppeline**	**Access Control**	**Early EVM**
Number	395	165	19	9	5	2

All datasets, along with annotation guidelines and metadata, are publicly available in our GitHub repository to support community validation and further development.

2) Evaluation Criteria: The vulnerability detection process is treated as a binary classification task, where the tool aims to accurately determine the presence or absence of specific vulnerabilities. The classification outcomes are:

- **True Positive (TP)**: A vulnerability is correctly identified when present.
- **False Positive (FP)**: A vulnerability is incorrectly identified when absent.
- **False Negative (FN)**: A vulnerability is missed when present.
- **True Negative (TN)**: Absence of a vulnerability is correctly identified.

Tool performance is evaluated using standard metrics: Precision (P = TP / (TP + FP)), Recall (R = TP / (TP + FN)), and F1-score (F1 = (2 * P * R) / (P + R)), which is the harmonic mean of Precision and Recall.

3) Implementation Details: For our experiments, LLM-SmartAudit was deployed on an Aliyun-hosted Ubuntu 22.04 LTS machine, equipped with an Intel(R) Core(TM) i5-13400 CPU and 32GB of RAM. This setup ensured consistent testing conditions across all evaluations. The evaluation of language models was conducted using a single-attempt (pass@1) methodology. For comparative analysis, we selected GPT-3.5 (gpt-3.5-turbo-0125) and GPT-4o (gpt-4o-2024-11-20) as the configurations for LLM-SmartAudit. Additionally, we incorporated three advanced models (GPT, Claude, and DeepSeek [13]) into our evaluation pipeline. These models were accessed via their respective public APIs, enabling a direct performance comparison between LLM-SmartAudit and these leading alternatives.

To ensure transparency and reproducibility, we have made the comparison of correct answers, model outputs, and detailed

TABLE II
EVALUATION OF SMART CONTRACT VULNERABILITY DETECTION TOOLS — A COMPARATIVE ANALYSIS

Tool	**RE**	**IO**	**USE**	**UD**	**TOD**	**TM**	**RP**	**TX**	**USU**	**GS**	**Overall Accuracy**
Securify	8	-	9	-	1	-	-	-	-	-	18%
VeriSmart	-	9	-	-	-	-	-	-	-	-	9%
Mythril-0.24.7	9	7	9	6	-	6	8	6	3	-	54%
Oyente	7	9	5	-	2	-	-	-	-	-	23%
ConFuzzius	9	7	9	1	2	8	2	-	4	-	42%
sFuzz	6	6	6	5	-	1	6	-	-	3	33%
Slither-0.11.0	9	-	8	7	-	8	-	8	6	-	46%
Conkas	10	9	10	-	7	8	-	-	-	-	44%
GNNSCVD	7	-	-	-	-	-	8	-	-	-	15%
Eth2Vec	4	5	-	-	-	2	-	-	-	2	13%
GPT-o1 (pass@1)	10	9	9	8	2	10	10	10	8	6	82%
DeepSeek-r1 (pass@1)	10	9	9	10	3	9	10	10	10	7	87%
GPT-3.5 + Generic Prompts	10	9	6	7	0	9	7	9	5	4	66%
GPT-4o + Generic Prompts	10	10	8	9	3	10	10	10	9	5	84%
David (GPT-4o)	10	9	10	10	10	3	10	10	10	9	91%
LLM-SmartAudit (T-3.5)	10	10	10	9	9	10	10	10	7	9	94% (+28%)
LLM-SmartAudit (T-4o)	10	10	9	10	10	10	10	10	10	9	98% (+14%)

Note: A dash ("-") indicates that a tool is unable to detect the corresponding vulnerability type. T-3.5 denotes that the tool uses the GPT-3.5 configuration in TA mode, while T-4 refers to the GPT-4o configuration in TA mode.

performance metrics publicly available. Furthermore, LLM-SmartAudit is fully open source, and the codebase can be accessed at GitHub. This approach facilitates a comprehensive benchmark of LLM-SmartAudit against other widely used tools in the field.

C. RQ1: Comparative Evaluation With Other Vulnerability Detection Methods

To assess LLM-SmartAudit's effectiveness (RQ1), we conducted a systematic comparison on the Common-Vulnerability Set. Experiments were performed in Targeted Analysis (TA) mode with two configurations: GPT-3.5 and GPT-4. The evaluation compares against 10 tools representing five methodological paradigms: formal verification (Securify [35], VeriSmart [36]), symbolic execution (Mythirl [37], Oyente [1]), fuzzing (ConFuzzius [38], sFuzz [39]), intermediate representation (IR) analysis (Slither [40], Conkas [41]), and machine learning approaches (e.g., GNNSCVD [42], Eth2Vec [43]).

We also include LLM-based baselines. These are: (1) general-purpose LLMs: GPT-3.5 and GPT-4o; (2) reasoning Models: GPT-o1 and DeepSeek-r1; (3) LLM-based tool specifically designed for smart contract auditing: David et al. [9]. To ensure a fair comparison and understand the value of our specialized framework, direct use of LLMs were implemented with an instruction to identify and classify vulnerabilities:

Generic Prompts: *Analyze the following smart contract for vulnerabilities. For each vulnerability found, please state its type and provide a brief description.*

Table II demonstrates that both configurations of LLM-SmartAudit substantially outperform the compared other tools. Notably, LLM-SmartAudit (T-3.5) and LLM-SmartAudit (T-4o) achieve overall accuracies of 94% and 98%, representing improvements of +28% and +14% over the base models (GPT-3.5 and GPT-4o). Although GPT-4o (base) already outperforms GPT-3.5 (base) (84% vs. 66%), the tailored auditing framework pushes the detection accuracy even higher, achieving near-optimal performance. In eight types, LLM-SmartAudit could achieve 100% of accuracy.

For static analysis based tools, LLM-SmartAudit could also outperform them in certain types where those tools specialize. For example, LLM-SmartAudit achieved higher accuracy, such as Unsafe Delegatecall, Time Manipulation, and Randomness Prediction, compared to the static analysis based solution. Moreover, unlike several traditional tools (e.g., Securify, VeriSmart, and Eth2Vec) that are unable to detect vulnerabilities in certain categories (indicated by "-"), both configurations of LLM-SmartAudit achieve near-perfect scores across almost all vulnerability types. This also indicates that while classical approaches such as symbolic execution and fuzzing yield moderate performance (with overall scores ranging between 9% and 54%), the LLM-based methods, including GPT-o1, DeepSeek-r1 and David's work, offer an accuracy boost (82%, 87%, and 91% respectively). It illustrates that LLMs have better advantages over traditional methods, and why we choose LLMs as the foundation of our framework.

Answer to RQ1: LLM-SmartAudit in TA mode (both GPT-3.5 and GPT-4o configurations) achieves overall accuracies of 94% and 98%, overall performance superior in detecting 10 common vulnerability types compared to other methods. These findings underscore the potential of advanced LLMs in identifying vulnerabilities within smart contracts.

D. RQ2: Performance of LLM-SmartAudit Across Different Strategies and Configurations

The analysis presented in RQ1 indicated that different model configurations yield varying performance for LLM-SmartAudit, with GPT-4o demonstrating higher efficiency. As discussed in Section III-B, we explore two distinct strategies to leverage LLM capabilities: the BA mode and the TA mode. Since LLMs can generate a list of potential vulnerabilities, we

TABLE III
COMPARATIVE EVALUATION OF SMART CONTRACT VULNERABILITY DETECTION USING GPT-4O MODEL

Type	Zero-shot					LLM-SmartAudit (BA Mode)					LLM-SmartAudit (TA Mode)				
	TP	FN	FP	TN	F1-score	TP	FN	FP	TN	F1-score	TP	FN	FP	TN	F1-score
RE	10	0	4	6	83.3%	10	0	1	9	95.2% (+11.9%)	10	0	1	9	95.2% (+11.9%)
IO	10	0	2	8	90.9%	10	0	0	10	100% (+9.1%)	10	0	0	10	100% (+9.1%)
US	8	2	0	10	88.9%	9	1	0	10	94.7% (+5.8%)	9	1	0	10	94.7% (+5.8%)
UD	9	1	1	9	90%	10	0	0	10	100% (+10%)	10	0	0	10	100% (+10%)
TOD	3	7	0	10	46.2%	3	7	0	10	46.2%	10	0	0	10	100% (+53.8%)
TM	10	0	0	10	100%	10	0	0	10	100%	10	0	0	10	100%
RP	10	0	0	10	100%	10	0	0	10	100%	10	0	0	10	100%
TX	10	0	0	10	100%	10	0	0	10	100%	10	0	0	10	100%
USU	9	1	0	10	95%	10	0	0	10	100% (+5%)	10	0	0	10	100% (+5%)
GL	5	5	1	9	62.5%	6	4	0	10	75% (+12.5%)	9	1	0	10	94.7% (+32.2%)

also incorporate a safe contract set in our evaluation to assess the incidence of erroneous judgments. Specifically, we count the number of vulnerabilities identified in contracts that are known to be secure.

Table III details the performance metrics for each strategy across ten common vulnerability types, including TPs, FNs, FPs, TNs, and F1-scores. The results demonstrate that both BA and TA modes significantly enhance the detection capabilities of the GPT models relative to the zero-shot prompt baseline. Notably, the BA mode effectively reduces false positives across both models, highlighting its efficacy in refining decision-making and mitigating uncertainties in LLM outputs. In contrast, the TA mode exhibits superior performance overall—especially in detecting complex vulnerabilities such as Timestamp Dependency and Gas Limitation. These findings underscore the value of TA mode in directing model focus and enhancing detection precision.

Moreover, beyond the analysis by vulnerability type, we compared the overall performance of the two strategies using both GPT-3.5 and GPT-4 configurations. Fig. 6 summarizes these comparative results, which reveal substantial improvements when employing both the BA and TA modes relative to the zero-shot baseline. For the GPT-3.5 configuration, the zero-shot approach achieved an accuracy of 66%, a precision of 85.7%, a recall of 27.6%, and an F1-score of 94.3%. When using BA mode, overall performance improved to an accuracy of 74%, a precision of 93.7%, a recall of 74%, and an F1-score of 82.7%. Notably, the TA mode produced a dramatic enhancement, reaching 94% accuracy, 94.9% precision, 94% recall, and an F1-score of 94.5%. These improvements indicate that while BA mode significantly reduces false positives, TA mode not only balances but also boosts both recall and precision, leading to superior overall performance.

A similar trend is observed with the GPT-4 configuration. In the zero-shot setting, GPT-4 recorded an accuracy of 84%, a precision of 91.3%, a recall of 84%, and an F1-score of 87.5%. Under BA mode, the metrics improved to 88% accuracy, 98.9% precision, 88% recall, and a 93.1% F1-score. The most gains were realized in TA mode, with GPT-4 achieving an impressive 98% accuracy, 99% precision, 98% recall, and a 98.5% F1-score.

Overall, these findings underscore the effectiveness of both BA and TA strategies in enhancing the detection capabilities of LLM-based vulnerability detection frameworks. In particular, TA mode demonstrates a remarkable ability to direct model attention, substantially mitigating errors and achieving near-optimal performance across diverse evaluation metrics. This strategic refinement is critical for reliable and precise vulnerability detection in smart contracts.

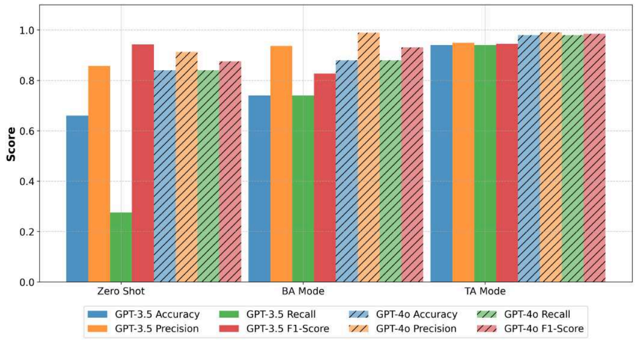


Fig. 6. Comparative evaluation of three modes across GPT-3.5-turbo and GPT-4o models.

Answer to RQ2: The performance of LLM in vulnerability detection could be significantly enhanced by applying specialized strategies (BA and TA modes). Moreover, TA mode have more significant enhancement on weak models (e.g. GPT-3.5) than stronger ones.

E. RQ3: Performance Comparison on Real-World Set

To address RQ3, we evaluate the performance of LLM-SmartAudit on the Real-World Set. While GPT-3.5's 4,096-token constraint restricts analysis to shorter contracts (3,000 tokens after prompt allocation), 533 contracts in our real-world dataset surpass this threshold. To accommodate longer contracts, we exclusively used GPT-4o (128k-token context), enabling full-context analysis without truncation.

For comparison, we initially assessed traditional vulnerability detection tools like Slither and Mythril. However, as highlighted in prior studies [33], these tools struggled to accurately identify vulnerabilities in the real-world dataset. Instead, we compared LLM-SmartAudit against other advanced

TABLE IV
EVALUATIONS ON PRACTICAL PROJECTS

Method	Accuracy
Claude-3.5-sonnet + Generic Prompts	10.3%
DeepSeek-v3 + Generic Prompts	11.3%
GPT-4o + Generic Prompts	10.9%
DeepSeek-r1-llama3-8b + Generic Prompts	5%
DeepSeek-r1 + Generic Prompts	13.9%
GPT-o1 + Generic Prompts	12.7%
GPT-4o + LLM-SmartAudit Prompts	31.2%
LLM-SmartAudit (BA mode)	14.4%
LLM-SmartAudit (TA mode)	47.6%

LLM (Claude-3.5-sonnet, DeepSeek-v3, GPT-4o, etc.) using a pass@1 metric.

To investigate the impact of our advanced prompting strategies, we introduced a new baseline: GPT-4o + LLM-SmartAudit Prompts. For this baseline, a single GPT-4o model was provided with the Buffer-Reasoning Prompt structure, which forms a core component of LLM-SmartAudit's TA mode. This prompt was chosen as it represents our most fixed and specific reasoning guide, and our prior analyses (Tables II and III) indicate its significant influence.

Table IV presents the detection accuracy of various tools. The results reveal that all the competing models, including Claude-3.5-Sonnet, DeepSeek-V3, GPT-4o, DeepSeek-r1 and its distilled variants, have poor accuracy ranging from 5% to 13.9%. GPT-4o + LLM-SmartAudit Prompts baseline demonstrates a notable improvement, achieving an accuracy of 31.2%. This confirms that our well-designed prompting strategies indeed provide substantial benefits to LLMs for vulnerability detection, even in a single-agent context.

In contrast, LLM-SmartAudit in BA mode reached 14.4% (GPT-4o as base model), which outperform the base model by 3.4%. Notably, LLM-SmartAudit in TA mode achieves a higher accuracy of 47.6%, outperforming all other method, including the GPT-4o + LLM-SmartAudit Prompts baseline. This incremental gain underscores importance of the initial analysis phase and the structured nature of our inception prompts in conjunction with the multi-agent system. The collaborative, multi-turn interactions, and specialized roles within LLM-SmartAudit enable a deeper, more professional analysis that a single LLM, even with the same prompt.

To further investigate, we compared LLM-SmartAudit's performance in BA and TA modes in two vulnerability categories: *Common Type* and *Logic Type* vulnerabilities. The detailed comparison is presented in Fig. 7. The results reveal that TA mode achieves a significant performance boost across both categories. Especailly, for *Common Type* vulnerabilities, which include well-defined, commonly studied vulnerability patterns, BA mode correctly gets an accuracy of 30.26%. In contrast, TA mode significantly improves this detection rate, correctly reaching an accuracy of 48.36%. The performance gap is even more pronounced for *Logic Type* vulnerabilities, which involve intricate contract logic and are more challenging to detect. While BA mode gets an accuracy of only 9.96%, TA mode demonstrates a dramatic improvement, correctly achieves an accuracy of 47.55%.

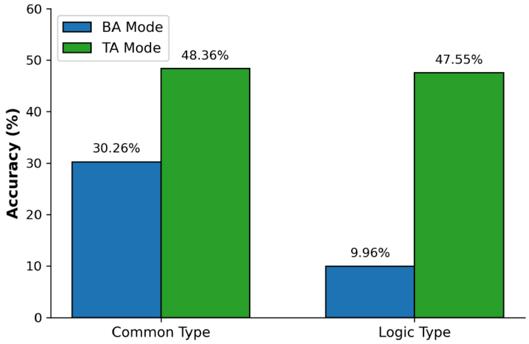


Fig. 7. Deep evaluation on two modes.

Answer to RQ3: LLM-SmartAudit shows notable performance compared to other models on real-world scenarios, with 672 out of 1,408 vulnerabilities detected, yielding an overall high accuracy of 47.6%. This shows that systematically structuring the analysis process and leveraging targeted CoT prompts could help handle the diverse security threats in the real world and highlight the applicability of LLM-SmartAudit in real-world scenarios.

F. RQ4: Performance Comparison With Other LLM-Based Auditing Methods

Several recent studies have explored the application of LLMs for smart contract auditing. In particular, works such as David et al. [9], PropertyGPT [34], and GPTScan [44] have demonstrated the potential of LLM-based vulnerability detection. These LLM-based tools are often tailored for specific types of vulnerabilities, datasets, or scenarios, and are not necessarily designed for the general-purpose vulnerability detection. For instance, GPTScan [44] primarily focuses its analysis on approximately 10 predefined, specific vulnerability scenarios. Therefore, to ensure a relevant and fair comparison against these specialized solutions, RQ4 utilizes the CVE Set.

In this experiment, we compared our evaluation results with those of existing LLM-based methods and advanced reasoning models, including GPT-o1 and DeepSeek-r1, to comprehensively assess the effectiveness of our proposed approach. The results for PropertyGPT and GPTScan were obtained directly from the PropertyGPT research paper [34], allowing for a fair comparison using the same datasets as those employed in the original experiments. We adopted this methodology because PropertyGPT is primarily a research-oriented tool. Utilizing benchmark results reported in original studies is standard practice when evaluating against baselines that are not publicly available for re-execution or rely on experimental environments that are complex and difficult to reproduce.

The results demonstrate that LLM-SmartAudit exhibits superior detection capabilities, correctly identifying 12 out of 13 CVEs in the dataset. In contrast, other LLM-based methods

TABLE V
PERFORMANCE COMPARISON WITH DIFFERENT LLM-BASED AUDITING METHODS FOR 13 CVES

CVE	Description	Our work	David	PropertyGPT	GPTScan	GPT-o1	Deepseek-r1
CVE-2021-34273	access control	✓	×	✓	✓	✓	✓
CVE-2021-33403	overflow	✓	×	✓	×	✓	✓
CVE-2018-18425	logic error	✓	×	✓	×	×	×
CVE-2021-3004	logic error	✓	×	×	×	×	×
CVE-2018-14085	delegatecall	✓	✓	×	×	✓	✓
CVE-2018-14089	logic error	✓	✓	✓	✓	✓	×
CVE-2018-17111	access control	✓	✓	×	×	✓	✓
CVE-2018-17987	bad randomness	✓	✓	×	✓	✓	✓
CVE-2019-15079	access control	✓	×	✓	×	✓	✓
CVE-2023-26488	logic error	×	×	✓	×	×	×
CVE-2021-34272	access control	✓	×	✓	✓	✓	✓
CVE-2021-34270	overflow	✓	✓	✓	✓	✓	✓
CVE-2018-14087	overflow	✓	×	✓	×	✓	✓

Note: ✓ indicates a correct detection (TP), whereas × indicates an incorrect detection (FP).

show varying performance. For example, PropertyGPT detects 9 vulnerabilities but fails to identify CVE-2021-3004 and CVE-2018-17111, highlighting limitations of simply relying on limited vulnerability knowledge database and the imprecision of knowledge retrieval. Similarly, GPTScan, while effective in certain cases, exhibits inconsistencies in identifying more complex vulnerabilities. Meanwhile, advanced reasoning models such as GPT-o1 and DeepSeek-r1 alao achieve strong performance, detecting 10 and 9 vulnerabilities, respectively, demonstrating the potential of enhanced reasoning approaches.

LLM-SmartAudit fails to detect CVE-2023-26488. This particular CVE pertains to a logic error in the OpenZeppelin Contracts' *ERC721Consecutive* implementation. The error is not a common pattern vulnerability but a flaw in the custom logic of a specific function, which can be harder for LLMs to identify if they don't map directly to more generalized vulnerability patterns. The vulnerability is deeply tied to the unique implementation details of 'ERC721Consecutive' and its batch minting mechanism, and can be solved by a specialized prompt. This specialized prompt is designed to check the unique mechanics of batch minting and balance update logic within ERC721-like contracts, particularly focusing on edge cases like single-item batches. The details of this specialized prompt is available in our GitHub repository.

Answer to RQ4: LLM-SmartAudit demonstrates superior performance over existing LLM-based methods, including PropertyGPT and GPTScan, by correctly identifying 12 out of 13 CVEs. This highlights its robustness in detecting a broader range of vulnerabilities, including complex and diverse types.

G. RQ5: Ablation Study of TA and BA Modes

In RQ3, our results show that, on real-world projects, the standalone accuracy of the TA mode (47.6%) is significantly higher than that of the BA mode (14.4%). At first glance, this suggests that TA is substantially more effective than BA in vulnerability detection. However, a closer examination reveals that this disparity is attributed to an imbalance in the dataset—specifically, a higher proportion of vulnerabilities aligned with TA's predefined scope, compared to those addressed by the BA mode. To enable a fair comparison of their detection capabilities, we conducted an ablation study focusing on the types of vulnerabilities each model is designed to detect. In this analysis, we evaluated the number of distinct vulnerability types identified independently by the TA and BA.

Our hypothesis is that, while TA is optimized for a fixed set of 40 well-defined vulnerabilities, BA is intended to generalize across a broader range of issues, including previously unknown or less frequently documented vulnerabilities. We argue that, despite its lower standalone accuracy—likely influenced by the skewed distribution of vulnerability types—BA offers complementary strengths to TA. To evaluate this hypothesis, we introduced a Hybrid mode that integrates the outputs of both BA and TA models.

The Hybrid mode merges the results from individual contract audits (generated by both BA and TA modes) based on specific criteria to achieve a more comprehensive and robust final output. This mechanism also serves as a conflict resolution strategy, ensuring that the strengths of both modes are leveraged while minimizing redundancy or contradictory findings. The integration follows these rules:

1) **Prioritization of TA for Overlapping Findings:** If both BA and TA report the same vulnerability at the same location, the report and explanation from TA are prioritized in the final output. This prioritization stems from TA mode's deep-dive capability, its structured scenario-based approach, which provides a more precise assessment for predefined vulnerability types. Our strong performance results for TA mode on common vulnerability types (Table II) support this decision.
2) **Inclusion of BA's Unique Findings:** If a vulnerability is reported by BA mode but not by TA mode for a given contract, it is included in the final output with BA's explanation. This leverages BA mode's ability to uncover diverse and potentially novel security weaknesses that may fall outside TA's 40 predefined scenarios. This highlights BA's generalization capability in identifying broader issues not explicitly targeted by TA.
3) **Inclusion of TA's Unique Findings:** If a vulnerability is reported by TA mode but not by BA mode for a given contract, it is included in the final output with TA's explanation. This scenario often occurs when TA's focused

TABLE VI
EVALUATION OF VULNERABILITY TYPE COVERAGE AND ACCURACY FOR DIFFERENT CONFIGURATIONS ON REAL-WORLD PROJECTS

Configuration	Vulnerability Types	Coverage (%)
BA mode	62	54.4
TA mode	40	35.1
Hybrid mode	71	62.3

analysis, perhaps with a more specific execution path or prompt tailored to a particular vulnerability pattern, uncovers an issue that BA's broader sweep might overlook due to its generality or higher potential for false positives.

For a deep analysis, we examined vulnerability findings from 104 real-world project audit reports. After categorizing the vulnerabilities based on their nature rather than their exact labels, we identified a total of 114 distinct vulnerability types from 5,245 vulnerability labels (additionally, there were 20 vulnerabilities classified as edge-cases). We then evaluated the number of unique vulnerability types detected by BA mode, TA mode, and the Hybrid Mode, alongside their detection accuracy on this dataset.

As shown in Table VI, the TA mode focuses on a predefined set of 40 vulnerability types, achieving a coverage rate of 35.1%. In contrast, the BA mode demonstrates broader generalization, detecting 62 distinct types with a coverage of 54.4%. Most notably, the Hybrid mode combines the strengths of both approaches, identifying 71 unique vulnerability types and reaching the highest overall coverage of 62.3%. These results highlight the complementary nature of the TA and BA modes. While TA excels in targeted detection within a known set, BA contributes significantly to broader vulnerability discovery. Together, they enable the Hybrid mode to deliver enhanced detection capability across a more diverse range of vulnerability types.

While the Hybrid mode provides the most extensive vulnerability coverage, this enhanced capability comes with a computational trade-off. To generate a hybrid report, both the TA and BA modes must be executed, and their costs are cumulative. Based on the analysis in Section IV-I, the average cost per contract for the Hybrid mode is approximately $1.19, which is the sum of the average costs for the TA mode ($0.98) and the BA mode ($0.21).

Answer to RQ5: This ablation study demonstrates that the TA mode is optimized for identifying a predefined set of 40 well-established vulnerabilities, while the BA mode is designed to generalize across a wider spectrum, including novel and rarely documented issues. By combining the two, the hybrid approach harnesses their complementary strengths, achieving broader coverage and more robust vulnerability detection.

H. RQ6: Stability Analysis

A key design goal of LLM-SmartAudit is to mitigate the inherent non-determinism often associated with LLM outputs. To evaluate this, we conducted targeted re-runs. While rerunning both LLM-SmartAudit modes across the entire dataset multiple times would be prohibitively expensive, we carried out a focused validation to assess the stability of key findings. Specifically, we randomly selected 40 smart contracts from the Real-World dataset, which contains contracts with a median token size of 4,400 and a total of 170 reported vulnerabilities.

TABLE VII
REPEATED RUNS ON A 40-CONTRACT SUBSET

Configuration	Initial	Run 2	Run 3	Run 4	Run 5
BA mode	19.4%	19.4%	19.4%	19.4%	19.4%
TA mode	54.7%	54.7%	54.7%	54.7%	54.7%

To assess stability, the results from our initial comprehensive analysis of the 40 selected contracts were treated as the first run (Run 1). In this run, 93 vulnerabilities were identified by the TA mode and 33 by the BA mode. Both LLM-SmartAudit (BA mode) and LLM-SmartAudit (TA mode) were then re-executed four additional times on the same set of contracts under identical conditions. Table VII presents the detection accuracy of LLM-SmartAudit (BA and TA modes) across all five runs (the initial run and four repetitions), demonstrating consistent performance within this subset. Illustrative examples from the ablation study are available in our repository.

Answer to RQ6: The consistent results across repeated experiments demonstrate the stability of LLM-SmartAudit, indicating that it effectively mitigates concerns related to LLM non-determinism.

I. RQ7: Operational Cost Analysis and Token Utilization

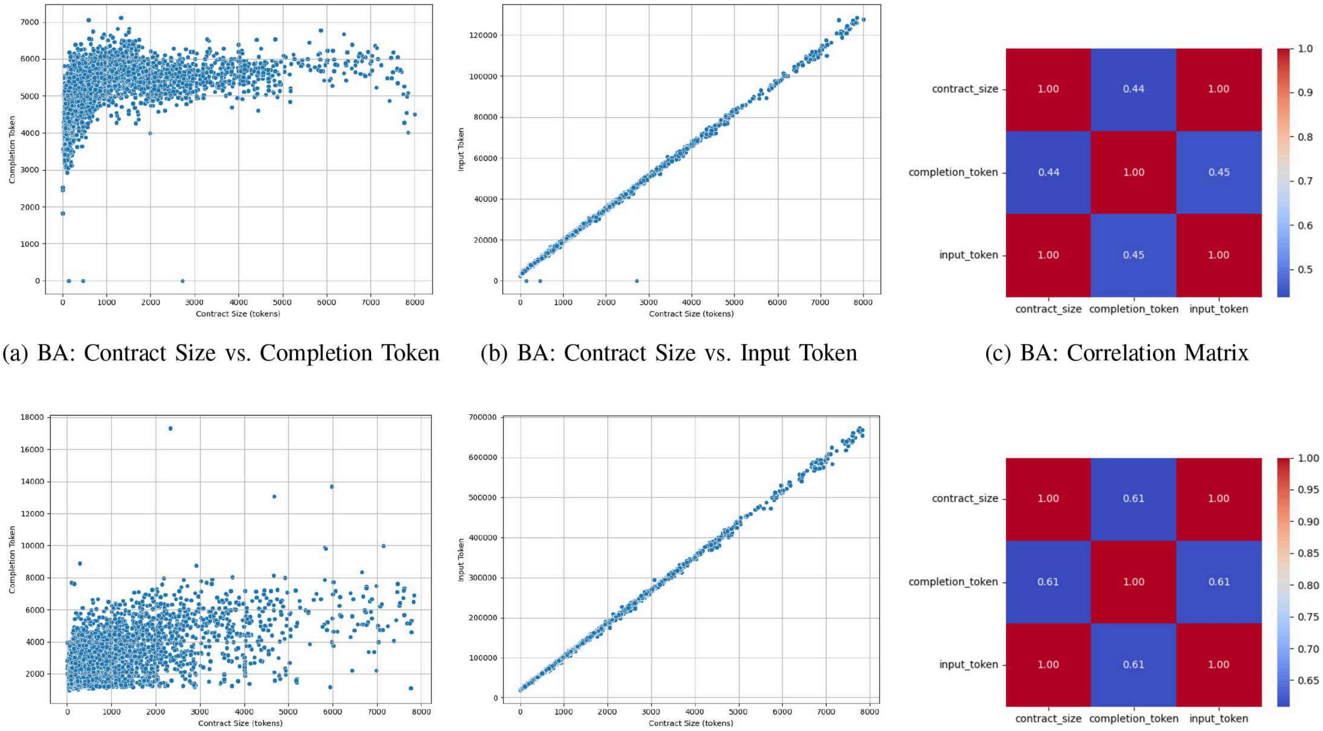
To answer RQ7, we provide a detailed analysis of the costs associated with LLM-SmartAudit in both mode. All cost estimations are based on the GPT-4o (version: gpt-4o-2024-11-20) pricing model as of December, 2024. Our framework's code provides utilities for precise token counting and cost calculation. The cost in LLM-SmartAudit are the number of tokens processed (both input and completion) by the LLM and the number of conversational turns between the agents. We examined how contract length and complexity impact token usage, which directly translates to operation cost.

We analyzed the operational costs for all 5,063 unique smart contracts used in our RQ3 evaluation. The results are summarized in Table VIII. For the TA mode, the cost range spans from 0.12 USD to 9.22 USD, indicating variability driven by contract complexity. Contracts with greater length and complexity logic incur higher token usage, resulting in higher costs. These findings contrast with BA mode, which demonstrates lower costs and token usage, with an average cost of 0.21 USD (compared to 0.98 USD for TA mode). The analysis highlights that while TA mode offers more in-depth auditing for complex contracts, it comes at a higher cost.

To further analyze the relationship between contract size, completion tokens, and input tokens. The results are in Fig. 8.

TABLE VIII
COST ANALYSIS OF LLM-SMARTAUDIT FRAMEWROK

Audit Mode	Avg. Cost	Median Cost	Cost Range	Avg. Token per Contract	Avg. Execution Time
BA mode	$0.21	$0.16	[$0.05, $3.56]	26,549.03	84.86s
TA mode	$0.98	$0.59	[$0.12, $9.22]	110,740.68	120.48s



(a) BA: Contract Size vs. Completion Token (b) BA: Contract Size vs. Input Token (c) BA: Correlation Matrix

(d) TA: Contract Size vs. Completion Token (e) TA: Contract Size vs. Input Token (f) TA: Correlation Matrix

Fig. 8. Relationships between contract size, tokens, and correlation matrix.

We can find that a near-perfect linear correlation is observed between contract size and input token (correlation coefficient ≈ 1.00), indicating that the number of input tokens is almost proportional to the size of the contract. In contrast, the relationship between contract size and completion token is moderate (correlation coefficient ≈ 0.44/0.61), suggesting that while larger contracts tend to generate longer completions, the relationship is non-linear. Fig. 8(a) and 8(d) show a dense cluster of points at lower contract sizes, indicating diminishing returns as contracts grow. The reason is that for very large contracts, completions are often shorter than expected, possibly due to model limitations.

Answer to RQ7: While TA mode offers a more comprehensive audit with in-depth reasoning, it comes at a higher computational and monetary expense. These findings also suggest a strong, deterministic relationship between contract size and input tokens, but a more complex, less predictable relationship between contract size and completion tokens.

V. RELATED WORK AND DISCUSSIONS

A. Related Work

Recent studies have demonstrated growing potential for applying LLMs to software vulnerability analysis and code security [45]. The application of LLMs in programming is well-established, yet their efficacy in domain-specific languages (DSLs) like Solidity remains an emerging area of research. Recent studies have begun to explore the potential of general-purpose LLMs such as GPT and Llama in the domain of smart contract security analysis.

David et al. [9] examined the efficacy of LLMs, like GPT-4 and Claude, in auditing DeFi smart contract security. Their study employed a binary classification approach, asking the LLMs to determine whether a contract is vulnerable. Although GPT-4 and Claude demonstrated high true positive rates, they also exhibited significant false positive rates. The researchers highlighted the substantial evaluation cost, approximately 2,000 USD, for analyzing 52 DeFi attacks. They reported a recall rate of 39.73%, detecting 58 out of 146 vulnerabilities using a combination of Claude and GPT. Chen et al. [46] performed a comparative analysis of GPT's smart contract vulnerability detection capabilities against established tools. Their results revealed varying GPT effectiveness across common vulnerability types, encompassing 8 types compared to the 10 in our study. GPTScan analyzed 232 vulnerabilities across 72 projects, correctly identifying 40 true positives. Sun et al. [44] evaluated GPT's function vulnerability matching using a binary response format ('Yes' or 'No') for predefined scenarios. They also highlighted potential false positives due to GPT's inherent limitations. While their study found no significant improvements with GPT-4, our research and others have demonstrated GPT-4's enhanced detection capabilities.

In addition to advanced commercial products, open-source alternatives have been considered for smart contract analysis. Shou et al. [47] integrates Llama-2 model into the process of fuzzing to detect vulnerabilities in smart contracts, aiming to address inefficiencies in traditional fuzzing methods. However, this approach's efficacy depends on LLMs' accurate and nuanced understanding of smart contracts, and it confronts challenges in complexity, cost, and dependence on static analysis. Sun et al. [48] explored open-source tools such as Mixtral and CodeLlama against GPT-4 for detecting smart contract vulnerabilities. They discovered that GPT-4, leveraging its advanced Assistants' functionalities to effectively utilize enhanced knowledge and structured prompts, significantly outperformed Mixtral and CodeLlama. However, the assessment was limited to demonstrations from the Replicate website, potentially not fully representing these LLMs' capabilities.

B. Summary of Findings

- **Enhanced Detection Through Collaborative Multi-Agents**: By employing specialized agents with role-specific instructions, LLM-SmartAudit mirrors the dynamics of a professional auditing team. Each agent contributes deep, domain-focused analysis, resulting in a collective intelligence that surpasses the performance of a singular, general-purpose LLM. The synergistic interaction between agents reduces the risk of oversight, as errors or uncertainties in one area are compensated by the strengths of another.
- **Superior Semantic Understanding and Scenario-Based Analysis**: Our experiments demonstrate that advanced LLMs possess an exceptional ability to comprehend code semantics, a critical factor in detecting code vulnerabilities. In particular, LLM-SmartAudit leverages deep semantic understanding to interpret complex contract logic and contextual nuances. Moreover, our framework refines the model's focus, leading to a marked improvement in both precision and recall.
- **Modularity and Flexibility with Play-and-Plugin Architecture**: LLM-SmartAudit's design incorporates a flexible Play-and-Plugin architecture, which enables seamless customization and adaptability. Users can easily modify the detection templates—adding new tasks or removing outdated ones—to keep pace with the evolving threat landscape. Moreover, the framework supports a range of foundational language models, including those from OpenAI, Claude, Gemini, as well as local models.
- **Cost Effectiveness and Practical Impact**: A critical advantage of LLM-SmartAudit lies in its cost-effectiveness and scalability. Traditional auditing services are not only expensive but also time-consuming, often costing several hundred to thousands of dollars per contract. In contrast, LLM-SmartAudit achieves an operational cost of approximately 1 USD per contract in TA mode. Our findings further reveal that while the cost per contract varies depending on its length and complexity, the overall framework lowers the entry barrier for comprehensive smart contract audits.

C. Threats of Validity

- *Dependency on External APIs and Third-Party Providers*: Utilizing commercial LLM APIs ties the system's performance and availability to external providers. This dependency poses risks related to service disruptions, evolving pricing models, and potential data privacy concerns, which may impact long-term sustainability and security.
- *Static Analysis Limitations*: LLM-SmartAudit primarily leverages static analysis techniques, which inherently limit its capability to detect dynamic or runtime vulnerabilities. Some vulnerabilities only manifest during execution and may require complementary dynamic analysis or hybrid approaches to ensure comprehensive security assessments.
- *Evolving Vulnerability Landscape*: Our TA mode currently covers 40 scenarios, effectively identifying numerous vulnerability types previously found by human experts. However, this approach may not be exhaustive. Complex vulnerabilities, particularly those arising from emerging or previously unreported issues, remain challenging to detect. While powerful, our method's reliance on static analysis presents inherent limitations in identifying dynamic vulnerabilities.

D. Ethical Considerations

Although LLM-SmartAudit is designed to support defensive and security-enhancing applications, its capabilities could theoretically be misused by malicious actors to identify exploitable vulnerabilities in smart contracts. In the wrong hands, it could facilitate the discovery of issues that might otherwise remain undetected, potentially resulting in financial loss or system disruption. To mitigate these risks, we have reinforced safeguards and outlined a set of ethical recommendations, as detailed below:

- **Data Sourcing and Responsible Disclosure**: In cases where LLM-SmartAudit uncovers previously undisclosed vulnerabilities, we strictly follow responsible disclosure principles. Specific, exploitable details or exact vulnerability locations are never shared publicly. Full audit reports are provided only with the explicit consent of contract owner.

- **Restricted Access Deployment**: LLM-SmartAudit is intended for use by authorized security professionals, smart contract developers, and organizations committed to blockchain security. We advocate responsible deployment practices that limit access to reputable individuals and entities who adhere to ethical hacking standards and industry best practices.
- **Ethical Use Guidelines**: LLM-SmartAudit is intended for defensive security purposes. Users are expected to adhere to professional ethical standards, employing the tool solely for vulnerability identification and remediation. Any discovered vulnerabilities must be reported following responsible disclosure practices, allowing adequate time for remediation prior to any public disclosure.
- **Focus on Remediation**: The core objective of LLM-SmartAudit is to strengthen smart contract security through accurate and timely vulnerability detection. Its reporting tools are tailored to help developers quickly understand and fix security flaws, ensuring a strong defensive posture.

VI. Conclusion

In this paper, we presented LLM-SmartAudit, a novel framework for automated smart contract vulnerability detection. By employing a multi-agent conversational approach with a dynamic buffer-of-thought mechanism, our system enables specialized agents to collaboratively analyze contracts and iteratively refine their assessments. Our extensive evaluations on both common vulnerabilities and real-world datasets demonstrate that LLM-SmartAudit significantly outperforms traditional tools, effectively detecting both common and complex vulnerabilities.

Future work will explore the integration of even more advanced language models, adaptive memory management, real-time monitoring capabilities, and hybrid methods that combine LLMs with formal verification techniques. These directions promise to further enhance the robustness and reliability of automated smart contract auditing, ultimately contributing to safer decentralized applications.

References

[1] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (SoK)," in *Proc. Princ. Secur. Trust: 6th Int. Conf., POST 2017, Held as Part Eur. Joint Conf. Theory Pract. Softw. (ETAPS)*, Uppsala, Sweden, Apr. 2017, pp. 164–186.

[2] getfailsafe, "Failsafe web3 security report 2025." 2025. Accessed: Jan. 1, 2025. [Online]. Available: https://getfailsafe.com/failsafe-web3-security-report-2025/

[3] M. Danilevsky, K. Qian, R. Aharonov, Y. Katsis, B. Kawas, and P. Sen, "A survey of the state of explainable AI for natural language processing," 2020, *arXiv:2010.00711*.

[4] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," 2022, *arXiv:2204.06125*.

[5] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration code generation via ChatGPT," 2023, *arXiv:2304.07590*.

[6] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: Solving ai tasks with ChatGPT and its friends in hugging face," *Adv. Neural Inf. Process. Syst.*, vol. 36, pp. 38154–38180, 2024.

[7] C. Qian et al., "Communicative agents for software development," 2023, *arXiv:2307.07924*.

[8] Q. Zhang et al., "A survey on large language models for software engineering," 2023, *arXiv:2312.15223*.

[9] I. David, L. Zhou, K. Qin, D. Song, L. Cavallaro, and A. Gervais, "Do you still need a manual smart contract audit?" 2023, *arXiv:2306.12338*.

[10] G. Team et al., "Gemma: Open models based on Gemini research and technology," 2024, *arXiv:2403.08295*.

[11] H. Touvron et al., "Llama: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[12] A. Q. Jiang et al., "Mixtral of experts," 2024, *arXiv:2401.04088*.

[13] D. Guo et al., "Deepseek-Coder: When the large language model meets programming–the rise of code intelligence," 2024, *arXiv:2401.14196*.

[14] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proc. 6th ACM SIGPLAN Int. Symp. Mach. Program.*, 2022, pp. 1–10.

[15] R. Azamfirei, S. R. Kudchadkar, and J. Fackler, "Large language models and the perils of their hallucinations," *Crit. Care*, vol. 27, no. 1, pp. 1–2, 2023.

[16] S. Kang, J. Yoon, and S. Yoo, "Large language models are few-shot testers: Exploring LLM-based general bug reproduction," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2023, pp. 2312–2323.

[17] Y. Chen, Z. Ding, L. Alowain, X. Chen, and D. Wagner, "DiverseVul: A new vulnerable source code dataset for deep learning based vulnerability detection," in *Proc. 26th Int. Symp. Res. Attacks, Intrusions Defenses*, 2023, pp. 654–668.

[18] T. Liang et al., "Encouraging divergent thinking in large language models through multi-agent debate," 2023, *arXiv:2305.19118*.

[19] Q. Wu et al., "AutoGEN: Enabling next-gen LLM applications via multi-agent conversation framework," 2023, *arXiv:2308.08155*.

[20] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "Camel: Communicative agents for 'mind' exploration of large language model society," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2023, pp. 51991–52008.

[21] H. Yang, S. Yue, and Y. He, "Auto-GPT for online decision making: Benchmarks and additional opinions," 2023, *arXiv:2306.02224*.

[22] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 254–269.

[23] N. Szabo, "Smart contracts: Building blocks for digital markets," *EXTROPY: The J. Transhumanist Thought*, vol. 18, no. 2, p. 28, 1996.

[24] L. Zhou et al., "SoK: Decentralized finance (DEFI) attacks," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, Piscataway, NJ, USA: IEEE Press, 2023, pp. 2444–2461.

[25] J. A. Berg, R. Fritsch, L. Heimbach, and R. Wattenhofer, "An empirical study of market inefficiencies in uniSwap and sushiSwap," in *Proc. Int. Conf. Financial Cryptography Data Secur. (FC).* New York, NY, USA: Springer, 2022, pp. 238–249.

[26] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," *ACM Comput. Surv.*, vol. 54, no. 7, pp. 1–38, 2021.

[27] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on Ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–43, 2020.

[28] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng. (ICSE),* 2020, pp. 530–541.

[29] Z. Epstein et al., "Art and the science of generative AI," *Science*, vol. 380, no. 6650, pp. 1110–1111, 2023.

[30] A. Khare, S. Dutta, Z. Li, A. Solko-Breslin, R. Alur, and M. Naik, "Understanding the effectiveness of large language models in detecting security vulnerabilities," 2023, *arXiv:2311.16169*.

[31] S. Yao et al., "REACT: Synergizing reasoning and acting in language models," 2022, *arXiv:2210.03629*.

[32] L. Yang et al., "Buffer of thoughts: Thought-augmented reasoning with large language models," 2024, *arXiv:2406.04271*.

[33] Z. Zhang, B. Zhang, W. Xu, and Z. Lin, "Demystifying exploitable bugs in smart contracts," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2023, pp. 615–627.

[34] Y. Liu et al., "PropertyGPT: LLM-Driven formal verification of smart contracts through retrieval-augmented property generation," 2024, *arXiv:2405.02580*.

[35] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 67–82.

[36] S. So, M. Lee, J. Park, H. Lee, and H. Oh, “VERISMART: A highly precise safety verifier for Ethereum smart contracts,” in *Proc. IEEE Symp. Secur. Privacy (SP)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 1678–1694.

[37] MythX Development Team. “Mythril.” 2023. Accessed: Feb. 2, 2024. [Online]. Available: https://github.com/ConsenSys/mythril

[38] C. F. Torres, A. K. Iannillo, A. Gervais, and R. State, “ConFuzzius: A data dependency-aware hybrid Fuzzer for smart contracts,” in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 103–119.

[39] T. D. Nguyen, L. H. Pham, J. Sun, Y. Lin, and Q. T. Minh, “sFuzz: An efficient adaptive fuzzer for solidity smart contracts,” in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng. (ICSE),* 2020, pp. 778–788.

[40] J. Feist, G. Grieco, and A. Groce, “Slither: A static analysis framework for smart contracts,” in *Proc. IEEE/ACM 2nd Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 8–15.

[41] N. Veloso, “conkas.” Mar. 2021. Accessed: Feb. 2, 2024. [Online]. Available: https://github.com/nveloso/conkas

[42] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, “Smart contract vulnerability detection using graph neural networks,” in *Proc. 29th Int. Conf. Int. Joint Conf. Artif. Intell.*, 2021, pp. 3283–3290.

[43] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, “Eth2vec: learning contract-wide code representations for vulnerability detection on Ethereum smart contracts,” in *Proc. 3rd ACM Int. Symp. Blockchain Secure Crit. Infrastruct.*, 2021, pp. 47–59.

[44] Y. Sun et al., “GPTscan: Detecting logic vulnerabilities in smart contracts by combining GPT with program analysis,” in *Proc. IEEE/ACM 46th Int. Conf. Softw. Eng. (ICSE),* 2024, pp. 1–13.

[45] P. Liu et al., “Exploring ChatGPT’s capabilities on vulnerability management,” in *Proc. 33rd USENIX Secur. Symp. (USENIX Secur.)*, 2024, pp. 811–828.

[46] C. Chen et al., “When ChatGPT meets smart contract vulnerability detection: How far are we?” 2023, *arXiv:2309.05520*.

[47] C. Shou, J. Liu, D. Lu, and K. Sen, “Llm4fuzz: Guided fuzzing of smart contracts with large language models,” 2024, *arXiv:2401.11108*.

[48] Y. Sun et al., “Llm4vuln: A unified evaluation framework for decoupling and enhancing LLMs’ vulnerability reasoning,” 2024, *arXiv:2401.16185*.