Avaliação de LLMs na detecção de vulnerabilidades em contratos inteligentes

Lis Loureiro Sousa
Dept. de Ciências Exatas e Tecnológicas
Univers. Estadual do Sudoeste da Bahia
Vitória da Conquista, BA, Brasil
lis-ls@hotmail.com
ORCID: 0009-0000-2387-6433

Cauê Rodrigues de Aguiar
Dept. de Ciências Exatas e Tecnológicas
Univers. Estadual do Sudoeste da Bahia
Vitória da Conquista, BA, Brasil
cauaguiar@outlook.com.br
ORCID: 0009-0002-1150-2375

Breno Novais Couto
Dept. de Ciências Exatas e Tecnológicas
Univers. Estadual do Sudoeste da Bahia
Vitória da Conquista, BA, Brasil
brenonovaiscouto@gmail.com
ORCID: 0009-0001-7046-9993

Ângela Gabriele de Souza Silva
Dept. de Ciências Exatas e Tecnológicas
Univers. Estadual do Sudoeste da Bahia
Vitória da Conquista, BA, Brasil
angelagabrielesouza@gmail.com
ORCID: 0009-0004-0912-2511

Hélio Lopes dos Santos
Dept. de Ciências Exatas e Tecnológicas
Univers. Estadual do Sudoeste da Bahia
Vitória da Conquista, BA, Brasil
heliosantos@uesb.edu.br
ORCID: 0009-0007-7709-3830

***Resumo*—A auditoria de contratos inteligentes é essencial para garantir a segurança de aplicações descentralizadas, prevenindo falhas críticas em ambientes imutáveis como blockchains. Este estudo tem como objetivo avaliar a eficácia de Modelos de Linguagem de Grande Escala (LLMs) na detecção automatizada de vulnerabilidades em contratos inteligentes. Para tanto, 40 contratos de diferentes categorias foram analisados por quatro LLMs — GPT-4o, DeepSeek-R1, Llama-3.3 e Gemini 2.0 Flash — utilizando um prompt unificado para extrair e classificar vulnerabilidades por severidade. O desempenho foi mensurado por precisão, recall, F1-Score e erro médio absoluto, comparando as detecções com auditorias de referência. O melhor modelo alcançou 10,36% de precisão e 22,48% de recall, indicando que os LLMs ainda requerem aprimoramentos para uso autônomo confiável.**

***Keywords*—Contratos inteligentes, LLMs, blockchain, segurança, vulnerabilidades.**

I. Introdução

Modelos de Linguagem de Grande Escala (LLMs) têm se destacado como ferramentas promissoras na engenharia de software, tanto em seus modelos comerciais quanto em iniciativas de código aberto, de modo que têm demonstrado eficácia em aplicações críticas, como segurança digital e auditoria de contratos inteligentes [1].

A auditoria de *smart contracts* é fundamental para garantir a segurança de aplicações descentralizadas, pois, apesar de muitas vulnerabilidades não resultarem necessariamente em exploração real, a natureza imutável da *blockchain* torna essencial a identificação preventiva de falhas para mitigar riscos antes que possam ser explorados efetivamente [2]. Somente em 2024, vulnerabilidades exploradas em contratos inteligentes foram responsáveis por 14% das perdas totais no espaço Web3, totalizando US$ 308,7 milhões, segundo o *The Hacken 2024 Web3 Security Report* [3]. Nesse cenário, ferramentas automatizadas e precisas são essenciais para a detecção precoce de vulnerabilidades, mitigando riscos e fortalecendo a confiabilidade do sistema [4].

Diante desse contexto, a seguinte pergunta de pesquisa orienta este estudo: em que medida LLMs são capazes de detectar, de forma autônoma, vulnerabilidades em contratos inteligentes quando comparados a auditorias realizadas por empresas especializadas? Para respondê-la, este trabalho avalia o desempenho de quatro modelos: DeepSeek-R1, GPT-4o, Llama 3.3 e Gemini 2.0 Flash, na detecção de vulnerabilidades em *smart contracts*, comparando seus resultados com auditorias de referência e entre si. A contribuição central deste trabalho é, portanto, de natureza avaliativa. Em vez de propor uma nova técnica ou ferramenta, esta pesquisa foca em fornecer uma análise empírica sobre a viabilidade dos LLMs como alternativa ou complemento às soluções convencionais de auditoria, mapeando suas capacidades e limitações atuais no estado da arte. Para tanto, o desempenho dos modelos é quantificado por meio de métricas como precisão, *recall* e erro médio absoluto na classificação de severidade. As principais contribuições deste trabalho são: (i) uma avaliação comparativa de quatro LLMs na detecção de vulnerabilidades em contratos inteligentes estratificados por categoria; (ii) a quantificação do desempenho dos modelos por meio de métricas padronizadas, incluindo concordância interavaliadores; e (iii) a discussão de diretrizes para a integração de LLMs em fluxos híbridos de auditoria.

II. Fundamentação teórica

Em 1996, Nicki Szabo cunhou o termo *smart contract*, definindo-o como protocolos computadorizados que executam termos contratuais automaticamente, reduzindo a necessidade de intermediários confiáveis. No ano seguinte, em 1997, o autor expandiu a definição, descrevendo-os como protocolos criptográficos e algorítmicos para automatizar transações legais e econômicas.

Além disso, a partir da introdução do Ethereum em 2014 por Vitalik Buterin, cria-se uma nova perspectiva em cima da utilização dos *smart contracts*, dado que nesta nova plataforma de *blockchain*, eles teriam o papel de *scripts* que seriam executados de forma determinística em um ambiente descentralizado de modo a permitir acordos sem necessidade de confiança e com aplicações em finanças descentralizadas (DeFi), tokens não fungíveis (NFTs), governança, entre outros.

Devido à imutabilidade e popularidade crescente dos *smart contracts* com o surgimento do Ethereum, tornou-se necessário o emprego de mecanismos de segurança para minimizar vulnerabilidades nos contratos [5]. Em 2024, o *The Hacken 2024 Web3 Security Report* enfatizou o impacto desse risco contínuo ao evidenciar que 14% das perdas totais no espaço Web3 devem-se às vulnerabilidades exploradas nos *smart contracts*, totalizando uma perda de 308,7 milhões de dólares naquele ano.

Desse modo, as principais causas de vulnerabilidades em *smart contracts* estão associadas à alta visibilidade, opacidade dos contratos, imutabilidade do código, execução automatizada sem controle adequado e imaturidade da linguagem Solidity[1], especialmente no tratamento de exceções e reentrância [6]. Visto isso, tais características técnicas expõem os contratos a riscos de segurança significativos em ambientes descentralizados.

Nesse viés, a complexidade crescente dos *smart contracts* exige que as auditorias evoluam em precisão e abrangência. Métodos tradicionais muitas vezes falham em identificar vulnerabilidades contextuais ou emergentes, especialmente aquelas relacionadas à execução automatizada e à ordem de transações. Assim, é necessário incorporar ferramentas complementares, como análise dinâmica e LLMs, que ampliam a capacidade de detecção. Além disso, [7] evidencia que a atualização constante das práticas de auditoria torna-se essencial para garantir a segurança nesse ambiente.

Por conseguinte, em estudos recentes, o potencial dos LLMs na auditoria automatizada de contratos inteligentes tem sido evidenciado, com destaque para sua capacidade como ferramenta identificadora de vulnerabilidades complexas que não são detectadas por abordagens tradicionais [4]. Dessa forma, a utilização desses modelos amplia significativamente a flexibilidade e a adaptabilidade da auditoria de contratos inteligentes e permite que o processo de detecção seja aplicado a uma gama maior de contextos [8].

III. Trabalhos Relacionados

No campo da análise automatizada de contratos inteligentes, ferramentas tradicionais como Mythril, Slither e Oyente têm sido amplamente utilizadas para verificação estática e simbólica. [9] realizaram uma avaliação empírica dessas ferramentas, constatando que, em conjunto, elas podem alcançar até 42% de precisão na detecção de vulnerabilidades, embora apresentem limitações na identificação de falhas contextuais e lógicas. Em paralelo, o uso de LLMs na segurança de software tem se expandido para além de contratos inteligentes, abrangendo tarefas como detecção de vulnerabilidades em código convencional, geração de testes de segurança e análise de código malicioso [1].

[10] avaliaram LLMs na detecção de vulnerabilidades em contratos inteligentes DeFi, testando o GPT-4 e o Claude com contratos reais e sintéticos vulneráveis por meio de prompting estruturado. Os modelos identificaram 40% das vulnerabilidades conhecidas e alcançaram 78,7% em testes de mutação, apresentando potencial como ferramenta auxiliar de auditoria, apesar das altas taxas de falsos positivos e da necessidade de revisão humana.

Ademais, no trabalho proposto por [11], foram avaliados cinco LLMs (GPT-4o-mini, Gemini, Claude, Yi-large, Qwen-plus) com contratos em Solidity v0.8, utilizando três bases de dados, constatando que *prompts* específicos reduzem falsos positivos em mais de 60%, embora persistam dificuldades na detecção de falhas complexas, como reentrância.

Em outra aplicação, [12] propõe o *iAudit*, um *framework* que combina *fine-tuning* de modelos de linguagem e agentes baseados em LLMs para auditoria intuitiva de contratos inteligentes com justificativas em duas etapas: identificar vulnerabilidades com base em múltiplos *prompts* e *majority voting*, e gerar explicações sobre as causas das vulnerabilidades. O sistema, então, é treinado com um conjunto de dados balanceado, contendo 1.734 amostras positivas (funções vulneráveis) e 1.810 amostras negativas (funções seguras), e utiliza a técnica de *Low-Rank Adaptation* (LoRA) para ajuste eficiente de parâmetros, os quais demonstram desempenho superior em relação a modelos de aprendizado por *prompt* e *fine-tuning* tradicional. Entretanto, sua eficácia está centrada na detecção de vulnerabilidades lógicas, podendo ser menos adequada para falhas convencionais, como *reentrancy* e *overflow*.

[13], em abordagem semelhante, propõe o *PropertyGPT*, uma ferramenta baseada em modelos de linguagem que gera automaticamente propriedades formais para a verificação de contratos inteligentes. Assim, é possível reutilizar propriedades existentes com recuperação aumentada por contexto. Os resultados indicam alta efetividade, com 80% de *recall*, identificação de nove CVEs reais e descoberta de 12 vulnerabilidades inéditas. Contudo, persistem limitações quanto à detecção de vulnerabilidades contextuais, ao foco restrito em propriedades de segurança e à necessidade de validação manual.

Nesse intérim, o presente estudo diferencia-se dos trabalhos relacionados por propor uma abordagem híbrida de auditoria que combina Large Language Models (LLMs) com ferramentas tradicionais de verificação estática. Enquanto pesquisas anteriores [4], concentraram-se na classificação de vulnerabilidades específicas em smart contracts utilizando redes neurais supervisionadas, o presente modelo integra a interpretação semântica fornecida por LLMs com critérios objetivos de rastreio sintático.

Desse modo, o método utilizado se assemelha mais aos trabalhos de [12] e [11] pois foi evidenciada a análise semântica. Não obstante, diferentemente de estudos que tratam os LLMs apenas como geradores de texto ou classificadores binários, esta pesquisa explora seu potencial de analisar a eficácia dos modelos de LLMs buscando também fornecer justificativas

[1]Principal linguagem de programação para contratos inteligentes na *blockchain* Ethereum e outras redes compatíveis com sua Máquina Virtual (EVM).

interpretáveis para cada alerta identificado, com intuito de qualifica-las em processos de auditoria.

IV. Metodologia

A metodologia deste estudo foi elaborada para garantir rigor acadêmico, reprodutibilidade e transparência a partir da revisão sistemática da literatura sobre as vulnerabilidades, ferramentas e *benchmarks* de [14] e do levantamento sistemático de engenharia de *prompt* de [15]. Assim, o processo iniciou com a seleção de um conjunto de 40 contratos inteligentes estratificados em quatro categorias principais: onze contratos DeFi, onze contratos NFTs, nove contratos de governança e nove contratos sem vulnerabilidades detectadas. A seleção baseou-se em auditorias públicas disponíveis em sites *Open Source*, principalmente o *GitHub* e em demais plataformas especializadas, priorizando contratos auditados após 2022 para assegurar relevância técnica.

Para o desenvolvimento e a validação da abordagem proposta, foram empregados *datasets* compostos por contratos inteligentes públicos e relatórios de auditoria emitidos por empresas especializadas em segurança *blockchain*. Esses materiais foram obtidos em fontes amplamente reconhecidas, como *Hacken*, *Quantstamp* e *QuillAudits*, além de repositórios oficiais hospedados no GitHub e no Bitbucket.

A seleção contemplou projetos de diferentes naturezas e níveis de complexidade, de modo a garantir diversidade estrutural e semântica no corpus analisado. Entre eles, incluem-se os contratos dos projetos *Athens Token* [16], [17], *Crypto-Today ERC20/ERC1155 Voting* [18], [19], *Openware Yellow Network* [20], *ClearSync* [21], e *ZKRace ERC20* [22].

Complementarmente, foram integrados contratos provenientes dos projetos *Bloqhouse Technologies RWA* [23], *Token Shares Solidity* [24], e *Ethereum Towers Staking* [25], [26], além de amostras do *Tengoku Senso* [27] e do *TGK Smart Contracts Audit* [28]. Por fim, também foram considerados o *Sequence Smart Wallet* [29], [30], o protocolo *Taiko* [31], [32] e o projeto *Meta Monkey* [33], ampliando a representatividade e robustez dos experimentos.

Esses *datasets* englobam uma ampla variedade de padrões de implementação, incluindo tokens ERC-20 e ERC-1155, contratos de *staking*, carteiras digitais e mecanismos de governança descentralizada, oferecendo uma base empírica diversificada para análise comparativa. Todos os contratos foram escritos em Solidity e as vulnerabilidades presentes incluem, entre outras, reentrância, controle de acesso inadequado, manipulação de oráculos, *overflow/underflow* e falhas em lógica de negócio.

Posto isso, para garantir a rastreabilidade e facilidade de interpretação, foram organizados os contratos e as análises no repositório GitHub[2] , cada contrato foi organizado em pastas estruturadas, contendo documentações, código-fonte, relatório de auditoria oficial e subpastas para análises individuais dos contratos.

Foram selecionados quatro modelos de linguagem, abrangendo soluções comerciais e de código aberto: GPT-4o (OpenAI), DeepSeek-R1 (DeepSeek), Llama-3.3-70B-Instruct (Meta) e Gemini 2.0 Flash (Google). A escolha considerou diversidade arquitetural, desempenho em *benchmarks* de código e viabilidade operacional, de modo a avaliar *trade-offs* entre escala de parâmetros, precisão analítica e custo computacional. Todos os modelos foram acessados por meio de suas interfaces web oficiais, sem ajuste fino ou configuração de temperatura.

O protocolo experimental envolveu a execução padronizada utilizando-se de um *prompt* unificado (Figura 1) desenvolvido e formulado para a auditoria de contratos inteligentes com base em técnicas de *prompt engineering*, com o objetivo de atingir respostas estruturadas e claras; dentre elas, *Structured Chain-of-Thought* (SCoT) [34], *Zero-shot* [35] e *Logical CoT* (LogiCoT) [36]. No *prompt* solicita-se a listagem de vulnerabilidades no código Solidity de forma breve, sem exemplos, e com a exigência de categorizar as vulnerabilidades de acordo com sua severidade, classificadas em '*High*', '*Medium*', '*Low*', '*Undetermined*' ou '*Informational*'. Para garantir a imparcialidade e a consistência, o mesmo *prompt* foi utilizado para todos os modelos. Essa abordagem foi adotada para assegurar que as diferenças de desempenho observadas pudessem ser atribuídas unicamente às capacidades inerentes de cada LLM, evitando a complexidade de discernir se um resultado superior decorre do modelo em si ou da otimização do *prompt*.

> Atue como um auditor de smart contracts.
> 1. Liste vulnerabilidades neste código Solidity de forma breve, sem exemplos, no formato Título e Descrição.
> 2. Separe por categorias cada uma como High, Medium, Low, Undetermined ou Informational.
>
> **Contrato:**

Figura 1. *Prompt* usado para auditoria automática de *smart contracts*.

Sobre o tratamento do contexto dos contratos, a análise foi feita copiando o código-fonte de cada contrato diretamente no *prompt*, sem *Retrieval-Augmented Generation* (RAG)[3] ou truncamento, pensando em simular o cenário de uso mais direto e simples de um LLM para auditoria.

A extração das vulnerabilidades de referência (*ground truth*) foi realizada a partir dos relatórios de auditoria em PDF feitos por empresas especializadas, os quais foram analisados manualmente para garantir a precisão das categorias de vulnerabilidade e as respectivas classificações de severidade. As respostas dos LLMs foram comparadas a essas auditorias, classificando cada detecção como Verdadeiro Positivo (VP) para casos nos quais o LLM acertou parcialmente ou completamente a vulnerabilidade, Falso Positivo (FP) para casos em que a LLM criou vulnerabilidades inexistentes no contrato ou Falso Negativo (FN) para quando as vulnerabilidades reais não foram encontradas pela LLM. Após a classificação, foi realizada a soma das diferenças nos erros de classificação de severidade. O procedimento de correspondência entre as detecções dos LLMs e as vulnerabilidades do ground truth seguiu critérios semânticos: cada vulnerabilidade reportada pelo modelo foi comparada individualmente às vulnerabilidades documentadas no relatório de auditoria oficial, verificando-se a equivalência

[2]Disponível em: https://anonymous.4open.science/r/Dataset-Smart-Contracts-Anonymous-04D4/

[3]RAG é uma técnica que aprimora LLMs ao permitir a consulta a bases de conhecimento externas antes de gerar uma resposta, o que resulta em análises mais precisas e contextualizadas [37].

quanto ao tipo de falha (e.g., reentrância, controle de acesso) e à localização no código (função ou trecho afetado). Uma detecção foi classificada como VP quando o tipo de vulnerabilidade identificado pelo LLM correspondia, total ou parcialmente, a uma vulnerabilidade presente no relatório, independentemente da formulação textual utilizada pelo modelo. Detecções sem correspondência semântica com nenhuma vulnerabilidade real foram classificadas como FP, e vulnerabilidades do relatório não mencionadas por nenhuma saída do modelo foram registradas como FN.

Para garantir a confiabilidade das classificações, a cada contrato, dois avaliadores (autores deste estudo) realizaram a análise das respostas dos LLMs em comparação com o *ground truth* extraído dos relatórios de auditoria em PDF. Cada avaliador classificou as detecções dos LLMs de forma independente e atribuiu categorias de VP, FP e FN. O coeficiente Kappa de Cohen (κ) foi calculado para medir a concordância interavaliadores nessas categorizações, seguindo a escala de [38]. Após treinamento prévio com um manual de codificação, os avaliadores alcançaram κ = 0.8832, indicando uma concordância quase perfeita. Discordâncias residuais foram resolvidas por consenso entre os próprios avaliadores, sem necessidade de um terceiro especialista.

As métricas de avaliação utilizadas para quantificar o desempenho dos LLMs foram: Precisão, que mede a proporção de detecções corretas entre todas as vulnerabilidades reportadas pelo modelo; *Recall*, que avalia a proporção de vulnerabilidades reais efetivamente identificadas; *F1-Score*, a média harmônica entre precisão e *recall*; e o Erro Médio Absoluto (MAE), que calcula a diferença média entre as classificações de severidade previstas e reais, utilizando a escala numérica: *High*=4, *Medium*=3, *Low*=2, *Informational*=1, *Undetermined*=0. Essas métricas fornecem uma visão geral da capacidade das LLMs em analisar as vulnerabilidades de *smart contracts*.

Por fim, para assegurar a equidade na comparação entre os modelos, todos os testes foram conduzidos sob condições experimentais semelhantes visando eliminar variáveis externas que pudessem influenciar os resultados para que as diferenças observadas nas métricas de desempenho fossem atribuídas exclusivamente às capacidades intrínsecas de cada LLM. Cabe ressaltar que cada contrato foi analisado uma única vez por cada modelo, focando na resposta típica em vez de sua variabilidade estocástica.

V. Resultados e Discussões

A partir do processo de análise dos 40 contratos, pôde-se extrair os seus respectivos VP, FP e FN de modo a gerar informação doravante dos dados, nos quais estão sintetizados por LLM na Tabela I. Nesta tabela, o melhor desempenho em cada métrica é destacado em negrito para facilitar a comparação.

Nesse contexto, a **Precisão** mede a proporção de vulnerabilidades apontadas pelo LLM que eram de fato reais, indicando a confiabilidade de seus alertas. O ***Recall*** avalia a capacidade do modelo em identificar o total de vulnerabilidades existentes, refletindo sua abrangência. O ***F1-Score*** oferece a média harmônica entre precisão e recall, que reflete o equilíbrio entre a utilidade (*recall*) e a confiabilidade (precisão) das detecções do modelo. Embora baseados nas contagens de **VP** e **FP**, estes foram omitidos da tabela em prol da clareza. Em contrapartida, a contagem bruta de **FN** foi mantida, pois quantifica diretamente o risco crítico de vulnerabilidades não detectadas. Por fim, o **Erro Médio Absoluto** quantifica a divergência média entre a severidade atribuída pelo LLM e a oficial para os casos corretamente identificados.

Tabela I. Desempenho dos modelos LLM

LLM	Precisão	*Recall*	*F1-Score*	FN	Erro Médio Absoluto
DeepSeek-R1	**0.1036**	**0.2248**	**0.1418**	**0.7752**	0.8621
GPT-4o	0.0871	**0.2248**	0.1255	**0.7752**	1.1724
Gemini 2.0 Flash	0.0714	0.1163	0.0885	0.8837	**0.6667**
Llama 3.3	0.0621	0.2093	0.0957	0.7907	1.0741

Posto isso, com base nas informações coletadas, a DeepSeek-R1 demonstrou a maior precisão entre os modelos em identificar vulnerabilidades das auditorias oficiais, 10,36% indicam que nove em cada dez alertas são incorretos, o que inviabiliza o uso autônomo. Para comparação, em conjunto, ferramentas estáticas de detecção de vulnerabilidades em *smart contracts* podem chegar a 42% de precisão [9], enquanto *frameworks* mais elaborados, com *machine learning*, podem atingir 91% [39].

Visto isso, ainda é possível obter as especificidades de cada LLM nesse caso de uso específico. O Gemini na análise de problemas em contratos limpos obteve a menor quantidade de FPs, e na classificação das vulnerabilidades, foi o modelo que melhor performou, com o índice de erro médio absoluto de 0.6667 unidades aproximadas, todavia, o GPT-4o teve o maior índice (1.17). Na análise, o DeepSeek-R1 e o GPT-4o compartilharam o maior *recall* (22.48%), detectando um em cada cinco casos positivos reais e, em contraste, o Gemini 2.0 Flash teve desempenho significativamente inferior (11.63%), falhando em 88.37% dos casos positivos. O DeepSeek-R1 também destacou-se com o maior *F1-Score* (14.18%), seguido pelo GPT-4o (12.55%). Nesse viés, a baixa harmonização entre precisão e *recall* (<15%) reforça a necessidade de técnicas de pós-processamento para aplicações práticas.

Esses resultados são consistentes com os achados de [10], que reportaram 40% de identificação de vulnerabilidades conhecidas com GPT-4, e com [11], que evidenciaram a persistência de dificuldades na detecção de falhas complexas como reentrância. Em comparação, abordagens mais elaboradas como o *iAudit* [12], que combinam *fine-tuning* e *majority voting*, e o *PropertyGPT* [13], que alcançou 80% de *recall* com geração de propriedades formais, sugerem que a integração de técnicas complementares é determinante para superar as limitações observadas no uso isolado de LLMs.

Nesse contexto, os resultados apresentam um cenário alarmante na capacidade de persuasão do modelo [40], visto que, mesmo com a comparação com o *ground truth*, uma pessoa que não tem conhecimento técnico no assunto passaria por dificuldades em entender o que é verdade ou não.

Assim, no que tange ao desempenho limitado observado relaciona-se a múltiplos fatores conjunturais. Primeiramente, os LLMs apresentam limites intrínsecos quando confrontados com código que exige raciocínio sobre execução determinística e estados de máquina, pois sua arquitetura é otimizada para padrões estatísticos de linguagem, não para semântica executável completa. Em segundo lugar, a ausência de um maior

contexto de execução impediu que os modelos avaliassem comportamentos dinâmicos que permitem apontar muitas vulnerabilidades só se manifestam sob condições de execução específicas. Adicionalmente, a complexidade sintática e idiomática de contratos reais e a ambiguidade na descrição natural das vulnerabilidades nos relatórios de referência aumentam a dificuldade de alinhamento entre o que se espera detectar e o que o modelo relata. Por fim, a sensibilidade ao *prompt* e à ordem das instruções pode ter desviado parte das respostas para interpretações superficiais, elevando falsos positivos e falsos negativos.

Nesse contexto, o papel mais promissor dos LLMs está em atuar como componentes integrados em *pipelines* híbridos de auditoria, e não como substitutos do auditor humano. Com base nos resultados obtidos e na literatura analisada, propõe-se um fluxo operacional de auditoria híbrida composto por cinco etapas: (1) pré-processamento do código-fonte, incluindo *chunking* por função ou contrato e enriquecimento contextual via RAG com documentação e padrões de vulnerabilidades conhecidos; (2) análise automatizada por múltiplos LLMs, aplicando o mesmo *prompt* padronizado a cada modelo para gerar relatórios independentes de vulnerabilidades; (3) pós-processamento por *majority voting*, consolidando as detecções convergentes entre os modelos e filtrando alertas isolados de baixa confiança; (4) verificação complementar por ferramentas estáticas e simbólicas (e.g., Slither, Mythril), responsáveis pela validação formal das vulnerabilidades candidatas; e (5) revisão humana supervisionada, na qual auditores especializados avaliam os alertas remanescentes com suporte das evidências geradas nas etapas anteriores. Esse fluxo visa maximizar a cobertura de detecção enquanto reduz a taxa de falsos positivos, combinando a flexibilidade semântica dos LLMs com o rigor formal das ferramentas tradicionais.

Dessa forma, os resultados deste estudo reforçam que, quando utilizados isoladamente, os LLMs ainda apresentam limitações significativas para auditorias autônomas, mas demonstram grande potencial como ferramentas de co-auditoria. A combinação de modelos como o GPT-4 com técnicas de *chunking* e escopos contextuais tem alcançado até 76,5% de respostas parcial ou totalmente corretas [41], indicando que a adoção de abordagens híbridas pode aprimorar substancialmente a precisão e a eficiência nos processos de auditoria de contratos inteligentes.

VI. Considerações Finais

Os resultados deste estudo indicam que, embora LLMs apresentem potencial significativo para apoiar a auditoria de contratos inteligentes, sua precisão atual ainda é insuficiente para aplicações críticas de forma autônoma. Os modelos são capazes de identificar esparsas vulnerabilidades, mas possuem um alto índice de erro, o que compromete sua confiabilidade como única ferramenta de verificação, tornando seu uso autônomo ineficiente. Logo, a pesquisa, ao evidenciar de maneira sistemática os limites e as possibilidades dos LLMs nesse contexto, sugere caminhos para avanços futuros, como o uso de modelos mais robustos e a integração com técnicas como análise estática e simbólica.

Além disso, para fomentar a continuidade das investigações, todos os dados e artefatos experimentais foram disponibilizados publicamente, incentivando a transparência e a colaboração científica. O estudo revela que LLMs, apesar de promissores como ferramentas complementares na auditoria de contratos inteligentes, têm aplicação autônoma limitada por baixa precisão (10,36% no melhor caso) e altos falsos positivos. A contribuição original está na avaliação comparativa de modelos em contratos estratificados por complexidade e na proposta de fluxos híbridos integrando análise estática, revisão humana e pós-processamento. Cabe reconhecer, contudo, algumas limitações inerentes ao desenho da pesquisa. A principal delas refere-se à execução única de cada teste por contrato e por modelo, o que restringe a estimação de intervalos de confiança e a mensuração da variabilidade estocástica típica de LLMs. Considerando que modelos generativos podem produzir variações para um mesmo *prompt*, a ausência de replicações sugere cautela na generalização dos resultados, embora não comprometa as tendências observadas.

Adicionalmente, a correspondência entre as saídas dos LLMs e o *ground truth* foi baseada em critérios semânticos validados por elevada concordância interavaliadores (= 0.8832), o que reforça a consistência do procedimento, ainda que permaneça algum grau residual de julgamento qualitativo.

Por fim, destacam-se desafios metodológicos próprios do campo, como a complexidade de capturar vulnerabilidades interdependentes e a sensibilidade das respostas ao *prompting*, aspectos que abrem espaço para aprofundamentos em pesquisas futuras.

Referências

[1] Z. Wei, J. Sun, Z. Zhang, and X. Zhang, "LLM-SmartAudit: Advanced Smart Contract Vulnerability Detection," *arXiv (Cornell University)*, 10 2024. [Online]. Available: http://arxiv.org/abs/2410.09381

[2] D. Perez and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1325–1341. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/perez

[3] Hacken, "The Hacken 2024 Web3 Security Report," 12 2024. [Online]. Available: https://hacken.io/insights/2024-security-report/

[4] C. Chen, J. Su, J. Chen, Y. Wang, T. Bi, J. Yu, Y. Wang, X. Lin, T. Chen, and Z. Zheng, "When ChatGPT meets Smart Contract Vulnerability detection: How far are we?" *ACM Transactions on Software Engineering and Methodology*, 11 2024. [Online]. Available: https://doi.org/10.1145/3702973

[5] D. He, Z. Deng, Y. Zhang, S. Chan, Y. Cheng, and N. Guizani, "Smart Contract vulnerability analysis and security audit," *IEEE Network*, vol. 34, no. 5, pp. 276–282, 7 2020. [Online]. Available: https://doi.org/10.1109/mnet.001.1900656

[6] H. Zhou, A. M. Fard, and A. Makanju, "The state of Ethereum Smart Contracts Security: Vulnerabilities, countermeasures, and tool support," *Journal of Cybersecurity and Privacy*, vol. 2, no. 2, pp. 358–378, 5 2022. [Online]. Available: https://doi.org/10.3390/jcp2020019

[7] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, "Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract," *IEEE Access*, vol. 10, pp. 6605–6621, 1 2022. [Online]. Available: https://doi.org/10.1109/access.2021.3140091

[8] S. Hu, T. Huang, F. İlhan, S. F. Tekin, and L. Liu, "Large Language Model-Powered Smart Contract Vulnerability Detection: New Perspectives," *arXiv (Cornell University)*, 1 2023. [Online]. Available: https://arxiv.org/abs/2310.01152

[9] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," *ICSE*, 6 2020. [Online]. Available: https://doi.org/10.1145/3377811.3380364

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

[11] Z. Xiao, Q. Wang, H. Pearce, and S. Chen, "Logic meets magic: Llms cracking smart contract vulnerabilities," 2025. [Online]. Available: https://arxiv.org/abs/2501.07058

[12] W. Ma, D. Wu, Y. Sun, T. Wang, S. Liu, J. Zhang, Y. Xue, and Y. Liu, "Combining fine-tuning and llm-based agents for intuitive smart contract auditing with justifications," *arXiv preprint arXiv:2403.16073*, 2024.

[13] Y. Liu, Y. Xue, D. Wu, Y. Sun, Y. Li, M. Shi, and Y. Liu, "Propertygpt: Llm-driven formal verification of smart contracts through retrieval-augmented property generation," *arXiv preprint arXiv:2405.02580*, 2025, to appear in Network and Distributed System Security (NDSS) Symposium, Internet Society.

[14] G. Iuliano and D. N. Dario, "Smart Contract Vulnerabilities, Tools, and Benchmarks: An Updated Systematic Literature review," *arXiv (Cornell University)*, 12 2024. [Online]. Available: http://arxiv.org/abs/2412.01719

[15] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A Systematic survey of prompt engineering in large language Models: Techniques and applications," *arXiv (Cornell University)*, 2 2024. [Online]. Available: http://arxiv.org/abs/2402.07927

[16] Quantstamp, "Athens token smart contract audit," https://certificate.quantstamp.com/full/athens.pdf, 2023.

[17] StormX, "Athens token smart contracts," https://github.com/stormxio/athens-token/tree/7ed63ba12f03c4e7856eb5845a5a234d0f806bd2/contracts, 2023.

[18] Hacken, "Cryptotoday erc20/erc1155 voting smart contract audit," https://hacken.io/audits/cryptotoday/sca-cryptotoday-erc20-erc1155-voting-feb2022/, 2022.

[19] CryptoToday, "Cryptotoday contracts," https://github.com/cryptotoday com/contracts/tree/548c1ef24d996a3adc0557638601d099a5ef745d, 2022.

[20] Hacken, "Openware yellow network smart contract audit," https://hacken.io/audits/openware-yellow-network/sca-yellow-network-erc20-mar2023/, 2023.

[21] Layer-3, "Clearsync smart contracts," https://github.com/layer-3/clearsync/tree/5b86a2134d295ac11af97d4f239782222e95fe24/contracts, 2023.

[22] Hacken, "Zkrace erc20 smart contract audit," https://hacken.io/audits/zkrace/sca-zkrace-erc20-mar2024/, 2024.

[23] ——, "Bloqhouse technologies rwa smart contract audit," https://hacken.io/audits/bloqhouse-technologies-b-v/sca-bloqhouse-technologies-rwa-mar2023/, 2023.

[24] A. Persson, "Token shares solidity contracts," https://bitbucket.org/alfredpersson/token-shares-solidity/src/cbdc7c0d6162346b96cf62cb2ff93c15f416819e/, 2023.

[25] Hacken, "Ethereum towers staking smart contract audit," https://hacken.io/audits/ethereum-towers/sca-ethereum-towers-staking-jun2022/, 2022.

[26] E. Towers, "Ethereum towers contracts," https://github.com/ethereumtowers/contracts/tree/94eb48031a02455bb3c48285ffe41fbbe3498079, 2022.

[27] Quantstamp, "Tengoku senso smart contract audit certificate," https://certificate.quantstamp.com/full/tengoku-senso/5361cc88-760a-4571-8284-7951b4dbbff4/index.html, 2023.

[28] A. Sharma, "Tgk smart contracts audit," https://github.com/AkshaySharma96/TGK-Smart-Contracts-Audit/tree/68f99d348ee637d90ba91b2996d1e132f7cf4268, 2023.

[29] Quantstamp, "Sequence smart wallet audit report," https://certificate.quantstamp.com/full/sequence-smart-wallet.pdf, 2023.

[30] 0xSequence, "Sequence wallet contracts," https://github.com/0xsequence/wallet-contracts/tree/7492cb33cea25696355a0e2a76f1fe9ea2adfbbd, 2023.

[31] QuillAudits, "Taiko smart contract audit," https://www.quillaudits.com/leaderboard/taiko, 2024.

[32] T. Labs, "Taiko mono smart contracts," https://github.com/taikoxyz/taiko-mono/tree/based_contestable_zkrollup, 2024.

[33] QuillAudits, "Meta monkey smart contract audit," https://www.quillaudits.com/leaderboard/meta-monkey, 2024.

[34] J. Li, G. Li, Y. Li, and Z. Jin, "Structured Chain-of-Thought prompting for code generation," *arXiv (Cornell University)*, 1 2023. [Online]. Available: https://arxiv.org/abs/2305.06599

[35] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, and et al., "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[36] X. Zhao, M. Li, W. Lu, C. Weber, J. H. Lee, K. Chu, and S. Wermter, "Enhancing Zero-Shot Chain-of-Thought Reasoning in Large Language Models through Logic," *arXiv (Cornell University)*, 1 2023. [Online]. Available: https://arxiv.org/abs/2309.13339

[37] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, H. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, vol. 2, no. 1, 2023.

[38] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, p. 159, 3 1977. [Online]. Available: https://doi.org/10.2307/2529310

[39] L. S. H. Colin, P. M. Mohan, J. Pan, and P. L. K. Keong, "An integrated smart contract vulnerability detection tool using Multi-Layer Perceptron on Real-Time Solidity smart contracts," *IEEE Access*, vol. 12, pp. 23 549–23 567, 1 2024. [Online]. Available: https://doi.org/10.1109/access.2024.3364351

[40] I. Amaro, A. Della Greca, R. Francese, G. Tortora, and C. Tucci, *AI Unreliable Answers: a case study on ChatGPT*, 1 2023. [Online]. Available: https://doi.org/10.1007/978-3-031-35894-4_2

[41] Y. Liu, Y. Xue, D. Wu, Y. Sun, Y. Li, M. Shi, and Y. Liu, "PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation," *arXiv (Cornell University)*, 5 2024. [Online]. Available: http://arxiv.org/abs/2405.02580