Robustez de LLMs na Detecção de Vulnerabilidades em Contratos Inteligentes com Transformações no Código

Rafael Santa Rosa Alves, Marco Amaral Henriques

Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Campinas – São Paulo – Brasil

rsralves@dac.unicamp.br, maah@unicamp.br

Abstract. *Large Language Models (LLMs) have shown promising results in code security analysis, but their evaluation can be influenced by non-executable textual cues. This work evaluates the robustness of LLMs in detecting smart contract vulnerabilities under semantics-preserving code transformations. Comparing four LLM models on original contracts and on variants with removed comments, renamed identifiers, and both transformations combined, we observed F1 score reductions of up to 99.5%. This demonstrates that different, yet semantically equivalent, representations of the same code can produce substantially distinct diagnoses.*

Resumo. *Grandes Modelos de Linguagem (LLMs) têm apresentado resultados promissores na análise de segurança de código, mas sua avaliação pode ser influenciada por pistas textuais não executáveis. Este trabalho avalia a robustez de LLMs na detecção de vulnerabilidades em contratos inteligentes sob transformações de código que preservam a semântica. Na comparação de quatro modelos de LLM, sobre contratos originais e em variantes sem comentários, com identificadores renomeados e com ambas as transformações combinadas, constataram-se reduções de F1 de até 99,5%, mostrando que diferentes representações do mesmo código, mas semanticamente equivalentes, podem produzir diagnósticos substancialmente distintos.*

1. Introdução

Grandes Modelos de Linguagem (*Large Language Models* – LLMs) vêm sendo empregados em geração, revisão e análise de código [Hou et al. 2024, Zhou et al. 2025]. No contexto específico de contratos inteligentes, há resultados promissores na identificação e localização de vulnerabilidades [Sun et al. 2024]. Entretanto, esses resultados são normalmente obtidos sobre uma representação fixa de cada programa, exatamente como disponibilizada pelo benchmark.

Essa prática não permite avaliar se o diagnóstico permanece estável diante de transformações que preservam o comportamento do programa. Comentários podem ser removidos, identificadores podem ser renomeados e o código pode ser reformatado sem

Trabalho realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001. O ChatGPT (OpenAI) apoiou a depuração do código e a revisão ortográfica do manuscrito. Os autores validaram e assumem responsabilidade pelo conteúdo.

que uma vulnerabilidade deixe de existir. Se alterações desse tipo modificam substancialmente a resposta do modelo, o desempenho medido pode depender de pistas superficiais da representação, e não apenas da lógica responsável pela falha. Essa possibilidade é especialmente relevante em segurança, pois decisões inconsistentes podem limitar a confiança na auditoria automatizada.

Contratos inteligentes constituem um estudo de caso interessante por reunirem características importantes para esta avaliação. A plataforma SmartBugs Curated fornece programas Solidity com vulnerabilidades anotadas em nível de linha [Durieux et al. 2020], granularidade necessária para a tarefa de localização e pouco comum em conjuntos de detecção, geralmente rotulados por função ou arquivo. Os contratos também são curtos o bastante para serem submetidos integralmente em uma única requisição, eliminando o fracionamento do código como variável interveniente. Além disso, o conjunto continua sendo empregado na avaliação de LLMs, de modo que a questão investigada incide sobre uma prática corrente. As características investigadas podem não ser exclusivas desse domínio e, portanto, este trabalho não presume sua generalização e restringe as conclusões ao conjunto avaliado.

Mais especificamente, comparamos o desempenho de quatro LLMs em quatro versões semanticamente equivalentes do SmartBugs Curated: original, sem comentários, com identificadores renomeados e com ambas as transformações. O prompt, os parâmetros de inferência e o critério de avaliação são mantidos constantes. Assim, a variável experimental é a representação do contrato.

As contribuições são: (i) um protocolo controlado e reprodutível para avaliar robustez por meio de variantes pareadas de código; (ii) uma análise empírica com modelos de diferentes escalas, incluindo execuções locais e via NVIDIA NIM; e (iii) a disponibilização do código, das variantes do conjunto e das saídas brutas. O objetivo não é propor um novo detector, mas medir uma limitação que avaliações convencionais podem ocultar.

2. Trabalhos Relacionados

LLMs e vulnerabilidades em contratos inteligentes. Trabalhos recentes utilizam LLMs para classificar contratos, localizar linhas vulneráveis e produzir explicações, geralmente sobre conjuntos tradicionais [Boi et al. 2024, Chen et al. 2025, Mandana et al. 2025]. Essas avaliações demonstram o potencial dos modelos, mas não isolam necessariamente influência de informações textuais não executáveis dos contratos.

Robustez de modelos de código. Transformações que preservam a semântica já foram usadas para gerar exemplos adversariais [Yefet et al. 2020] e avaliar a generalização de modelos neurais de programas [Rabin et al. 2021]. Esses estudos estabelecem que mudanças lexicais e sintáticas podem afetar modelos de código. O presente trabalho parte dessa evidência, mas investiga uma questão experimental distinta: o efeito dessas transformações sobre LLMs generativos empregados como auditores, com saída estruturada, localização exata de linha e métricas de detecção.

Artefatos e sensibilidade ao prompt. O aprendizado de atalhos e a exploração de artefatos de anotação podem produzir desempenho elevado sem que o modelo aprenda as propriedades pretendidas [Geirhos et al. 2020, Gururangan et al. 2018]. LLMs também são sensíveis à formulação e à formatação do prompt [Sclar et al. 2024]. Nosso estudo

não pretende caracterizar essa segunda forma de sensibilidade. Para impedir que ela confunda a comparação entre variantes, o mesmo prompt, reproduzido no Apêndice A, é utilizado em todas as execuções.

3. Metodologia

3.1. Conjunto de dados e transformações

A avaliação rigorosa da eficácia de ferramentas de segurança e modelos de linguagem depende criticamente da disponibilidade de conjuntos de dados devidamente rotulados [Achjian and Junior 2025]. Neste trabalho, utilizamos o SmartBugs Curated, em sua versão mais recente com 143 contratos Solidity e 207 vulnerabilidades já rotuladas e organizadas segundo as categorias do DASP Top 10. A escolha decorre de ser um conjunto de referência amplamente adotado, ainda usado em estudos recentes com LLMs [da Costa et al. 2026, Chen et al. 2025, Boi et al. 2024] e de fornecer anotações em nível de linha. Essas anotações provêm da distribuição oficial e permitem compararmos diretamente as respostas produzidas para versões equivalentes do mesmo contrato.

Aplicamos duas transformações elementares e sua combinação, todas preservando a semântica operacional e a numeração das linhas. A remoção de comentários usa um analisador de estados caractere a caractere que elimina comentários de linha e de bloco, inclusive as marcações de vulnerabilidade da distribuição original, e mantém todas as quebras de linha de modo que cada linha de código permanece em sua posição original. A renomeação de identificadores percorre a árvore sintática abstrata e substitui variáveis, funções e nomes de contrato por rótulos genéricos (x001, x002, . . .), preservando a consistência referencial, ou seja, todas as ocorrências de um identificador recebem o mesmo novo nome de forma que o programa permanece válido e equivalente.

Cruzando as duas transformações com suas ausências, obtemos quatro variantes: **orig** (conjunto original), **sem-com** (sem comentários), **renom** (identificadores genéricos) e **ambas**. Esse conjunto é o ponto central da metodologia: orig vs. sem-com isola o efeito dos comentários; orig vs. renom, o dos identificadores; e ambas mede o efeito conjunto. As transformações não eliminam toda a informação superficial; ainda permanecem a estrutura, as chamadas de API (p. ex., call.value) e os padrões sintáticos típicos de cada classe, de modo que a degradação medida é um limite inferior da dependência de pistas.

3.2. Modelos

Avaliamos quatro modelos de pesos abertos com escalas e arquiteturas distintas: llama-3.3-70b-instruct, gpt-oss-120b, deepseek-v4-pro 1.6T e llama-3.1-8b-instruct. Os três primeiros executaram via NVIDIA NIM, plataforma que expõe um endpoint compatível com a API da OpenAI e o último, foi executado localmente com o Ollama em uma GPU NVIDIA RTX 4060 de 8 GB, fornecendo um contraste local de menor escala.

Modelos proprietários de fronteira não foram incluídos. Considerando o volume de tokens medido no experimento e os preços de API vigentes durante seu planejamento, a execução da matriz completa ultrapassaria US$ 2.000 em opções como GPT-5.5 e Claude Opus 4.8. Priorizamos, portanto, modelos de pesos abertos que permitissem repetir todas as configurações dentro dos recursos disponíveis. Como a comparação conduzida é sempre *intramodelo* sob representações equivalentes, a validade do achado não depende de os modelos avaliados serem os de melhor desempenho absoluto na tarefa. Mesmo que

essa escolha limite o alcance das conclusões, entendemos que há uma boa probabilidade de que modelos grandes só reforçarão os achados de modelos bem mais modestos. Talvez tivéssemos conclusões mais limitadas se os resultados mostrados fossem oriundos apenas de modelos maiores e mais refinados.

3.3. Protocolo experimental

Cada execução submete um contrato ao modelo e solicita uma saída JSON contendo as vulnerabilidades presentes e as linhas correspondentes. O fluxo é orquestrado com LangGraph, garantindo isolamento de contexto, ou seja, cada contrato é processado de forma independente, sem histórico compartilhado.

Dado que a formulação do prompt é fator reconhecido de variação de desempenho [Sclar et al. 2024], utilizamos o mesmo prompt em todos os modelos, variantes e execuções, sem ajuste por modelo ou conjunto. Ele enumera as categorias do DASP Top 10 e especifica o formato de saída, seu texto integral consta do Apêndice A. As diferenças entre variantes não podem, assim, ser atribuídas a mudanças na formulação da tarefa.

Todos os experimentos usaram temperatura 0, o que corresponde a decodificação gulosa e elimina a aleatoriedade de amostragem, mas não torna a execução determinística. Em GPUs, reduções paralelas utilizam aritmética de ponto flutuante, que não é associativa, de modo que diferentes ordens de somação podem produzir resultados ligeiramente distintos. Já em serviços de inferência em nuvem, o batching dinâmico modifica a composição dos lotes, o que pode alterar os kernels utilizados e a ordem das reduções.

Além disso, dois dos modelos avaliados, o GPT-OSS-120B e o DeepSeek-V4-Pro, adotam arquitetura de mistura de especialistas (*Mixture of Experts*), em que cada *token* é encaminhado a um subconjunto de especialistas. Nesses casos, o próprio roteamento pode variar com a composição do lote, somando-se às fontes de aleatoriedade anteriores. Tais diferenças podem inverter a escolha do token mais provável em quase-empates, e a geração autorregressiva amplifica a divergência. Por isso, cada configuração foi executada três vezes, reportando-se a média.

3.4. Métricas e reprodutibilidade

Uma predição é considerada verdadeiro positivo apenas quando a categoria e a linha indicada correspondem exatamente à anotação de referência. Não é utilizada margem de tolerância. Indicações sem correspondência são falsos positivos, enquanto vulnerabilidades não identificadas são falsos negativos. A avaliação utilizou métricas tradicionais de recuperação de informação com precisão, revocação e F1, sendo F1 a medida principal por sintetizar o compromisso entre cobertura e excesso de alertas [Manning 2008].

4. Resultados e Discussão

A Tabela 1 apresenta o desempenho dos modelos nas quatro variantes[1]. A remoção de comentários reduz F1 em uma ordem de grandeza ou mais em todos os modelos e a combinação das transformações leva o desempenho a valores próximos de zero.

Efeito da remoção de comentários: Essa transformação produziu a maior degradação consistente. Em relação ao conjunto original, o F1 caiu 99,5% para o Llama

[1]O código e demais artefatos estão organizados no repositório `https://github.com/regras/smart-contracts-vulnerabilities-by-llm`

Tabela 1. Precisão (P), Revocação (R) e F1 por variante.

Modelo	orig			renom			sem-com			ambas		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Llama 3.1 8B	**0,737**	0,779	**0,757**	**0,656**	0,713	**0,684**	0,004	0,005	0,004	0,003	0,005	0,004
Llama 3.3 70B	0,252	0,461	0,326	0,174	0,346	0,232	0,004	0,010	0,006	0,003	0,010	0,005
GPT-OSS-120B	0,379	0,823	0,519	0,343	**0,808**	0,481	**0,040**	**0,093**	**0,056**	**0,045**	**0,113**	**0,065**
DeepSeek-V4-Pro	0,475	**0,947**	0,632	0,333	0,655	0,441	0,007	0,019	0,010	0,002	0,006	0,005

3.1, 98,2% para o Llama 3.3, 89,2% para o GPT-OSS e 98,4% para o DeepSeek-V4-Pro. Como o código executável e as linhas vulneráveis permaneceram inalterados, a diferença mostra que os comentários exerceram influência substancial sobre os diagnósticos. O resultado não prova ausência de raciocínio sobre o código, mas demonstra que o desempenho não é invariável a informações não executáveis.

Efeito da renomeação: A renomeação isolada teve efeito menor que a remoção de comentários nos quatro modelos. O F1 caiu 9,6% no Llama 3.1, 28,8% no Llama 3.3, 7,3% no GPT-OSS e 30,2% no DeepSeek-V4-Pro. A combinação permaneceu próxima da variante sem comentários, sugerindo que, neste conjunto, a presença dos comentários é o principal fator entre as duas transformações avaliadas.

Escala dos modelos: O menor modelo avaliado, Llama 3.1 8B, obteve o maior F1 no conjunto original (0,757), mas caiu para 0,004 após as duas transformações. Na mesma comparação, o Llama 3.3 70B passou de 0,326 para 0,005. Esse contraste pode indicar, especificamente entre os dois modelos da família Llama, que o desempenho superior do modelo menor no conjunto original está associado a uma maior influência das pistas presentes na representação, e não necessariamente a uma maior capacidade de identificar vulnerabilidades. Além disso, como ambos atingiram valores próximos de zero após as ambas transformações, há um possível efeito de piso que reduz a capacidade de distinguir seus graus de degradação. Os modelos maiores também apresentaram quedas expressivas: o GPT-OSS, por exemplo, passou de 0,519 para 0,065. Assim, os resultados não sustentam que o aumento de escala, isoladamente, elimine a sensibilidade à representação.

Observações exploratórias: Foram feitos testes preliminares com comentários adversariais que afirmavam incorretamente que trechos vulneráveis eram seguros e as respostas mudaram de maneira relevante, indicando que esse cenário merece investigação. Entretanto, uma avaliação adversarial adequada exigiria diferentes formulações, posições, intensidades e modelos de atacante. Por limitações de espaço, tempo e cobertura experimental, não apresentamos esses testes e os reservamos para trabalho futuro.

Hipóteses alternativas: A degradação observada poderia, em princípio, ter outras causas. Uma delas é o deslocamento de distribuição [Moreno-Torres et al. 2012], mas os dados não a favorecem, já que a renomeação produz código bem mais atípico que a remoção de comentários e, ainda assim, degrada muito menos. Outra é a contaminação do conjunto [Sainz et al. 2023], público desde 2020, que não pode ser descartada com os dados disponíveis. Ela é compatível com a interpretação proposta, já que a memorização estaria ancorada na forma textual original. Em qualquer dos casos, a implicação prática se mantém. Pontuações com representação original superestimam desempenho com representações equivalentes.

4.1. Mitigação preliminar por representação do contrato

Observamos que os modelos frequentemente apontavam linhas em branco ou deslocadas em relação ao trecho descrito, evidenciando dificuldade inerente dos LLMs em realizar o mapeamento posicional preciso em nível de linha sobre código em texto plano [Fu and Tantithamthavorn 2022]. Avaliamos, então, uma alteração estritamente de representação: prefixar cada linha com seu respectivo número, sem modificar o código. Ponderando pela perda absoluta de F1 em `sem-com` e `ambas` (Tabelas 1 e 2), a prefixação recupera 50,5% da degradação, com variação de 23% a 88% entre modelos. Como ela corrige a ancoragem sem restituir pistas textuais, em média 49,5% da perda reflete dependência dessas pistas.

Tabela 2. F1 com prefixação de linhas nas variantes transformadas.

Modelo	`renom`		`sem-com`		`ambas`	
	padrão	prefixado	padrão	prefixado	padrão	prefixado
Llama 3.1 8B	0,684	0,410	0,004	0,196	0,004	0,175
Llama 3.3 70B	0,232	0,731	0,006	0,286	0,005	0,270
GPT-OSS-120B	0,481	0,553	0,056	0,410	0,065	0,385
DeepSeek-V4-Pro	0,441	0,665	0,010	0,338	0,005	0,274

5. Conclusões e Trabalhos Futuros

Avaliamos a robustez de LLMs para detecção de vulnerabilidades em contratos inteligentes sob transformações que preservam o comportamento dos programas. Mantendo prompt, parâmetros e critério de avaliação constantes, observamos que a remoção de comentários reduziu a F1 em mais de 89% nos quatro modelos. A renomeação apresentou impacto menor, com F1 reduzindo entre 7,3% e 30,2%, em todos eles, enquanto sua combinação com a remoção de comentários manteve o desempenho próximo de zero.

Os achados mostram que pontuações obtidas em uma única representação de um benchmark não caracterizam, por si só, a robustez de um auditor baseado em LLM. Para o SmartBugs Curated, informações não executáveis influenciam substancialmente os diagnósticos. Avaliações futuras devem reportar não apenas desempenho no conjunto original, mas também estabilidade sob variantes semanticamente equivalentes. Além disso, as transformações removem apenas parte das pistas superficiais. Chamadas de API, operadores, fluxo de controle e outros padrões sintáticos permanecem disponíveis. Os resultados devem ser interpretados como evidência de sensibilidade às propriedades modificadas, e não como uma medida completa da compreensão semântica dos modelos.

A avaliação restringe-se aos 143 contratos Solidity do SmartBugs Curated, e as conclusões valem para esse conjunto, sob os modelos e o prompt avaliados. Nada nos mecanismos descritos é específico de Solidity ou de Contratos Inteligentes, mas não dispomos de evidência para afirmar que o fenômeno se reproduz em outros domínios, essa verificação fica como trabalho futuro. O uso de um único prompt controla sua influência na comparação entre variantes, porém não mede a sensibilidade a diferentes formulações. Estudos futuros devem cruzar transformações do código com múltiplos prompts e incluir modelos proprietários quando houver recursos. Também pretendemos conduzir estudos específicos sobre comentários adversariais e integração com ferramentas estáticas, separando claramente robustez de representação, resistência a ataques e análise híbrida.

Referências

Achjian, R. and Junior, M. S. (2025). Building a labeled smart contract dataset for evaluating vulnerability detection tools' effectiveness. In *Anais Estendidos do XXV Simpósio Brasileiro de Cibersegurança*, pages 1–10, Porto Alegre, RS, Brasil. SBC.

Boi, B., Esposito, C., and Lee, S. (2024). Vulnhunt-gpt: a smart contract vulnerabilities detector based on openai chatgpt. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, pages 1517–1524.

Chen, C., Su, J., Chen, J., Wang, Y., Bi, T., Yu, J., Wang, Y., Lin, X., Chen, T., and Zheng, Z. (2025). When chatgpt meets smart contract vulnerability detection: How far are we? *ACM Transactions on Software Engineering and Methodology*, 34(4):1–30.

da Costa, E. V. A., Guzman, C. A. M., de Abreu, O. B. M., Sousa, J. C., and Braga, A. (2026). Análise comparativa do desempenho de llms e ferramentas de segurança na detecção de vulnerabilidades em contratos inteligentes. In *Workshop em Blockchain: Teoria, Tecnologias e Aplicações (WBlockchain)*, pages 1–14. SBC.

Durieux, T., Ferreira, J. F., Abreu, R., and Cruz, P. (2020). Empirical review of automated analysis tools on 47,587 Ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 530–541.

Fu, M. and Tantithamthavorn, C. (2022). Linevul: A transformer-based line-level vulnerability prediction. In *Proceedings of the 19th international conference on mining software repositories*, pages 608–620.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. (2020). Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2:665–673.

Gururangan, S., Swayamdipta, S., Levy, O., Schwartz, R., Bowman, S. R., and Smith, N. A. (2018). Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 107–112.

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., and Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79.

Mandana, E., Vlahavas, G., and Vakali, A. (2025). Evullm: ethereum smart contract vulnerability detection using large language models. *Electronics*, 14(16):3226.

Manning, C. D. (2008). *Introduction to information retrieval*. Syngress Publishing,.

Moreno-Torres, J. G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. V., and Herrera, F. (2012). A unifying view on dataset shift in classification. *Pattern recognition*, 45(1):521–530.

Rabin, M. R. I., Bui, N. D. Q., Wang, K., Yu, Y., Jiang, L., and Alipour, M. A. (2021). On the generalizability of neural program models with respect to semantic-preserving program transformations. *Information and Software Technology*, 135:106552.

Sainz, O., Campos, J., García-Ferrero, I., Etxaniz, J., de Lacalle, O. L., and Agirre, E. (2023). Nlp evaluation in trouble: On the need to measure llm data contamination

for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10776–10787.

Sclar, M., Choi, Y., Tsvetkov, Y., and Suhr, A. (2024). Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *Proceedings of the 12th International Conference on Learning Representations*.

Sun, Y., Wu, D., Xue, Y., Liu, H., Wang, H., Xu, Z., Xie, X., and Liu, Y. (2024). Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis. In *Proceedings of the IEEE/ACM 46th international conference on software engineering*, pages 1–13.

Yefet, N., Alon, U., and Yahav, E. (2020). Adversarial examples for models of code. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–30.

Zhou, X., Cao, S., Sun, X., and Lo, D. (2025). Large language model for vulnerability detection and repair: Literature review and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–31.

A. Prompt utilizado nas avaliações

```
Analyze the Solidity contract below and identify all security vulnerabilities.

Requirements

- Consider only vulnerabilities present in the provided code.
- Report every vulnerability you identify.
- For each vulnerability provide:
  - vulnerability type
  - line number
  - short description
- Line numbering starts at line 1 of the provided input.
- Count every line exactly as provided, including blank lines and comments.
- If no vulnerabilities are found, return an empty list.

Use only the following vulnerability categories:

- arithmetic
- access_control
- reentrancy
- denial_of_service
- front_running
- bad_randomness
- time_manipulation
- unchecked_low_level_calls
- short_addresses
- other

Output

Return only valid JSON.
```