Aplicação de Large Language Models na Geração Automática de Testes de Software: Uma Revisão Sistemática da Literatura

Samuel Ryan da Fonseca Anunciação[1] , Luis Fernando Maia Santos Silva[1]

[1] Instituto Federal de Educação, Ciência e Tecnologia do Maranhão (IFMA)
Campus Caxias - MA - Brasil
samuel.ryan@acad.ifma.edu.br
luis.maia@ifma.edu.br

Abstract. *Manual test generation is costly, motivating the adoption of Large Language Models (LLMs). This Systematic Literature Review (SLR), based on Kitchenham's methodology, analyzed 20 primary studies and identified a transition toward multi-stage architectures using Retrieval-Augmented Generation (RAG) and iterative repair. The analyzed approaches achieved up to 77.67% line coverage and reduced manual effort by 45.8%. Despite challenges such as hallucinations and context limitations, LLMs have consolidated themselves as effective support for automated test generation.*

Resumo. *A geração manual de testes é onerosa, impulsionando o uso de Large Language Models (LLMs). Esta Revisão Sistemática da Literatura (RSL), baseada na metodologia de Kitchenham, analisou 20 estudos e identificou a evolução para arquiteturas multiestágio com Retrieval-Augmented Generation (RAG) e reparo iterativo. As abordagens analisadas alcançaram até 77,67% de cobertura de linha e redução de 45,8% no esforço manual. Apesar de desafios como alucinações e limites de contexto, os LLMs consolidam-se como suporte eficaz para geração automática de testes.*

1. Introdução

A garantia da qualidade de software demanda processos de teste complexos e onerosos, consumindo entre 30% e 50% do esforço de desenvolvimento [Barr et al. 2015]. Embora técnicas tradicionais, como *Search-Based Software Testing* (SBST), ampliem a cobertura estrutural, elas apresentam limitações na captura da intenção semântica do código e na geração de oráculos de teste [Yuan et al. 2024].

Com o avanço dos *Large Language Models* (LLMs), surgiram novas possibilidades para geração automática de testes, impulsionadas pela capacidade desses modelos em compreender e sintetizar código [Brown et al. 2020]. Estudos recentes demonstram ganhos relevantes em cobertura e detecção de defeitos, principalmente quando os modelos são integrados a mecanismos de recuperação de contexto e reparo iterativo [Zhang et al. 2025a, Li et al. 2026].

Apesar do crescimento da área e da existência de revisões voltadas principalmente para testes unitários, ainda são escassas revisões sistemáticas focadas na viabilidade técnica de LLMs em sistemas de larga escala e em múltiplos tipos de teste. Assim, esta Revisão Sistemática da Literatura (RSL), conduzida segundo Kitchenham

[Kitchenham 2004], analisa 20 estudos primários publicados entre 2022 e 2026 para responder: (RQ1) quais arquiteturas e abordagens são utilizadas; (RQ2) qual a efetividade técnica reportada; e (RQ3) quais benefícios e limitações são observados.

2. Fundamentação Teórica e Desafios

Os LLMs baseiam-se na arquitetura Transformer e em mecanismos de autoatenção [Vaswani et al. 2017]. Técnicas recentes incluem *Low-Rank Adaptation* (LoRA), utilizada para especialização eficiente de modelos, e arquiteturas multiestágio, caracterizadas pela combinação de etapas como recuperação de contexto, análise estática e reparo iterativo durante a geração de testes.

Na geração automática de testes, abordagens tradicionais como SBST [Harman and McMinn 2010], execução simbólica [Cadar and Sen 2013] e ferramentas como EvoSuite [Fraser and Arcuri 2011] enfrentam o problema do oráculo, isto é, a dificuldade em produzir asserções semanticamente corretas [Barr et al. 2015].

Embora os LLMs avancem nesse aspecto, persistem limitações como janela de contexto restrita, alucinações e risco de *data leakage* em benchmarks públicos, como Defects4J [Just et al. 2014]. Além disso, estudos reportam recorrência de *test smells* em suítes geradas automaticamente [Alves et al. 2025], incluindo redundância de casos de teste e baixa coesão entre cenários avaliados.

3. Metodologia

Esta RSL seguiu as diretrizes de Kitchenham [Kitchenham 2004], contemplando planejamento, execução e síntese dos dados. O protocolo foi estruturado segundo a estratégia PICOC, considerando: sistemas de software complexos ou de larga escala (*Population*); técnicas baseadas em LLMs para geração automática de testes (*Intervention*); comparação com ferramentas tradicionais ou testes manuais quando disponível (*Comparison*); métricas como cobertura, *Mutation Score*, executabilidade e detecção de defeitos (*Outcome*); e estudos empíricos publicados entre 2022 e 2026 (*Context*).

As buscas foram realizadas nas bases ACM Digital Library, IEEE Xplore e Scopus, considerando estudos publicados entre 2022 e 2026. A string de busca combinou termos relacionados a LLMs, geração automática de testes e qualidade de software. O termo "unit test generation" foi incluído para ampliar o recall, embora a revisão contemple também testes GUI e API. Foram considerados estudos publicados em inglês.

Foram incluídos estudos empíricos que aplicam LLMs à geração automática de testes em sistemas complexos. Excluíram-se estudos secundários, literatura cinzenta e trabalhos sem validação experimental.

Ao todo, 148 estudos foram identificados. Após remoção de 16 duplicatas e aplicação dos critérios de seleção, 20 estudos primários compuseram o corpus final da revisão. A triagem foi conduzida por um único revisor, seguindo critérios previamente definidos no protocolo para reduzir vieses de seleção e garantir rastreabilidade metodológica. Devido às restrições de espaço do formato reduzido, os resultados são apresentados de forma sintética, sem discussão individual de todos os estudos primários analisados.

Os estudos foram avaliados por seis critérios de qualidade (Q1–Q6), relacionados à clareza metodológica, definição de métricas, evidências quantitativas e discussão de

limitações. Os escores de qualidade foram utilizados como suporte à interpretação crítica dos resultados e à análise da robustez metodológica dos estudos selecionados.

A Figura 1 apresenta o fluxo PRISMA do processo de seleção dos estudos.

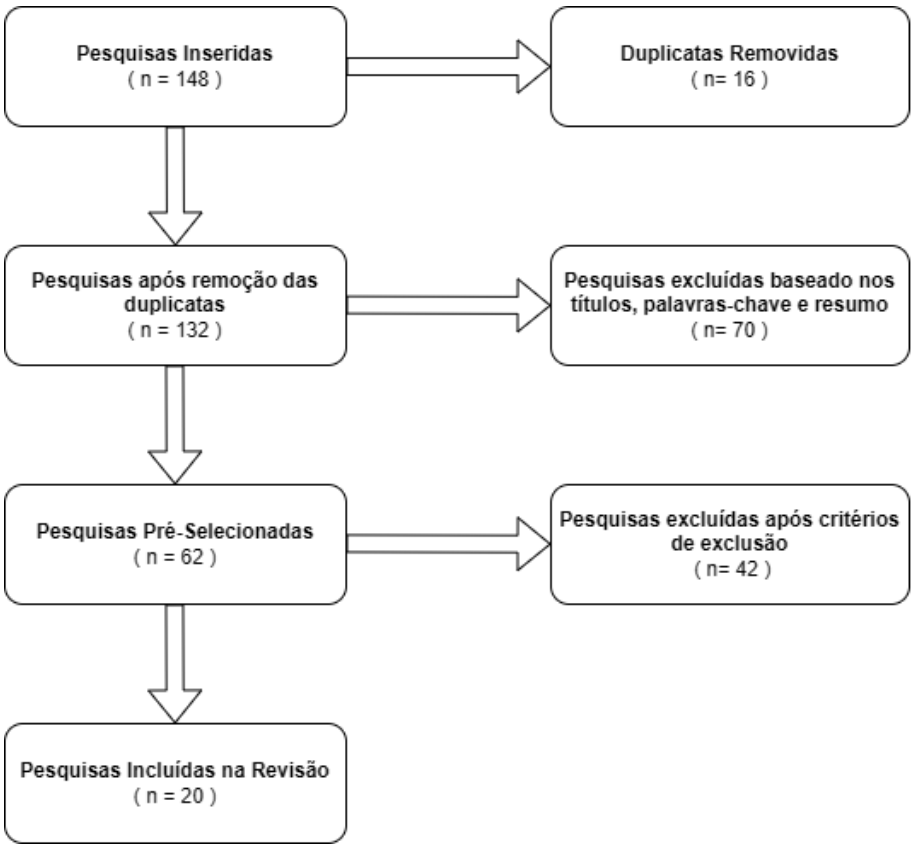

Figura 1. Fluxo PRISMA do processo de seleção dos estudos.

A Tabela 1 apresenta a síntese da avaliação de qualidade dos estudos.

Tabela 1. Síntese da avaliação de qualidade dos estudos

Faixa de Score	Quantidade de Estudos
6.0	10
5.5	5
5.0	4
4.5	1

Os resultados indicam predominância de estudos com elevada qualidade metodológica, com 15 dos 20 estudos alcançando pontuação igual ou superior a 5,5 nos critérios avaliados.

4. Resultados e Discussão

A análise dos estudos evidencia uma transição de abordagens baseadas apenas em prompting para arquiteturas multiestágio integradas a mecanismos de *Retrieval-Augmented Generation* (RAG), análise estática e reparo iterativo [Zhang et al. 2025c,

Zhang et al. 2025a]. Estratégias *few-shot* apresentaram melhor desempenho que abordagens *zero-shot*, com ganhos de até 28,3 pontos percentuais na taxa de compilação [Auer et al. 2024].

Quanto à efetividade técnica, o framework STRUT alcançou 77,67% de cobertura de linha em programas C [Liu et al. 2025], enquanto abordagens baseadas em *mutation testing* atingiram *Mutation Score* de até 93,57% [Moradi Dakhel et al. 2024]. Também foram observadas reduções de até 45,8% no esforço manual de criação de testes [Zhang et al. 2025b].

Os principais benefícios incluem maior capacidade de capturar intenção semântica e automatizar geração de oráculos a partir de documentação [Hossain et al. 2025]. Entretanto, persistem desafios relacionados à janela de contexto, alucinações, dependência de APIs proprietárias e incidência de *test smells* em suítes geradas [Alves et al. 2025].

Os resultados também evidenciam que a efetividade dos LLMs depende mais da engenharia de contexto e integração arquitetural do que apenas do tamanho do modelo. A combinação entre recuperação de informações, análise estática e reparo iterativo mostrou-se mais consistente em sistemas complexos do que abordagens baseadas exclusivamente em prompting.

5. Ameaças à Validade

As principais ameaças incluem risco de *data leakage* em benchmarks públicos e limitação das buscas a bases indexadas. Além disso, a forte dependência de benchmarks como Defects4J reduz a generalização para sistemas industriais.

Métricas estáticas, como *CodeBLEU* e *Exact Match*, também podem não refletir corretamente a qualidade funcional dos testes. Para mitigar essa limitação, priorizaram-se estudos que reportam métricas dinâmicas, como cobertura e detecção de defeitos.

6. Considerações Finais

Esta Revisão Sistemática da Literatura analisou a aplicação de *Large Language Models* na geração automática de testes de software em sistemas de maior escala. Os resultados indicam uma evolução para arquiteturas multiestágio baseadas em RAG e reparo iterativo, permitindo ganhos relevantes de cobertura e redução de esforço manual.

Apesar dos avanços, persistem desafios relacionados à confiabilidade lógica, alucinações, custo computacional e dependência de APIs proprietárias. Nesse contexto, os LLMs mostram maior efetividade como suporte assistido, atuando em conjunto com análise estática e validação humana.

Como trabalhos futuros, destacam-se técnicas de compressão baseadas em *Abstract Syntax Tree* (AST) e o uso de *Small Language Models* executados localmente, visando reduzir custos operacionais, limitações de contexto e riscos de privacidade.

Referências

Alves, V., Bezerra, C. I. M., Machado, I. d. C., Rocha, L., Virgínio, T., and Silva, P. (2025). Quality assessment of Python tests generated by large language models. In *Proc. EASE*.

Auer, M., Moreno, I. A., and Fraser, G. (2024). LLMs for automated unit test generation and assessment in Java: The AgoneTest framework. In *Proc. AST*.

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., and Yoo, S. (2015). The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS 33)*.

Cadar, C. and Sen, K. (2013). Symbolic execution for software testing: three decades later. *Communications of the ACM*, 56(2):82–90.

Fraser, G. and Arcuri, A. (2011). EvoSuite: Automatic test suite generation for object-oriented software. In *Proc. FSE*, pages 416–419.

Harman, M. and McMinn, P. (2010). A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Transactions on Software Engineering*, 36(2):226–247.

Hossain, S. B., Taylor, R., and Dwyer, M. B. (2025). Doc2OracLL: Investigating the impact of documentation on LLM-based test oracle generation. *Proceedings of the ACM on Software Engineering*, 2(FSE).

Just, R., Jalali, D., and Ernst, M. D. (2014). Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proc. ISSTA*, pages 437–440.

Kitchenham, B. (2004). Procedures for performing systematic reviews. Technical Report TR/SE-0401, Keele University and NICTA.

Li, T., Cui, C., Huang, R., Towey, D., and Ma, L. (2026). Large language models for automated web-form-test generation: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 35(3).

Liu, J., Li, C., Chen, R., Li, S., Gu, B., and Yang, M. (2025). STRUT: Structured seed case guided unit test generation for C programs using LLMs. *Proceedings of the ACM on Software Engineering*, 2(ISSTA).

Moradi Dakhel, A., Nikanjam, A., Majdinasab, V., Khomh, F., and Desmarais, M. C. (2024). Effective test generation using pre-trained large language models and mutation testing. *Information and Software Technology*, 171:107468.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS 30)*.

Yuan, Z., Liu, M., Ding, S., Wang, K., Chen, Y., Peng, X., and Lou, Y. (2024). Evaluating and improving ChatGPT for unit test generation. *Proceedings of the ACM on Software Engineering*, 1(FSE).

Zhang, Q., Fang, C., Zheng, Y., Zhang, Y., Zhao, Y., Huang, R., Zhou, J., Yang, Y., Zheng, T., and Chen, Z. (2025a). Improving deep assertion generation via fine-tuning retrieval-augmented pre-trained language models. *ACM Transactions on Software Engineering and Methodology*, 34(7).

Zhang, Q., Kang, R., Liu, Y., and Cao, X. (2025b). Large language model-enhanced test case generation. In *Proc. CBASE*. IEEE.

Zhang, Z., Liu, X., Lin, Y., Gao, X., Sun, H., and Yuan, Y. (2025c). Reference-based retrieval-augmented unit test generation. *ACM Transactions on Software Engineering and Methodology*, 34(9).