JIC JORNADA DE INICIAÇÃO CIENTÍFICA E TECNOLÓGICA XV EDIÇÃO
Integridade Científica e Combate à desinformação
20 a 22 de outubro
uffs.edu.br/jic
UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

GERAÇÃO DE CÓDIGO DE LINGUAGENS DE PROGRAMAÇÃO UTILIZANDO MODELOS DE LINGUAGEM AMPLA (LLMS)

LUIZ PAULO GRAFETTI TERRES [1,2]*, SAMUEL DA SILVA FEITOSA[3]

1 Introdução

Modelos de linguagem são peça fundamental para o campo de processamento da linguagem natural, valendo-se de métodos matemáticos para predição de elementos de linguagem (palavras, frases) baseados em contexto (WANG et al., 2024). Os recentes modelos de linguagem ampla (LLMs) ganharam especial notoriedade quando atingiram um público maior, ao serem disponibilizados de forma online e com interface simplificada, por empresas como a OpenAI (Chatgpt) e Google (Gemini). LLMs são baseados na arquitetura *Transformer*, apresentada por Vaswani et al. (2017), e são treinados usando centenas de bilhões de parâmetros e aprimorados para seguir instruções, além de, segundo Wang et al (2024), aderir aos valores humanos. Na engenharia de *software*, mais especificamente no desenvolvimento de *software*, esses modelos tem sido aplicados, entre outras finalidades, para sugestão de código e para geração de código a partir de descrição em linguagem natural (JIANG et al., 2024).

Para o desenvolvimento de *software*, compiladores são ferramentas fundamentais. Compiladores são responsáveis por otimizar e traduzir o código-fonte, produzido por um programador ou por um LLM, em código binário executável pelos componentes físicos do computador. Na condição de *software*, compiladores estão suscetíveis a erros e *bugs*, podendo causar graves falhas de segurança em todo o ecossistema de desenvolvimento de aplicações (MILLER et al., 1990; BAUER et al., 2015). *Fuzzing* surge como uma técnica de teste dinâmico de compiladores, a fim de aprimorar a implementação desses, identificando suas falhas antes que sejam exploradas. *Input fuzzing*, a opção mais popularmente aplicada para teste de compiladores, consiste em compilar uma grande quantidade de casos de teste (para compiladores, códigos-fonte) aleatórios ou semi-aleatórios, buscando por comportamentos inesperados do compilador ou resultados inconsistentes no binário gerado (MANÈS, 2021). Nesses casos, o caso de teste é revisitado, em investigação das causas para o possível *bug*.

[1] Graduando em Ciência da Computação, Universidade Federal da Fronteira Sul, *campus* Chapecó, contato: luiz.terres@estudante.uffs.edu.br
[2] Grupo de Pesquisa: IDT - Inovação e Desenvolvimento Tecnológico.
[3] Doutor em Computação, Universidade Federal da Fronteira Sul, **Orientador**.

A geração de casos de teste efetivos é vital para uma campanha de *fuzzing* ter sucesso em encontrar falhas no compilador. Entradas completamente aleatórias são geralmente pouco efetivas, causando erros de sintaxe prematuros e ativando poucas áreas da implementação do compilador (MCKEEMAN, 1998). Tradicionalmente, *fuzzers* utilizam ferramentas geradoras de código, em abordagens de geração de novos casos de teste ou de mutação de casos de teste existentes (MANÈS, 2021). Algumas dificuldades são associadas a essas ferramentas na literatura, como dificuldade técnica de construção, manutenção e ampliação do escopo das ferramentas (CHEN et al., 2020) e soluções pouco generalistas (XIA et al., 2024). Esta pesquisa explora como modelos de linguagem ampla, como ferramentas capazes de geração de código, podem encaixar-se nesse contexto. Esses modelos podem superar a capacidade das ferramentas tradicionais na geração de casos de teste efetivos para *fuzzing* de compiladores?

Este estudo investiga a viabilidade e explora as dificuldades e vantagens do uso de modelos de linguagem ampla para tarefas do processo de teste dinâmico de compiladores, por meio de uma revisão sistemática das soluções produzidas na literatura e o desenvolvimento de uma ferramenta para teste de compiladores da linguagem de programação Go.

2 Objetivos

O objetivo geral deste trabalho é explorar e avaliar a eficácia da integração de modelos de linguagem ampla ao processo de teste dinâmico (*fuzzing*) para detecção de falhas na implementação de compiladores. Especificamente: produzir uma Revisão Sistemática da Literatura (RSL) sobre o uso de LLMs em testes de compiladores; desenvolver uma ferramenta para *fuzzing* diferencial de compiladores da linguagem de programação Go, que utilize um modelo de linguagem ampla na geração de casos de teste; e comparar resultados da ferramenta desenvolvida com soluções tradicionais para compiladores Go, por meio de métricas de cobertura de código alcançada pelos casos de teste gerados e *bugs* encontrados.

3 Metodologia

3.1 Revisão sistemática da literatura

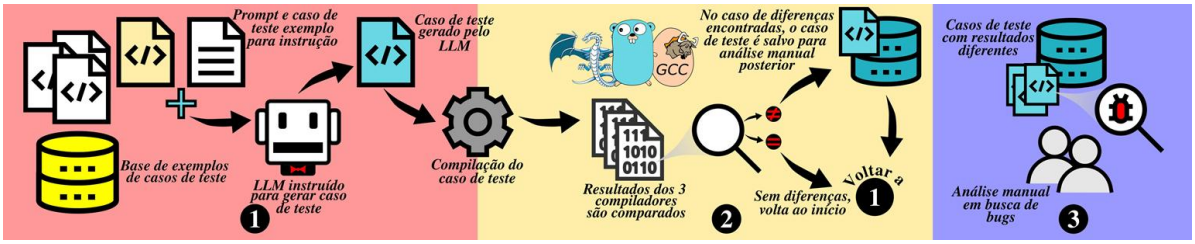
Uma revisão sistemática da literatura (RSL) busca apresentar uma pesquisa reprodutível sobre um tema na literatura, selecionando trabalhos relevantes para responder às perguntas de pesquisa. A pesquisa, sobre o emprego de modelos de linguagem ampla para tarefas no processo de teste de compiladores, foi conduzida na língua inglesa, usando as bases de dados *online*: *ACM Digital Library* e *Scopus*. Para buscar por trabalhos relevantes, uma *search string* foi

construída para selecionar artigos que contivessem, em qualquer lugar do corpo, as palavras "*fuzz*", "*fuzzer*", "*fuzzing*", "*test*", "*tester*" e "*testing*" a até 5 palavras de distância da palavra "*compiler*", e que contivessem "*large language model*" ou seu acrônimo "*LLM*" (ou suas formas plurais). Critérios de inclusão e exclusão explícitos filtraram artigos encontrados, restando artigos que atendem todos os critérios: pesquisa primária; publicado após 2019; afirmam foco em validação ou manipulação de bugs de compiladores; usam LLMs para alguma tarefa no processo de teste de compiladores; e escritos em inglês ou português. Esses artigos foram então cuidadosamente analisados para responder às seguintes perguntas de pesquisa: quais LLMs e compiladores são mais testados na literatura, quais são as tarefas dos LLMs nos testes e quão competitivas são as soluções baseadas neles e seus desafios.

3.2 Fuzzer diferencial de compiladores Go

A Figura 1 apresenta o funcionamento da ferramenta desenvolvida pelos autores para teste diferencial de compiladores da linguagem Go usando um LLM como gerador de testes.

Figura 1: Funcionamento do *fuzzer* diferencial de compiladores Go.



Fonte: Autores, 2025

Na etapa de geração de casos de teste (1), o LLM é instruído por meio de um *prompt* com caso de exemplo (*one-shot prompting*). Inicialmente a base de exemplos está populada com dados gerados pela pesquisadora Gu (2024). Os casos gerados pelo LLM (modelo StarCoder2 7B) são compilados por 3 compiladores (GOLLVM, GCCGO e o compilador oficial Go) e os resultados de compilação são comparados (teste diferencial) (2). O compilador oficial é instrumentalizado para a captura da métrica de cobertura de código atingida por caso de teste processado. Casos de teste gerados que alcançam sucesso na métrica de cobertura de código, ou apresentarem diferença de compilação, são adicionados à base de dados de exemplo para próximas iterações. Ao final, todos os casos de teste que geraram resultados de compilação diferentes são manualmente avaliados pelos autores, para identificação de causas para os possíveis *bugs*. Nossa ferramenta será comparada ao *go-fuzz*, *fuzzer* tradicional e oficial da linguagem Go.

4 Resultados e Discussão

4.1 Revisão sistemática da literatura (RSL)

Realizada na língua inglesa, a RSL selecionou um total de 10 trabalhos, publicados em jornais e conferências da *ACM*, *IEEE*, *Tech Science Press* e *Springer Nature*. Os trabalhos selecionados desenvolvem ferramentas para teste de compiladores (80%) ou processamento de *bugs* de compiladores (20%). Entre as tarefas mais populares para emprego dos LLMs estão a geração de casos de teste (50%), priorização de casos de teste (10%), geração de programas geradores de código (usados em *fuzzing* tradicional) (10%), sugestão de *flags* de compilação em estratégia de *option fuzzing* (10%), isolamento de *bugs* e redução de programas com *bug* (20%). A família de LLMs GPT está entre as mais populares, e os compiladores C, como o GCC estão entre os mais testados. As soluções usando LLMs foram frequentemente consideradas superiores às tradicionais, por métricas de cobertura de código e falhas encontradas. O principal desafio relatado entre os autores refere-se ao alto custo de inferência de LLMs mais potentes, que poderiam gerar casos de teste mais relevantes. Os resultados encontrados com a RSL reforçam a viabilidade do desenvolvimento de um *fuzzer* com LLM para Go e a comparação dos resultados ao seu *fuzzer* tradicional oficial, *go-fuzz*.

4.2 Fuzzer diferencial de compiladores Go

Até o momento, a solução foi parcialmente implementada pelos autores, e passa por preliminares do experimento. Essa subseção discute o que já foi implementado e o ainda em etapa de exploração baseados na Figura 1, apontando caminhos futuros. A base de exemplos iniciais apresenta código com cobertura promissora. A estrutura do *prompt* para geração de novos programas, no entanto, ainda não está rigidamente definida. Os compiladores GCCGO e GOLLVM não suportam a versão mais recente da linguagem de programação Go. Informações adicionais sobre essa implementação podem guiar o LLM, visando maior compatibilidade entre código gerado e versão suportada pelos compiladores. A comparação dos binários, gerados pela compilação de casos de teste, ocorre observando seus resultados de execução. A instrumentalização do compilador oficial Go permite que casos de teste gerados entrem na fila de exemplos, ordenados por seus valores de cobertura. Os processos de compilação e execução dos binários são limitados em tempo máximo de 1 minuto, visando evitar execuções infinitas, como as causadas por código com lógica de término indefinido. Resultados diferentes entre os compiladores indicam inconsistência e são salvos para análise posterior. Em trabalhos futuros, LLMs mais robustos e técnicas de instrução avançadas poderiam ser explorados para refinar a característica de teste diferencial da nossa solução.

5 Conclusão

Os resultados encontrados indicam um caminho promissor para a automação de testes de segurança em *software* de base usando LLMs. Soluções da literatura apresentam resultados favoráveis quando empregando esses modelos, indicando a possibilidade de simplificação em processos tradicionalmente de alta dificuldade técnica, e direcionando pesquisadores para o avanço de técnicas de instrução dos LLMs e balanço de custo computacional desses modelos.

Referências Bibliográficas

BAUER, Scott; CUOQ, Pascal; REGEHR, John. Deniable backdoors using compiler bugs. **International Journal of PoC||GTFO**, n. 8, p. 7-9, 2015. Acesso em: 10 ago. 2025.

CHEN, Junjie *et al*. A Survey of Compiler Testing. **ACM Comput. Surv.**, v. 53, n. 1, p. 4:1-4:36, 2020. Acesso em: 10 ago. 2025.

GU, Qiuhan. LLM-Based Code Generation Method for Golang Compiler Testing. *In*: **Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2023, p. 2201–2203. (ESEC/FSE 2023). Disponível em: <https://doi.org/10.1145/3611643.3617850>. Acesso em: 20 fev. 2025.

JIANG, Juyong *et al*. A Survey on Large Language Models for Code Generation. 2024. Disponível em: <http://arxiv.org/abs/2406.00515>. Acesso em: 20 fev. 2025.

MANÈS, Valentin J.M. *et al*. The Art, Science, and Engineering of Fuzzing: A Survey. **IEEE Transactions on Software Engineering**, v. 47, n. 11, p. 2312–2331, 2021. Acesso em: 10 ago. 2025.

MCKEEMAN, W. M. Differential Testing for Software. **Digit. Tech. J.**, 1998. Disponível em: <https://www.semanticscholar.org/paper/Differential-Testing-for-Software-McKeeman/fc881e8d0432ea8e4dd5fda4979243cac5e4b9e3>. Acesso em: 10 ago. 2025.

MILLER, Barton P.; FREDRIKSEN, Lars; SO, Bryan. An empirical study of the reliability of UNIX utilities. **Commun. ACM**, v. 33, n. 12, p. 32–44, 1990. Acesso em: 10 ago. 2025.

VASWANI, Ashish; *et al*. Attention is All you Need. *In*: **Advances in Neural Information Processing Systems**. [s.l.]: Curran Associates, Inc., 2017, v. 30. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>. Acesso em: 10 ago. 2025.

WANG, Zichong *et al*. History, development, and principles of large language models: an introductory survey. **AI and Ethics**, 2024. Disponível em: <https://doi.org/10.1007/s43681-024-00583-7>. Acesso em: 20 fev. 2025.

XIA, Chunqiu Steven *et al*. Fuzz4All: Universal Fuzzing with Large Language Models. *In*: **Proceedings of the IEEE/ACM 46th International Conference on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2024, p. 1–13. (ICSE '24). Disponível em: <https://dl.acm.org/doi/10.1145/3597503.3639121>. Acesso em: 20 fev. 2025.

Palavras-chave: Compilador; LLM; Geração de código; Fuzzing; Teste Differencial;

Nº de Registro no sistema Prisma: PES-2024-0131

Financiamento

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL