16º Congresso de Inovação, Ciência e Tecnologia do IFSP - 2025

INTEGRAÇÃO DO CHATGPT NA OTIMIZAÇÃO DE ANALISADORES ESTÁTICOS

AUGUSTO D. SASAKI[1], LUIZ C. JUNIOR[2]

[1] Graduando em Bacharelado em Engenharia da Computação, Bolsista PIBIFSP, IFSP, Campus Piracicaba, augusto.sasaki@ifsp.aluno.edu.br.
[2] Doutorado em Ciência da Computação, UFSCar, cavamura@ifsp.edu.br.
Área de conhecimento (Tabela CNPq): 1.03.00.00-7 Ciência da Computação

RESUMO: Este trabalho investiga a viabilidade do uso do *ChatGPT* como ferramenta auxiliar na análise estática de código, com foco na ampliação do conjunto de alertas e na priorização daqueles com maior probabilidade de corresponderem a defeitos reais. Foram utilizados analisadores estáticos de código aberto para *Java* aplicados a projetos do repositório *Defects4J*. O *ChatGPT* foi instruído a reproduzir os resultados dessas ferramentas, além de agrupar, organizar e priorizar os alertas gerados. Os resultados demonstraram que 68,92% dos alertas eram exclusivos de cada ferramenta, enquanto 31,08% foram compartilhados, evidenciando que a combinação de diferentes analisadores ampliou a cobertura da análise. Quanto à priorização, dos 257 alertas identificados, 25 (9,72%) foram corretamente classificados como prioritários, com apenas 1,55% de falhas em alertas que deveriam receber prioridade. Contudo, 20,23% representaram falsas priorizações. Observou-se ainda que 76,92% dos defeitos conhecidos foram detectados total ou parcialmente. Conclui-se que o *ChatGPT* apresenta potencial para otimizar a análise estática ao apoiar a identificação e organização de alertas, reduzindo esforços do desenvolvedor. Entretanto, limitações como variação de respostas, restrições de contexto e casos de defeitos não priorizados indicam que seu uso deve ocorrer de forma supervisionada.

PALAVRAS-CHAVE: inteligência artificial; qualidade de software; filtragem de alertas; revisão de código; chatbot;

INTEGRATION OF CHATGPT IN THE OPTIMIZATION OF STATIC ANALYZERS

ABSTRACT: This study investigates the feasibility of using ChatGPT as an auxiliary tool in static code analysis, focusing on expanding the set of alerts and prioritizing those most likely to correspond to real defects. Open-source static analyzers for Java were applied to projects from the Defects4J repository. ChatGPT was instructed to reproduce the results of these tools, as well as to group, organize, and prioritize the generated alerts. The results showed that 68.92% of the alerts were exclusive to each tool, while 31.08% were shared, highlighting that the combination of different analyzers broadens the coverage of the analysis. Regarding prioritization, of the 257 identified alerts, 25 (9.72%) were correctly classified as high priority, with only 1.55% of failures in alerts that should have received priority. However, 20.23% represented false prioritizations. It was also observed that 76.92% of the known defects were detected either fully or partially. The study concludes that ChatGPT has the potential to optimize static analysis by supporting the identification and organization of alerts, thus reducing developer effort. Nevertheless, limitations such as response variability, context restrictions, and cases of non-prioritized defects indicate that its use should occur under supervision.

KEYWORDS: artificial intelligence; software quality; alert filtering; code review; chatbot.

INTRODUÇÃO

A análise estática de código é uma prática fundamental na engenharia de software, voltada à identificação de defeitos e vulnerabilidades no código fonte, contribuindo para a melhoria da qualidade de sistemas. Apesar de sua importância, essa técnica enfrenta desafios, tais como a variação de resultados entre diferentes ferramentas e a alta incidência de falsos positivos, esses fatores

dificultam a interpretação e priorização dos alertas pelos desenvolvedores. Nesse contexto, o uso de *chatbots* de inteligência artificial, como o *ChatGPT*, surge como uma potencial alternativa para o aprimoramento da análise estática, permitindo o agrupamento e filtração dos alertas de diferentes fontes, aumento da taxa de verdadeiros positivos e redução de falsos positivos.

A pesquisa propõe um estudo de viabilidade que combina a revisão de literatura e experimentação prática, utilizando analisadores estáticos e conjuntos de código com falhas conhecidas, de forma a avaliar de maneira sistemática o potencial do *ChatGPT* na melhoria do processo de análise de código. Para guiar a investigação, foram definidas as seguintes questões de pesquisa:

RQ1: O *chatbot ChatGPT* é capaz ou não de ampliar o conjunto de alertas gerando um domínio de falhas estendido a partir dos resultados provenientes de diferentes analisadores estáticos?

RQ2: O *chatbot ChatGPT* é capaz ou não de identificar os alertas com maior probabilidade de serem verdadeiros-positivos e, assim, estabelecer um critério de ranqueamento dos alertas baseado nessa probabilidade?

MATERIAL E MÉTODOS

Esta pesquisa estruturou seu processo metodológico no estudo de viabilidade da aplicação do *ChatGPT* (CHATGPT, 2022) como recurso auxiliar em análise estática de código. Inicialmente, foram realizadas buscas bibliográficas para identificar estudos relevantes que tratassem da relação entre inteligência artificial e análise estática de código. Foram utilizadas as bases de dados Scopus e ACM Digital Library, empregando combinações de termos como “Static Analysis”, “ChatGPT”, “Prioritizing” e “Warnings”, conforme estudos que investigam o uso de chatbots em análise estática (OZTURK et al., 2023) e a priorização de alertas em ferramentas automatizadas (CAVAMURA JÚNIOR et al., 2021). Durante as buscas, foi realizada a leitura exploratória com o objetivo de selecionar apenas os trabalhos que possuíam as características adequadas ao tema central da pesquisa. Em seguida, procedeu-se a leitura seletiva, nela foram identificados dois artigos que mais se alinham à proposta do estudo.

New Tricks to Old Codes: Can AI Chatbots Replace Static Code Analysis Tools? avalia a aplicação do ChatGPT como alternativa a analisadores estáticos para a detecção de vulnerabilidades de segurança em aplicações web. *WarningsFIX: a Recommendation System for Prioritizing Warnings Generated by Automated Static Analyzers* propõe estratégias de priorização de alertas visando reduzir a sobrecarga dos desenvolvedores diante de grandes volumes de alertas gerados por analisadores estáticos.

Esses estudos selecionados serviram de referência para a seleção inicial das ferramentas de análise empregadas e o estabelecimento dos critérios de priorização considerados nesta pesquisa. Além disso evidenciam duas lacunas: a ausência de investigações sobre a aplicação do ChatGPT para a integração de diferentes ferramentas de análise estática tradicionais e a falta de propostas que façam uso de modelos de linguagem natural para priorização de alertas. O presente estudo diferencia-se ao investigar essa integração de ferramentas e nova abordagem de priorização, visando apoiar o desenvolvedor de forma mais eficiente.

Na etapa prática da pesquisa, foram selecionados analisadores estáticos gratuitos e de código aberto para a linguagem *Java*, sendo escolhidos os seguintes: *SpotBugs* (SPOTBUGS, 2015), *PMD* (PMD, 2002) e *Checkstyle* (CHECKSTYLE, 2001). A definição destes se baseou na relevância dessas ferramentas na comunidade acadêmica e profissional, bem como na possibilidade de integração com a inteligência artificial utilizada. Foi utilizado como conjunto de dados o repositório *Defects4J* (Defects4J*, 2014)*, composto por projetos Java com bugs reais documentados e classificados como verdadeiros-positivos. Para a realização dos experimentos, foram selecionados após estudo, especificamente os projetos *Cli* (commons-cli) e *Lang* (commons-lang), por apresentarem um menor número de linhas de código e menor complexidade estrutural em comparação a outros sistemas do repositório.

Os experimentos realizados consistiram em interações iterativas com o *ChatGPT* 3.5, modelo de linguagem desenvolvido pela *OpenAI*. Foram fornecidos trechos de código dos projetos selecionados e instruções explícitas para que o *ChatGPT* atuasse como cada um dos analisadores estáticos definidos. O padrão de instrução adotado foi: “Atue como a ferramenta de análise estática (*PMD*, *SpotBugs* ou *Checkstyle*) e faça a análise do código acima, identificando defeitos no código,

sua resposta deve ser exatamente a mesma que a ferramenta mencionada forneceria". Após isso, as respostas geradas foram organizadas e sistematizadas.

RESULTADOS E DISCUSSÃO

O *ChatGPT* foi instruído a agrupar os alertas identificados, verificando quais ferramentas os haviam reportado e classificá-los em níveis de prioridade, conforme demonstrado na Figura 1. Essa sistematização permitiu a construção do diagrama da Figura 2 e tabelas comparativas que serviram de base para a análise dos resultados obtidos.

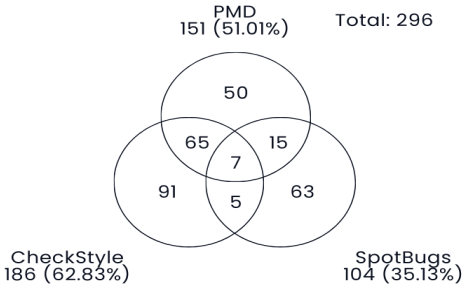
FIGURA 1. Exemplo de tabela gerada pelo *ChatGPT* exibindo os resultados da análise estática.

LINHA	ALERTA	PMD	SPOTBUGS	CHECKSTYLE	PRIORIDADE
30	Null Dereference (str might be null)	Não	Sim	Não	ALTA
35	Duplicate String Literal	Sim	Não	Não	ALTA
28	Simplify Boolean Return	Sim	Não	Não	BAIXA
46	Possible Heap Pollution (incorrect casting)	Não	Sim	Não	MÉDIA
31	Line exceeds 80 characters	Não	Não	Sim	BAIXA
38, 44	Cyclomatic complexity too high	Não	Não	Sim	MÉDIA

A RQ1 investigou se o ChatGPT é capaz de ampliar o conjunto de alertas a partir de resultados de diferentes analisadores estáticos. Observou-se que 68,92% dos alertas eram exclusivos de cada ferramenta, enquanto 31,08% foram detectados por mais de uma ferramenta simultaneamente. O baixo número de ocorrências de alertas compartilhados entre as três ferramentas sugere que há uma grande diversidade nos critérios de análise de cada uma, ou seja, no seu conjunto de regras, em especial entre *SpotBugs* e as demais, indicando que não há uma ferramenta única capaz de capturar todos os possíveis problemas.

Além disso, a quantidade de alertas únicos para cada ferramenta mostra que ao combinar diferentes analisadores estáticos, é possível ampliar a cobertura da análise, isso reforça a hipótese sugerida nos de que o *ChatGPT* pode contribuir na detecção de maior número de alertas ao agregar resultados provenientes de diversos analisadores estáticos. Portanto, os resultados obtidos proveem resposta à questão de pesquisa RQ1, pois demonstram que o *ChatGPT* foi capaz de expandir o conjunto de alertas usando alertas provenientes de analisadores distintos e mostrou-se capaz de eliminar redundâncias encontradas nos casos onde mais de uma ferramenta gera o mesmo alerta.

FIGURA 2. Diagram de Venn comparando os alertas obtidos pelo uso de diferentes ferramentas de análise estática com o *ChatGPT*.



A RQ2 avaliou se o *ChatGPT* é capaz de identificar os alertas com maior probabilidade de serem verdadeiros-positivos e estabelecer um critério de ranqueamento baseado nessa probabilidade. Entre 39 defeitos conhecidos em 34 códigos analisados, 30 (76,92%) foram encontrados ou parcialmente encontrados, enquanto 9 (23,07%) passaram despercebidos. Os resultados parcialmente encontrados referem-se a casos em que o *ChatGPT* conseguiu detectar o defeito, mas com alguma imprecisão, seja na linha exata do código ou na categoria do erro.

Observa-se que, embora o *ChatGPT* apresente boa capacidade de detecção, sua precisão depende da complexidade do código e da natureza do defeito. A quantidade de defeitos parcialmente

identificados indica que o modelo consegue reconhecer padrões de erro, mas nem sempre fornece uma análise completa. Durante o estudo, verificou-se que quanto maior a complexidade do código e quanto menos isolado estiver o defeito, maior a probabilidade de compreensão parcial do erro ou ocorrência de falso negativo.

TABELA 1. Resultados da detecção de defeitos de código pela análise executada utilizando *ChatGPT*.

Total de Códigos	Total de Defeitos	Encontrados*	Parcialmente Encontrados*	Não encontrados**
34	39	14	16	9

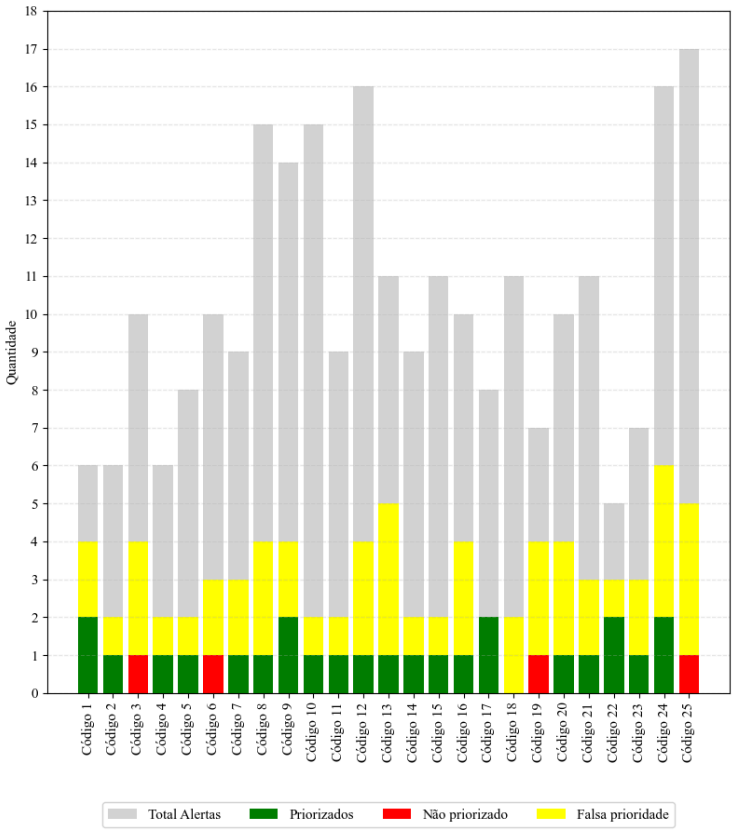
*:alertas verdadeiros-positivos.
**:alertas falso-negativo.

A priorização realizada apresentou alta acurácia, com apenas 1,55% de falhas em alertas que deveriam receber prioridade. Para os casos em que o defeito foi detectado, os alertas foram classificados em três categorias de priorização:

- Priorizado: alerta de defeito real classificado como alta prioridade;
- Não priorizado: verdadeiro-positivo encontrado, mas sem alta prioridade;
- Falsa priorização: falso-positivo erroneamente indicado como possível verdadeiro-positivo.

O gráfico 1 apresenta a análise detalhada da priorização dos alertas realizada pelo *ChatGPT*. Ao todo, foram gerados 257 alertas, dos quais 25 (9,72%) foram corretamente classificados como priorizados. Apenas 4 alertas (1,55%) que deveriam receber prioridade não foram identificados como tal, indicando uma baixa taxa de falhas. Por outro lado, houve 52 alertas (20,23%) classificados erroneamente como priorizados, configurando falsas prioridades. Considerando a priorização em relação à quantidade de alertas de cada código, 29,96% dos alertas foram tratados como de alta prioridade. Esses resultados demonstram que o *ChatGPT* consegue identificar quase todos os alertas relevantes e fornecer uma priorização confiável, embora ainda existam limitações na distinção precisa de prioridade, principalmente devido à atribuição de importância baseada em padrões superficiais sem levar em conta o contexto específico do código. Em alguns casos, isso fez com que alertas fossem considerados críticos mesmo quando seu impacto real era menor ou inexistente.

GRÁFICO 1. Priorização de Alertas Gerados pelo *ChatGPT*



Por se tratar de um modelo de linguagem baseado em probabilidade, o *ChatGPT* pode gerar respostas diferentes mesmo para entradas idênticas ou pequenas alterações na formulação da pergunta ou no histórico da conversa. No contexto de análise estática, isso é uma limitação, pois compromete a consistência e reprodutibilidade dos resultados. Diferente das ferramentas tradicionais, que produzem os mesmos alertas para uma mesma entrada garantindo previsibilidade, o *ChatGPT* pode apresentar variações que dificultam a confiabilidade da análise.

Outra limitação relevante é o limite de tokens e sua capacidade de compreensão do contexto. A versão 3.5 utilizada nesta pesquisa suporta até 4096 tokens por mensagem, considerando tanto a entrada quanto a resposta gerada. Em projetos de software com múltiplos arquivos, classes e métodos complexos, esse limite impede a análise completa, restringindo-a a trechos específicos de código. Além disso, muitos defeitos de software não estão isolados em um único trecho, sendo necessário compreender a interação entre diferentes partes do sistema. Embora seja possível contornar parcialmente esse limite enviando múltiplas mensagens, a compreensão do contexto diminui à medida que o código se torna mais extenso e complexo, afetando a precisão da análise.

CONCLUSÕES

O presente estudo teve como objetivo avaliar a viabilidade do uso do *ChatGPT* na análise estática de código, considerando a ampliação do conjunto de alertas e a priorização daqueles com maior probabilidade de serem verdadeiros-positivos. Os resultados obtidos demonstram que o modelo de inteligência artificial cumpre consistentemente o objetivo de integrar alertas de diferentes analisadores, ampliando a cobertura de falhas em relação à execução isolada de ferramentas convencionais.

Além disso, a análise da priorização evidencia que o *ChatGPT* é capaz de orientar de forma eficaz a revisão de código, identificando a maioria dos alertas críticos, no entanto, a presença de falsas priorizações indica limitações na distinção precisa do contexto do código, destacando que o modelo tende a se basear em padrões superficiais ao definir a relevância dos alertas. Adicionalmente, a ocorrência de defeitos reais não priorizados, ainda que rara, indica que o *ChatGPT* não pode ser utilizado de forma totalmente autônoma, pois existe o risco de que alguns erros no software permaneçam não detectados.

Conclui-se que o uso de chatbots de inteligência artificial na análise estática pode otimizar o processo de inspeção de código, oferecendo suporte ao desenvolvedor na identificação de defeitos reais e na redução do tempo gasto com alertas irrelevantes. Ao mesmo tempo, apontam para desafios a serem superados, como a consistência de respostas e as restrições de compreensão de contexto em códigos extensos e complexos, que devem ser considerados para uma aplicação prática eficiente.

CONTRIBUIÇÕES DOS AUTORES

A.D.S. foi responsável pela curadoria e análise dos dados, a aplicação da metodologia, validação dos resultados e a redação do trabalho. L.C.J. contribuiu com a conceitualização da pesquisa e supervisão do estudo.

Ambos os autores contribuíram com a revisão do trabalho e aprovaram a versão submetida.

AGRADECIMENTOS

Agradeço ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - Câmpus Piracicaba, bem como ao meu professor orientador, pelo apoio e pela oportunidade de participação no programa de iniciação científica, que possibilitou o meu desenvolvimento acadêmico e científico. Agradeço também ao Programa Institucional de Bolsas de Iniciação Científica e Tecnológica do IFSP (PIBIFSP) pelo suporte financeiro e pela experiência proporcionada.

REFERÊNCIAS

CAVAMURA, J. L.; BELGAMO, A.; MENDONÇA, V. R. L.; VINCENZI, A. M. R. WarningsFIX: a Recommendation System for Prioritizing Warnings Generated by Automated Static Analyzers. **Proceedings of the XIX Brazilian Symposium on Software Quality (SBQS '20)**. ACM, New York, NY, USA, p. 1–10, 2021.

OPENAI. **ChatGPT**, 2022. Chatbot que utiliza inteligência artificial e processamento de linguagem natural possibilitando conversas semelhantes às humanas. Acesso em: 05 mar. 2024. Online. Disponível em: https://chatgpt.com/.

CHECKSTYLE. **Checkstyle**, 2001. Ferramenta para auxílio na programação em Java que adere a um padrão de codificação. Página do projeto. Acesso em: 20 mar. 2024. Online. Disponível em: https://checkstyle.sourceforge.io.

RJUST. **Defects4j**, 2015. Biblioteca de código aberto para benchmarking de ferramentas de análise de defeitos em código Java. Repositório do projeto. Acesso em: 17 ago. 2025. Online. Disponível em: https://github.com/rjust/defects4j.

PMD. **Pmd**, 2002**.** Analisador de código estático multilinguagem. Página do projeto. Acesso em: 20 mar. 2024. Online. Disponível em: https://pmd.github.io.

SPOTBUGS. **SpotBugs**, 2016. Uma ferramenta de análise estática para detecção de erros em código Java. Página do projeto. Acesso em: 20 mar. 2024. Online. Disponível em: https://spotbugs.github.io.

OZTURK, O. S.; EKMEKCIOGLU, E.; CETIN, O.; ARIEF, B.; HERNANDEZ-CASTRO, J. New Tricks to Old Codes: Can AI Chatbots Replace Static Code Analysis Tools?. **European Interdisciplinary Cybersecurity Conference (EICC 2023)**, Stavanger, Norway, 14–15. 2023.