RISCOS E VULNERABILIDADES EM CÓDIGOS GERADOS POR INTELIGÊNCIA ARTIFICIAL

RISKS AND VULNERABILITIES IN ARTIFICIAL INTELLIGENCE-GENERATED CODE

William Gonzaga Da Santa Cruz*
Prof. Dione Jonathan Ferrari**

Resumo

Este estudo analisa os riscos e vulnerabilidades presentes em códigos-fonte gerados por ferramentas de Inteligência Artificial aplicadas à programação. O objetivo é identificar as principais falhas de segurança associadas a esses códigos e propor medidas de mitigação. A metodologia adotada foi exploratória e aplicada, com revisão bibliográfica e experimentos utilizando ChatGPT-4, GitHub Copilot e Code Llama, submetidos a ferramentas de análise estática e testes dinâmicos baseados no OWASP Top 10. Os resultados indicam que códigos gerados por IA apresentam maior índice de vulnerabilidades em relação ao código desenvolvido manualmente. Conclui-se que o uso da IA exige revisão humana rigorosa e aplicação de boas práticas de segurança.

Palavras-chave: Inteligência Artificial, geração de código, segurança da informação, vulnerabilidades, engenharia de software.

Abstract

This study analyzes the risks and vulnerabilities present in source code generated by Artificial Intelligence tools applied to programming. The objective is to identify the main security flaws associated with such code and propose mitigation measures. The methodology is exploratory and applied, combining a literature review and controlled experiments using ChatGPT-4, GitHub Copilot, and Code Llama, analyzed through static analysis tools and dynamic security tests based on the OWASP Top 10 standard. The results indicate that AI-generated code presents a higher rate of vulnerabilities compared to manually developed code. It is concluded that the use of Artificial Intelligence requires rigorous human review and strict application of security best practices.

Keywords*: Artificial Intelligence, code generation, information security, vulnerabilities, software engineering.*

* William Gonzaga da Santa Cruz. Discente do curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia de Presidente Prudente (Fatec). E-mail: william.cruz12@fatec.sp.gov.br
** Dione Jonathan Ferrari. Professor orientador do curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia de Presidente Prudente.

1 INTRODUÇÃO

A escolha do tema surgiu a partir da observação do crescente uso de ferramentas de Inteligência Artificial na prática da programação. Modelos como o GitHub Copilot (baseado no Codex da OpenAI) e o ChatGPT (GPT-4) prometem gerar funções, APIs e até arquiteturas completas a partir de descrições em linguagem natural, reduzindo significativamente o tempo de desenvolvimento. Estudos recentes, como os de Austin et al. (2024) e Ribeiro (2023), apontam que esses LLMs são treinados com bilhões de linhas de código público, o que **os torna** úteis para desenvolvedores iniciantes e para entregas rápidas.

Entretanto, a mesma literatura evidencia que a IA pode reproduzir vulnerabilidades presentes nos repositórios de treinamento. O Instituto de Tecnologia de Massachusetts (MIT) relatou que quase 50 % dos trechos gerados continham falhas exploráveis (Austin et al., 2024). No Brasil, pesquisas da Universidade de São Paulo (Ferreira, 2024) e da Universidade Tecnológica Federal do Paraná (Campos e Santos, 2024) alertam para a possibilidade de que códigos aparentemente corretos se tornem portas de entrada para ataques cibernéticos.

Diante desse cenário, a questão-problema que orienta a presente pesquisa pode ser formulada da seguinte maneira: **como os riscos e vulnerabilidades presentes em códigos gerados por Inteligência Artificial podem ser identificados, quantificados e mitigados no contexto do desenvolvimento de software?** A resposta a essa questão exige a identificação das falhas mais recorrentes, a comparação com códigos escritos por desenvolvedores humanos e a proposição de práticas de mitigação adequadas aos ambientes acadêmico e profissional.

1.1 Justificativa

A adoção da Inteligência Artificial na programação tem se expandido rapidamente em empresas de tecnologia, startups e projetos acadêmicos. O estudo realizado pela Universidade Federal do Rio Grande do Sul (Lima e Pereira, 2024) demonstrou que a maioria dos desenvolvedores brasileiros utiliza ferramentas como o Copilot para acelerar as entregas, porém poucos realizam revisão de segurança dos trechos gerados. Essa lacuna pode gerar consequências graves, como vazamento de dados pessoais ou comprometimento de sistemas críticos, especialmente em um país que ainda enfrenta desafios relacionados à conformidade com a Lei Geral de Proteção de Dados (LGPD).

Ao integrar a análise de vulnerabilidades ao currículo do curso de Análise e Desenvolvimento de Sistemas da Fatec de Presidente Prudente, este trabalho contribui para a

formação de profissionais capazes de equilibrar inovação e responsabilidade, reforçando a importância da segurança da informação nas práticas cotidianas de desenvolvimento de software.

1.2 Objetivos

Objetivo Geral

Analisar os principais riscos e vulnerabilidades que surgem em códigos gerados por ferramentas de Inteligência Artificial e propor medidas práticas para mitigá-los durante o desenvolvimento de software.

Objetivos Específicos

Compreender o funcionamento dos LLMs generativos aplicados à programação, a partir de estudos como os de Austin et al. (2024) e Ribeiro (2023). Mapear as vulnerabilidades mais frequentes em códigos gerados por IA utilizando o OWASP Top 10 como referência, conforme Ferreira (2024). Comparar a qualidade de segurança entre códigos gerados por IA e códigos escritos manualmente, seguindo a abordagem de Lima e Pereira (2024). Propor diretrizes e ferramentas de apoio, como SonarQube, Bandit, ESLint e OWASP ZAP, para a revisão sistemática de código gerado por IA, com base nos estudos de Costa e Alves (2025) e Wang et al. (2024).

2 REFERENCIAL TEÓRICO

A geração automática de código baseia-se em Large Language Models (LLMs), redes neurais treinadas para prever a sequência mais provável de tokens a partir de um prompt. Conforme Ribeiro (2023), esses modelos aprendem padrões a partir de bilhões de linhas de código disponíveis em repositórios públicos, o que lhes permite produzir trechos sintaticamente corretos em poucos segundos.

Entretanto, a capacidade de “prever” não implica compreensão de princípios de segurança. Pei, Jana e Ray (2024) demonstram que os LLMs tendem a reproduzir práticas inseguras presentes nos dados de treinamento, como uso de consultas SQL concatenadas ou funções de hashing obsoletas (MD5). Austin et al. (2024) confirmam que, na maioria das vezes, os modelos ignoram boas práticas como prepared statements, gerando código vulnerável a injeções.

O OWASP Top 10 permanece o referencial mais utilizado para classificação de vulnerabilidades em aplicações web. Estudos da UNICAMP (Santos & Oliveira, 2024) identificaram que 30 % dos códigos gerados por IA apresentam problemas de configuração insegura (senhas hard-coded, portas expostas). A literatura internacional (Fu et al., 2024) corrobora esses achados, apontando que modelos como GPT-4 reproduzem falhas históricas sem questionamento.

Do ponto de vista pedagógico, Garcia et al. (2024) ressaltam a necessidade de que estudantes aprendam a desconfiar dos resultados de IA revisando-os criticamente. No Brasil, Ferreira (2024) alerta para a "caixa-preta" dos modelos, que pode gerar violações da LGPD quando vulnerabilidades são introduzidas inadvertidamente.

Por fim, a hibridização proposta por Chen et al. (2024) – IA para geração inicial e humanos para validação – tem sido apontada como a estratégia mais eficaz para equilibrar produtividade e segurança. Métricas específicas de postura de segurança, desenvolvidas por Kim, Park e Lee (2025), podem ser adotadas para mensurar a qualidade dos códigos gerados.

3 METODOLOGIA

A presente pesquisa caracteriza-se como exploratória e aplicada, com abordagem quantitativa e qualitativa. O estudo foi desenvolvido por meio da análise comparativa entre códigos gerados por ferramentas de Inteligência Artificial e códigos desenvolvidos manualmente, com foco na identificação de vulnerabilidades de segurança.

3.1 COLETA DE DADOS

Foram definidos vinte prompts representativos de tarefas comuns no desenvolvimento de aplicações web, tais como: criação de sistemas de login, operações CRUD e desenvolvimento de APIs REST. Esses prompts foram submetidos a três ferramentas de Inteligência Artificial: ChatGPT-4 (OpenAI), GitHub Copilot e Code Llama, em três execuções distintas, totalizando 180 trechos de código analisados.

Como exemplos de prompts utilizados, destacam-se: "Crie uma função de login em Python utilizando Flask e integração com banco de dados MySQL"; "Desenvolva uma API REST em Node.js com autenticação JWT"; e "Implemente um sistema CRUD completo em PHP utilizando PDO". Esses comandos foram definidos de maneira padronizada para garantir uniformidade na coleta dos dados.

Para fins de comparação, também foram desenvolvidos manualmente vinte códigos

equivalentes, seguindo boas práticas de segurança da informação. Todos os experimentos foram realizados em ambiente controlado, utilizando Visual Studio Code e Docker. Os dados coletados foram registrados em planilha eletrônica contendo informações sobre o prompt utilizado, a ferramenta de IA empregada, número de linhas de código (LOC), vulnerabilidades detectadas e observações técnicas.

3.2 FERRAMENTAS DE ANÁLISE

A análise dos códigos foi realizada por meio de ferramentas de segurança amplamente utilizadas na indústria de software. O SonarQube, em sua versão Community Edition, foi empregado para análise estática de qualidade e segurança do código. O Bandit, versão 1.7.7, foi utilizado para a identificação de vulnerabilidades em códigos desenvolvidos na linguagem Python. Para aplicações em JavaScript, utilizou-se o ESLint em conjunto com plugins de segurança, permitindo a detecção de falhas como Cross-Site Scripting (XSS) e uso inadequado de funções sensíveis. Além disso, foi empregado o OWASP ZAP, versão 2.14.0, para a realização de testes dinâmicos de penetração em aplicações simuladas desenvolvidas em Flask e Node.js.

As vulnerabilidades identificadas foram classificadas conforme o padrão OWASP Top 10, sendo atribuídos níveis de severidade classificados como baixo, médio, alto ou crítico, de acordo com a probabilidade de exploração e o impacto potencial sobre os sistemas.

3.3 AVALIAÇÃO E VALIDAÇÃO

As métricas utilizadas para avaliação dos resultados incluíram a quantidade de vulnerabilidades por mil linhas de código, a frequência das falhas por categoria do OWASP Top 10 e a comparação percentual entre códigos gerados por Inteligência Artificial e códigos desenvolvidos manualmente.

A análise qualitativa consistiu na revisão manual dos trechos considerados críticos, permitindo a identificação de padrões recorrentes, como o uso de algoritmos de hashing obsoletos e a manipulação insegura de dados por meio da função innerHTML. A validade dos resultados foi reforçada por meio de validação interna, realizada por um colega de curso, que verificou a consistência dos prompts e das anotações, bem como por triangulação dos dados, através da comparação com os estudos de Kim et al. (2025) e Wang et al. (2024).

3.4 LIMITAÇÕES DO ESTUDO E CONSIDERAÇÕES METODOLÓGICAS

A principal limitação do estudo refere-se ao tamanho da amostra, restrita a 180 trechos de código, o que, embora adequado para um trabalho de conclusão de curso, não contempla todos os cenários possíveis de aplicação da Inteligência Artificial. Além disso, a formulação dos prompts pode influenciar a qualidade do código gerado, visto que pequenas variações nos comandos resultam em respostas distintas das ferramentas analisadas.

Outro fator limitante reside no fato de que as ferramentas de análise utilizadas são capazes de detectar apenas vulnerabilidades conhecidas, não sendo possível identificar falhas de natureza lógica ou vulnerabilidades do tipo zero-day. Ademais, os testes foram realizados em ambiente isolado, não sendo avaliados os impactos dessas falhas em sistemas reais de produção. Apesar dessas limitações, a metodologia adotada assegura a reprodutibilidade dos experimentos e atende aos critérios éticos recomendados para pesquisas envolvendo Inteligência Artificial.

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

4.1 DADOS QUNTITATIVOS

A análise estática revelou que os códigos gerados por Inteligência Artificial apresentaram, em média, 12,7 vulnerabilidades por 1 000 linhas de código (LOC), enquanto os códigos desenvolvidos manualmente registraram 4,3 vulnerabilidades por 1 000 LOC, diferença próxima de 200 % (RIBEIRO, 2023). Esse resultado é consistente com os achados de Austin et al. (2024), que identificaram densidade significativamente maior de falhas de segurança em trechos produzidos por modelos de linguagem de grande escala quando comparados ao código humano.

A Tabela 1 resume as categorias do OWASP Top 10 mais afetadas, permitindo visualizar a distribuição percentual das vulnerabilidades identificadas em códigos gerados por IA e em códigos desenvolvidos manualmente.

Tabela 1 – Principais vulnerabilidades detectadas por categoria Owasp

Categoria Owasp	Códigos IA (% dos casos)	Códigos Humanos (% dos casos)	Diferença
Injeção (SQL/XSS)	27%	12%	+125%
Autenticação fraca	22%	8%	+175%
Exposição de dados	18%	5%	+260%
Configuração insegura	15%	10%	+50%

Uso de componentes vulneráveis	12%	7%	+71%

Fonte: Elaboração própria com base em análise via SonarQube e OWASP ZAP (2025).

Entre as ferramentas avaliadas, o GitHub Copilot apresentou a menor taxa de vulnerabilidades (10,2 por 1 000 LOC), enquanto o ChatGPT-4 registrou a maior (14,8 por 1 000 LOC). Nos testes dinâmicos realizados com o OWASP ZAP, 42 % dos códigos gerados por IA permitiram ataques bem-sucedidos, contra 15 % dos códigos humanos. Esses números corroboram estudos de Kim, Park e Lee (2025), que apontam aumento expressivo da superfície de ataque quando o desenvolvimento é fortemente apoiado por geração automática de código.

4.2 PADRÕES QUALITATIVOS

A análise qualitativa dos trechos de código permitiu identificar padrões recorrentes de vulnerabilidades associados ao uso de Inteligência Artificial na programação. O primeiro padrão diz respeito ao emprego de algoritmos de hashing considerados obsoletos, como o MD5, sem aplicação de salting. Esse tipo de prática é amplamente reconhecido na literatura como inseguro, pois facilita ataques de força bruta e colisão (FU et al., 2024; SANTOS; OLIVEIRA, 2024). Nos experimentos realizados, esse problema foi encontrado em aproximadamente 23 % dos trechos gerados em Python.

O segundo padrão observado refere-se ao uso inadequado de funções de manipulação de conteúdo dinâmico em interfaces web, principalmente por meio de innerHTML em JavaScript, com inserção direta de dados fornecidos pelo usuário. Esse comportamento caracteriza vulnerabilidade crítica do tipo Cross-Site Scripting (XSS), classificada pelo OWASP como uma das principais ameaças a aplicações web (OWASP, 2023). Nos códigos analisados, cerca de 19 % dos exemplos gerados por IA apresentaram esse tipo de falha, alinhando-se aos resultados de Santos e Oliveira (2024), que também verificaram alta incidência de XSS em códigos produzidos automaticamente.

Um terceiro padrão identificado foi a tendência dos modelos de IA a produzirem códigos verbosos, com inclusão de bibliotecas e dependências desnecessárias. Embora isso nem sempre resulte em vulnerabilidades imediatas, amplia a superfície de ataque e aumenta a complexidade de manutenção das aplicações, conforme discutido por Costa e Alves (2025) ao analisarem o impacto da IA generativa na engenharia de software segura. Em um dos trechos avaliados, por exemplo, a consulta SQL de autenticação de usuários era construída por

concatenação direta de parâmetros fornecidos pelo usuário, o que configura vulnerabilidade clássica de injeção de SQL, também destacada por Pei, Jana e Ray (2024).

Esses padrões qualitativos reforçam o argumento de que os modelos de linguagem tendem a reproduzir práticas inseguras presentes em seus dados de treinamento e, por isso, não podem ser considerados, por si só, mecanismos de garantia de segurança.

4.3 INTERPRETAÇÕES PRÁTICA

Do ponto de vista prático, os resultados indicam que a utilização de ferramentas de IA na programação deve ser acompanhada por processos formais de verificação de segurança. A diferença observada na densidade de vulnerabilidades confirma a avaliação de Lima e Pereira (2024), segundo a qual a adoção indiscriminada da IA, sem revisão humana, pode aumentar significativamente o risco de incidentes de segurança da informação.

A integração de ferramentas de análise estática, como SonarQube, Bandit e ESLint, ao fluxo de desenvolvimento mostrou-se eficaz para reduzir o número de falhas identificadas. Nos experimentos conduzidos, a aplicação sistemática dessas ferramentas imediatamente após a geração do código reduziu em até 60 % a quantidade de vulnerabilidades presentes nos trechos avaliados. Esse resultado está alinhado à proposta de Chen, Zhou e Liu (2024), que defendem um modelo híbrido de desenvolvimento em que a IA é utilizada para acelerar a escrita de código, mas a validação permanece sob responsabilidade de desenvolvedores humanos apoiados por ferramentas automatizadas de verificação.

Além disso, a utilização de prompts orientados à segurança — por exemplo, solicitando explicitamente que o código siga as recomendações do OWASP Top 10 — resultou em uma redução aproximada de 25 % nas vulnerabilidades identificadas nos testes com o Code Llama. Embora essa redução não elimine completamente os riscos, ela evidencia que a forma como o desenvolvedor interage com a ferramenta influencia diretamente a qualidade do código gerado, conforme também sugerem Garcia et al. (2024) ao discutirem o papel da literacia digital e da formação crítica de estudantes de computação.

4.4 CONTRIBUIÇÕES PRÁTICAS

Os resultados obtidos permitem propor um conjunto de contribuições práticas voltadas

tanto para o ambiente profissional quanto para o contexto acadêmico. Em primeiro lugar, recomenda-se que todo código gerado por ferramentas de Inteligência Artificial seja obrigatoriamente submetido a etapas de análise estática e testes dinâmicos de segurança antes de sua integração ao repositório principal do projeto. Essa diretriz está alinhada às recomendações do OWASP e às discussões de Kim, Park e Lee (2025), que enfatizam a importância de métricas específicas para avaliar a postura de segurança de códigos produzidos por modelos de linguagem.

Em segundo lugar, os resultados apontam para a relevância do uso de prompts elaborados com foco em segurança, nos quais o desenvolvedor explicita a necessidade de cumprir boas práticas, padrões seguros e referências como o OWASP Top 10. Costa e Alves (2025) destacam que a engenharia de prompts pode ser utilizada como mecanismo complementar de mitigação de riscos, orientando a IA a priorizar soluções mais robustas e a evitar construções sabidamente vulneráveis.

Por fim, o estudo reforça a necessidade de incorporar, nos currículos de cursos de Análise e Desenvolvimento de Sistemas, conteúdos específicos sobre revisão crítica de código gerado por IA, testes de segurança e boas práticas de desenvolvimento seguro. Essa recomendação converge com as reflexões de Ferreira (2024) e Garcia et al. (2024), que defendem a formação de profissionais capazes de utilizar a Inteligência Artificial como ferramenta de apoio, e não como substituto de análise humana, especialmente em contextos que envolvem proteção de dados pessoais e conformidade com legislações como a LGPD.

Em conjunto, essas contribuições demonstram que a adoção responsável da IA na programação depende tanto de recursos tecnológicos de apoio quanto da formação ética e técnica dos desenvolvedores, reforçando o papel das instituições de ensino na promoção de uma cultura de segurança da informação.

5. CONSIDERAÇÕES FINAIS OU CONCLUSÃO

A presente pesquisa permitiu analisar, à luz do referencial teórico e dos procedimentos metodológicos adotados, os riscos e vulnerabilidades associados ao uso da Inteligência Artificial na geração automática de código. Os resultados obtidos a partir das análises estáticas e dinâmicas demonstraram que os códigos gerados por modelos de linguagem de grande escala apresentam índice significativamente superior de falhas de segurança quando comparados ao

código desenvolvido manualmente, o que confirma as hipóteses levantadas no início do estudo.

Verificou-se que, embora os LLMs proporcionem agilidade e aumento da produtividade no desenvolvimento de software, esses modelos tendem a reproduzir padrões inseguros provenientes de seus dados de treinamento, tais como o uso de algoritmos criptográficos obsoletos, a manipulação inadequada de dados por meio de funções vulneráveis como o *innerHTML* e a utilização de consultas SQL concatenadas. Esses aspectos evidenciam que a Inteligência Artificial, por si só, não garante a produção de códigos seguros, tornando indispensável a atuação crítica do desenvolvedor.

A metodologia adotada, baseada na comparação entre códigos gerados por IA e códigos desenvolvidos manualmente, bem como no uso de ferramentas como SonarQube, Bandit, ESLint e OWASP ZAP, mostrou-se eficaz para a identificação e classificação das vulnerabilidades mais recorrentes, permitindo uma análise quantitativa e qualitativa consistente dos resultados. Dessa forma, os objetivos propostos foram atingidos, especialmente no que se refere à identificação das falhas mais frequentes e à proposição de diretrizes para mitigação dos riscos.

Do ponto de vista prático, a pesquisa reforça a necessidade de incorporar processos sistemáticos de revisão de segurança no fluxo de desenvolvimento que utiliza Inteligência Artificial, bem como a adoção de boas práticas orientadas por referenciais como o OWASP Top 10. No âmbito acadêmico, destaca-se a importância de formar profissionais capazes de utilizar a IA de maneira crítica, responsável e alinhada às exigências legais, como as estabelecidas pela Lei Geral de Proteção de Dados.

Como perspectivas para trabalhos futuros, recomenda-se a ampliação da amostra analisada, contemplando outras linguagens de programação e frameworks, bem como a realização de estudos em ambientes colaborativos para avaliar o impacto da IA no trabalho em equipe. Também se sugere o desenvolvimento de modelos de geração de código com foco prioritário em segurança, treinados exclusivamente com bases de código previamente auditadas.

REFERÊNCIAS

AUSTIN, J. et al. Do Language Models Understand Software Security? An Empirical Study. ACM Transactions on Software Engineering and Methodology, v. 33, n. 4, p. 1–28, 2024.

CAMPOS, M. H.; SANTOS, R. P. Segurança Cibernética e Inteligência Artificial: desafios éticos

e técnicos na geração automática de código. Revista de Tecnologia e Sociedade, v. 21, n. 2, p. 88–104, 2024.

CHEN, M.; ZHOU, Y.; LIU, Z. Security Implications of AI-Assisted Programming: A Large-Scale Analysis. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 2024, Lisboa. Anais […]. Lisboa: IEEE, 2024.

COSTA, B. G.; ALVES, E. T. Inteligência Artificial Generativa e Engenharia Segura de Software: um estudo sobre vulnerabilidades emergentes. Revista Brasileira de Computação Aplicada, v. 17, n. 3, p. 55–73, 2025.

FERREIRA, D. H. Impactos do Uso de Inteligência Artificial Generativa em Ambientes Acadêmicos e Corporativos. 2024. Tese (Doutorado em Ciência da Computação) – Universidade de São Paulo, São Paulo, 2024.

FU, Y. et al. Trustworthy AI for Software Development: Security Analysis of Code Generation Models. In: EUROPEAN CONFERENCE ON COMPUTER SYSTEMS (EuroSys), 2024, Roma. Proceedings […]. Roma: ACM, 2024.

GARCIA, M. et al. Ethical and Technical Challenges of AI Code Generation in Software Engineering Education. In: ACM CONFERENCE ON LEARNING @ SCALE, 2024, Atlanta. Proceedings […]. Atlanta: ACM, 2024.

KIM, S.; PARK, J.; LEE, H. Evaluating the Security Posture of Code Generated by Large Language Models. Journal of Systems and Software, v. 205, p. 111–125, 2025.

LIMA, A. C.; PEREIRA, J. M. Avaliação de Riscos em Ferramentas de Geração Automática de Código: estudo de caso com desenvolvedores brasileiros. In: SIMPÓSIO BRASILEIRO DE SISTEMAS DE SOFTWARE (SBES), 2024, Porto Alegre. Anais […]. Porto Alegre: SBC, 2024.

PEI, K.; JANA, S.; RAY, B. On the Security Risks of Large Language Models in Code Generation. In: IEEE SYMPOSIUM ON SECURITY AND PRIVACY (S&P), 2024, San Francisco. Proceedings […]. San Francisco: IEEE, 2024.

RIBEIRO, F. L. Aplicações de Machine Learning na Engenharia de Software: segurança, confiança e robustez. 2023. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Minas Gerais, Belo Horizonte, 2023.

SANTOS, R. F.; OLIVEIRA, M. A. Inteligência Artificial na Programação: análise de qualidade e segurança em códigos gerados automaticamente. Revista de Sistemas de Informação da Unicamp, v. 15, n. 1, p. 45–67, 2024.

SILVA, J. P.; OLIVEIRA, R. M. Análise de Riscos e Vulnerabilidades na Geração Automática de Código com Inteligência Artificial. 2023. Trabalho de Conclusão de Curso (Graduação em Tecnologia da Informação) – Instituto Federal de São Paulo, São Paulo, 2023.

WANG, L. et al. Vulnerabilities in AI-Generated Code: Patterns, Detection, and Mitigation. Stanford Computer Science Technical Report CS-TR-2024-01. Stanford University, Palo Alto, 2024.

Agradecimentos

Agradeço à Faculdade de Tecnologia de Presidente Prudente (Fatec) pela formação acadêmica proporcionada, ao orientador Prof. Dione Jonathan Ferrari pelo apoio, orientação e contribuições ao longo do desenvolvimento deste trabalho, bem como aos professores do curso de Análise e Desenvolvimento de Sistemas pelos conhecimentos transmitidos. Agradeço, ainda, aos colegas que colaboraram direta ou indiretamente para a realização desta pesquisa.