An Evaluation of the Impact of Code Generation Tools on Software Development

Luiz Fernando Mendes Osorio
Universidade Federal do Piauí
Teresina, Piauí, Brasil
fernando.mendes@ifpi.edu.br

Pedro de A. dos Santos Neto
Universidade Federal do Piauí
Teresina, Piauí, Brasil
pasn@ufpi.edu.br

Guilherme Avelino
Universidade Federal do Piauí
Teresina, Piauí, Brasil
gaa@ufpi.edu.br

Werney Ayala Luz Lira
Instituto Federal do Piauí
Teresina, Piauí, Brasil
werney@ifpi.edu.br

Abstract

Context: The rise of AI-assisted tools like GitHub Copilot aims to improve productivity in software development, raising questions about their practical impact on developer performance and code quality. **Problem:** While AI-assisted tools are promoted for enhancing productivity, empirical evidence on their real-world impact remains limited. This study addresses whether such tools improve programming workflows, focusing on developers with varying experience levels and examining effects on coding quality and efficiency. **Solution:** An empirical study is conducted to assess Copilot's effectiveness through experiments with student developers, focusing on task efficiency and code correctness, guided by established indicators identified in the literature. **IS Theory:** Grounded in the Task-Technology Fit (TTF) theory, this research explores the extent to which Copilot aligns with programming tasks and meets user needs, ensuring its relevance to practical development scenarios. **Method:** A literature review identified key indicators for evaluating AI-assisted code generation, focusing on task completion time and code correctness. Data collection involved task-based experiments with students, followed by quantitative analysis comparing performance with and without Copilot. **Summary of Results:** Findings indicate that Copilot can significantly reduce task completion time. However, no statistically significant differences were observed in code correctness, suggesting that while Copilot improves efficiency, its recommendations require careful review to ensure quality. **Contributions and Impact on IS Field:** This study contributes empirical insights into AI-assisted coding's advantages and challenges. The results inform academia and industry about best practices for implementing such tools effectively, emphasizing their potential for accelerating development while maintaining the need for human oversight.

CCS Concepts

• **Computing methodologies** → **Natural language generation**;
• **Software and its engineering** → **Software maintenance tools**.

Keywords

AI-assisted code generation, Copilot, empirical evaluation, developer performance.

1 Introdução

Os sistemas de informação assumiram um papel central na vida moderna, transformando desde as interações pessoais até os processos mais complexos de negócios e serviços públicos. Um sistema de informação é um conceito amplo, que envolve a integração de pessoas, processos, dados e tecnologia para apoiar a tomada de decisão e a gestão organizacional. Em nosso dia a dia, sistemas de informação controlam a forma como nos comunicamos, trabalhamos, nos divertimos e até cuidamos da saúde. Esse cenário demonstra como o sistema de informação se tornou indispensável para o funcionamento da sociedade moderna, sendo um elemento essencial para a continuidade e evolução das atividades econômicas e sociais [14, 18].

Com essa importância crescente, surge uma demanda exponencial por sistemas de informação mais robustos, eficientes e confiáveis. Organizações de todos os setores precisam desenvolver sistemas que sejam escaláveis, capazes de processar grandes volumes de dados e que ofereçam segurança e desempenho superior. E além disso, esses sistemas de informação precisam ser entregues rapidamente para atender à competitividade do mercado e às expectativas dos consumidores [28]. O tempo de desenvolvimento dos softwares ligados a sistemas de informação, antes espaçado e linear, é agora acelerado por metodologias, que integram continuamente novas funcionalidades e atualizações ao longo do ciclo de vida do sistema de informação [31]. A necessidade de inovação constante coloca pressão sobre as equipes de desenvolvimento para criar e aprimorar sistemas de informação em um ritmo muito mais veloz e com margens de erro reduzidas.

Em consonância com essa aceleração da necessidade de sistemas de informação, pode-se constatar o desenvolvimento de ferramentas voltadas ao apoio à programação [24]. Entre as recentes inovações estão as ferramentas de geração automática de código, que foram impulsionadas pelos avanços nos modelos de linguagem de grande escala (LLM, do inglês *Large Language Models*) [32]. Um exemplo comercial dessas ferramentas é o GitHub Copilot, um assistente de programação baseado em inteligência artificial, criado pela OpenAI em parceria com o GitHub [7].

Dada a importância crescente das ferramentas de geração automática de código no desenvolvimento de software, este estudo busca preencher uma lacuna na literatura acadêmica, investigando a eficácia prática dessas ferramentas em cenários reais. Embora muitos estudos explorem as capacidades técnicas dessas inovações,

a maioria limita-se a utilizar as ferramentas em problemas de programação simples e isolados, que não exigem a integração com outros métodos [22, 25]. Além disso, é comum que essas pesquisas avaliem códigos gerados a partir de repositórios, como o GitHub, sem envolver desenvolvedores na geração desses códigos, o que faz com que a análise seja realizada apenas pelo pesquisador que conduz o experimento [13]. Dessa forma, este trabalho propõe-se a fornecer evidências concretas sobre como essas ferramentas influenciam a dinâmica do desenvolvimento de software, oferecendo uma visão prática sobre seu uso no mercado [23, 24, 30, 33].

Com base na motivação apresentada, este trabalho busca gerar evidências empíricas sobre a eficácia das ferramentas de geração de código em cenários reais de desenvolvimento. Embora promovidas como soluções inovadoras para aprimorar o desempenho no desenvolvimento de software [28], ainda existem lacunas na literatura quanto à sua aplicação prática. O problema central, portanto, reside em validar se essas ferramentas realmente contribuem para otimizar o processo de programação. Além disso, é fundamental identificar tanto os benefícios quanto as limitações dessas tecnologias, assegurando que seu uso se baseie em resultados concretos e adequados ao contexto real de desenvolvimento [26].

O objetivo geral deste estudo é avaliar se as ferramentas de geração de código efetivamente auxiliam os desenvolvedores a programar de forma mais eficaz. Para alcançar esse objetivo, foram definidos os seguintes objetivos específicos:

- Identificar indicadores utilizados para avaliar a eficácia e o impacto de ferramentas de geração de código por IA no desenvolvimento de software.
- Desenvolver e conduzir um estudo empírico com desenvolvedores, visando avaliar o impacto dessas ferramentas no desempenho ao longo do processo de desenvolvimento de *software*.

A estrutura deste trabalho é organizada conforme descrito a seguir. A seção 2 apresenta o Referencial Teórico, onde são explorados os principais conceitos e os indicadores usados para avaliar ferramentas de geração de código. Na seção 3, são discutidos os Trabalhos Relacionados, abordando o estado da arte dessas ferramentas. A metodologia adotada para a realização deste estudo é descrita na seção 4, detalhando as fases de investigação, coleta de dados, análise e apresentação dos resultados. Na seção 6 são apresentadas as ameaças à validade dos achados detes trabalho e as ações para mitigar essas ameaças. Os resultados obtidos com o estudo empírico e suas análises são apresentados na seção 5, seguidos por uma discussão sobre as descobertas principais. Por fim, em seção 7, são expostas as conclusões do trabalho, bem como sugestões para pesquisas futuras.

2 Referencial Teórico

Esta seção apresenta trabalhos relevantes que abordam os conceitos e o uso de ferramentas de geração de código, descrevendo sua evolução, os indicadores utilizados para avaliar seu desempenho e os critérios para a construção de estudos empíricos.

2.1 Evolução das Ferramentas de Geração de Código

A evolução das ferramentas de geração de código reflete o progresso constante das técnicas de automação no desenvolvimento de *software*, buscando otimizar o processo de programação e reduzir o esforço humano em tarefas repetitivas.

2.1.1 Primeiras Ferramentas de Geração de Código. As primeiras ferramentas de geração de código surgiram no contexto do desenvolvimento dos compiladores, com o objetivo de traduzir linguagens de alto nível para código executável em linguagens de máquina. Pioneiras como o Lex e o Yacc, criadas nos anos 1970 [20], representaram avanços fundamentais ao permitir a geração automática de analisadores sintáticos e léxicos com base em especificações formais [20, 27].

Com o passar dos anos, as necessidades dos desenvolvedores evoluíram, e as ferramentas de suporte à codificação passaram a incorporar funcionalidades mais sofisticadas para automatizar etapas do processo de programação. Em ambientes de desenvolvimento integrado (IDEs), por exemplo, funcionalidades de autocompletar se tornaram indispensáveis [15, 19]. Além do autocompletar, as IDEs modernas passaram a incluir recursos avançados de navegação de código. Ferramentas para localização de declarações de elementos (como a navegação até a declaração de um método a partir de uma chamada) e para exibir todas as referências a um elemento específico tornaram-se comuns, facilitando o entendimento e a modificação de códigos complexos. Algumas abordagens experimentais investigam o uso de informações dinâmicas do programa para aprimorar ainda mais a navegação e auxiliar na manutenção de *software* [15, 29]. Essas inovações não só aceleram o desenvolvimento como também minimizam erros e promovem a qualidade e a consistência do código produzido.

2.1.2 A Era dos Modelos de Linguagem e a Integração de IA. Com o avanço dos modelos de linguagem de grande escala - LLMs, como o GPT-3 [5] e, posteriormente, o Codex [7], desenvolvidos pela OpenAI, houve uma transformação significativa nas ferramentas de geração de código. Essas ferramentas trouxeram a capacidade de interpretar o contexto do código em desenvolvimento e sugerir automaticamente trechos de código, funções completas ou até mesmo a resolução de problemas complexos [11].

Diferente das gerações anteriores, essas ferramentas de IA baseadas em LLMs aprendem a partir de vastos conjuntos de dados e são capazes de gerar código que não apenas segue padrões sintáticos, mas também leva em consideração boas práticas de programação e soluções previamente bem-sucedidas [21, 24, 28]. Isso marcou um novo capítulo na automação do desenvolvimento de *software*, com uma ênfase na adaptação do código às necessidades imediatas do desenvolvedor, aumentando tanto a eficiência quanto a qualidade [24, 28].

2.1.3 Geração de Código via GitHub Copilot. O GitHub Copilot é uma ferramenta de geração de código assistido por IA desenvolvida pela OpenAI em colaboração com o GitHub [11]. Baseada no modelo Codex, que por sua vez se originou do GPT-3 [7], essa ferramenta utiliza técnicas de aprendizado profundo para sugerir trechos de código e soluções completas, ajudando desenvolvedores

a programar de forma mais eficiente. A base tecnológica do Copilot é fundamentada nos conceitos estabelecidos no trabalho de [32], que introduziu o mecanismo de *attention* e as *transformers*, responsáveis por um salto significativo na capacidade dos modelos de linguagem de processar e gerar texto com base em contexto.

Diferente de abordagens tradicionais de autocompletar, que se limitam a sugestões baseadas em padrões simples, o Copilot é capaz de entender o contexto do código em desenvolvimento, fornecendo sugestões que levam em conta não apenas a sintaxe, mas também o propósito funcional do código [26]. O modelo Codex, utilizado pelo Copilot, foi treinado em uma vasta quantidade de código-fonte disponível publicamente, o que lhe permite sugerir desde pequenas correções até métodos e classes completas em diversas linguagens de programação [7].

3 Trabalhos Relacionados

Esta seção aborda os principais trabalhos da literatura que discutem o impacto das ferramentas de geração de código assistido por IA, no processo de desenvolvimento de *software*.

A análise dos trabalhos relacionados revela uma diversidade de abordagens na avaliação do impacto das ferramentas de geração de código assistido por IA, como o GitHub Copilot, ChatGPT e outras [9] . Os estudos revisados abrangem desde investigações sobre a correção e compreensibilidade do código gerado [26, 34] até avaliações mais amplas de usabilidade e percepções dos desenvolvedores [10, 29].

Uma característica comum entre muitos dos trabalhos analisados é o foco na comparação entre soluções de código geradas pelas ferramentas de IA e aquelas criadas por humanos [9, 24]. Esses estudos frequentemente utilizam experimentos controlados e estudos empíricos para medir a precisão, eficiência e qualidade do código gerado, revelando que, embora as ferramentas de IA possam acelerar o processo de programação, elas ainda enfrentam desafios em tarefas mais complexas e em garantir uma qualidade consistente de código. Além disso, a importância de boas descrições (docstrings) e nomes de funções para melhorar as sugestões de código foi um ponto ressaltado em vários estudos [34].

Outra semelhança é a ênfase em estudos que avaliam a experiência de desenvolvedores novatos [16, 29], indicando que as ferramentas de IA podem ser úteis no processo de aprendizado de programação, mas também podem gerar uma dependência que compromete o desenvolvimento de habilidades fundamentais. Esses estudos são especialmente relevantes em contextos educacionais, destacando a necessidade de acompanhamento adequado quando essas ferramentas são usadas como auxiliares no ensino.

Entretanto, algumas diferenças são notáveis em relação ao escopo e aos contextos investigados. Enquanto muitos estudos se concentram na linguagem Python e em conjuntos de dados como o HumanEval [24, 34], outros exploram diferentes linguagens de programação e contextos de desenvolvimento, como o Java [9] e casos de uso específicos em empresas brasileiras [10]. Essa diversidade de abordagens é importante para entender como as ferramentas de geração de código se adaptam a diferentes cenários, desde a indústria até ambientes acadêmicos.

Comparado a esses estudos, o presente trabalho se diferencia por sua abordagem empírica, como será explicado na seção 4. A pesquisa envolve o uso de APIs para tarefas de programação, o que torna o código gerado mais complexo e próximo de cenários reais de manutenção, em contraste com a maioria dos estudos que avaliam apenas problemas de programação mais simples. Essa abordagem permite observar como as ferramentas de geração de código se comportam em contextos que exigem um maior nível de compreensão e organização do código.

Além disso, diferentemente de estudos que frequentemente avaliam a ferramenta sem fornecer instruções específicas, este trabalho inclui uma etapa de treinamento dos participantes, tanto no uso da ferramenta de geração de código quanto na estruturação do código das APIs utilizadas. Esse treinamento visa reduzir possíveis barreiras iniciais e oferecer uma base mais equitativa para todos os participantes, proporcionando uma análise mais precisa dos efeitos da ferramenta no desempenho dos desenvolvedores.

Em suma, enquanto os estudos revisados trazem contribuições valiosas para entender o potencial e as limitações das ferramentas de geração de código, a necessidade de pesquisas que explorem contextos mais complexos e próximos da realidade permanece evidente. Este estudo procura preencher essa lacuna, oferecendo uma análise empírica do impacto dessas ferramentas no desenvolvimento de *software*, proporcionando uma perspectiva prática e aplicada sobre seu uso em cenários de desenvolvimento diversificados e de maior complexidade.

4 Metodologia

Esta seção descreve os métodos empregados para conduzir o presente estudo, cujo objetivo principal é avaliar o impacto de ferramentas de geração de código assistido por IA no desenvolvimento de software. A metodologia empregada neste trabalho organizando-se em três fases principais: *Investigação*, *Coleta de Dados* e *Análise*. A Figura 1 ilustra o fluxo das atividades realizadas em cada fase. Cada uma dessas etapas é detalhada nas subseções a seguir.

Com base nos objetivos definidos, este trabalho busca responder às seguintes questões de pesquisa, que orientam a investigação e análise dos resultados:

- **QP1: Quais indicadores são comumente utilizados para avaliar a eficácia e o impacto de ferramentas de geração de código por IA no desenvolvimento de software?**
  Justificativa: Essa questão visa identificar os principais indicadores adotados na literatura para avaliar ferramentas de geração de código por IA, o que permitirá definir critérios para mensurar sua eficácia e impacto no processo de desenvolvimento.
- **QP2: Como o uso de ferramentas de geração de código impacta o desempenho dos desenvolvedores?**
  Justificativa: Esta questão foca no objetivo central desta pesquisa, que é avaliar empiricamente a eficácia das ferramentas de geração de código. Através de estudos empíricos, será possível identificar de que maneira essas ferramentas influenciam o desempenho de desenvolvedores com diferentes níveis de experiência.

Essas questões foram formuladas com o intuito de direcionar a investigação para uma análise detalhada da eficácia prática das ferramentas de geração de código em cenários de desenvolvimento

próximos do real. As respostas a essas questões permitirão avaliar os benefícios e limitações das ferramentas, fornecendo uma base para recomendações fundamentadas sobre sua aplicação no desenvolvimento de *software*.

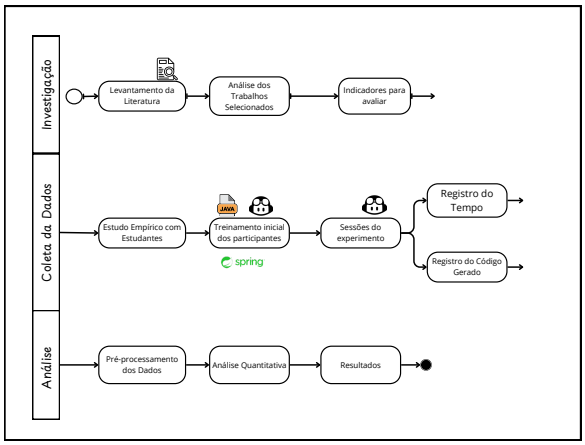

Figura 1: Etapas da metodologia de pesquisa para avaliação do impacto do GitHub Copilot no desenvolvimento de software.

4.1 Investigação

A fase de investigação teve como objetivo central realizar um levantamento bibliográfico para embasar as questões de pesquisa **QP1**. Esse levantamento da literatura focou em identificar os principais indicadores usados para avaliar ferramentas de geração de código. Este processo inicial é fundamental para estabelecer as bases teóricas e metodológicas do estudo, permitindo a seleção de indicadores aplicados nas fases subsequentes para avaliar ferramentas de geração de código.

Inicialmente, foram selecionados artigos relevantes que abordam o uso de ferramentas de geração de código assistido por IA, utilizando o *Google Scholar* como principal plataforma de busca, devido à sua ampla cobertura de bases de dados acadêmicas e facilidade de acesso. Em seguida, esses estudos foram analisados com o objetivo de identificar indicadores e métricas amplamente utilizados para avaliar a eficiência, usabilidade e eficácia dessas ferramentas. Essa análise orientou a definição dos parâmetros de avaliação aplicados na fase de coleta de dados. Os indicadores identificados foram, posteriormente, categorizados conforme descrito na seção 5.1.

4.2 Coleta de Dados

A fase de coleta de dados foi conduzida a partir de um estudo empírico. Essa abordagem permite avaliar como programadores de diferentes níveis de conhecimento interagem com a ferramenta de geração de código, proporcionando uma visão sobre o impacto dessa tecnologia no desenvolvimento de software.

4.2.1 Estudo Empírico. O estudo empírico teve como objetivo avaliar o desempenho de participantes ao utilizar o Copilot para implementar funcionalidades que foram apagadas de duas APIs. Essa abordagem permitiu investigar como a ferramenta pode impactar no processo de desenvolvimento.

O Copilot foi escolhido por ser uma ferramenta amplamente investigada por outros autores, além de estar disponível gratuitamente para estudantes por meio do programa *GitHub Education* [12]. Essa escolha também se justifica pela facilidade de integração do Copilot ao ambiente de desenvolvimento.

Para garantir um ambiente padronizado para todos os participantes, foi utilizado o Visual Studio Code (VSCode) na versão 1.90.1, juntamente com a extensão oficial do GitHub Copilot, versão 0.12. Diferentemente da configuração padrão, a funcionalidade de auto complete (Enable Auto Completions) foi desativada, garantindo que os participantes utilizassem exclusivamente a funcionalidade de chat do Copilot. Essa restrição foi imposta para isolar o impacto da assistência via chat no processo de desenvolvimento, evitando interferências de sugestões automáticas durante a implementação das funcionalidades ausentes.

4.2.2 Participantes. Os participantes do estudo empírico foram estudantes de Computação, nos níveis de graduação e pós-graduação. Todos os alunos das disciplinas Engenharia de Software e Programação Orientada a Objetos (Graduação) e Engenharia de Software (Pós-Graduação) foram convidados a participar, sem realização de sorteio. Antes do início das tarefas, foi conduzido um treinamento inicial: 4 encontros de 2h cada, para todos os participantes, abrangendo tanto o uso da ferramenta de geração de código quanto a estrutura e o funcionamento das APIs utilizadas (Java/Spring Boot). Esse treinamento foi essencial para assegurar que os participantes possuíssem a familiaridade necessária com a ferramenta e o domínio das tarefas, reduzindo o impacto de eventuais lacunas de conhecimento sobre o desempenho nas atividades propostas.

Para garantir a conformidade com as normas éticas, todos os participantes do estudo foram devidamente informados sobre os objetivos, procedimentos e possíveis implicações da pesquisa. Foi obtido o consentimento informado de cada participante, garantindo que eles estivessem cientes de sua participação voluntária e do direito de se retirar a qualquer momento, sem qualquer penalização.

4.2.3 Procedimento. O estudo empírico consistiu em uma sessão experimental, nal qual os participantes utilizaram ou não a ferramenta de geração de código para recriar métodos que foram excluídos das APIs fornecidas. Durante o experimento, os participantes realizaram as atividades sob supervisão, sendo coletados dados sobre o tempo de execução das tarefas e o código gerado, além disso, eles deveriam gravar a tela durante todo o experimento e disponibilizar a gravação juntamente com a solução para o problema.

Durante a sessão experimental, foram disponibilizadas aos participantes duas APIs desenvolvidas em Java/Spring Boot (locadora-api, disponível em [3] e associacao-api, disponível em [1]). Em cada API, foram suprimidos dois métodos, gerando os problemas identificados como #1, #2, #3 e #4. Com o objetivo de manter a proporcionalidade entre as abordagens com e sem o uso da ferramenta, os problemas foram distribuídos aleatoriamente entre os participantes, de modo que cada questão tivesse uma quantidade equilibrada de resoluções com e sem o Copilot, observe os dados na Tabela 1.

Os participantes foram orientados a recriar apenas os métodos que haviam sido suprimidos, sem adicionar novos métodos, já que apenas esses estavam ausentes na implementação original. Dessa

forma, os métodos auxiliares já presentes foram mantidos, garantindo que o comportamento esperado da aplicação fosse preservado. Embora os testes unitários não estivessem disponíveis, os participantes puderam validar suas implementações por meio de dados gerados automaticamente pela própria API ao ser iniciada. Essa validação foi realizada por meio de requisições à API, utilizando o Swagger[1]. A escolha por esse método de verificação foi determinada pelo curto período de tempo disponível para o treinamento dos participantes, e o conteúdo relacionado aos testes unitários não foi abordado devido a essa limitação.

4.2.4 Dados Coletados. Os dados coletados durante o estudo empírico incluíram:

- **Registro do Tempo de Execução**: Os participantes foram instruídos a registrar manualmente o horário de início e término de cada tarefa, tanto nas condições com o uso da ferramenta de geração de código quanto sem seu auxílio. Esse registro permitiu uma comparação direta do tempo necessário para completar as atividades em ambos os cenários, fornecendo dados para análise do impacto da ferramenta no tempo de execução.
- **Registro do Código Gerado**: O código produzido por cada participante foi coletado e submetido a testes unitários para avaliar sua correção. Esse processo foi utilizado exclusivamente para verificar se o código gerado ou criado pelo participante atendia aos critérios de funcionalidade esperados.

Apesar de o registro manual do intervalo de tempo gasto para concluir cada tarefa ser suscetível a erros, o vídeo enviado por cada participante foi utilizado como uma ferramenta adicional para validar a precisão dos tempos registrados. Esse vídeo forneceu uma confirmação adicional do tempo realmente gasto, permitindo uma análise mais confiável e mitigando possíveis imprecisões no processo de medição manual. Assim, esses dados serviram de base para a análise quantitativa, com o objetivo de avaliar o impacto da ferramenta.

4.3 Análise dos Dados

A fase de análise dos dados tem como objetivo responder à **QP2**, investigando como o uso de ferramentas de geração de código impacta o desempenho dos desenvolvedores em diferentes contextos. Nesta etapa, os dados coletados passaram por um pré-processamento, seguido de uma análise *quantitativa*.

4.3.1 Pré-processamento dos Dados. O pré-processamento representa o primeiro passo desta etapa, no qual as informações fornecidas pelos participantes nos formulários são verificadas e comparadas com outras fontes de dados, visando assegurar a fidedignidade e a consistência das respostas. Esse procedimento é essencial para reduzir possíveis vieses e garantir a confiabilidade das informações antes da aplicação das abordagens analíticas. Por exemplo, o horário de início e término foi conferido por meio dos vídeos disponibilizados pelos participantes, garantindo que o intervalo de tempo gasto estivesse correto.

[1]https://swagger.io/

4.3.2 Análise Quantitativa. A análise quantitativa focou em examinar o impacto do uso do GitHub Copilot no desempenho dos participantes, utilizando técnicas estatísticas apropriadas para investigar as diferenças entre os grupos com e sem o auxílio da ferramenta. O teste de Mann-Whitney U foi utilizado para comparar as medianas dos grupos, dado que os dados de tempo gasto e número de falhas nos testes não apresentaram distribuição normal, conforme verificado pelo teste de Shapiro-Wilk. Além disso, para avaliar a magnitude do efeito observado, foi calculado o *Cliff's delta*, uma medida não paramétrica que indica a proporção de pares ordenados entre os dois grupos.

Embora essas análises estatísticas revelem correlações entre o uso do Copilot e o desempenho dos participantes, é importante destacar que não se pode interpretar essas correlações como relações causais. Outros fatores, como a experiência prévia dos participantes com a ferramenta ou a natureza dos problemas apresentados, podem ter influenciado os resultados observados. Portanto, as correlações devem ser vistas como tendências e não como evidências de causalidade.

4.3.3 Resultados. O último passo desta abordagem é a divulgação dos resultados deste estudo por meio da produção de artigo científico. Esse documento inclui a metodoliga empregada, as análises detalhadas dos dados coletados, discussões sobre as limitações encontradas e recomendações práticas para o uso de ferramentas de geração de código em ambientes de desenvolvimento de software.

5 Resultados

Essa seção apresenta os resultados conforme direcionamento dado pelas questões de pesquisa apresentadas na seção 4

5.1 Avaliação de Ferramentas de Geração de Código

Para responder à **QP1: Quais indicadores são comumente utilizados para avaliar a eficácia e o impacto de ferramentas de geração de código por IA no desenvolvimento de software?**, a literatura apresenta uma série de indicadores que têm sido aplicados para avaliar diferentes aspectos dessas ferramentas. Esses indicadores permitem verificar a eficácia das tecnologias e identificar os desafios envolvidos em seu uso. A seguir, são destacados os principais indicadores encontrados na literatura, organizados conforme suas áreas de avaliação.

5.1.1 Qualidade do Código.

- **Validade do Código**: A capacidade do código de ser compilado sem erros é fundamental. O estudo de [9] destaca a importância da validade do código como um requisito essencial para o uso prático de ferramentas de IA no desenvolvimento.
- **Correção do Código**: Para que o código seja funcional, é necessário que produza os resultados esperados. O trabalho de [34] explora a correção do código gerado utilizando testes unitários e o *HumanEval Dataset* como referência.
- **Manutenibilidade do Código**: A facilidade de compreensão, modificação e manutenção do código ao longo do tempo é essencial. Indicadores como complexidade ciclomática e número de linhas de código são analisados em [24].

- **Code Smells**: são analisados por [34] para identificar problemas potenciais na manutenibilidade do código (como código duplicado ou métodos muito longos).
- **Auto-correção**: Analisa a capacidade das ferramentas de corrigirem seus próprios erros durante o processo de geração de código. Esse aspecto é explorado em [4], onde são comparadas as capacidades de auto-correção do Copilot, ChatGPT e Gemini.

5.1.2 Eficiência.

- **Tempo de Conclusão da Tarefa**: O tempo que um desenvolvedor gasta para concluir uma tarefa usando a ferramenta é um indicador direto de eficiência, medido em [16, 28].
- **Tempo de Execução do Código**: O desempenho do código gerado, em termos de tempo de execução, é avaliado por [25] para entender a eficiência do código gerado.
- **Número de Linhas de Código (LOC)**: O tamanho do código, medido em número de linhas, reflete sua concisão e complexidade. O estudo de [8] investiga a relação entre LOC e eficiência do código.

5.1.3 Outros Indicadores. Além das métricas de qualidade e eficiência, outros estudos sugerem indicadores adicionais que complementam a análise de ferramentas de geração de código:

- **Diversidade do Código**: A capacidade da ferramenta de gerar diferentes abordagens para uma mesma tarefa permite explorar soluções alternativas. [24] investigam essa métrica comparando soluções geradas por IA e desenvolvedores humanos.
- **Impacto no Aprendizado**: Em ambientes educacionais, é importante avaliar se a ferramenta facilita o aprendizado. O trabalho de [16] analisa o impacto da IA no aprendizado de conceitos e técnicas de programação.
- **Similaridade com o Código de Referência**: Compara o código gerado com implementações de referência, utilizando métricas como CodeBLEU e similaridade de Levenshtein, como observado em [9].
- **Reprodutibilidade**: Avalia a capacidade da ferramenta de gerar soluções consistentes em diferentes tentativas, analisada em [24].
- **Nível de Granularidade**: Refere-se à granularidade das sugestões, como tokens, linhas ou blocos de código, descrito em [30].
- **Resposta Recebida**: Em [24], essa métrica contabiliza o número de vezes que o Copilot gera código em resposta a um problema.
- **Optimalidade do Código**: Avalia se a ferramenta sugere algoritmos eficientes para um problema, como observado em [24].
- **Auto-correção**: Analisa a capacidade das ferramentas de corrigirem seus próprios erros durante o processo de geração de código. Esse aspecto é explorado em [4], onde são comparadas as capacidades de auto-correção do Copilot, ChatGPT e Gemini.

Com base na revisão da literatura, este trabalho adota como indicadores de avaliação o **tempo de conclusão da tarefa** e a **correção do código**. O tempo de conclusão da tarefa é amplamente utilizado para medir a eficiência das ferramentas de geração de código, avaliando o quanto elas contribuem para reduzir o tempo necessário para que o desenvolvedor finalize uma atividade [16, 28]. Já a correção do código refere-se à capacidade do código gerado de produzir os resultados esperados, sendo um aspecto essencial para validar a funcionalidade e a utilidade prática das ferramentas [34]. A seleção desses indicadores reflete o foco deste estudo em investigar tanto a eficiência quanto a funcionalidade do código gerado, proporcionando uma análise do impacto dessas ferramentas no desempenho de desenvolvimento.

Tabela 1: Número de respostas dos participantes por tarefas de programação com e sem uso do GitHub Copilot

Problema	Total de Respostas	Com Copilot	Sem Copilot
#1	46	22	24
#2	47	26	21
#3	46	22	24
#4	45	26	19
Total	**184**	**96**	**88**

5.2 Estudo Empírico

Para responder à **QP2: Como o uso de ferramentas de geração de código impacta o desempenho dos desenvolvedores?**, foi realizado um estudo empírico conforme descrito na metodologia. A coleta de dados resultou em um conjunto de informações sobre o desempenho nas tarefas realizadas com e sem o uso do Copilot. Esses dados foram organizados e analisados com o objetivo de extrair informações relevantes para responder às questões de pesquisa, focando especificamente na execução das tarefas, no tempo despendido e na quantidade de testes falhos nas atividades propostas. Os dados completos estão disponíveis em [2].

Participaram do estudo um total de **49** participantes (todos estudantes), que realizaram atividades de recriar os método apagados, numerados de #1 a #4 (coluna Problema na Tabela 1). No total, foram coletadas **184** respostas, distribuídas conforme apresentado na Tabela 1. Vale destacar que eram esperadas 196 (4 x 49) respostas, porém nem todos os participantes conseguiram enviar a solução para os quatro problemas.

Conforme apresentado na Tabela 1, cada questão contou com um número específico de respostas, tanto com o uso do Copilot quanto sem. Observa-se que o total de respostas com o Copilot foi de 96, enquanto sem o Copilot foi de 88.

Para uma análise mais detalhada, os participantes foram sorteados aleatoriamente para garantir uma distribuição equilibrada entre o número de problemas resolvidos com e sem o auxílio do Copilot. Embora os dados possam ser segmentados por turma (alunos das turmas 1 e 2 são do nível de graduação, enquanto alunos da turma 3 são da pós-graduação), tempo de experiência (dividido com base na mediana dos dados, 24 meses), e uso prévio do Spring, essas categorias não foram utilizadas para segmentar os participantes no sorteio. Esses dados estão resumidos na Tabela 2 e podem ser consultados no *dataset* [2] e são . O estudo foi estruturado para evitar viés, permitindo comparações justas entre os participantes,

que resolveram os problemas tanto com quanto sem o auxílio do assistente de IA.

Essa estratégia de sorteio aleatório revelou uma distribuição diversificada dos participantes, com uma maioria de alunos da graduação, sendo que uma parte significativa possuía mais de 24 meses de experiência em programação. O uso prévio do Spring foi relativamente baixo, indicando que uma fração dos participantes teve pouco ou nenhum contato com a tecnologia antes da realização das tarefas. Esse sorteio garantiu que a comparação entre os participantes fosse justa, levando em conta que todos haviam passado pelo mesmo treinamento inicial e tinham o conhecimento mínimo necessário para utilizar a ferramenta e a estrutura do sistema.

Tabela 2: Distribuição dos participantes por questão considerando turma, tempo de programação e uso do Spring

Categoria	#1	#2	#3	#4
Graduação	34	36	34	34
Pós-Graduação	12	11	12	11
> 24 meses de experiência	26	28	26	25
≤ 24 meses de experiência	20	19	20	20
Usaram Spring	4	4	4	4
Não Usaram Spring	42	43	42	41
Total	**138**	**46**	**105**	**79**

A análise desses dados fornece *insights* sobre como o Copilot influenciou o tempo de resolução, a taxa de sucesso na execução das tarefas. Além disso, a variação no número de respostas por questão pode indicar diferenças na complexidade das tarefas ou na familiaridade dos participantes com os conceitos envolvidos.

5.3 Análise dos Dados

Nesta subseção, é realizada uma análise exploratória dos dados coletados com o objetivo de observar características gerais das variáveis e identificar padrões ou tendências. Esse exame inicial é essencial para contextualizar as condições experimentais com e sem o uso do GitHub Copilot e avaliar o impacto dessa ferramenta no desempenho dos participantes nas tarefas de programação.

5.3.1 Estatísticas Descritivas. Para iniciar a análise exploratória, foram calculadas estatísticas descritivas das variáveis principais, incluindo *tempo gasto* e *número de testes falhos*. As medidas de tendência central (média e mediana) e de dispersão (desvio padrão) foram utilizadas para caracterizar a distribuição dos dados.

Essas estatísticas fornecem uma visão inicial sobre o desempenho médio dos participantes, permitindo observar, por exemplo, se o tempo médio gasto na realização das tarefas é reduzido com o uso do Copilot e se há uma diferença significativa no número de Testes Falhos nas duas condições experimentais (com e sem Copilot). A Tabela 3 apresenta um resumo das estatísticas descritivas das variáveis mencionadas.

5.3.2 Análise Visual dos Indicadores Normalizados. A Figura 2 apresenta a distribuição normalizada dos dois indicadores avaliados neste estudo: *Tempo Gasto* e *Número de Testes Falhos*, divididos entre as condições "Com Copilot"e "Sem Copilot". A normalização dos

Tabela 3: Estatísticas descritivas do tempo gasto (em minutos) e número de testes unitários falhos, comparando grupos com e sem utilização do GitHub Copilot

Indicador	Não Usou Copilot	Usou Copilot
$\text{Média}_{\text{Tempo Gasto}}$	30.45	**24.73**
$\text{Desvio Padrão}_{\text{Tempo Gasto}}$	**23.05**	25.03
$\text{Mediana}_{\text{Tempo Gasto}}$	23.5	**16.0**
$\text{Média}_{\text{Testes Falhou}}$	**1.48**	1.65
$\text{Desvio Padrão}_{\text{Testes Falhou}}$	**1.84**	2.44
$\text{Mediana}_{\text{Testes Falhou}}$	1.0	1.0

dados foi realizada utilizando a técnica de Min-Max Scaling, que transforma os valores para um intervalo entre 0 e 1, permitindo comparações diretas entre indicadores de escalas diferentes. Esta abordagem foi necessária porque os dois indicadores possuem escalas distintas, sendo o *Tempo Gasto* medido em minutos e o *Número de Testes Falhos* representado por valores inteiros discretos, o que poderia comprometer a análise visual caso os dados não fossem ajustados.

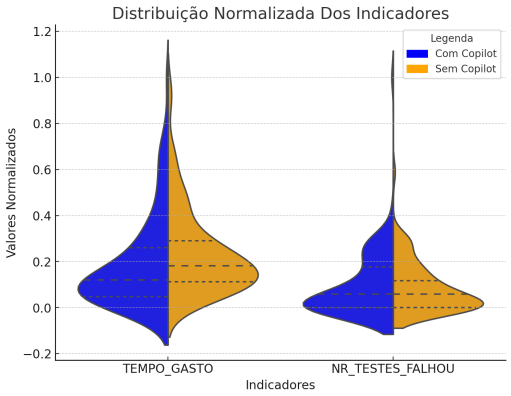

Figura 2: Distribuição Normalizada Dos Indicadores por Uso do Copilot

Ao observar o **Tempo Gasto**, nota-se que a distribuição para "Com Copilot"apresenta uma concentração maior de valores em intervalos inferiores, indicando uma redução no tempo necessário para completar as tarefas em comparação à condição "Sem Copilot". O perfil da distribuição sugere que o uso do Copilot pode melhorar a eficiência dos desenvolvedores, uma tendência já apontada pelas análises estatísticas apresentadas anteriormente.

Já no **Número de Testes Falhos**, as distribuições normalizadas para ambas as condições mostram uma maior uniformidade, sem diferenças visíveis significativas entre as duas categorias. Isso reforça a conclusão das análises anteriores, indicando que o uso do Copilot não impactou de forma relevante a correção funcional do código produzido.

Por fim, a análise visual dos gráficos normalizados corrobora as conclusões quantitativas: enquanto o Copilot demonstra um impacto positivo na eficiência (*Tempo Gasto*), sua influência na

qualidade do código (*Número de Testes Falhos*) é limitada. A normalização permitiu uma comparação mais justa entre os indicadores, evidenciando claramente os efeitos observados.

5.4 Análise Descritiva dos Dados

A Tabela 3 apresenta as estatísticas descritivas das variáveis *tempo gasto* e *número de testes falhos*, separadas entre as condições de uso e não uso do Copilot pelos participantes. A análise comparativa entre esses dois grupos possibilita uma avaliação do impacto da ferramenta nos desempenhos observados.

Na análise do **tempo gasto**, a *média* para os participantes que utilizaram o Copilot foi de 24,73 minutos, enquanto para aqueles que não utilizaram a ferramenta, a média foi de 30,45 minutos. Esse dado sugere uma redução média no tempo de execução das tarefas para os participantes com suporte da ferramenta. Observa-se, ainda, que o *desvio padrão* é próximo entre os dois grupos (25,03 para o grupo com Copilot e 23,05 para o grupo sem Copilot), indicando variações similares no tempo de execução em ambos os casos. A *mediana* também reflete essa tendência, sendo menor para o grupo que utilizou o Copilot (16 minutos) em comparação com o grupo que não utilizou (23,5 minutos), o que reforça a hipótese de que o uso da ferramenta pode ter contribuído para uma execução mais rápida das tarefas.

Na análise do **número de testes que falharam**, a *média* para os participantes que utilizaram o Copilot foi de 1,65, enquanto para aqueles que não utilizaram a ferramenta, a média foi de 1,48. Esse dado sugere um leve aumento no número de falhas para os participantes com suporte da ferramenta. Observa-se, ainda, que o *desvio padrão* é maior no grupo com Copilot (2,44 em comparação com 1,84 no grupo sem Copilot), indicando uma maior variação no número de falhas entre os participantes que utilizaram a ferramenta. A *mediana* também reflete essa observação, sendo similar entre os dois grupos (1,0 em ambos), sugerindo que, apesar do aumento na média e na variabilidade de falhas para os usuários do Copilot, a maioria dos participantes teve um número semelhante de falhas. Esse achado pode ser explicado pela introdução de métodos auxiliares, que geraram inconsistências, pois os participantes foram orientados a recriar unicamente o método excluído e a utilizar os métodos auxiliares disponíveis em outros pontos dos sistemas.

Em resumo, os dados descritivos sugerem que o uso do Copilot pode reduzir o tempo necessário para completar as tarefas, mas também pode introduzir variações na qualidade do código gerado, conforme indicado pelo número de testes que falharam. Essa análise descritiva inicial fornece uma base para o aprofundamento com análises inferenciais que investiguem a significância estatística dessas diferenças observadas.

5.4.1 Análise de Normalidade e Teste de Hipótese. Para determinar a normalidade dos dados, foi aplicado o teste de Shapiro-Wilk nos tempos gastos e no número de testes falhos dos grupos com e sem o uso do Copilot. Abaixo estão os resultados obtidos:

- **Tempo Gasto (Usou Copilot)**: $p = 4.98 \times 10^{-10}$
- **Tempo Gasto (Não Usou Copilot)**: $p = 1.52 \times 10^{-8}$
- **Número de Testes Falhos (Usou Copilot)**: $p = 2.76 \times 10^{-13}$
- **Número de Testes Falhos (Não Usou Copilot)**: $p = 4.53 \times 10^{-10}$

Os valores de p inferiores ao nível de significância convencional ($\alpha = 0.05$) [6] indicam que a hipótese nula de normalidade dos dados pode ser rejeitada. Assim, os dados não apresentam distribuição normal.

Diante disso, optou-se pelo **teste de Mann-Whitney U** (ou *Wilcoxon rank-sum*), que é adequado para comparar grupos não paramétricos [6, 17] e permite avaliar a diferença nas medianas entre os grupos com e sem o uso do Copilot.

Resultados do Teste de Mann-Whitney U:

- **Tempo Gasto**: $U = 3148.0$, $p = 0.0029$ — indica uma diferença estatisticamente significativa no tempo gasto entre os grupos.
- **Número de Testes Falhos**: $U = 4075.0$, $p = 0.866$ — não indica diferença significativa entre os grupos para o número de Testes Falhos.

Interpretação: Este teste avalia se uma variável tende a ter valores maiores ou menores em um grupo em comparação a outro, sem a necessidade de pressupor distribuição normal. A análise revelou que o uso do Copilot impacta significativamente o tempo de execução das tarefas, mas não apresenta diferença significativa no número de testes que falharam em relação ao grupo que não utilizou a ferramenta.

5.4.2 Cliff's Delta. Para complementar a análise estatística realizada com o teste de Mann-Whitney U, foi calculado o *Cliff's delta* para as variáveis *Tempo Gasto* e *Número de Testes Falhos*, a fim de avaliar a magnitude do efeito. O *Cliff's delta* é uma medida não paramétrica que indica a proporção de pares ordenados entre dois grupos, sendo útil para comparar distribuições de maneira independente de sua escala.

Os resultados são apresentados a seguir:

- **Tempo Gasto:** O valor de $\delta = -0.254$ indica um efeito *pequeno* (*small*). Isso sugere que há uma tendência de os tempos de execução sem o uso do Copilot serem maiores do que aqueles com o uso da ferramenta, corroborando os resultados do teste de Mann-Whitney U, que indicaram uma diferença estatisticamente significativa.
- **Número de Testes Falhos:** O valor de $\delta = -0.014$ indica um efeito *negligenciável* (*negligible*). Este resultado reforça que não há diferenças relevantes entre os grupos com e sem Copilot em relação ao número de testes falhos, conforme também observado no teste de Mann-Whitney U.

Esses resultados complementam a análise anterior, evidenciando que, enquanto o Copilot contribui para uma maior eficiência em termos de tempo, seu impacto na qualidade do código gerado, medido pelo número de testes falhos, é insignificante. O uso do *Cliff's delta* permite, portanto, uma interpretação mais detalhada da magnitude das diferenças entre os grupos.

5.5 Discussão dos Achados

A análise dos resultados obtidos permite algumas considerações sobre o impacto do uso do Copilot no contexto do desenvolvimento de software. Os dados descritivos e os testes de hipótese indicaram uma redução no tempo necessário para completar as tarefas quando os participantes utilizaram o Copilot. Esse achado está alinhado com a literatura, que sugere que ferramentas de geração de

código assistido por IA podem atuar como facilitadores no desenvolvimento, permitindo que os desenvolvedores concluam tarefas mais rapidamente ao delegarem partes rotineiras ou repetitivas do código à ferramenta, como apresentado em [28].

Por outro lado, o número de testes que falharam, avaliado por meio de testes unitários, não apresentou diferença estatisticamente significativa entre os grupos que usaram e não usaram o Copilot. Esse resultado sugere que, apesar da eficiência temporal proporcionada pela ferramenta, sua utilização não necessariamente melhora a qualidade do código em termos de correção funcional. De fato, essa conclusão vai ao encontro de estudos que indicam que a assistência ao código pode oferecer soluções rápidas, mas requer a análise crítica do desenvolvedor para manter a qualidade desejada [34].

No entanto, é importante destacar que a qualidade do código pode ter sido influenciada pela familiaridade dos participantes com a ferramenta e pela natureza das tarefas propostas. Os participantes podem ter tido um conhecimento limitado do Copilot ou talvez não tenham utilizado suas sugestões de maneira ideal, o que pode ter impactado o número de testes falhos. Além disso, as tarefas apresentadas podem ter variado em complexidade, o que também poderia influenciar a eficácia da ferramenta na geração de código funcional. Essas variáveis precisam ser consideradas para uma análise mais detalhada do impacto da ferramenta na qualidade do código produzido.

Essas observações reforçam a importância de orientar os desenvolvedores quanto ao uso consciente e criterioso das sugestões de código oferecidas pelo Copilot.

6 Ameaças à Validade

A validade deste estudo pode ser impactada por diferentes fatores que devem ser considerados na interpretação dos resultados. As ameaças foram categorizadas em validade interna, externa, de construção e de conclusão.

Validade Interna: A validade interna pode ter sido comprometida pela variabilidade no nível de experiência dos participantes, que eram estudantes com conhecimentos diversos em programação. Embora todos tenham recebido o mesmo treinamento sobre o uso do GitHub Copilot, a familiaridade com a ferramenta e as habilidades prévias em programação podem ter influenciado os resultados. Além disso, o ambiente controlado do experimento pode não representar totalmente as condições reais de desenvolvimento de software.

Validade Externa: A generalização dos resultados (da primeira etapa) para contextos de desenvolvimento industrial ou com desenvolvedores mais experientes é limitada. Como o estudo foi realizado exclusivamente com estudantes, é possível que os resultados não reflitam completamente o impacto do Copilot em ambientes de trabalho profissionais.

Validade de Construção: O estudo utilizou indicadores, como o tempo de execução e o número de testes falhos, que foram selecionadas com base na literatura. No entanto, outras medidas poderiam ser consideradas para uma análise mais abrangente.

Validade de Conclusão: As análises estatísticas foram realizadas com base em uma amostra limitada de estudantes, o que pode afetar a robustez dos resultados. Assim, interpretações generalizadas devem ser feitas com cautela, considerando as limitações impostas pelo tamanho da amostra e pela variabilidade dos dados coletados.

Essas ameaças à validade destacam a necessidade de estudos complementares em contextos variados e com diferentes perfis de desenvolvedores, a fim de verificar a aplicabilidade dos resultados encontrados em cenários reais de desenvolvimento de software.

7 Conclusões e Trabalhos Futuros

Este trabalho investigou o impacto do uso de ferramentas de geração de código assistido por *Inteligência Artificial* (*IA*), com foco na avaliação do *GitHub Copilot*. A pesquisa foi estruturada em duas partes principais: um levantamento da literatura para identificar os principais indicadores utilizados na avaliação dessas ferramentas e um estudo empírico para investigar o efeito da ferramenta em tarefas de desenvolvimento de *software*.

Com base no levantamento da literatura, foram selecionados dois indicadores principais para análise: o *tempo de execução das tarefas* e o *número de testes falhos*. A análise dos dados coletados no estudo empírico indicou que o uso do Copilot pode reduzir o tempo necessário para concluir tarefas de programação. No entanto, não houve diferença estatisticamente significativa no número de testes que falharam entre os grupos que utilizaram e não utilizaram o Copilot, sugerindo que o uso da ferramenta não impactou diretamente a qualidade do código em termos de correção.

Esses achados corroboram a literatura, que sugere que, embora o uso de *IA* assistiva possa acelerar o desenvolvimento, ele não substitui a supervisão humana para garantir a qualidade do código gerado. O estudo empírico também destaca a importância de orientar os desenvolvedores quanto ao uso crítico das sugestões geradas pela *IA*.

Trabalhos futuros podem incluir um estudo de caso em uma empresa de desenvolvimento de *software*, permitindo a análise do impacto do Copilot em um ambiente corporativo com desenvolvedores experientes. Esse estudo traria uma visão mais prática sobre o uso da ferramenta em situações reais de desenvolvimento, onde a experiência dos profissionais e a complexidade das tarefas têm influência direta nos resultados. Adicionalmente, futuras pesquisas podem explorar o uso do *Copilot* em tarefas que exigem maior criatividade ou resolução de problemas complexos, avaliando se a ferramenta realmente contribui para soluções inovadoras.

Finalmente, este estudo contribui para o entendimento do impacto das ferramentas de geração de código assistido por *IA*, destacando tanto os benefícios quanto as limitações de seu uso. O *GitHub Copilot* mostrou-se eficaz na redução do tempo de desenvolvimento, mas os resultados também sugerem que a supervisão e a revisão humana permanecem essenciais para garantir a qualidade do código. Espera-se que esta pesquisa forneça uma base para futuras investigações e para a implementação consciente dessas ferramentas em diferentes contextos de desenvolvimento de *software*.

Este trabalho dialoga com temas como **Sistemas de Informação e Inteligência Artificial (Generativa, LLM, PLN, entre outros)**, ao investigar o impacto de tecnologias emergentes baseadas em *LLMs* no desenvolvimento de software. Também contribui para **Paradigmas, modelagem, design, engenharia e avaliação de**

sistemas de informação, ao propor indicadores empíricos para medir a eficácia dessas ferramentas.

Referências

[1] Anônimo Autor. 2024. Associacao API. https://drive.google.com/drive/folders/1tgG_WjNST7J3CxAgoShNFLi2Vc1WHdYg?usp=sharing.

[2] Anônimo Autor. 2024. Dados Experimento. https://docs.google.com/spreadsheets/d/1pvIE3YiuXLUPz4w30JReIgDi4BSHFwKHDkELTRFvkxE/edit?usp=sharing.

[3] Anônimo Autor. 2024. Locadora API. https://drive.google.com/drive/folders/1tgG_WjNST7J3CxAgoShNFLi2Vc1WHdYg?usp=sharing.

[4] Samuel Silvestre Batista, Bruno Branco, Otávio Castro, and Guilherme Avelino. 2024. Code on Demand: A Comparative Analysis of the Efficiency Understandability and Self-Correction Capability of Copilot ChatGPT and Gemini. In *Proceedings of the XXIII Brazilian Symposium on Software Quality (SBQS '24)*. Association for Computing Machinery, New York, NY, USA, 351–361. https://doi.org/10.1145/3701625.3701673

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] https://arxiv.org/abs/2005.14165

[6] Gary Charness, Uri Gneezy, and Michael A. Kuhn. 2012. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization* 81, 1 (2012), 1–8. https://doi.org/10.1016/j.jebo.2011.08.009

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374 Disponível em https://arxiv.org/abs/2107.03374.

[8] Vincenzo Corso, Leonardo Mariani, Daniela Micucci, and Oliviero Riganelli. 2024. Assessing AI-Based Code Assistants in Method Generation Tasks. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (Lisbon, Portugal) *(ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 380–381. https://doi.org/10.1145/3639478.3643122

[9] Vincenzo Corso, Leonardo Mariani, Daniela Micucci, and Oliviero Riganelli. 2024. Generating Java Methods: An Empirical Assessment of Four AI-Based Code Assistants. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension* (Lisbon, Portugal) *(ICPC '24)*. Association for Computing Machinery, New York, NY, USA, 13–23. https://doi.org/10.1145/3643916.3644402

[10] Nicole Davila, Igor Wiese, Igor Steinmacher, Lucas Lucio da Silva, Andre Kawamoto, Gilson Jose Peres Favaro, and Ingrid Nunes. 2024. An Industry Case Study on Adoption of AI-based Programming Assistants. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (Lisbon, Portugal) *(ICSE-SEIP '24)*. Association for Computing Machinery, New York, NY, USA, 92–102. https://doi.org/10.1145/3639477.3643648

[11] GitHub. 2021. Introducing GitHub Copilot: your AI pair programmer. https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/. Accessed: 2024-02-25.

[12] GitHub. 2024. GitHub Education. https://github.com/education. Acesso em: 9 nov. 2024.

[13] Xiaodong Gu, Meng Chen, Yalan Lin, Yuhan Hu, Hongyu Zhang, Chengcheng Wan, Zhao Wei, Yong Xu, and Juhong Wang. 2024. On the Effectiveness of Large Language Models in Domain-Specific Code Generation. *ACM Trans. Softw. Eng. Methodol.* 34 (Oct. 2024), 1–22. https://doi.org/10.1145/3697012 Just Accepted.

[14] James D Herbsleb and Deependra Moitra. 2001. Global software development. *IEEE software* 18, 2 (2001), 16–20.

[15] Rasha Ahmad Husein, Hala Aburajouh, and Cagatay Catal. 2025. Large language models for code completion: A systematic literature review. *Computer Standards & Interfaces* 92 (2025), 103917. https://doi.org/10.1016/j.csi.2024.103917

[16] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. https://doi.org/10.1145/3544548.3580919

[17] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering* 22, 2 (2017), 579–630. https://doi.org/10.1007/s10664-016-9437-5

[18] Rob Kitchin and Martin Dodge. 2014. *Code/space: Software and everyday life.* Mit Press, USA.

[19] Yun Young Lee, Sam Harwell, Sarfraz Khurshid, and Darko Marinov. 2013. Temporal code completion and navigation. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE Press, USA, 1181–1184. https://doi.org/10.1109/ICSE.2013.6606673

[20] John R Levine, Tony Mason, and Doug Brown. 1992. *Lex & yacc.* "O'Reilly Media, Inc.", USA.

[21] Werney Lira, Pedro Santos Neto, and Luiz Osorio. 2024. Uma análise do uso de ferramentas de geração de código por alunos de computação. In *Anais do IV Simpósio Brasileiro de Educação em Computação* (Evento Online). SBC, Porto Alegre, RS, Brasil, 63–71. https://doi.org/10.5753/educomp.2024.237427

[22] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D. Le, and David Lo. 2024. Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. *ACM Trans. Softw. Eng. Methodol.* 33, 5, Article 116 (June 2024), 26 pages. https://doi.org/10.1145/3643674

[23] JunSeong Moon, RaeEun Yang, SoMin Cha, and Seong Baeg Kim. 2023. chatGPT vs Mentor : Programming Language Learning Assistance System for Beginners. In *2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS)*. 2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS), Penang, Malaysia, 106–110. https://doi.org/10.1109/ICSECS58457.2023.10256295

[24] Arghavan MORADI DAKHEL, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming (Jack) Jiang. 2023. GitHub Copilot AI pair programmer: Asset or Liability? *Journal of Systems and Software* 203 (2023), 111734. https://doi.org/10.1016/j.jss.2023.111734

[25] Mohamed Nejjar, Luca Zacharias, Fabian Stiehle, and Ingo Weber. 2024. LLMs for science: Usage for code generation and data analysis. *Journal of Software: Evolution and Process* n/a, n/a (2024), e2723. https://doi.org/10.1002/smr.2723 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2723

[26] Nhan Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories* (Pittsburgh, Pennsylvania) *(MSR '22)*. Association for Computing Machinery, New York, NY, USA, 1–5. https://doi.org/10.1145/3524842.3528470

[27] Francisco Ortin, Jose Quiroga, Oscar Rodriguez-Prieto, and Miguel Garcia. 2022. An empirical evaluation of Lex/Yacc and ANTLR parser generation tools. *PLOS ONE* 17, 3 (03 2022), 1–16. https://doi.org/10.1371/journal.pone.0264326

[28] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv:2302.06590 [cs.SE] https://arxiv.org/abs/2302.06590

[29] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans. Comput.-Hum. Interact.* 31, 1, Article 4 (Nov. 2023), 31 pages. https://doi.org/10.1145/3617367

[30] Ben Puryear and Gina Sprint. 2022. Github Copilot in the Classroom: Learning to Code with AI Assistance. *J. Comput. Sci. Coll.* 38, 1 (nov 2022), 37–47.

[31] Varun Shah. 2019. Towards Efficient Software Engineering in the Era of AI and ML: Best Practices and Challenges. *International Journal of Computer Science and Technology* 3, 3 (2019), 63–78.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[33] Han Xue and Yanmin Niu. 2023. Exercise Generation and Student Cognitive Ability Research Based on ChatGPT and Rasch Model. *IEEE Access* 11 (2023), 1–1. https://doi.org/10.1109/ACCESS.2023.3325741

[34] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. arXiv:2304.10778 [cs.SE] https://arxiv.org/abs/2304.10778