Avaliação Comparativa do GitHub Copilot e do Amazon CodeWhisperer na Geração Automática de Código-Fonte

Pedro C. Miranda[1], **Michelle Hanne S. de Andrade**[2]

[1]Engenharia de Software - Pontifícia Universidade Católica de Minas Gerais (PUC - Minas)
Belo Horizonte – MG – Brasil

[2]Departamento de Engenharia de Software – Pontifícia Universidade Católica de Minas Gerais (PUC Minas). Belo Horizonte, MG.

pcmiranda@sga.pucminas.br, michelleandrade@pucminas.br

Abstract. *Technology companies such as Amazon and OpenAI have been investing in the creation of Large Language Models (LLMs) that assist developers in generating source code. This work aims to evaluate the effectiveness of the automatic source code generation tools GitHub Copilot and Amazon CodeWhisperer. These tools use language models trained with public GitHub source code to generate suggestions based on natural language descriptions. This study collected 33 programming problems from the LeetCode website in three programming languages: Python, JavaScript, and Java. Subsequently, for each language, source code was generated using GitHub Copilot and Amazon CodeWhisperer. Then, the generated source codes were submitted and analyzed by the LeetCode platform. The results indicate that GitHub Copilot presented a higher accuracy rate (77.78%) compared to Amazon CodeWhisperer (64.8%), although no significant differences were found in terms of the efficiency and readability of the generated code.*

Resumo. *Empresas de tecnologia como Amazon e OpenAI têm investido na criação de Large Language Models (LLMs) que auxiliam desenvolvedores a gerarem códigos-fonte. Este trabalho procura avaliar a eficácia das ferramentas de geração automática de código-fonte GitHub Copilot e Amazon CodeWhisperer. Essas ferramentas usam modelos de linguagem treinados com código-fonte público do GitHub para gerar sugestões a partir de descrições em linguagem natural. Este estudo coletou 33 problemas de programação do site LeetCode, em três linguagens de programação: Python, Javascript e Java. Posteriormente, foi gerado para cada linguagem o código-fonte com o uso do GitHub Copilot e Amazon CodeWhisperer. Em seguida, os códigos-fonte gerados foram submetidos e analisados pela plataforma LeetCode. Os resultados indicam que o GitHub Copilot apresentou uma taxa de assertividade superior (77,78%) em comparação ao Amazon CodeWhisperer (64,8%), embora não tenha ocorrido diferenças significativas em termos de eficiência e legibilidade dos códigos gerados.*

1. Introdução

O preenchimento automático de código-fonte é uma funcionalidade muito útil para ambientes de desenvolvimento integrados, que pode acelerar substancialmente as atividades

de codificação. O desenvolvimento recente de *Large Language Models* (LLMs) como o GPT-3 abriu novas oportunidades para superar as limitações das técnicas existentes de geração de código-fonte [Vaithilingam et al. 2022]. LLMs são algoritmos de aprendizado profundo que podem realizar uma variedade de tarefas de processamento de linguagem natural, como reconhecimento, resumo, tradução, previsão e geração de texto, fundamentando-se no conhecimento adquirido a partir de conjuntos de dados massivos. [Abukhalaf et al. 2023].

Um exemplo de LLM é o GitHub Copilot, lançado em 2021 pela GitHub, um colaborador virtual alimentado por Inteligência Artificial (IA) que pode gerar código-fonte a partir de uma descrição em linguagem natural [Nguyen and Nadi 2022].

Imai (2022) demonstrou que o GitHub Copilot aumenta a produtividade dos desenvolvedores, no auxilio de geração de código-fonte. Outro sistema similar é o Amazon CodeWhisperer, lançado em 2022, que também pode traduzir uma descrição de problema para código-fonte [Mastropaolo et al. 2023]. Diversos estudos investigam as capacidades ferramentais de LLMs [Mastropaolo et al. 2023, Chen et al. 2021, Nguyen and Nadi 2022, Siddiq et al. 2023] no contexto de geração de código-fonte. No entanto, esses estudos não determinam qual é a melhor ferramenta para propor trechos de código-fonte. Portanto, o problema abordado neste estudo aponta a carência de trabalhos relativos à geração de código-fonte que apresentam evidências em consonância com as métricas de Engenharia de Software, como assertividade, eficiência e legibilidade.

É importante avaliar o desempenho dessas ferramentas para a área da Engenharia de Software, especialmente quando a IA desempenha o papel de *Pair-programming* [Imai 2022]. Essas ferramentas possuem o potencial de acelerar o processo de codificação, reduzir erros humanos e ampliar o acesso à programação para aqueles que não têm experiência técnica avançada [Wermelinger 2023]. No entanto, para que essas ferramentas sejam amplamente adotadas e confiáveis, é fundamental avaliar sua capacidade de gerar código-fonte correto e legível [Imai 2022].

O objetivo geral deste trabalho é avaliar a eficácia das sugestões do GitHub Copilot e Amazon CodeWhisperer na geração de código-fonte a partir de descrições em linguagem natural. Para alcançar o objetivo geral, foram definidos três objetivos específicos: i) avaliar o índice de assertividade para cada ferramenta; ii) analisar a tempo de execução do código-fonte gerado de cada ferramenta; e iii) avaliar a legibilidade do código-fonte gerado por cada ferramenta.

Este estudo é parte do Trabalho de Conclusão de Curso, e busca contribuir para o avanço do conhecimento sobre as capacidades e limitações dessas tecnologias, bem como orientar seu uso na Engenharia de Software. Para isso, foram avaliados o desempenho do GitHub Copilot e do Amazon CodeWhisperer em 33 problemas de programação em três diferentes linguagens (Java, JavaScript e Python). Os resultados mostraram uma vantagem para o GitHub Copilot em assertividade, com 77,78% contra 64,8% do Amazon CodeWhisperer. No entanto, as análises de eficiência e legibilidade não mostraram diferenças significativas entre as ferramentas.

O trabalho está estruturado em cinco partes distintas. A Seção 2 aborda os principais estudos relacionados. A Seção 3 detalha a metodologia utilizada no experimento. A Seção 4 apresenta os resultados obtidos e uma análise detalhada dos dados. Por fim, a

Seção 5 ressalta as considerações finais e sugere trabalhos futuros.

2. Trabalhos Relacionados

O principal trabalho relacionado ao tema é dos autores Nguyen e Nadi (2022). Eles realizam um estudo empírico para avaliar a precisão e a compreensibilidade das sugestões de código-fonte sintetizado pelo GitHub Copilot. O estudo usa 33 questões do LeetCode, um site de problemas de programação, para criar consultas para o GitHub Copilot em quatro linguagens diferentes: Python, Java, JavaScript e C. A avaliação da precisão das sugestões do GitHub Copilot é feita usando os casos de teste fornecidos pelo LeetCode e a compreensibilidade usando as métricas de complexidade cognitiva e ciclomática do SonarQube. Eles apontaram que as sugestões do GitHub Copilot em Java têm a maior precisão (57%) enquanto JavaScript tem a menor (27%).

O artigo escrito por Chen et al. (2021) apresenta Codex, um modelo de linguagem treinado com código-fonte público do GitHub, e estuda suas capacidades de escrita de código-fonte Python. Os autores propuseram uma nova métrica de correção funcional baseada em testes unitários, e cria um conjunto de avaliação chamado HumanEval com 164 problemas de programação originais. Os resultados mostram que o Codex supera modelos LLMs anteriores que não são especializados para código-fonte, bem como sistemas de *autocomplete* de código-fonte existentes, e pode gerar soluções corretas para a maioria dos problemas do HumanEval.

Yetistiren et al. (2022) realizam um estudo empírico para avaliar a qualidade do código-fonte gerado pelo GitHub Copilot. O estudo usa o conjunto de dados HumanEval, que contém 164 problemas de programação em Python, para criar consultas para o GitHub Copilot em diferentes modos: balanceado, criativo e preciso. A avaliação da qualidade do código-fonte gerado pelo GitHub Copilot é feita usando três métricas: validade, precisão e eficiência. Os resultados mostram que o GitHub Copilot conseguiu gerar código-fonte válido com uma taxa de sucesso de 91,5% e código-fonte correto com uma taxa de sucesso de 28,7%.

Denny et al. (2023) abordam em seu estudo o impacto do GitHub Copilot, na educação de programação. Os autores utilizam um protocolo para interagir com o GitHub Copilot e avaliar seu desempenho em 166 problemas em Python. Esse protocolo envolvia a formulação de descrições de problemas em linguagem natural e a análise das soluções geradas. Eles descobriram que o GitHub Copilot resolve cerca de metade dos problemas na primeira tentativa e resolve 60% dos problemas restantes apenas com mudanças na descrição do problema em linguagem natural.

3. Metodologia

Nesta pesquisa, foi adotada uma abordagem quantitativa. Dado o escopo da pesquisa, que visa comparar a eficácia de ferramentas de geração automática de código-fonte.

Baseado nos objetivos específicos deste estudo, foram elaboradas três questões de pesquisa (QP), e para cada uma foi elaborada hipóteses, sendo: a Hipótese nula (H0) e Hipótese alternativa (H1). A Hipótese nula (H0) de que não há diferença significativa entre os softwares (GitHub Copilot e Amazon CodeWhispere) e H1 sendo que o GitHub Copilot apresenta vantagem em comparação ao Amazon CodeWhisperer.

- (QP1) *Qual ferramenta apresenta maior assertividade nas suas sugestões?*
- (QP2) *Qual ferramenta gera as sugestões mais eficientes?*
- (QP3) *Qual ferramenta gera códigos-fonte mais legíveis?*

Na QP1, avalia-se qual ferramenta apresenta maior taxa de sugestões de código-fonte corretas, capazes de resolver o problema descrito em linguagem natural. Para validar, é necessário que o código-fonte sugerido seja aprovado em todos os casos de teste propostos na ferramenta LeetCode. Para avaliar a eficiência dos códigos-fonte e responder a QP2, avalia-se o tempo de execução de cada problema na plataforma LeetCode. Por último, na QP3 avalia-se a legibilidade do código-fonte gerado pelas ferramentas, são utilizadas medições de complexidade ciclomática e cognitiva.

A complexidade ciclomática é uma métrica que quantifica a quantidade de caminhos independentes no código-fonte, valores mais altos indicam um código-fonte mais difícil de testar e manter. Já a complexidade cognitiva avalia a dificuldade de compreensão do código-fonte, considerando a quantidade de conceitos e estruturas que um programador precisa entender.

3.1. Arranjo experimental

Para esse estudo foi usado os problemas de programação que estão disponíveis no site do LeetCode[1]. O LeetCode é uma plataforma online que oferece uma ampla coleção de problemas de algoritmos de programação, agrupados por níveis de dificuldade (*easy, medium e hard*). O LeetCode fornece os parâmetros para a assinatura da função de cada problema, bem como, realiza a validação da assertividade e eficiência com casos de teste.

Na primeira etapa da pesquisa, optou-se por uma amostra de 33 problemas do LeetCode, a mesma quantidade dos autores Nguyen e Nadi (2022). Foram selecionados 11 problemas de cada nível de dificuldade (*easy, medium e hard*), organizados pelo critério de maior percentual de aceitação na plataforma. A aceitação é uma métrica associada a cada problema, informando o percentual de envios que foram devidamente aceitos no LeetCode.

Para automatizar o processo de extração de informações, foi desenvolvido um *script* em Python (versão 3.10.12) que utilizou o GraphQL[2] da plataforma LeetCode para obter os enunciados e as assinaturas dos 33 problemas selecionados. Esses dados foram utilizados para gerar códigos-fonte correspondentes em três linguagens distintas: Python, Javascript e Java. A escolha das linguagens de programação é uma limitação deste estudo, porém, optou-se por utilizar as mesmas linguagens usadas no trabalho de Nguyen e Nadi (2022).

Durante a segunda etapa da pesquisa, os arquivos de assinatura de cada problema foram carregados aos projetos no Visual Studio Code[3], com as extensões do GitHub Copilot e Amazon CodeWhisperer instaladas. Desse modo, os códigos-fonte foram gerados separadamente nas três linguagens de programação.

Na terceira fase do processo, dedicada à validação da precisão e eficácia, os códigos-fonte gerados foram submetidos manualmente ao LeetCode para uma avaliação

[1] https://leetcode.com/
[2] https://graphql.org/
[3] https://code.visualstudio.com/

por meio de casos de teste específicos da plataforma. Caso o código-fonte tenha atingido 100% de acerto, o LeetCode mostra o tempo de execução. Foi registrado para cada problema, o percentual de casos de teste que o código-fonte acertou e o tempo de execução em milissegundos (ms).

A validação da legibilidade, por sua vez, foi conduzida com o uso do software SonarQube[1]. Essa ferramenta permitiu uma análise das métricas de complexidade ciclomática e cognitiva dos trechos de código-fonte.

Os artefatos deste estudo encontram em https://zenodo.org/records/13686929 [Miranda 2024].

3.2. Métricas

Para avaliar a eficácia das ferramentas GitHub Copilot e Amazon CodeWhisperer na geração automática de código-fonte, foram adotadas três métricas principais: assertividade, eficiência e legibilidade. Essas métricas também foram adotadas no trabalho relacionado de Nguyen e Nadi (2022).

A assertividade foi medida pelo percentual de códigos-fonte sugeridos que obtiveram êxito em todos os casos de teste, em relação ao total de problemas. Essa métrica foi estratificada por linguagem de programação e nível de dificuldade, proporcionando uma análise detalhada do desempenho de cada ferramenta em diferentes contextos.

A eficiência das sugestões foi avaliada com base no tempo de execução do código-fonte gerado. Foram avaliados somente os códigos-fonte que obtiveram 100% de acerto em todos os casos de teste, considerando ambas as ferramentas. Esses valores foram submetidos ao teste de Shapiro-Wilk para verificar a normalidade da distribuição com 95% de intervalo de confiança. Se a distribuição for normal, foi aplicado o Teste T de Student para comparar as médias dos tempos; caso contrário, o Teste U de Mann-Whitney, não paramétrico [Barbetta et al. 2010]. Esses testes serão utilizados para aceitar ou rejeitar a hipótese nula.

A legibilidade foi analisada usando as métricas de complexidade ciclomática e cognitiva. Conforme indicado por Nguyen e Nadi (2022), a complexidade ciclomática crescente implica mais ramificações no código-fonte, exigindo mais casos de teste. Assim, como na métrica de eficiência, serão apenas considerados resultados que obtiveram 100% de acerto em ambas as ferramentas.

4. Resultados e Discussão

Essa seção detalhará os dados coletados e resultados obtidos.

4.1. Caracterização do Conjunto de Dados

Primeiramente, foi realizada a análise de distribuição da porcentagem de acerto para cada problema, levando em consideração o GitHub Copilot e o Amazon CodeWhisperer. Como demonstrado na Figura 1, cada histograma mostra a frequência de acerto percentual de todos os problemas selecionados nesta pesquisa.

A Figura 1(a) apresenta a distribuição da porcentagem de acerto para a ferramenta GitHub Copilot. Observa-se que o intervalo com a maior frequência foi de 90 a 100%,

[1]https://www.sonarsource.com/

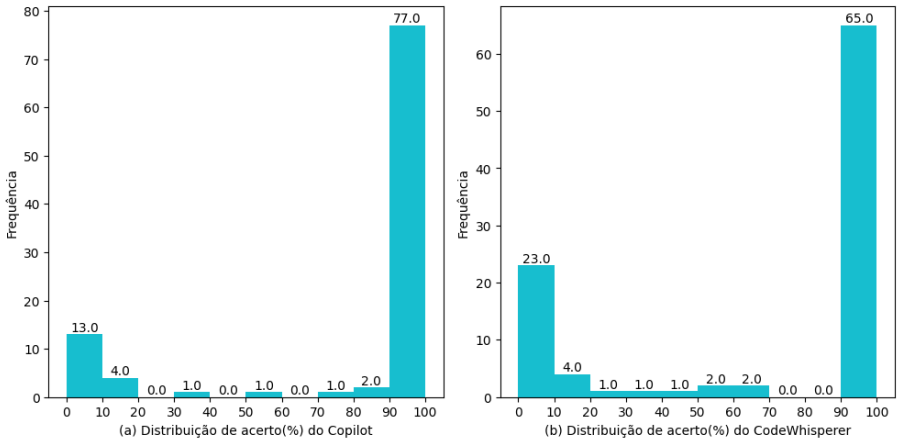

Figura 1. Histograma do percentual de acerto

com 77 ocorrências, indicando uma alta precisão nessa faixa de acertos. O segundo intervalo mais frequente foi de 0 a 10%, com 13 ocorrências.

Por outro lado, a Figura 1(b) mostra a distribuição da porcentagem de acerto para a ferramenta Amazon CodeWhisperer. Neste caso, o intervalo com a maior frequência foi o mesmo do GitHub Copilot, de 90 a 100%, com 65 ocorrências. No intervalo de 0 a 10%, houve 23 ocorrências, superior ao número do GitHub Copilot.

Com relação a distribuição por linguagem de programação, no GitHub Copilot a maioria das sugestões em Java, JavaScript e Python apresentou uma taxa de acerto de 100%, com 27, 25 e 25 ocorrências, respectivamente. Poucas sugestões tiveram resultado abaixo de 20%, indicando uma alta precisão geral. No entanto, o Amazon CodeWhisperer, apesar de também apresentar um grande número de sugestões corretas, com 26 ocorrências para Java, 21 para JavaScript e 18 para Python, mostrou maior sugestões abaixo de 20%.

Já com relação ao nível de dificuldade, o GitHub Copilot possui maior frequência de acertos (90% a 100%) em todos os níveis: 30 ocorrências nos problemas *easy*, 23 nos *medium* e 24 nos *hard*. No entanto, há 6 ocorrências com baixos resultados (0 a 10%) de acertividade. O Amazon CodeWhisperer apresenta uma distribuição semelhante, mas com menos ocorrências no intervalo de 90% a 100% para problemas *easy* (27 ocorrências) e *medium* (25 ocorrências). Nos problemas *hard*, o Amazon CodeWhisperer exibe uma distribuição mais dispersa, com 13 ocorrências tanto no intervalo de 90% a 100% quanto no intervalo de 0 a 10%.

A Figura 2 exibe os *boxplots* que representam a distribuição do tempo de execução, da complexidade ciclomática e da complexidade cognitiva dos códigos-fonte gerados por cada ferramenta. É importante ressaltar que os dados considerados foram somente aqueles em que ambas as ferramentas alcançaram uma taxa de acerto de 100% no mesmo problema.

Na Figura 2(a) mostra para o GitHub Copilot, os valores dos quartis e da mediana são os seguintes: o primeiro quartil (Q1) está em 3,50 ms, a mediana (Q2) em 32,00 ms e o terceiro quartil (Q3) em 65,75 ms. É notável a presença de *outliers*, como no problema "*Deepest Leaves Sum*"(169,0 ms) e "*Number of Submatrices That Sum to Target*"(549,0

ms), ambos em Python. Em contrapartida, para o Amazon CodeWhisperer, os valores são ligeiramente diferentes: Q1 em 4,25 ms, Q2 em 43,50 ms e Q3 em 67,75 ms. Da mesma forma, foram identificados *outliers* os mesmos *outliers* com valores diferentes.

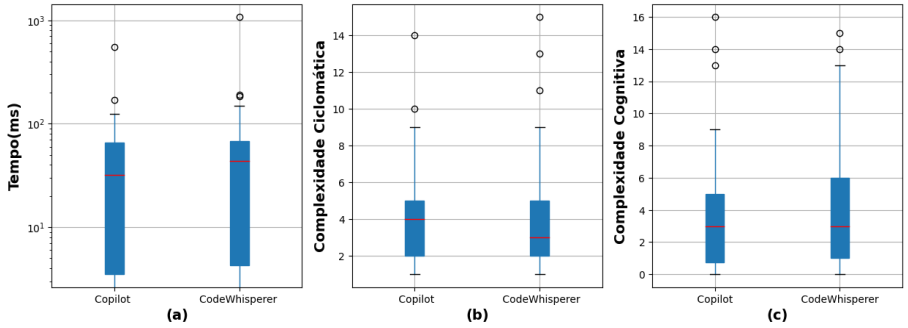

Figura 2. Distribuição dos dados

O boxplot exibido na Figura 2(b) de complexidade ciclomática no caso do GitHub Copilot, os quartis e a mediana assumem os seguintes valores: Q1 localiza-se em 2, Q2 em 4 e o Q3 em 5. Destaca-se a presença de *outliers* nos problemas "*Unique Paths III*"(10) em Python e "*Cherry Pickup II*"(14) em Java. Em contrapartida, para o Amazon CodeWhisperer, os valores apresentam similaridades: Q1 também em 2, Q2 em 3 e o Q3 em 5,0. Foram identificados *outliers* nos problemas "*Unique Paths III*"(11) em Python, "*N-Queens II*"(15) em JavaScript e "*Cherry Pickup II*"(13) em Java.

A Figura 2(c) ilustra a distribuição para complexidade cognitiva. Para GitHub Copilot, Q1 está em 0,75, Q2 em 3 e o Q3 em 5. Foram identificados *outliers* nos problemas "*Unique Paths III*"(14) em Python, "*Number of Submatrices That Sum to Target*"(13) em Python e "*Cherry Pickup II*"(16) em Java. Por outro lado, para Amazon CodeWhisperer, os valores são: Q1 em 1,0, Q2 em 3,0 e Q3 em 6,0. Os *outliers* foram encontrados nos problemas "*Unique Paths III*"(14) em Python e "*N-Queens II*"(15) em JavaScript.

4.2. Análise dos resultados

A seguir, são apresentadas as análise dos resultados relacionados às questões de pesquisa.

(QP1) *Qual ferramenta apresenta maior assertividade nas suas sugestões?*

Para responder à QP1, foram considerados apenas problemas com 100% de acerto. Observa-se que, do total de 99 problemas gerados e submetidos ao LeetCode, o GitHub Copilot acertou 77,78%, ou seja, 77 problemas, enquanto o Amazon CodeWhisperer alcançou 64,8%, acertando 64 dos 99 problemas. Assim, houve variação percentual de 16,88% entre as duas ferramentas.

Essa diferença é caracterizada na Tabela 1, onde são apresentados esses valores do GitHub Copilot em relação ao Amazon CodeWhisperer para as linguagens e níveis de dificuldades. Esses percentuais são apresentados em relação a amostra de 11 problemas. Pode-se observar que o Amazon CodeWhisperer é melhor no nível *medium* para as linguagens Java e JavaScript. Ambas as ferramentas se igualam em Java no nível *easy* e em Python no nível *medium*. Nos demais casos, a ferramenta GitHub Copilot é superior.

(QP2) *Qual ferramenta gera as sugestões mais eficientes?*

Linguagem	**Java**			**Javascript**			**Python**		
Dificuldade	*easy*	*medium*	*hard*	*easy*	*medium*	*hard*	*easy*	*medium*	*hard*
Copilot(%)	90,9	72,7	81,8	90,9	72,7	63,6	90,9	63,6	72,7
CodeWhisperer(%)	90,9	81,8	54,5	81,8	81,8	27,3	72,7	63,6	27,3

Tabela 1. Problemas com 100% de acerto por linguagem e dificuldade

Ao abordar a QP2, exploramos a eficiência das sugestões geradas pelas ferramentas GitHub Copilot e Amazon CodeWhisperer. Para essa análise, consideramos os dados de tempo de execução em milissegundos. Os resultados dos testes de Shapiro-Wilk revelam que esses dados não seguem uma distribuição normal, com *p-values* muito baixos (6,34e-12 para GitHub Copilot e 2,37e-14 para Amazon CodeWhisperer), indicando uma assimetria na distribuição em torno da média.

Além disso, ao realizar o Teste U de Mann-Whitney, com um *p-value* de 0,956, não há evidências suficientes para rejeitar H0, ou seja, não há diferença significativa entre as distribuições de tempo de execução de GitHub Copilot e Amazon CodeWhisperer. Por esse motivo aceita-se a hipótese nula.

A análise das medianas do tempo de execução do código-fonte por linguagem e dificuldade, conforme apresentado na Tabela 2, revela algumas tendências. Em Java, tanto o GitHub Copilot quanto o Amazon CodeWhisperer têm medianas relativamente baixas em comparação com outras linguagens. Para JavaScript e Python, as ferramentas mostram medianas maiores de tempo de execução, mas as medianas continuam bem próximas entre as ferramentas. A única exceção percebida é em relação ao Python no nível *hard*, contudo, a diferença não mostra ser estatisticamente significativa, conforme os resultados avaliados no Teste U de Mann-Whitney.

Linguagem	**Java**			**Javascript**			**Python**		
Dificuldade	*easy*	*medium*	*hard*	*easy*	*medium*	*hard*	*easy*	*medium*	*hard*
Copilot(ms)	1,0	5,0	23,0	60,0	89,0	62,0	23,0	45,0	34,0
CodeWhisperer(ms)	0,5	6,0	25,0	60,0	87,0	56,0	22,5	50,0	70,0

Tabela 2. Medianas do tempo de execução do código-fonte por linguagem e dificuldade

QP3) *Qual ferramenta gera códigos-fonte mais legíveis?*

Ao investigar a legibilidade dos códigos-fonte gerados pelo GitHub Copilot e Amazon CodeWhisperer, usou-se as métricas de complexidade ciclomática e cognitiva. Os testes de Shapiro-Wilk revelam que esses dados não seguem uma distribuição normal, com *p-value* muito baixo para ambas as métricas, sugerindo uma assimetria na distribuição em torno da média.

Além disso, ao realizar o Teste U de Mann-Whitney para ambas as métricas, encontramos *p-value* de 0,876 para complexidade ciclomática e 0,953 para complexidade cognitiva. Esses resultados indicam que não há evidências suficientes para rejeitar a hipótese nula, ou seja, não há diferença significativa entre as distribuições de complexidade ciclomática e cognitiva entre GitHub Copilot e Amazon CodeWhisperer.

O estudo de Nguyen e Nadi (2022) também avaliou a legibilidade, e, comparando com este estudo, os valores medianos das complexidades encontrados pelos autores foram superiores. A mediana encontrada para a complexidade ciclomática foi de 5, enquanto para a complexidade cognitiva foi de 6 em todas as linguagens. A abordagem definida para a escolha dos problemas no LeetCode pode ter sido determinante para essa diferença.

Os Testes U de Mann-Whitney não indicam diferenças estatisticamente significativas nas distribuições de complexidade ciclomática e cognitiva entre GitHub Copilot e Amazon CodeWhisperer. Corroborando com o Teste U, a análise das medianas mostra que nas linguagens Java e JavaScript a complexidadade cognitiva são similares (3 cargas congitivas). A análise ciclomática também apresenta dados próximos para a mediana nas linguagens Java e JavaScript, 3,5 no GitHub Copilot e 3,0 no Amazon CodeWhisperer. Já na linguagem Pyhton ocorre maior diferença, na complexidade ciclomática, o Amazon CodeWhisperer apresenta mediana de 2,5, enquanto o GitHub Copilot 4,0. Na complexidade cognitiva, a mediana do GitHub Copilot foi de 1,5, enquanto o Amazon CodeWhisperer de 0,5, sugerindo maior legibilidade do código.

Com base na proximidade observada entre as medianas de complexidade ciclomática e cognitiva foi realizado um cálculo de correlação de Pearson. O resultado para GitHub Copilot foi de 0,85 e para Amazon CodeWhisperer de 0,90, indicando uma forte correlação entre as duas métricas em ambas as ferramentas.

5. Conclusão

Este estudo conduziu uma análise detalhada das ferramentas de geração automática de código-fonte, GitHub Copilot e Amazon CodeWhisperer, usando um conjunto de 33 problemas de programação do LeetCode em Python, JavaScript e Java. Os resultados apresentados demonstram as capacidades e limitações dessas ferramentas, contribuindo como estudo para a Engenharia de Software.

Em termos de assertividade, o GitHub Copilot mostrou um desempenho superior em comparação com o Amazon CodeWhisperer. O GitHub Copilot apresentou uma taxa de acerto de 77,78%, enquanto o Amazon CodeWhisperer de 64,8%. Isso indica que o GitHub Copilot é mais eficiente em fornecer sugestões corretas que acertam 100% dos casos de teste propostos no LeetCode. A análise estratificada revelou que o GitHub Copilot teve um desempenho particularmente superior em Java e Python, enquanto o Amazon CodeWhisperer mostrou uma ligeira vantagem em JavaScript e em problemas de nível *medium*.

Em relação à eficiência, medida pelo tempo de execução do código-fonte gerado, os testes estatísticos não indicaram diferenças significativas entre as duas ferramentas. Apesar das diferenças na assertividade, a eficiência do código-fonte final produzido é similar. Quanto à legibilidade, avaliada pelas métricas de complexidade ciclomática e cognitiva, novamente, não foram encontradas diferenças estatisticamente significativas entre o GitHub Copilot e o Amazon CodeWhisperer.

Para pesquisas futuras, recomenda-se incluir maior variedade de linguagens de programação, além de aumentar o tamanho da amostra. A rápida evolução de LLMs na geração de códigos-fonte é uma limitação deste trabalho, por isso, torna-se necessário testar outras ferramentas de geração automática de código-fonte.

Referências

Abukhalaf, S., Hamdaqa, M., and Khomh, F. (2023). On codex prompt engineering for ocl generation: An empirical study. *arXiv preprint arXiv:2303.16244*.

Barbetta, P. A., Reis, M. M., and Bornia, A. C. (2010). *Estatística: para cursos de engenharia e informática*.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code.(2021). *arXiv preprint arXiv:2107.03374*.

Denny, P., Kumar, V., and Giacaman, N. (2023). Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 1136–1142.

Imai, S. (2022). Is github copilot a substitute for human pair-programming? an empirical study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ICSE '22, page 319–321, New York, NY, USA. Association for Computing Machinery.

Mastropaolo, A., Pascarella, L., Guglielmi, E., Ciniselli, M., Scalabrino, S., Oliveto, R., and Bavota, G. (2023). On the robustness of code generation techniques: An empirical study on github copilot. In *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23, page 2149–2160. IEEE Press.

Miranda, P. (2024). [DataSet] Avaliação Comparativa do GitHub Copilot e do Amazon CodeWhisperer na Geração Automatica de Código-Fonte.

Nguyen, N. and Nadi, S. (2022). An empirical evaluation of github copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 1–5, New York, NY, USA. Association for Computing Machinery.

Siddiq, M. L., Samee, A., Azgor, S. R., Haider, M. A., Sawraz, S. I., and Santos, J. C. S. (2023). Zero-shot prompting for code complexity prediction using github copilot. In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, pages 56–59.

Vaithilingam, P., Zhang, T., and Glassman, E. L. (2022). Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI EA '22, New York, NY, USA. Association for Computing Machinery.

Wermelinger, M. (2023). Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, page 172–178, New York, NY, USA. Association for Computing Machinery.

Yetistiren, B., Ozsoy, I., and Tuzun, E. (2022). Assessing the quality of github copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2022, page 62–71, New York, NY, USA. Association for Computing Machinery.