

ORIGINAL

Comparative Study of AI Code Generation Tools: Quality Assessment and Performance Analysis

Estudio Comparativo de Herramientas de Generación de Código por IA: Evaluación de Calidad y Análisis de Desempeño

Michael Alexander Florez Muñoz¹ , Juan Camilo Jaramillo De La Torre¹ , Stefany Pareja López¹ , Stiven Herrera¹ , Christian Andrés Candela Uribe¹ 

¹Universidad del Quindío. Colombia.

Cite as: Florez Muñoz MA, Jaramillo De La Torre JC, Pareja López S, Herrera S, Candela Uribe CA. Comparative Study of AI Code Generation Tools: Quality Assessment and Performance Analysis. LatIA. 2024; 2:104. <https://doi.org/10.62486/latia2024104>

Submitted: 04-02-2024

Revised: 10-05-2024

Accepted: 15-08-2024

Publi: 16-08-2024

Editor: Prof. Dr. Javier González Argote 

ABSTRACT

Artificial intelligence (AI) code generation tools are crucial in software development, processing natural language to improve programming efficiency. Their increasing integration in various industries highlights their potential to transform the way programmers approach and execute software projects. The present research was conducted with the purpose of determining the accuracy and quality of code generated by artificial intelligence (AI) tools. The study began with a systematic mapping of the literature to identify applicable AI tools. Databases such as ACM, Engineering Source, Academic Search Ultimate, IEEE Xplore and Scopus were consulted, from which 621 papers were initially extracted. After applying inclusion criteria, such as English-language papers in computing areas published between 2020 and 2024, 113 resources were selected. A further screening process reduced this number to 44 papers, which identified 11 AI tools for code generation. The method used was a comparative study in which ten programming exercises of varying levels of difficulty were designed and the results obtained from 4 of them are presented. The identified tools generated code for these exercises in different programming languages. The quality of the generated code was evaluated using the SonarQube static analyzer, considering aspects such as safety, reliability and maintainability. The results showed significant variations in code quality among the AI tools. Bing as a code generation tool showed slightly superior performance compared to others, although none stood out as a noticeably superior AI. In conclusion, the research evidenced that, although AI tools for code generation are promising, they still require a pilot to reach their full potential, giving evidence that there is still a long way to go.

Keywords: Artificial Intelligence; Coding Assistants; Code Generation.

RESUMEN

Las herramientas de generación de código con inteligencia artificial (IA) son cruciales en el desarrollo de software, procesando lenguaje natural para mejorar la eficiencia en la programación. Su creciente integración en diversas industrias destaca su potencial para transformar la manera en que los programadores abordan y ejecutan proyectos de software. La presente investigación se realizó con el propósito de determinar la precisión y calidad del código generado por herramientas de inteligencia artificial (IA). El estudio comenzó con un mapeo sistemático de la literatura para identificar las herramientas de IA aplicables. Se consultaron bases de datos como ACM, Engineering Source, Academic Search Ultimate, IEEE Xplore y Scopus, de donde se extrajeron inicialmente 621 artículos. Tras aplicar criterios de inclusión, como artículos en inglés de áreas de la computación publicados entre 2020 y 2024, se seleccionaron 113 recursos. Un proceso de tamizaje adicional redujo esta cifra a 44 artículos, que permitieron identificar 11 herramientas de IA para la generación de código. El método utilizado fue un estudio comparativo en el que se diseñaron diez ejercicios

de programación con diversos niveles de dificultad de los cuales se presentan los resultados obtenidos de 4 de ellos. Las herramientas identificadas generaron código para estos ejercicios en diferentes lenguajes de programación. La calidad del código generado fue evaluada mediante el analizador estático SonarQube, considerando aspectos como seguridad, fiabilidad y mantenibilidad. Los resultados mostraron variaciones significativas en la calidad del código entre las herramientas de IA. Bing como herramienta de generación de código mostró un rendimiento ligeramente superior en comparación con otras, aunque ninguna destacó como una IA notablemente superior. En conclusión, la investigación evidenció que, aunque las herramientas de IA para la generación de código son prometedoras, aún requieren de un piloto para alcanzar su máximo potencial, dando a evidenciar que aún queda mucho por avanzar.

Palabras clave: Inteligencia Artificial; Asistentes de Codificación; Generación de Código.

INTRODUCTION

Artificial intelligence (AI) has experienced exponential development in recent decades, transforming virtually every aspect of our lives (Finnie-Ansley et al., 2022; Lee, 2020). From virtual assistants to recommender systems, AI has demonstrated its ability to process and analyze large amounts of data (Gruson, 2021; Ruiz Baquero, 2018), identify complex patterns, and make informed decisions. Thus, it has become a fundamental tool in various fields (Yadav & Pandey, 2020).

One of the most promising fields in which AI has a significant impact is the generation of functional and efficient software code (Yan et al., 2023; Hernández-Pinilla & Mendoza-Moreno, n.d.). The importance of code generation using AI lies in its ability to save time and effort (Marar, 2024; Koziolek et al., 2023) while speeding up the software development process and reducing the possibility of human error (Chemnitz et al., 2023; Alvarado Rojas, 2015). In addition, these tools can help understand and learn new programming languages, which is particularly useful for novice developers (Llanos et al., 2021; De Giusti et al., 2023).

Code generation using AI involves using machine learning models and algorithms to create, modify, or improve the source code of a computer program (Wolfschwenger et al., 2023).

(Wolfschwenger et al., 2023; Azaiz et al., 2023). This technology can potentially revolutionize software development by speeding up the coding process, improving code quality, and facilitating collaboration between developers (Pasquinelli et al., 2022; Tseng et al., 2023).

This study focuses on evaluating the accuracy and quality of code generated by 11 of the AI platforms. The process includes code generation to solve ten tests with an incremental difficulty level. Subsequently, the resulting codes are evaluated using static code analysis tools. The objective is to compare the performance of the platforms and select the most representative and capable of generating functional and quality code.

This paper has the following sections: methods, results, conclusions, and bibliography.

METHOD

The research was carried out in four main phases: identification of artificial intelligences, code generation, code evaluation and analysis of results.

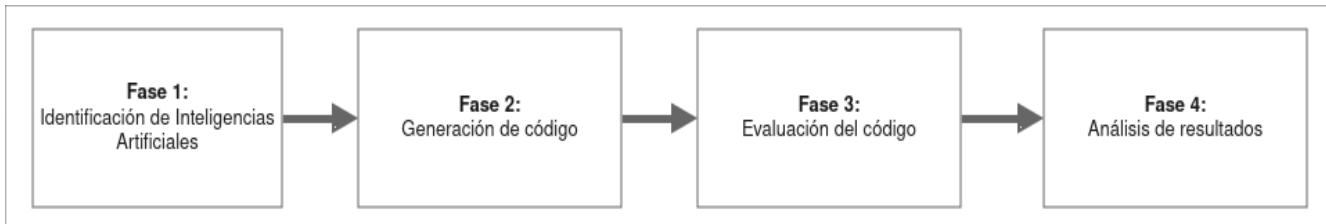


Figure 1. Phases of the process for the construction of the SMS

Phase 1: identification of Artificial Intelligences

The main objective of this phase is to define the artificial intelligence-based tools that will be used to perform the code generation, therefore, a systematic mapping of the literature to identify artificial intelligence capable of code generation (Macchi & Solari, 2012; Carrizo & Moller, 2018; Kitchenham et al., 2010). A search was conducted in several databases, including ACM, Engineering Source, Academic Search Ultimate, IEEE Xplore, and Scopus, resulting in 621 relevant scholarly articles and online resources. After applying inclusion criteria based on articles and proceedings in English about the areas of engineering and computer science published between 2020 and 2024, the total number of resources was reduced to 113. Then, a screening process was performed, resulting in 44 articles of interest, which allowed the identification of the 11 artificial intelligences for code generation: CodeGPT, Replit, Textsynth, Amazon CodeWhisperer, Bing, ChatGPT, Claude, Codeium, Gemini, GitHub Copilot and Tabnine.

Phase 2: code generation

The objective of this phase is that the previously selected tools were tested, for which ten programming exercises were designed with different difficulty levels, covering a wide range of tasks and concepts. The seminar teacher elaborated on these exercises: Assistants for code generation. Subsequently, the 11 identified code-generating tools were asked to solve these exercises in different programming languages, such as PHP, Python, Java, JavaScript, TypeScript, C#, Kotlin, Go, and Ruby.

The exercises designed for evaluation include: 1. sort a set of elements, 2. search for an element within a set, 3. count the occurrences of a specific element, 4. register users, 5. implement a REST API for user management (CRUD), 6. Develop a GUI for this user management CRUD, 7. perform unit testing for the REST API, 8. perform testing for the user management GUI, 9. implement a REST API for user management with token validation, and 10. create a GUI for this protected CRUD.

Phase 3: code evaluation

The main objective of this phase is to determine the quality of the code generated by each of the tools in the different languages, taking into account the effectiveness and reliability of each tool in terms of adherence to good programming practices and code performance optimization. To carry out those mentioned above, a review was performed by the students using a static code analyzer (Jenkins and SonarQube), which allowed different tests to verify the quality of the code, taking into account aspects of security, reliability, and maintainability of the code, in order to subsequently determine which artificial intelligence has higher quality when generating code.

Phase 4: analysis of results

The main objective of this phase is to contrast the results obtained, that is, which are the most adequate tools to generate quality and functional code. An analysis was generated considering all the information obtained in the phase code evaluation.

RESULTS

This section details the results obtained in the different stages of the project development. Specific search strings were designed for each selected database using the previously defined terms. This approach ensured relevant coverage of the scope of the study.

Search Chains

To optimize the collection of relevant documents related to automatic code generation within the field of artificial intelligence, specific search strings were developed for each database, adjusting according to previously established terms (table 1).

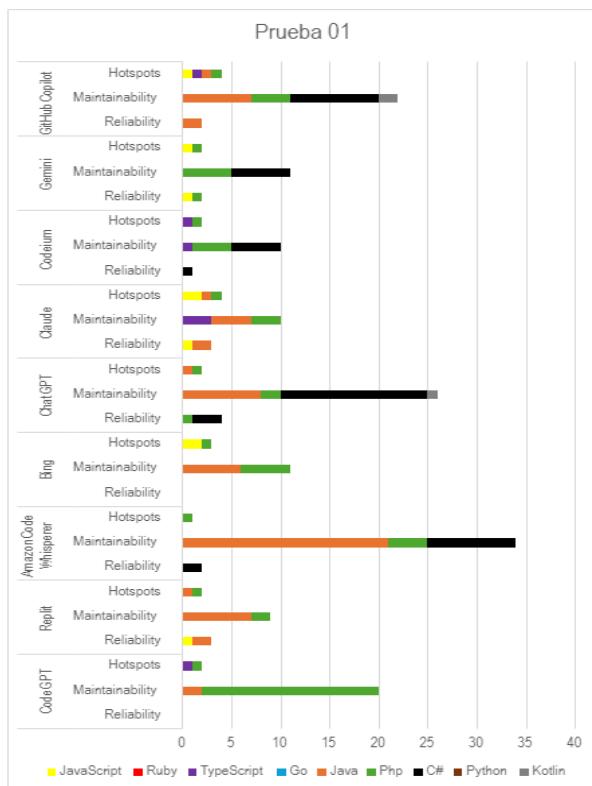
Table 1. Search strings

Database	Search string
Academic Search Ultimate	TI (("artificial intelligence" OR "intelligent systems" OR ai) AND ("code generation" OR "automatic code creation" OR "automatic code development" OR "software autogeneration" OR "code assistants" OR "code generator" OR "automatic coding")) OR AB (("artificial intelligence" OR "intelligent systems" OR ai) AND ("code generation" OR "automatic code creation" OR "automatic code development" OR "software autogeneration" OR "code assistants" OR "code generator" OR "automatic coding")) OR KW (("artificial intelligence" OR "intelligent systems" OR ai) AND ("code generation" OR "automatic code creation" OR "automatic code development" OR "software autogeneration" OR "code assistants" OR "code generator" OR "automatic coding"))
ACM	[[[Title: "artificial intelligence"] OR [Title: "intelligent systems"] OR [Title: ai]] AND [[Title: "code generation"] OR [Title: "automatic code creation"] OR [Title: "automatic code development"] OR [Title: "software autogeneration"] OR [Title: "code assistants"] OR [Title: "code generator"] OR [Title: "automatic coding"]]]] OR [[[Abstract: "artificial intelligence"] OR [Abstract: "intelligent systems"] OR [Abstract: ai]] AND [[Abstract: "code generation"] OR [Abstract: "automatic code creation"] OR [Abstract: "automatic code development"] OR [Abstract: "software autogeneration"] OR [Abstract: "code assistants"] OR [Abstract: "code generator"] OR [Abstract: "automatic coding"]]]] OR [[[Keywords: "artificial intelligence"] OR [Keywords: "intelligent systems"] OR [Keywords: ai]] AND [[Keywords: "code generation"] OR [Keywords: "automatic code creation"] OR [Keywords: "automatic code development"] OR [Keywords: "software autogeneration"] OR [Keywords: "code assistants"] OR [Keywords: "code generator"] OR [Keywords: "automatic coding"]]]]
Engineering source	TI (("artificial intelligence" OR "intelligent systems" OR ai) AND ("code generation" OR "automatic code creation" OR "automatic code development" OR "software autogeneration")

	OR “code assistants” OR “code generator” OR “automatic coding”)) OR AB ((“artificial intelligence” OR “intelligent systems” OR ai) AND (“code generation” OR “automatic code creation” OR “automatic code development” OR “software autogeneration” OR “code assistants” OR “code generator” OR “automatic coding”)) OR KW ((“artificial intelligence” OR “intelligent systems” OR ai) AND (“code generation” OR “automatic code creation” OR “automatic code development” OR “software autogeneration” OR “code assistants” OR “code generator” OR “automatic coding”))
IEEE Xplore	(“Document Title”:“artificial intelligence” OR “Document Title”:“intelligent systems” OR “Document Title”:ai) AND (“Document Title”:“code generation” OR “Document Title”:“automatic code creation” OR “Document Title”:“automatic code development” OR “Document Title”:“software autogeneration” OR “Document Title”:“code assistants” OR “Document Title”: “code generator” OR “Document Title”:“automatic coding”) OR (“Abstract”:“artificial intelligence” OR “Abstract”:“intelligent systems” OR “Abstract”:ai) AND (“Abstract”:“code generation” OR “Abstract”:“automatic code creation” OR “Abstract”:“automatic code development” OR “Abstract”:“software autogeneration” OR “Abstract”:“code assistants” OR “Abstract”: “code generator” OR “Abstract”:“automatic coding”) OR (“Author Keywords”:“artificial intelligence” OR “Author Keywords”:“intelligent systems” OR “Author Keywords”:ai) AND (“Author Keywords”:“code generation” OR “Author Keywords”:“automatic code development” OR “Author Keywords”:“software autogeneration” OR “Author Keywords”:“code assistants” OR “Author Keywords”: “code generator” OR “Author Keywords”:“automatic coding”)
Scopus	TITLE-ABS-KEY ((“artificial intelligence” OR “intelligent systems” OR ai) AND (“code generation” OR “automatic code creation” OR “automatic code development” OR “software autogeneration” OR “code assistants” OR “code generator” OR “automatic coding”))

Table 2. Results of the database search

	Academic Search Ultimate	ACM	Engineering source	IEEE Xplore	Scopus	Total
No exclusion criteria	19	41	11	68	482	621
With exclusion criteria	12	29	7	30	35	113
Screening	7	12	0	15	10	44
Participation	15,91 %	27,27 %	0 %	34,09 %	22,73 %	100 %

**Figure 1.** Test 01

After executing the queries, 621 studies were obtained, of which 44 were finally selected.

The following results are from the most relevant tests of the different AI code-generating tools. This graph shows the tools involved in the tests and their results when passed through the SonarQube tool that reviews the source code statically.

This test 01 uses different artificial intelligences to generate the code to capture a person's data, connect to the database, create the table if it does not exist, and save the person's data in the database.

In this evaluation, it is important to highlight that all the tools analyzed presented optimal performance in programming languages such as Python, Go, and Ruby. Likewise, it was observed that the code analysis tool offered more recommendations in languages such as Java, PHP, and C#.

On the other hand, it is worth noting that Codeium presented the lowest number of recommendations, standing out among the other tools. However, for the test, which is relatively simple, there are quite a few recommendations. In contrast, Amazon Code Whisperer was the tool that generated the highest number of recommendations according to the analysis performed.

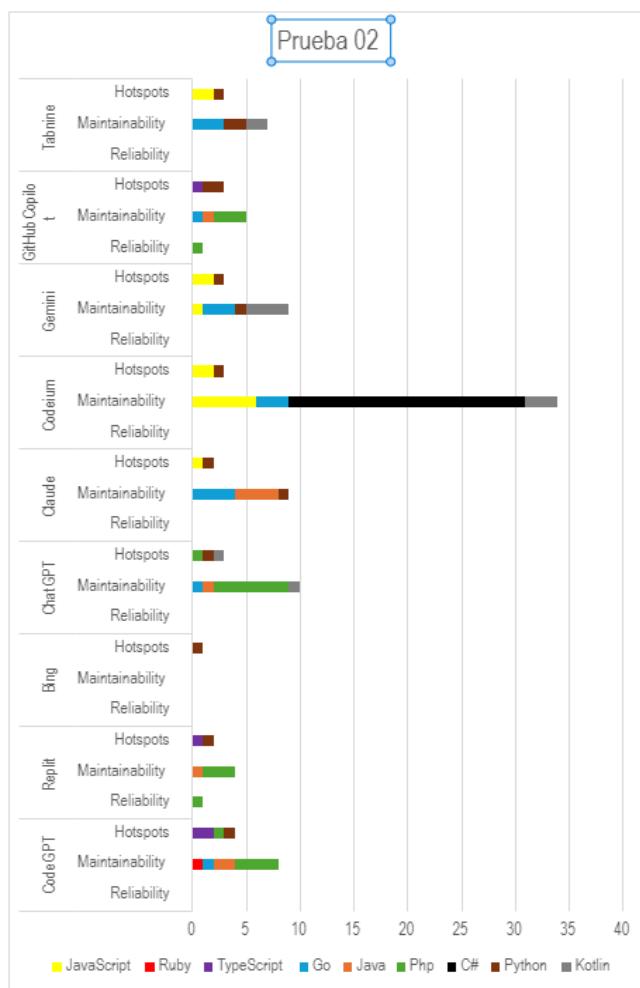


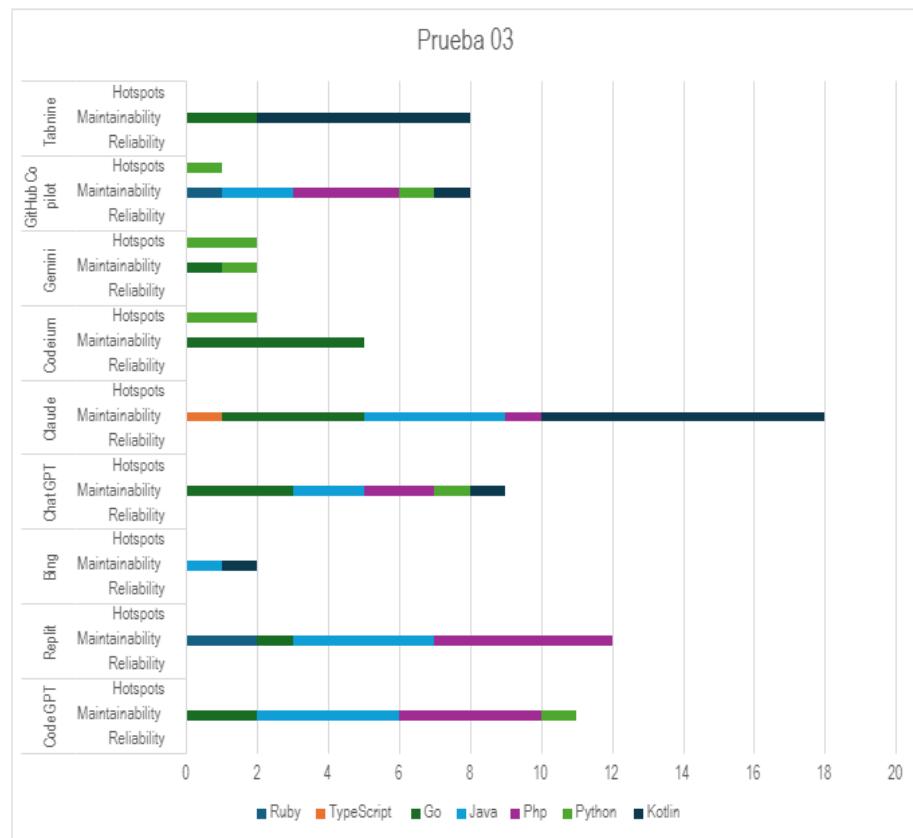
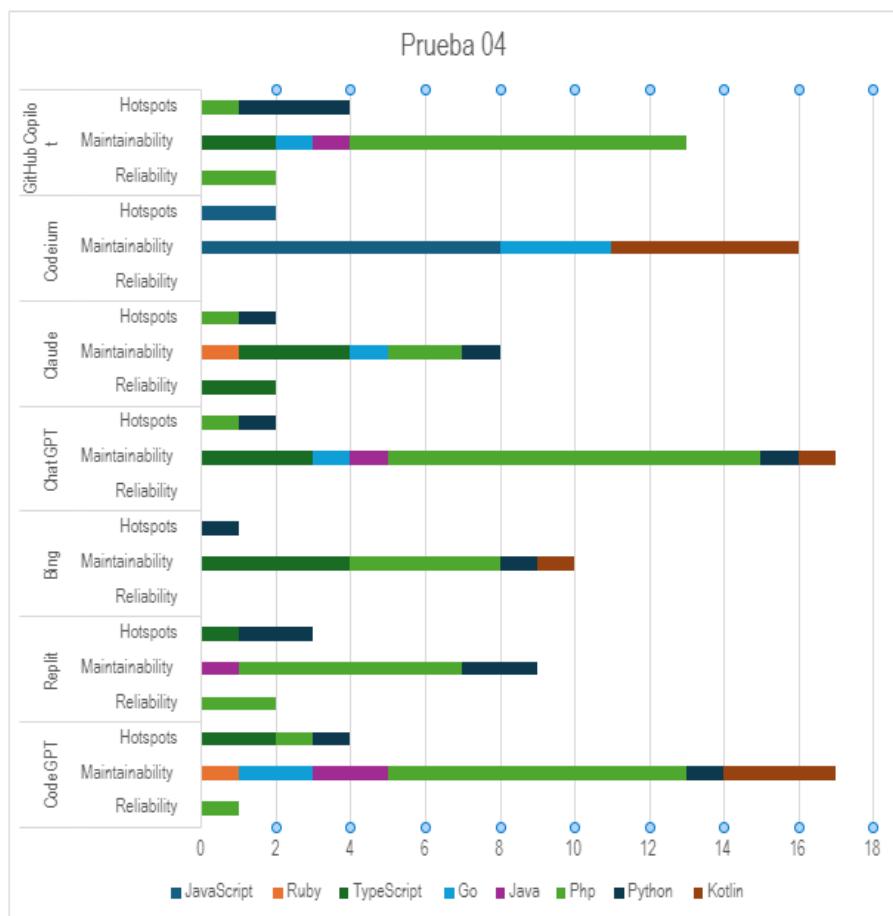
Figure 2. Test 02

This test 02 uses different artificial intelligences to generate the code necessary to complete the CRUD operations to manage users.

In this evaluation, it is important to highlight that all the tools analyzed presented optimal performance in programming languages such as Ruby and TypeScript. Likewise, it was observed that the code analysis tool offered more recommendations in languages such as Go, PHP, and C#.

On the other hand, it is worth noting that Bing presented the lowest number of recommendations, standing out among the other tools. In contrast, Codeium was the tool that generated the highest number of recommendations according to the analysis performed.

This test 03 uses different artificial intelligences to generate the appropriate code to test the functionalities of the CRUD generated in past tests. In this evaluation, it is important to highlight that all the tools analyzed presented an optimal performance in programming languages such as Ruby and TypeScript. Likewise, it was observed that the code analysis tool offered more recommendations in languages such as Go, Java, Kotlin, and PHP.

**Figure 3.** Test 03**Figure 4.** Test 04

On the other hand, it is worth noting that Bing and Gemini presented the lowest number of recommendations, standing out among the other tools. In contrast, Claude was the tool that generated the highest number of recommendations according to the analysis performed.

This test 04 uses different artificial intelligences to generate the appropriate code to perform login and route protection functionalities.

In this evaluation, it is important to highlight that all the tools analyzed presented optimal performance in programming languages such as Ruby and Java. It was also observed that the code analysis tool offered more recommendations in the PHP language.

On the other hand, it is worth noting that Bing presented the lowest number of recommendations, standing out among the other tools. In contrast, CodeGPT was the tool that generated the highest number of recommendations according to the analysis performed.

Finally, the total count of reports generated for each tool in the various tests performed is presented. This count includes only the tools that participated in all tests, excluding Textsynth, Amazon CodeWhisperer, Gemini, and Tabnine, which presented difficulties in generating functional code (table 3).

Table 3. Quality reports of the code generated for each tool				
	Reliability	Maintainability	Hotspots	Total
Replit	19	80	7	106
Bing	15	86	8	109
Claude	23	88	15	126
CodeGPT	14	110	12	136
Codeium	11	121	11	143
ChatGPT	10	160	8	178
GitHub Copilot	15	181	14	210

CONCLUSIONS

This paper presents the results of an SMS-type research focused on identifying the best AI tools to generate functional and quality code. Studies in Spanish and English belonging to the areas of engineering and computer science, published between 2020 and 2024, were considered. Through this study, it is evident that the various artificial bits of intelligence evaluated still need to present a stable behavior regarding the quality and functionality of the code they generate since the analysis results were not constant for any of the tools in the ten tests performed. It was observed that most of the errors reported were related to the quality of the code, in particular, characteristics that affect its maintainability.

According to the results obtained with this work, it stands out that the tool with the best performance in generating functional and quality code was Replit, with 106 reports found. This result is based on the contrast between the number of tests each AI used and the total number of errors reported by the static code analysis tool (SonarQube). Likewise, it was observed that the GitHub Copilot tool presented the lowest performance in the evaluation, with 210 reports found. Despite being one of the least problematic in generating functional code, it presented quality failures in the analysis results.

During phase 3 of the study, the following incidents were identified: the Textsynth and Amazon CodeWhisperer tools were discarded after the third and fourth tests due to problems with the usage plan and limitations in the interaction with them. Although Gemini and Tabnine showed adequate behavior during the generation and evaluation phases, they faced difficulties when generating functional code for one of the tests, which prevented their integration into the results obtained with the other tools.

BIBLIOGRAPHIC REFERENCES

1. Alvarado Rojas, M. E. (2015). UNA MIRADA A LA INTELIGENCIA ARTIFICIAL.
2. Revista Ingeniería, Matemáticas y Ciencias de La Información, 2(3). <http://ojs.urepublicana.edu.co/index.php/ingenieria/article/view/234>
3. Azaiz, I., Deckarm, O., & Strickroth, S. (2023). AI-Enhanced Auto-Correction of Programming Exercises: How Effective is GPT-3.5? International Journal of Engineering Pedagogy (IJEP), 13(8), 67-83. <https://doi.org/10.3991/ijep.v13i8.45621>
4. Carrizo, D., & Moller, C. (2018). Estructuras metodológicas de revisiones sistemáticas de literatura en Ingeniería de Software: un estudio de mapeo sistemático. Ingeniare.

5. Revista Chilena de Ingeniería, 26, 45-54. <https://doi.org/10.4067/S0718-33052018000500045>
6. Chemnitz, L., Reichenbach, D., Aldebes, H., Naveed, M., Narasimhan, K., & Mezini, M. (2023). Towards Code Generation from BDD Test Case Specifications: A Vision. 2023 IEEE/ACM 2nd International Conference on AI Engineering - Software Engineering for AI (CAIN), 139-144. <https://doi.org/10.1109/CAIN58948.2023.00031>
7. De Giusti, L. C., Villarreal, G. L., Ibañez, E. J., & De Giusti, A. E. (2023). Aprendizaje y enseñanza de programación: El desafío de herramientas de Inteligencia Artificial como ChatGPT. Web
8. Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. Proceedings of the 24th Australasian Computing Education Conference, 10-19. <https://doi.org/10.1145/3511861.3511863>
9. Gruson, D. (2021). Big Data, inteligencia artificial y medicina de laboratorio: la hora de la integración. Advances in Laboratory Medicine / Avances En Medicina de Laboratorio, 2(1), 5-7. <https://doi.org/10.1515/almed-2021-0014>
10. Hernández-Pinilla, D. V., & Mendoza-Moreno, J. F. (n.d.). La Inteligencia Artificial como una herramienta para potenciar el desarrollo de software.
11. Kitchenham, B., Pretorius, R., Budgen, D., Brereton, Pearl., Turner, M., Niazi, M., & Linkman, S. (2010). Systematic literature reviews in software engineering - A tertiary study. Information and Software Technology, 52(8), 792-805. <https://doi.org/10.1016/j.infsof.2010.03.006>
12. Kozolek, H., Gruener, S., & Ashiwal, V. (2023). ChatGPT for PLC/DCS Control Logic Generation. 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), 1-8. <https://doi.org/10.1109/ETFA54631.2023.10275411>
13. Lee, R. S. T. (2020). Artificial Intelligence in Daily Life. Springer Singapore. <https://doi.org/10.1007/978-981-15-7695-9>
14. Llanos Mosquera, J. M., Hidalgo Suarez, C. G., & Bucheli Guerrero, V. A. (2021). Una revisión sistemática sobre aula invertida y aprendizaje colaborativo apoyados en inteligencia artificial para el aprendizaje de programación. Tecnura, 25(69), 196-214. <https://doi.org/10.14483/22487638.16934>
15. Macchi, D., & Solari, M. (2012). Mapeo sistemático de la literatura sobre la Adopción de Inspecciones de Software.
16. Marar, H. W. (2024). Advancements in software engineering using AI. Computer Software and Media Applications, 6(1), 3906. <https://doi.org/10.24294/csma.v6i1.3906>
17. Pasquinelli, M., Cafassi, E., Monti, C., Peckaitis, H., & Zarauza, G. (2022). Cómo una máquina aprende y falla - Una gramática del error para la Inteligencia Artificial. Hipertextos, 10(17), 13-29. <https://doi.org/10.24215/23143924e054>
18. Ruiz Baquero, P. E. (2018). Avances en inteligencia artificial y su impacto en la sociedad. <https://repository.upb.edu.co/handle/20.500.11912/4942>
19. Tseng, A., Hahnemann, L., Humberto, M. E., Gaitan, P., Antonio, M., & Acosta, P. (2023). ChatGPT desarrollando código fuente de software para historias de usuarios en una consultora, Lima, 2023. Repositorio Institucional - UCV. <https://repositorio.ucv.edu.pe/handle/20.500.12692/133838>
20. Wolfschwenger, P., Sabitzer, B., & Lavicza, Z. (2023). Integrating Cloud-Based AI in Software Engineers' Professional Training and Development. 2023 IEEE Frontiers in Education Conference (FIE), 1-5. <https://doi.org/10.1109/FIE58773.2023.10343391>
21. Yadav, R. K., & Pandey, M. (2020). Artificial Intelligence and its Application in Various Fields. International Journal of Engineering and Advanced Technology, 9(5), 1336-1339. <https://doi.org/10.35940/ijeat.D9144.069520>

22. Yan, D., Gao, Z., & Liu, Z. (2023). A Closer Look at Different Difficulty Levels Code Generation Abilities of ChatGPT. 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), 1887-1898. <https://doi.org/10.1109/ASE56229.2023.00096>

FINANCING

The authors did not receive financing for the development of this research.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest.

AUTHORSHIP CONTRIBUTION

Conceptualization: Michael Alexander Florez Muñoz, Juan Camilo Jaramillo De La Torre, Stefany Pareja López, Stiven Herrera, Christian Andrés Candela Uribe.

Investigation: Michael Alexander Florez Muñoz, Juan Camilo Jaramillo De La Torre, Stefany Pareja López, Stiven Herrera, Christian Andrés Candela Uribe.

Methodology : Michael Alexander Florez Muñoz, Juan Camilo Jaramillo De La Torre, Stefany Pareja López, Stiven Herrera, Christian Andrés Candela Uribe.

Drafting - original draft: Michael Alexander Florez Muñoz, Juan Camilo Jaramillo De La Torre, Stefany Pareja López, Stiven Herrera, Christian Andrés Candela Uribe.

Writing - proofreading and editing: Michael Alexander Florez Muñoz, Juan Camilo Jaramillo De La Torre, Stefany Pareja López, Stiven Herrera, Christian Andrés Candela Uribe.