

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.

Citar como: 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>

Recibido: 04-02-2024

Revisado: 10-05-2024

Aceptado: 15-08-2024

Publicado: 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.

INTRODUCCIÓN

La inteligencia artificial (IA) ha experimentado un desarrollo exponencial en las últimas décadas, transformando prácticamente todos los aspectos de nuestras vidas (Finnie-Ansley et al., 2022; Lee, 2020). Desde asistentes virtuales hasta sistemas de recomendación, la IA ha demostrado su capacidad para procesar y analizar grandes cantidades de datos (Gruson, 2021; Ruiz Baquero, 2018), identificar patrones complejos y tomar decisiones informadas. Convirtiéndose así, en una herramienta fundamental en diversos ámbitos (Yadav & Pandey, 2020).

Uno de los campos más prometedores en los que la IA está teniendo un impacto significativo es en la generación de código software funcional y eficiente (Yan et al., 2023; Hernández-Pinilla & Mendoza-Moreno, n.d). La importancia de la generación de código mediante IA radica en su capacidad para ahorrar tiempo y esfuerzo (Marar, 2024; Koziol et al., 2023), al mismo tiempo que acelera el proceso de desarrollo de software y reduce la posibilidad de errores humanos (Chemnitz et al., 2023; Alvarado Rojas, 2015). Además, estas herramientas pueden ayudar a facilitar la comprensión y el aprendizaje de nuevos lenguajes de programación, lo que resulta particularmente útil para los desarrolladores principiantes (Llanos et al., 2021; De Giusti et al., 2023).

La generación de código mediante IA implica el uso de modelos y algoritmos de aprendizaje automático para crear, modificar o mejorar el código fuente de un programa informático (Wolfschwenger et al., 2023; Azaiz et al., 2023). Esta tecnología tiene el potencial de revolucionar la forma en que se desarrolla el software, al acelerar el proceso de codificación, mejorar la calidad del código y facilitar la colaboración entre desarrolladores (Pasquinelli et al., 2022; Tseng et al., 2023).

Este estudio se enfoca en evaluar la precisión y calidad del código generado por 11 de las plataformas de IA. El proceso incluye la generación de código para resolver diez pruebas, con un nivel de dificultad incremental. Posteriormente, se evalúan los códigos resultantes utilizando herramientas de análisis estático de código. El objetivo es comparar el rendimiento de las plataformas y seleccionar las más representativas y capaces en la generación de código funcional y de calidad.

MÉTODO

La investigación se llevó a cabo en cuatro fases principales: identificación de las inteligencias artificiales, generación de código, evaluación de código y análisis de resultados.

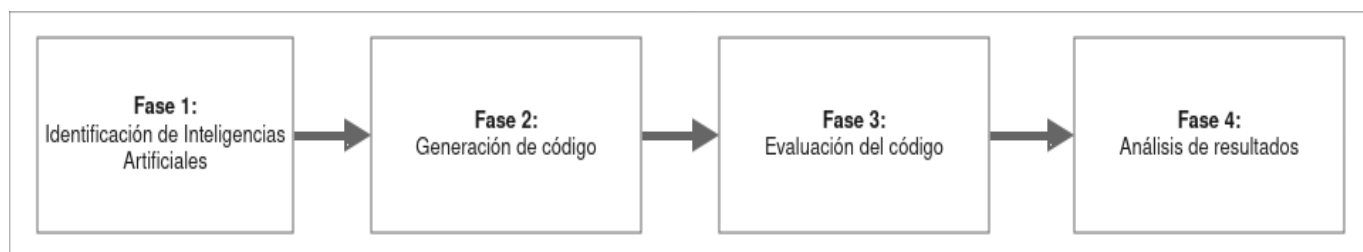


Figura 1. Fases del proceso para la construcción del SMS

Fase 1: identificación de las Inteligencias Artificiales

El principal objetivo de esta fase es definir las herramientas basadas en inteligencia artificial que serán utilizadas para realizar la generación de código, por ello se realizó un mapeo sistemático de la literatura para identificar las inteligencias artificiales capaces de generar código (Macchi & Solari, 2012; Carrizo & Moller, 2018;

Kitchenham et al., 2010). Se llevó a cabo una búsqueda en varias bases de datos que incluyen: ACM, Engineering Source, Academic Search Ultimate, IEEE Xplore y Scopus, lo que resultó en 621 artículos académicos y recursos en línea relevantes. Después de aplicar criterios de inclusión basados en artículos y proceedings en inglés, pertenecientes a las áreas de ingeniería y ciencias de la computación, publicados entre 2020 y 2024, el total de recursos se redujo a 113. Luego, se realizó un proceso de tamizaje que resultó en 44 artículos de interés, lo que permitió identificar las 11 inteligencias artificiales para generación de código: CodeGPT, Replit, Textsynth, Amazon CodeWhisperer, Bing, ChatGPT, Claude, Codeium, Gemini, GitHub Copilot y Tabnine.

Fase 2: generación de código

El objetivo de esta fase es que las herramientas seleccionadas anteriormente fueran puestas a prueba, para lo que se diseñaron diez ejercicios de programación con diferentes niveles de dificultad, abarcando una amplia gama de tareas y conceptos. Estos ejercicios fueron elaborados por el docente del seminario: Asistentes para generación de código. Posteriormente, se solicitó a cada una de las 11 herramientas identificadas que generan código para resolver estos ejercicios en diferentes lenguajes de programación, como: PHP, Python, Java, JavaScript, TypeScript, C#, Kotlin, Go y Ruby.

Los ejercicios diseñados para la evaluación incluyen: 1. ordenar un conjunto de elementos, 2. buscar un elemento dentro de un conjunto, 3. contar las ocurrencias de un elemento específico, 4. registrar usuarios, 5. implementar una API REST para la gestión de usuarios (CRUD), 6. Desarrollar una interfaz gráfica para este CRUD de gestión de usuarios, 7. realizar pruebas unitarias para la API REST, 8. realizar pruebas para la interfaz gráfica de gestión de usuarios, 9. implementar una API REST para la gestión de usuarios con validación de token, y 10. crear una interfaz gráfica para este CRUD protegido.

Fase 3: evaluación de código

Esta fase tiene como objetivo principal determinar la calidad del código generado por cada una de las herramientas en los diferentes lenguajes, teniendo en cuenta la efectividad y la fiabilidad de cada una de las herramientas en términos de adherencia a buenas prácticas de programación y optimización del rendimiento del código. Para llevar a cabo lo anteriormente mencionado se realizó una revisión por parte de los estudiantes haciendo uso de un analizador estático de código (Jenkins y SonarQube), lo cual permitía mediante distintas pruebas verificar la calidad del código teniendo en cuenta aspectos de seguridad, fiabilidad y mantenibilidad del código, para así determinar posteriormente cual inteligencia artificial cuenta con mayor calidad a la hora de generar código.

Fase 4: análisis de Resultados

El objetivo principal de esta fase es contrastar los resultados obtenidos, es decir; cuales son las herramientas más adecuadas para generar código de calidad y funcional. Se generó un análisis teniendo en cuenta toda la información obtenida en la fase: evaluación de código.

RESULTADOS

En esta sección se detallan los resultados obtenidos en las distintas etapas del desarrollo del proyecto. Se diseñaron cadenas de búsqueda específicas para cada una de las bases de datos seleccionadas, utilizando los términos previamente definidos. Este enfoque aseguró una cobertura relevante del ámbito de estudio.

Cadenas de búsqueda

Para optimizar la recolección de documentos relevantes relacionados con la generación automática de código dentro del campo de la inteligencia artificial, se desarrollaron cadenas de búsqueda específicas para cada base de datos, ajustándose conforme a los términos previamente establecidos (tabla 1).

Tabla 1. Cadenas de búsqueda	
Base de datos	Cadena de búsqueda
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 creation" 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"))

Tras la ejecución de las consultas se obtuvieron 621 estudios de los cuales finalmente se seleccionaron 44 (tabla 2).

Tabla 2. Resultado de ejecución: búsqueda en bases de datos

	Academic Search Ultimate	ACM	Engineering source	IEEE Xplore	Scopus	Total
Sin criterios de exclusión	19	41	11	68	482	621
Con criterios de exclusión	12	29	7	30	35	113
Tamizaje (Screening)	7	12	0	15	10	44
Participación	15,91 %	27,27 %	0 %	34,09 %	22,73 %	100 %

A continuación, se mostrarán los resultados obtenidos de las pruebas más relevantes realizadas por las diferentes herramientas de IA generadoras de código, en esta gráfica se observa las herramientas implicadas en las pruebas y sus resultados al pasarlos por la herramienta SonarQube que revisa el código fuente de manera estática.

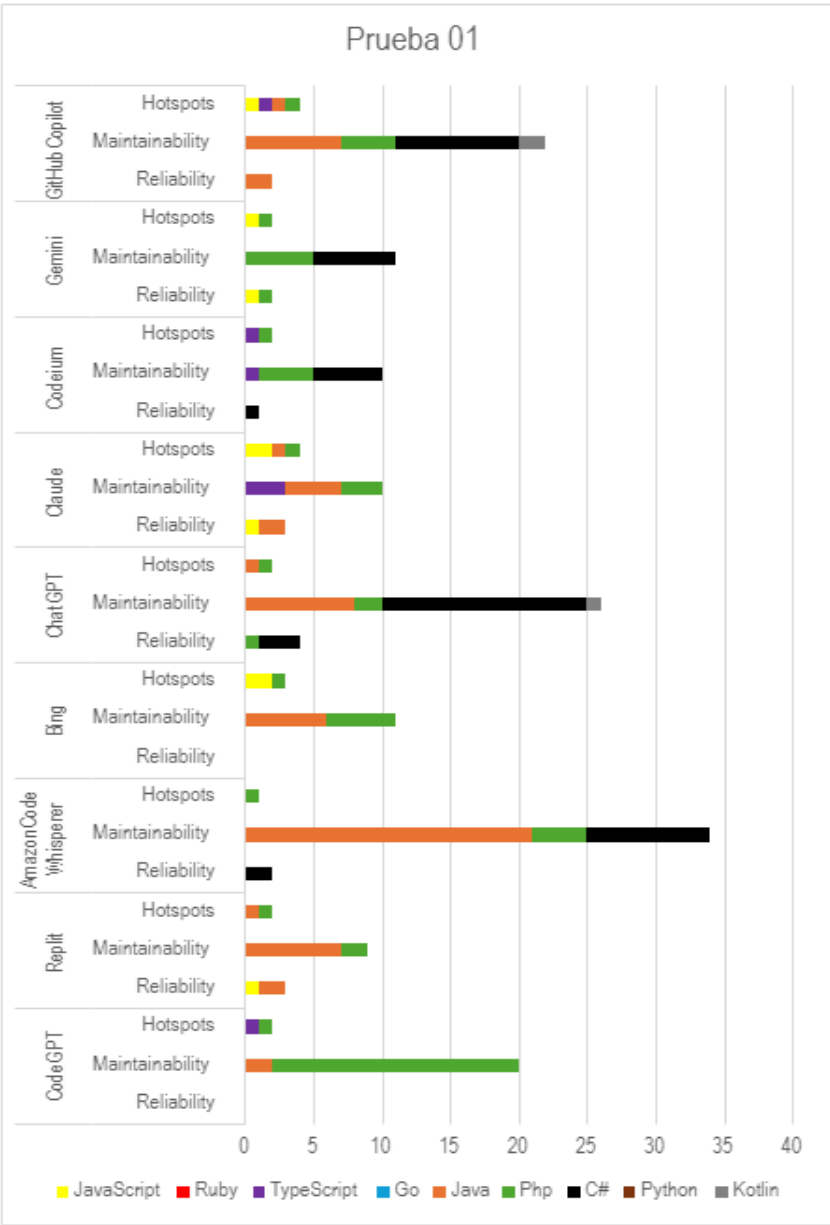


Figura 1. Prueba 01

Esta prueba 01 consiste en utilizar las diferentes inteligencias artificiales para generar el código para capturar los datos de una persona, hacer la conexión a la base de datos, crear la tabla si no existe y guardar los datos de la persona en la base de datos.

En esta evaluación, es importante destacar que todas las herramientas analizadas presentaron un rendimiento óptimo en lenguajes de programación como Python, Go y Ruby. Asimismo, se observó que en lenguajes como Java, PHP y C#, la herramienta de análisis de código ofreció un mayor número de recomendaciones.

Por otro lado, cabe destacar que Codeium presentó la menor cantidad de recomendaciones, sobresaliendo entre las demás herramientas, aunque para la prueba que es relativamente sencilla son bastantes recomendaciones. En contraste, Amazon Code Whisperer fue la herramienta que generó el mayor número de recomendaciones según el análisis realizado.

En la prueba 02 consiste en utilizar las diferentes inteligencias artificiales para generar el código necesario para completar las operaciones CRUD para gestionar usuarios.

En esta evaluación, es importante destacar que todas las herramientas analizadas presentaron un rendimiento óptimo en lenguajes de programación como Ruby y TypeScript. Asimismo, se observó que en lenguajes como Go, PHP y C#, la herramienta de análisis de código ofreció un mayor número de recomendaciones.

Por otro lado, cabe destacar que Bing presentó la menor cantidad de recomendaciones, sobresaliendo entre las demás herramientas. En contraste, Codeium fue la herramienta que generó el mayor número de recomendaciones según el análisis realizado.

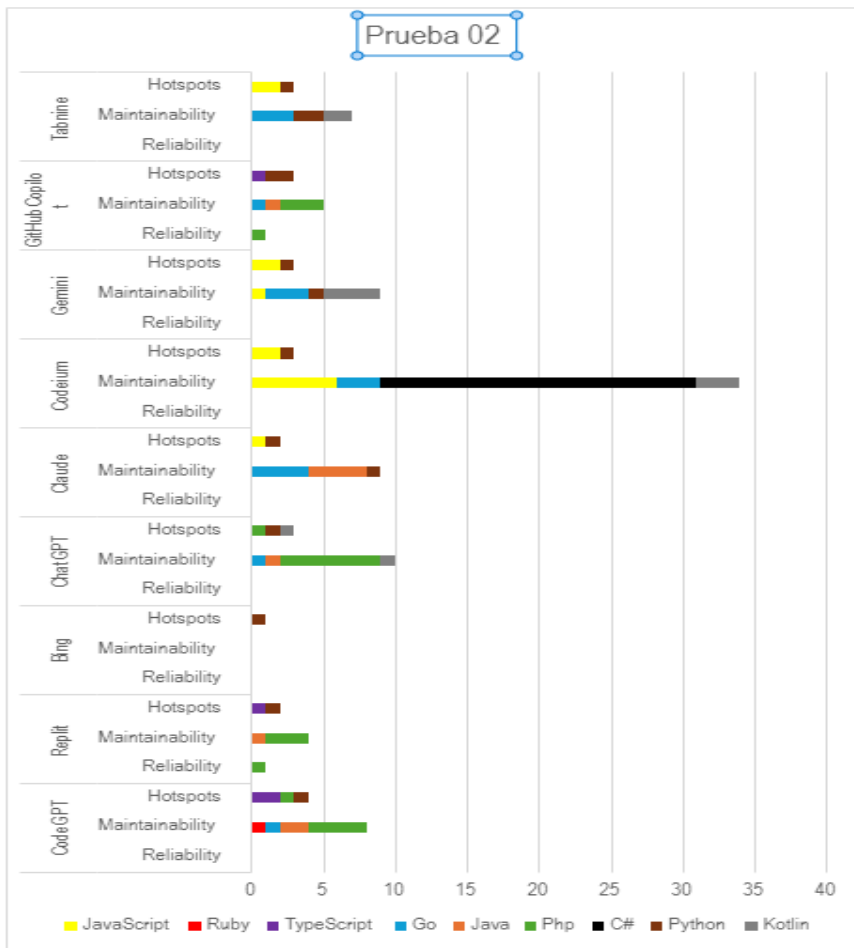


Figura 2. Prueba 02

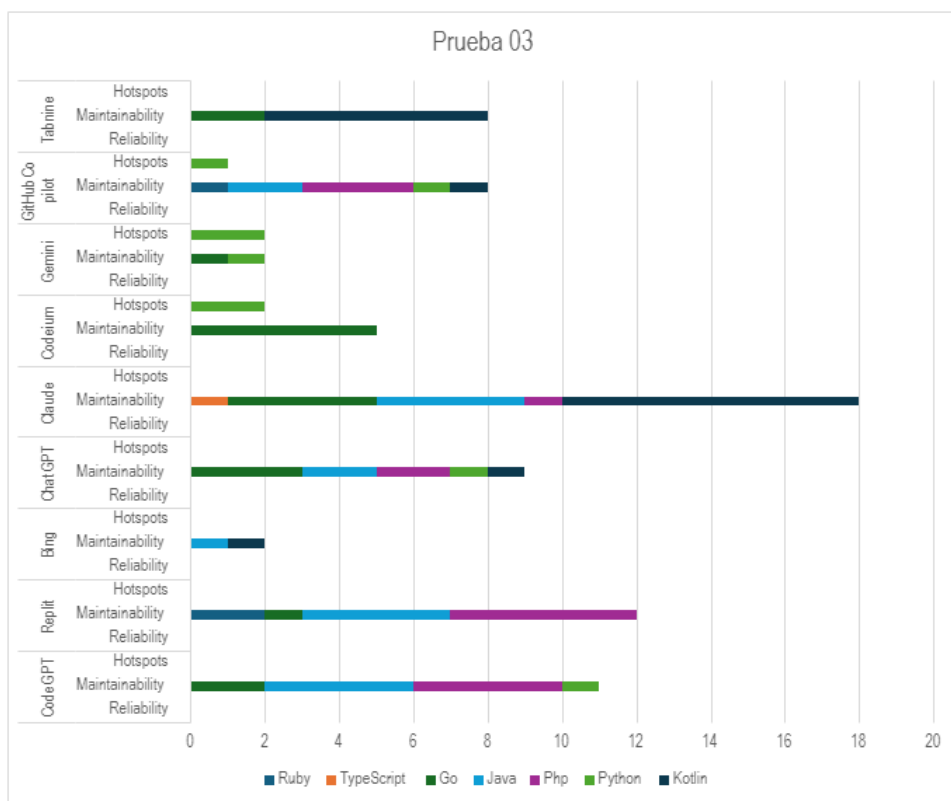


Figura 3. Prueba 03

Esta prueba 03 consiste en utilizar las diferentes inteligencias artificiales para generar el código adecuado para testear las funcionalidades del CRUD generado en pruebas pasadas.

En esta evaluación, es importante destacar que todas las herramientas analizadas presentaron un rendimiento óptimo en lenguajes de programación como Ruby y TypeScript. Asimismo, se observó que en lenguajes como Go, Java, Kotlin y Php la herramienta de análisis de código ofreció un mayor número de recomendaciones.

Por otro lado, cabe destacar que Bing y Gemini presentaron la menor cantidad de recomendaciones, sobresaliendo entre las demás herramientas. En contraste, Claude fue la herramienta que generó el mayor número de recomendaciones según el análisis realizado.

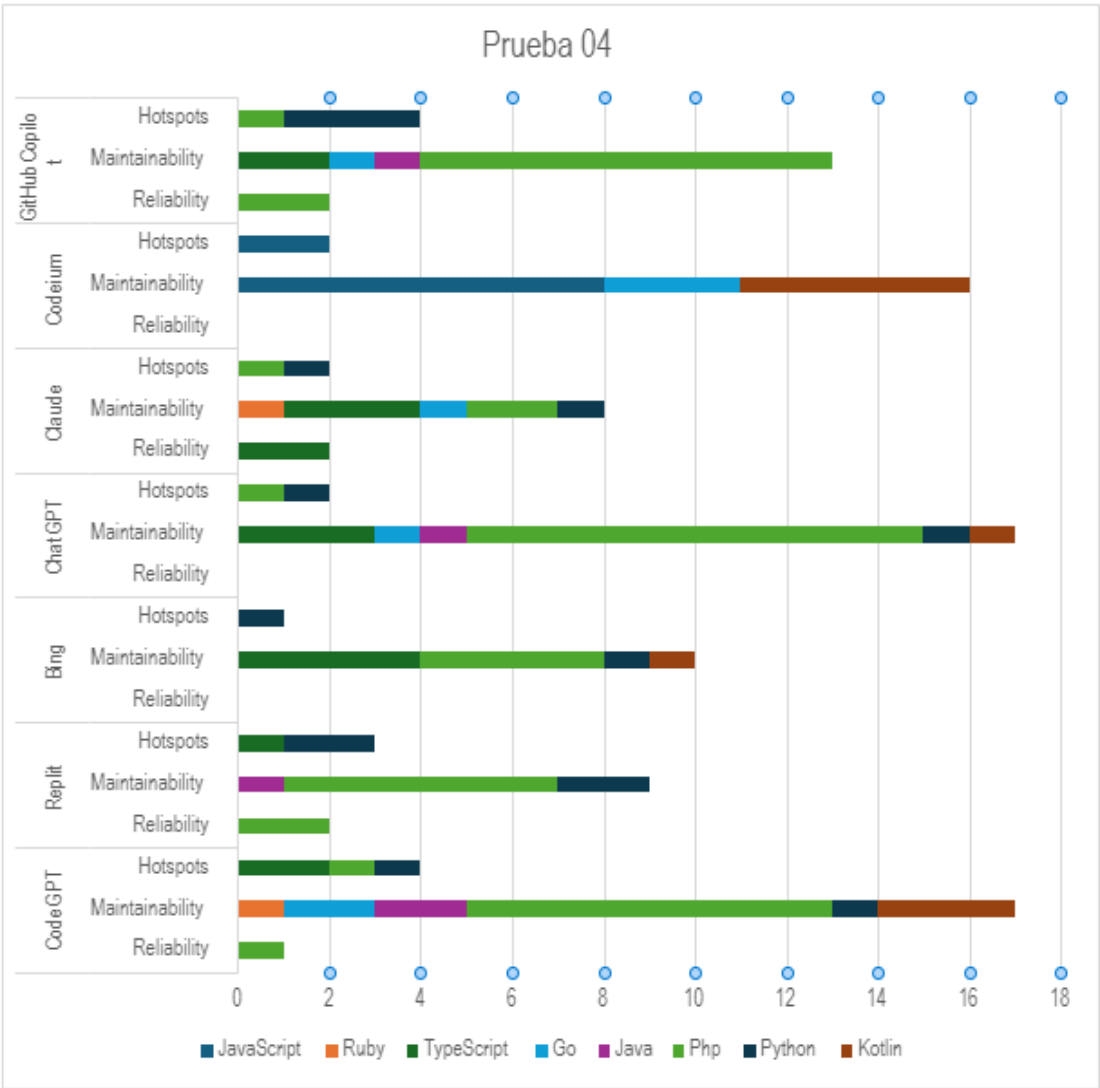


Figura 4. Prueba 04

Tabla 3. Reportes de calidad del código generado para cada herramienta				
	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

Esta prueba 04 consiste en utilizar las diferentes inteligencias artificiales para generar el código adecuado para realizar funcionalidades como inicio de sesión y protección de rutas.

En esta evaluación, es importante destacar que todas las herramientas analizadas presentaron un rendimiento óptimo en lenguajes de programación como Ruby y Java. Asimismo, se observó que en el lenguaje PHP, la herramienta de análisis de código ofreció un mayor número de recomendaciones.

Por otro lado, cabe destacar que Bing presentó la menor cantidad de recomendaciones, sobresaliendo entre las demás herramientas. En contraste, CodeGPT fue la herramienta que generó el mayor número de recomendaciones según el análisis realizado.

Finalmente, se presenta el recuento total de reportes generados para cada herramienta en las diversas pruebas realizadas. Este recuento incluye únicamente a las herramientas que participaron en todas las pruebas, excluyendo aquellas como Textsynth, Amazon CodeWhisperer, Gemini y Tabnine, las cuales presentaron dificultades en la generación de código funcional (tabla 3).

CONCLUSIONES

Este trabajo presenta los resultados de una investigación de tipo SMS centrada en identificar las mejores herramientas IA para generar código funcional y de calidad. Se consideraron estudios en español e inglés pertenecientes a las áreas de ingeniería y ciencias de la computación, publicados en el período de 2020 y 2024. A través de este estudio, se evidencia que las diversas inteligencias artificiales evaluadas aún no presentan un comportamiento estable en cuanto a la calidad y funcionalidad del código que generan, ya que los resultados del análisis no fueron constantes para ninguna de las herramientas en las diez pruebas realizadas. Se observó que la mayoría de errores reportados estaban relacionados con la calidad del código, en particular, características que afectan su mantenibilidad.

Según los resultados obtenidos con este trabajo, se destaca que la herramienta con mejor desempeño en la generación de código funcional y de calidad fue Replit, con un total de 106 reportes encontrados. Este resultado se da a partir del contraste entre la cantidad de pruebas en las que fue utilizada cada IA y el total de errores que fueron reportados por la herramienta de análisis estático de código (SonarQube). Asimismo, se observó que la herramienta de GitHub Copilot presentó el desempeño más bajo en la evaluación, teniendo un resultado de 210 reportes encontrados. A pesar de ser una de las que menos problemas tuvo para generar código funcional, este presentaba fallos de calidad que se manifestaron con los resultados del análisis.

Durante la fase 3 del estudio, se identificaron los siguientes incidentes: las herramientas Textsynth y Amazon CodeWhisperer fueron descartadas después de la tercera y cuarta prueba, respectivamente, debido a inconvenientes con el plan de uso y limitaciones en la interacción con ellas. Aunque Gemini y Tabnine mostraron un comportamiento adecuado durante las fases de generación y evaluación, enfrentaron dificultades al generar código funcional para una de las pruebas, lo que impidió su integración en los resultados obtenidos con las demás herramientas.

REFERENCIAS BIBLIOGRÁFICAS

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. Kozirolek, 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>

FINANCIACIÓN

No existe financiación para el presente trabajo.

CONFLICTO DE INTERES

Los autores declaran que no existe conflicto de interés.

CONTRIBUCIÓN DE AUTORÍA

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

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

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

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

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

Redacción-revisión y edición: Michael Alexander Florez Muñoz, Juan Camilo Jaramillo De La Torre, Stefany Pareja López, Stiven Herrera, Christian Andrés Candela Uribe.