Research Club Scientia Vol. 4, N°3
Frecuencia: Julio - Septiembre
ISSN: 3103 - 1315
DOI: https://doi.org/10.65415/xfce7x48

Impacto de la inteligencia artificial generativa en la calidad del código fuente: estudio cuasiexperimental

Impact of Generative Artificial Intelligence on Source Code Quality: A Quasi-Experimental Study

Impatto dell'intelligenza artificiale generativa sulla qualità del codice sorgente: studio quasi-sperimentale

Rodrigo José Pazmiño Pérez[1]
E-mail: rjpazmino@uagraria.edu.ec
ORCID: https://orcid.org/0009-0000-9507-6865

Norma Valencia Castillo
E-mail: nvalencia@uagraria.edu.ec
ORCID: https://orcid.org/0009-0003-6675-1069

Romulo Steev Vélez Martínez
Email: rvelezm@unemi.edu.ec
ORCID: https://orcid.org/0009-0009-6928-0016

Correspondencia: rjpazmino@uagraria.edu.ec

Artículo de Investigación

***Recibido:** 1 de junio 2026 * **Aceptado:** 14 de julio 2026 * **Publicado:** 4 de agosto 2026

I. Universidad Agraria del Ecuador.
II. Universidad Agraria del Ecuador.
III. Universidad Estatal de Milagro.

Resumen

La inteligencia artificial generativa se incorporó con rapidez a los entornos de programación, pero sus efectos sobre la calidad del código no son uniformes. Este estudio evalúa, mediante un diseño cuasiexperimental, el cambio asociado al uso de un asistente generativo en corrección funcional, mantenibilidad, complejidad, legibilidad, seguridad y productividad. Se construyeron 128 registros sintéticos —64 de control y 64 experimentales— con mediciones pretest y postest. El desenlace principal fue el porcentaje de pruebas unitarias superadas; los modelos ANCOVA emplearon errores robustos HC3, ajuste de Holm e intervalos bootstrap para la g de Hedges. En la simulación, la asistencia con IA produjo una diferencia ajustada de 5,11 puntos porcentuales en pruebas superadas (IC 95 %: 3,22–7,01; p ajustada < 0,001; g = 0,94) y redujo el tiempo en 12,77 minutos (IC 95 %: −15,93 a −9,61). También mejoró complejidad, legibilidad, documentación y Pylint. No se observaron beneficios concluyentes en vulnerabilidades ni duplicación. Los resultados delimitan un escenario reproducible y no constituyen evidencia obtenida de participantes reales; sirven para formular y someter a prueba un protocolo empírico posterior.

Palabras clave: inteligencia artificial generativa; calidad del software; código fuente; ingeniería de software; simulación.

Abstract

Generative artificial intelligence has rapidly entered programming environments, yet its effects on source-code quality remain uneven. This study evaluates, through a quasi-experimental design using entirely simulated data, the change associated with a generative coding assistant in functional correctness, maintainability, complexity, readability, security, and productivity. The dataset contains 128 synthetic records—64 control and 64 experimental—with pretest and posttest measurements. The primary outcome was the percentage of unit tests passed. ANCOVA models used HC3 robust errors, Holm adjustment, and bootstrap confidence intervals for Hedges' g. In the simulation, AI assistance yielded an adjusted increase of 5.11 percentage points in tests passed (95% CI: 3.22–7.01; adjusted p < .001; g = 0.94) and a 12.77-minute reduction in completion time (95% CI: −15.93 to −9.61). Complexity, readability, documentation, and Pylint scores also improved. No conclusive benefit appeared for vulnerabilities or duplication. These outcomes describe a reproducible scenario rather than observations from real participants and should be used to design a subsequent empirical protocol.

Keywords: generative artificial intelligence; software quality; source code; software engineering; simulation.

Introducción

Los modelos generativos de código han desplazado la asistencia automática desde la finalización de tokens hacia una interacción capaz de interpretar requisitos, proponer funciones completas, explicar fragmentos y sugerir pruebas. Esta transición modifica el proceso de construcción de software porque introduce una fuente externa de decisiones sobre estructura, dependencias y estilo. La revisión sistemática de Hou et al. (2024) muestra que los grandes modelos de lenguaje ya intervienen en múltiples tareas de ingeniería de software, mientras que Husein et al. (2025) documentan una expansión paralela de las investigaciones sobre completado de código. En consecuencia, la pregunta relevante dejó de ser si estas herramientas producen texto ejecutable y pasó a centrarse en las condiciones bajo las cuales ayudan a producir software sostenible.

La calidad del código fuente es un constructo multidimensional. La corrección observable mediante pruebas es necesaria, pero no resume propiedades como mantenibilidad, legibilidad, modularidad, seguridad o duplicación. ISO/IEC 25010:2023 distingue características del producto que deben valorarse de forma conjunta, pues una solución funcional puede acumular deuda técnica o exposición a fallos. En el plano operativo, la complejidad ciclomática de McCabe (1976) y los indicadores de mantenibilidad propuestos por Coleman et al. (1994) ofrecen antecedentes consolidados para traducir parte de esas propiedades a métricas. La evaluación de asistentes generativos exige, por tanto, evitar que una sola tasa de acierto se interprete como sinónimo de calidad integral.

Los primeros estudios empíricos reflejan esa heterogeneidad. Nguyen y Nadi (2022) observaron variación en la corrección de las sugerencias de Copilot según el dominio y el lenguaje, y Yetiştiren et al. (2022) compararon validez, corrección y eficiencia en problemas algorítmicos. Moradi Dakhel et al. (2023) encontraron que el asistente puede producir soluciones competitivas, aunque su comportamiento no es estable en todas las categorías. Wermelinger (2023) llegó a una conclusión semejante al estudiar problemas introductorios: el rendimiento depende del enunciado, la interacción y la revisión. Estas diferencias impiden extrapolar una ganancia promedio a cualquier equipo o repositorio.

La productividad constituye otro foco de interés. Ebert y Louridas (2023) describen oportunidades para acelerar tareas repetitivas, documentación y exploración de alternativas, pero advierten que el valor profesional depende de la supervisión. Banh et al. (2025) sitúan la transformación en un plano organizacional más amplio, donde la automatización altera coordinación, aprendizaje y control de calidad. La reducción del tiempo puede ser valiosa solo si no traslada el esfuerzo hacia depuración, revisión de dependencias o corrección de defectos tardíos. Por esa razón, el presente trabajo analiza simultáneamente duración y propiedades internas del código.

También interviene la manera en que el programador formula, acepta y modifica una sugerencia. Denny et al. (2023) demostraron que la construcción del prompt influye en la resolución de ejercicios de programación inicial. Barke et al. (2023) describieron dos patrones de interacción

—aceleración y exploración— que producen relaciones distintas con el código sugerido. En lugar de considerar al modelo como un generador autónomo, conviene entender el resultado como una producción socio-técnica: la herramienta propone, el usuario selecciona y el entorno de verificación revela o deja pasar defectos.

Metodología

Diseño y alcance de la simulación

Se desarrolló un estudio cuasiexperimental computacional con dos grupos no equivalentes y mediciones pretest–postest. La unidad de análisis fue un registro sintético que representa, exclusivamente con fines metodológicos, a una persona universitaria que resuelve tareas de programación. Los registros no corresponden a individuos reales, no se recopilaron datos personales ni se ejecutó una intervención en ninguna institución.

La base contiene 128 unidades distribuidas en partes iguales: 64 en control y 64 en condición experimental. El escenario representa seis semanas de trabajo con cuatro tareas equivalentes de Python. En control se supone acceso al editor, documentación técnica y pruebas locales; en la condición experimental se añade un asistente generativo capaz de proponer código, explicaciones y casos de prueba. El usuario hipotético conserva la responsabilidad de aceptar, modificar o rechazar la sugerencia. Se simuló un cumplimiento del protocolo y un número de interacciones con la IA únicamente para describir exposición, no para atribuir conductas reales.

La generación utilizó una semilla fija (20260720) y distribuciones acotadas en rangos plausibles. Edad, semestre, experiencia y competencia inicial formaron la estructura basal. Las métricas pretest se construyeron con correlaciones moderadas respecto de competencia y experiencia, más componentes aleatorios específicos de cada indicador. Las mediciones postest combinaron el valor basal, una mejora general atribuible a práctica, un componente diferencial de grupo y error residual. La magnitud diferencial se programó para ser mayor en corrección, documentación y tiempo; más moderada en mantenibilidad, complejidad y legibilidad; y mínima o desfavorable en vulnerabilidades y duplicación.

Los límites físicos o conceptuales se aplicaron después de cada transformación. Las escalas de 0 a 100 y de 0 a 10 se truncaron en su intervalo válido; conteos, tiempos y densidades se mantuvieron no negativos. Los cambios se calcularon como postest menos pretest, de modo que un valor positivo indica aumento. Para complejidad, olores, vulnerabilidades, tiempo, duplicación y errores, una reducción representa mejor calidad; esta inversión se consideró al construir la figura de tamaños de efecto. El archivo complementario conserva tanto las medidas como los cambios para que el análisis pueda repetirse sin regenerar la muestra.

El estudio buscó producir grupos semejantes al inicio sin imponer igualdad exacta. Esto permite probar el ajuste por línea base propio de un diseño cuasiexperimental. Por tratarse de una simulación metodológica, no se realizó un cálculo de potencia para inferir efectos en una población real.

El tamaño n = 128 se eligió para proporcionar estabilidad numérica a modelos con cinco covariables y para mantener una base revisable fila por fila. Cualquier investigación posterior deberá justificar su tamaño muestral con un efecto mínimo relevante, pérdidas previstas y restricciones del contexto real.

Metodología

Variables y operacionalización

El desenlace principal fue el porcentaje de pruebas unitarias superadas, interpretado como aproximación a corrección funcional. En un estudio empírico, la prueba debería construirse antes de observar las soluciones, cubrir casos normales y límites, y ejecutarse en un entorno reproducible. La simulación representa ese resultado en una escala de 0 a 100. Se incluyeron además los errores funcionales restantes para distinguir una tasa proporcional de un conteo de fallos.

La mantenibilidad se expresó con un índice de 0 a 100 inspirado en la combinación de volumen, complejidad y tamaño propuesta en la literatura de métricas (Coleman et al., 1994). La complejidad ciclomática se definió como número de caminos independientes por función, siguiendo el principio de McCabe (1976). Los olores de código y las vulnerabilidades se normalizaron por mil líneas de código (KLOC), lo que reduce el sesgo derivado del tamaño. La duplicación se registró como porcentaje. Estas variables representan salidas que podrían obtenerse mediante análisis estático, pero en el presente trabajo no proceden de una ejecución real de SonarQube, Bandit ni herramientas equivalentes.

Legibilidad, modularidad y documentación se expresaron en escalas de 0 a 100. La legibilidad postest incluyó dos evaluaciones sintéticas, cuyo promedio se usó como puntuación final. La modularidad representa separación de responsabilidades, cohesión y acoplamiento; la documentación integra docstrings, nombres y comentarios pertinentes. Pylint se simuló en su escala de 0 a 10. En una aplicación real, estas rúbricas deberán acompañarse de descriptores observables y evaluadores cegados a la condición para minimizar expectativas.

El tiempo de resolución se midió en minutos y representa la duración desde la lectura de la tarea hasta la entrega de una solución que ejecuta el conjunto de pruebas. No incluye aprendizaje previo ni revisión diferida. La productividad no se definió como tiempo aislado, sino como la relación entre reducción temporal y preservación de la corrección. Esta distinción responde a los costos de lectura, evaluación y reparación señalados en estudios de interacción con asistentes (Mozannar et al., 2024; Vaithilingam et al., 2022).

La competencia inicial de 0 a 100 y los meses de experiencia se incorporaron como covariables. El nivel inicial —bajo, medio o alto— se derivó de la competencia para facilitar descripciones y análisis exploratorios. La exposición a la IA se representó mediante interacciones y cumplimiento solo en el grupo experimental.

El control de calidad comprobó unicidad del identificador, ausencia de valores perdidos y coherencia algebraica de todos los cambios. Se conservaron 128 filas, sin duplicados ni celdas vacías. Para la legibilidad se calculó el coeficiente de correlación intraclase de acuerdo absoluto entre las dos valoraciones. **Metodología**

Plan de análisis

Cada desenlace postest se estimó mediante un modelo lineal de covarianza que incluyó grupo, valor pretest, competencia inicial y meses de experiencia. Los errores estándar se calcularon con el estimador robusto HC3 para disminuir la sensibilidad a heterocedasticidad. El coeficiente de grupo expresa la diferencia ajustada entre condición experimental y control en la unidad original. Se presentaron intervalos del 95 % y valores p bilaterales. El ajuste de Holm se aplicó conjuntamente a los 12 desenlaces para limitar el error por comparaciones múltiples.

Como medida comparable se calculó la g de Hedges sobre las puntuaciones de cambio, con corrección por tamaño muestral. Los intervalos del 95 % se obtuvieron mediante 4.000 remuestreos bootstrap estratificados por grupo. En la visualización, los signos de las métricas donde un valor menor es favorable se invirtieron, de modo que valores positivos siempre indiquen beneficio relativo de la IA. La magnitud se interpretó como información continua y no mediante umbrales rígidos, siguiendo la recomendación de reportar tamaños de efecto con su incertidumbre (Lakens, 2013).

Se especificó un análisis exploratorio de moderación para el desenlace principal mediante la interacción grupo × competencia inicial. También se estimó, dentro del grupo experimental, la correlación entre minutos ahorrados y cambio en pruebas superadas. Estos análisis no fueron usados para redefinir la hipótesis principal y se informan sin interpretación causal. Todos los cálculos se reprodujeron desde la hoja «Datos complementarios»; las figuras derivan de las mismas columnas y de la semilla declarada.

Tabla 1. Comparabilidad basal de los grupos simulados

Variable	Control Media (DE)	Experimental Media (DE)	p	g
Edad (años)	20.61 (1.58)	20.97 (1.50)	0.189	0.23
Experiencia de programación (meses)	22.12 (9.09)	21.62 (8.91)	0.754	-0.06
Competencia inicial (0–100)	64.54 (9.83)	65.43 (10.13)	0.613	0.09
Pruebas unitarias pre (%)	60.72 (8.74)	61.30 (8.09)	0.697	0.07
Mantenibilidad pre (0–100)	61.63 (6.99)	63.05 (6.81)	0.247	0.20
Complejidad ciclomática pre	9.94 (1.49)	10.31 (1.38)	0.141	0.26
Vulnerabilidades pre/KLOC	2.05 (0.51)	2.10 (0.67)	0.615	0.09

Nota. p de Welch para diferencias de medias; g positiva indica mayor valor en el grupo experimental.

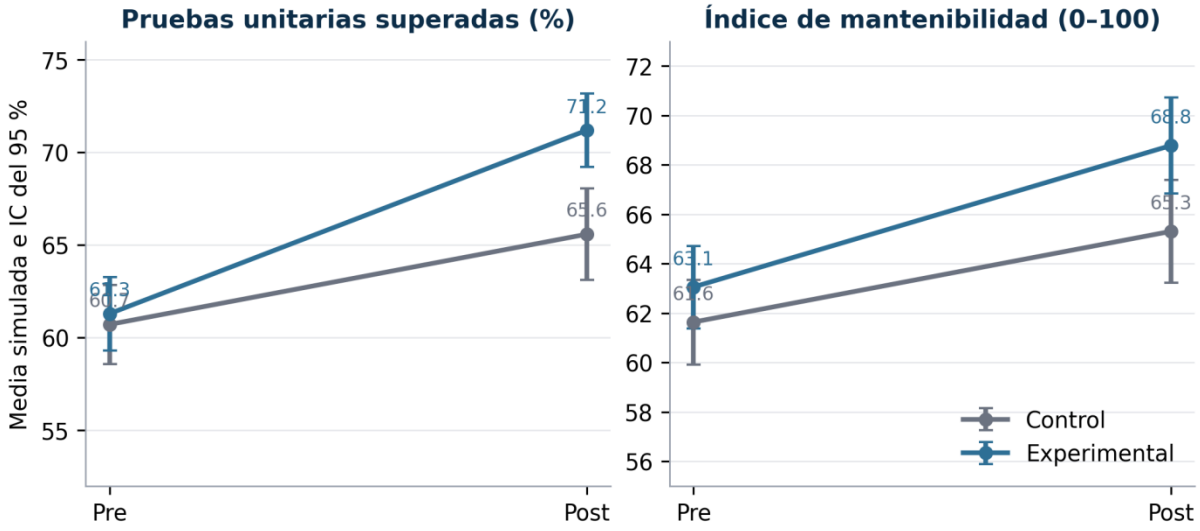
Resultados

Desenlace principal

Los 128 registros sintéticos se distribuyeron sin pérdidas entre control (n = 64) y experimental (n = 64). En el pretest, el control alcanzó 60,72 % de pruebas unitarias y el experimental 61,30 %. En el postest, las medias fueron 65,59 % y 71,20 %, respectivamente. La mejora bruta fue de 4,87 puntos en control y 9,90 en experimental, lo que produce una diferencia de cambios de 5,03 puntos porcentuales.

Después de ajustar por resultado basal, competencia y experiencia, la diferencia de grupo fue 5,11 puntos porcentuales (IC 95 %: 3,22–7,01). El valor p robusto fue < 0,001 y permaneció por debajo de 0,001 tras el ajuste de Holm. La g de Hedges del cambio fue 0,94 (IC bootstrap del 95 %: 0,59–1,30). En el escenario simulado, por tanto, la asistencia generativa se asocia con una mejora funcional clara y compatible con una magnitud moderada-alta.

Figura 1. Evolución pretest–postest de corrección funcional y mantenibilidad



Nota. Medias e intervalos de confianza del 95 % calculados sobre los datos complementarios.

La mantenibilidad siguió un patrón ascendente en ambos grupos: 61,63 a 65,30 en control y 63,05 a 68,78 en experimental. La diferencia ajustada fue 2,04 puntos (IC 95 %: 0,51–3,57). Aunque el valor p sin corrección fue 0,009, el ajuste conjunto produjo p = 0,056. Este contraste ilustra por qué la evidencia secundaria no debe resumirse mediante el resultado nominal de una sola prueba.

Resultados

Desenlaces secundarios

Los resultados ajustados muestran mejoras conservadas después de controlar la multiplicidad en complejidad, legibilidad, documentación, tiempo y Pylint. Mantenibilidad, modularidad y errores funcionales conservaron una dirección favorable, pero sus valores p de Holm quedaron ligeramente por encima de 0,05. Los olores de código disminuyeron en mayor medida con IA, aunque la evidencia corregida fue insuficiente. Vulnerabilidades y duplicación se desplazaron en dirección desfavorable para el grupo experimental y sus intervalos incluyeron el valor nulo.

Tabla 2. Efectos ajustados en los desenlaces postest

Desenlace	β ajustada	IC 95 %	p Holm	g del cambio
Pruebas unitarias (%)	5,11	[3,22; 7,01]	<0,001	0,94
Mantenibilidad	2,04	[0,51; 3,57]	0,056	0,48
Complejidad	-0,83	[-1,22; -0,43]	<0,001	-0,81
Olores/KLOC	-0,57	[-1,11; -0,02]	0,124	-0,37
Vulnerabilidades/KLOC	0,18	[0,00; 0,35]	0,124	0,32
Legibilidad	2,88	[1,13; 4,63]	0,010	0,57
Modularidad	2,42	[0,47; 4,36]	0,069	0,43
Documentación	4,59	[2,48; 6,71]	<0,001	0,80
Tiempo (min)	-12,77	[-15,93; -9,61]	<0,001	-1,46
Duplicación (%)	0,46	[-0,11; 1,03]	0,124	0,31
Pylint	0,32	[0,13; 0,51]	0,007	0,61
Errores funcionales	-0,56	[-1,00; -0,11]	0,069	-0,43

Nota. β se expresa en la escala original. En complejidad, olores, vulnerabilidades, tiempo, duplicación y errores, un coeficiente negativo es favorable. g mantiene el signo bruto del cambio. Errores HC3 y ajuste de Holm sobre 12 pruebas.

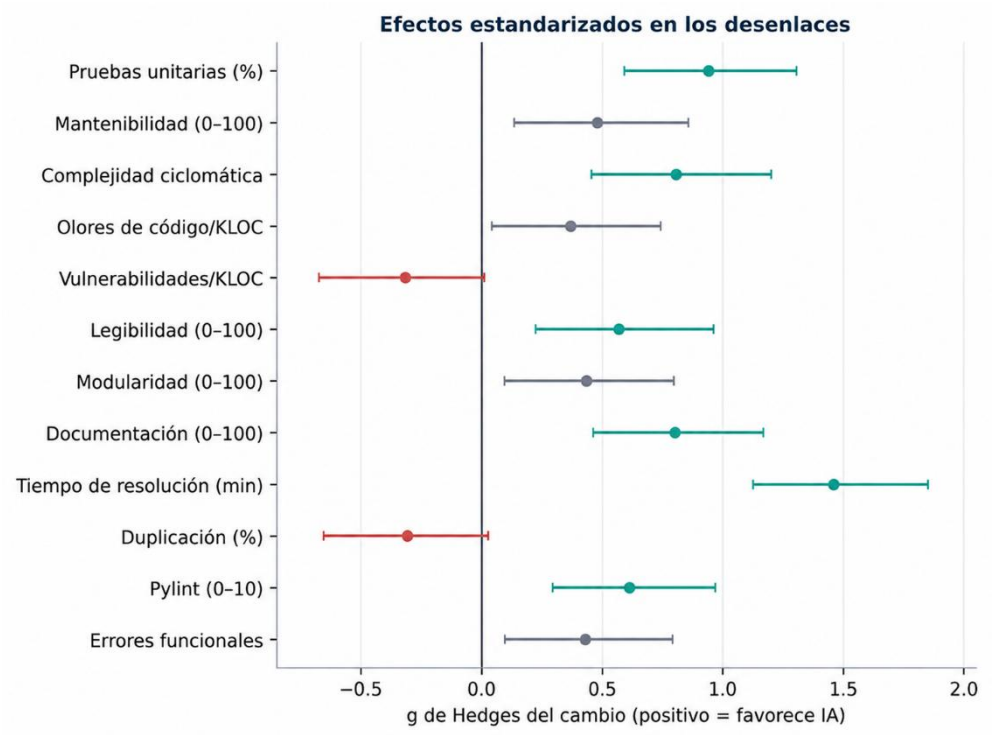
La reducción ajustada de complejidad fue −0,83 (IC 95 %: −1,22 a −0,43; p Holm < 0,001) y el aumento de documentación fue 4,59 puntos (IC 95 %: 2,48–6,71; p Holm < 0,001). El tiempo descendió 12,77 minutos adicionales (IC 95 %: −15,93 a −9,61; p Holm < 0,001). Legibilidad mejoró 2,88 puntos (p Holm = 0,010) y Pylint 0,32 (p Holm = 0,007).

Resultados

Magnitud, fiabilidad y heterogeneidad

Al reorientar las métricas para que un valor positivo favorezca la IA, los mayores efectos correspondieron a tiempo (g = 1,46), pruebas unitarias (0,94), complejidad (0,81) y documentación (0,80). Pylint y legibilidad mostraron magnitudes intermedias. Vulnerabilidades (−0,32) y duplicación (−0,31) se desplazaron en dirección desfavorable, con intervalos que incluyeron cero.

Figura 2. Tamaños de efecto estandarizados de los cambios

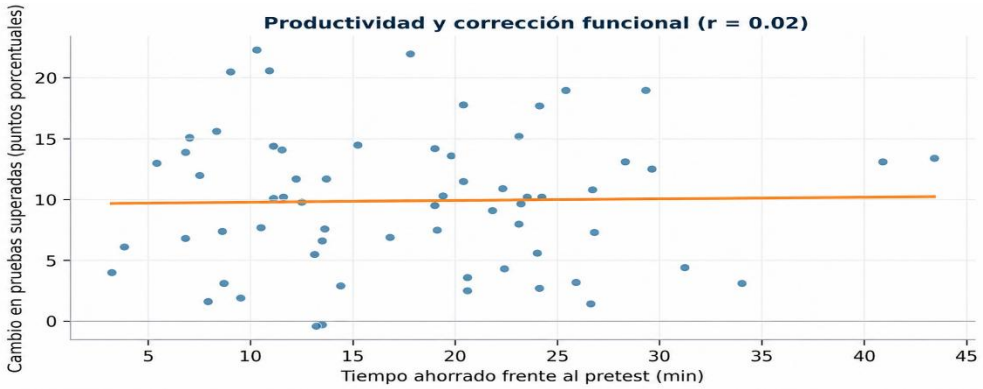


Nota. g de Hedges e IC bootstrap del 95 %. Valores positivos favorecen IA.

Fiabilidad y moderación. Las valoraciones de legibilidad alcanzaron ICC = 0,947. La interacción grupo × competencia fue −0,207 (IC 95 %: −0,376 a −0,038; p = 0,017), lo que recupera la heterogeneidad programada en la simulación. La correlación entre minutos ahorrados y cambio en pruebas fue nula (r = 0,021; p = 0,871): ahorrar más tiempo no implicó superar más pruebas.

Figura 3. Tiempo ahorrado y cambio funcional en el grupo experimental

Nota. Cada punto representa un registro sintético; la línea es un ajuste lineal descriptivo.



Discusión

Corrección funcional y productividad

El escenario simulado reproduce una posibilidad recurrente en la literatura: la IA generativa puede acelerar la resolución y elevar la proporción de pruebas superadas, pero la ganancia no se distribuye por igual entre todas las propiedades del código. La diferencia ajustada de 5,11 puntos en corrección y la reducción de 12,77 minutos son coherentes con una herramienta que completa estructuras rutinarias, propone casos de prueba y reduce el costo de búsqueda. Sin embargo, estos valores fueron generados mediante supuestos explícitos; su función es probar la lógica del protocolo, no cuantificar un rendimiento real de Copilot, ChatGPT u otro producto.

La mejora funcional coincide en dirección con estudios que documentan soluciones correctas en una proporción relevante de tareas, aunque también subrayan variabilidad entre problemas (Nguyen & Nadi, 2022; Yetiştiren et al., 2022). Moradi Dakhel et al. (2023) muestran que la valoración de Copilot depende del tipo de tarea y de la calidad considerada. Wermelinger (2023) observa que incluso problemas simples exigen comprobar el resultado, y Denny et al. (2023) evidencia que la formulación del pedido cambia el desempeño. El efecto no pertenece únicamente al modelo: emerge de la interacción entre requisito, prompt, propuesta, revisión y prueba.

La magnitud temporal es compatible con el potencial de automatizar código repetitivo descrito por Ebert y Louridas (2023). No obstante, la correlación prácticamente nula entre tiempo ahorrado y cambio en pruebas impide tratar ambas variables como intercambiables. Una persona puede aceptar rápidamente una solución incompleta o, por el contrario, invertir tiempo en revisar una alternativa que termina siendo más robusta. Los costos de lectura y decisión modelados por Mozannar et al. (2024) explican por qué el tiempo de tecleo representa solo una parte de la productividad.

Los patrones de aceleración y exploración identificados por Barke et al. (2023) ofrecen una lectura complementaria. En el modo de aceleración, la herramienta materializa una intención ya definida; en el exploratorio, también interviene en la comprensión del problema. El segundo patrón puede beneficiar a personas con menor competencia inicial, pero aumenta la dependencia de explicaciones y la dificultad para reconocer errores. La interacción negativa simulada entre grupo y competencia permite anticipar cómo evaluar esta hipótesis, aunque la confirmación requiere asignación real, medición previa válida y análisis preespecificado.

La literatura de experiencia de usuario refuerza esta prudencia. Vaithilingam et al. (2022) describen fricciones cuando la sugerencia no coincide con el modelo mental del programador; Liang et al. (2024) señalan desafíos de confianza y control; Prather et al. (2024) muestran que los novatos pueden sorprenderse por la aparente comprensión del sistema. Bird et al. (2022) y Ross et al. (2023) destacan el valor del intercambio conversacional, pero esa fluidez puede generar una sensación de corrección que debe someterse a pruebas. El protocolo futuro debería registrar no solo aceptación, sino modificaciones, rechazos y razones de decisión.

Discusión

Calidad interna, legibilidad y seguridad

La reducción de complejidad y el aumento de documentación en la simulación sugieren que un asistente puede aportar estructuras más regulares y comentarios iniciales. Haindl y Weinberger (2024) encontraron que el análisis estático permite detectar diferencias en código de programadores noveles con apoyo de ChatGPT, mientras que Rabbi et al. (2024) muestran la utilidad de revisar sistemáticamente fragmentos Python generados. Aun así, los indicadores estáticos no prueban por sí mismos que el diseño sea apropiado ni que el comportamiento en ejecución sea correcto. Funcionan mejor como señales complementarias dentro de una cadena de verificación.

La legibilidad merece una interpretación separada. Los estudios de Al Madi (2022) y Dantas et al. (2023) indican que la lectura humana puede distinguir propiedades que no aparecen en una tasa de pruebas. En la simulación, la mejora de 2,88 puntos y la alta concordancia entre evaluadores reflejan una rúbrica consistente, pero la consistencia no garantiza que la rúbrica capture comprensión, facilidad de cambio o adecuación al equipo. Una validación empírica debería incluir tareas de lectura y modificación, no solo puntuaciones de apariencia.

La no determinación descrita por Ouyang et al. (2025) introduce otra fuente de variación. Si una misma solicitud produce alternativas distintas, el resultado observado depende del momento, la configuración y la selección del usuario. Oertel et al. (2025) sostienen que revisar varias soluciones puede aumentar la probabilidad de encontrar una alternativa adecuada. Registrar versión del modelo, parámetros, prompt completo y número de intentos será indispensable para

reproducir una investigación real y para evitar que la herramienta se trate como una intervención estable a lo largo del tiempo.

Los resultados de vulnerabilidades y duplicación son deliberadamente cautelosos. Pearce et al. (2022) identificaron contribuciones inseguras; Perry et al. (2023) observaron que la asistencia puede alterar tanto el código como la percepción de seguridad; Asare et al. (2023) compararon la introducción de vulnerabilidades con el desempeño humano. El estudio de repositorios de Fu et al. (2025) muestra que las debilidades también aparecen fuera de tareas controladas. Frente a esa evidencia, una ganancia en pruebas funcionales no autoriza a asumir seguridad, porque las pruebas pueden omitir inyección, validación, autenticación o uso inseguro de bibliotecas.

La mantenibilidad, modularidad y los olores conservaron dirección favorable, pero perdieron significación tras Holm. Esta diferencia entre señal y evidencia concluyente es metodológicamente útil. Si se seleccionaran solo valores p sin corregir, el artículo afirmaría mejoras más amplias que las soportadas por el conjunto de contrastes. La g de Hedges y sus intervalos permiten conservar la información sobre magnitud e incertidumbre. En una muestra real, convendría predefinir una jerarquía de desenlaces y evitar que un gran número de métricas convierta hallazgos accidentales en conclusiones centrales.

Discusión

Implicaciones, amenazas a la validez y trabajo futuro

Para la docencia y los equipos de desarrollo, el patrón sugiere una regla práctica: permitir asistencia generativa debe ir acompañado de pruebas automáticas, análisis estático, revisión humana y criterios de seguridad. El modelo no debería recibir autoridad para integrar código por el solo hecho de producir una respuesta plausible. Una política madura puede exigir trazabilidad del prompt, identificación de fragmentos aceptados, ejecución de la batería de pruebas y revisión de dependencias. Estas medidas convierten a la IA en una fuente de propuestas dentro del proceso, no en un sustituto del aseguramiento de calidad.

La adopción también requiere diferenciar perfiles. La posible mayor ventaja entre unidades de menor competencia puede traducirse en apoyo al aprendizaje, pero también en aceptación acrítica. La supervisión debería pedir que el usuario explique la solución, identifique riesgos y modifique deliberadamente el código. En perfiles expertos, la ganancia podría concentrarse en velocidad y exploración. Esta segmentación debe evaluarse mediante interacciones preespecificadas, evitando conclusiones basadas en subgrupos formados después de observar los datos.

La amenaza principal a la validez externa es total: los registros son sintéticos. Las distribuciones y efectos reflejan decisiones del generador, de modo que no representan una universidad, país, curso, empresa o herramienta concreta. La validez de constructo también es limitada porque varias escalas resumen propiedades complejas y porque no se ejecutaron analizadores reales. La

validez interna no equivale a la de un experimento; el ajuste estadístico recupera relaciones programadas, no elimina confusión observada. Por estas razones, el artículo debe clasificarse como estudio metodológico o de simulación y no como ensayo con estudiantes.

Existen además límites estadísticos. El uso de modelos lineales para conteos y variables acotadas se justifica aquí por estabilidad y claridad, pero un estudio real podría requerir regresión binomial, Poisson o modelos mixtos por tarea. El bootstrap supone que los registros sintéticos son intercambiables dentro de grupo. El ajuste de Holm controla comparaciones, aunque reduce potencia en desenlaces secundarios. La moderación y la correlación productividad–corrección son exploratorias. Ninguna de estas estimaciones debería usarse para calcular beneficios económicos o decisiones institucionales sin datos observados.

El siguiente paso es registrar un protocolo prospectivo. Debe definir tareas equivalentes, versión del asistente, reglas de uso, pruebas ocultas, métricas, cegamiento de evaluadores, criterios de exclusión y plan de análisis. La asignación aleatoria sería preferible; si no es posible, se requieren estrategias de balance y sensibilidad a confusión. También conviene archivar prompts, respuestas y modificaciones con protección de datos. El archivo sintético entregado permite depurar ese protocolo, estimar cargas de captura y ensayar el análisis antes de involucrar personas.

Una segunda extensión debería estudiar mantenimiento diferido. El código puede superar pruebas al finalizar una sesión y volverse costoso de modificar semanas después. Medir comprensión, corrección de defectos, incorporación de requisitos y vulnerabilidades bajo revisión revelaría dimensiones que el postest inmediato no captura. La contribución de la presente simulación radica en hacer visibles esas decisiones antes de la recolección, no en anticipar el resultado que necesariamente obtendrá un estudio futuro.

Conclusiones

En el escenario cuasiexperimental, la asistencia con IA generativa se asoció con una mejora ajustada de la corrección funcional y una reducción considerable del tiempo de resolución. También aparecieron efectos favorables en complejidad, legibilidad, documentación y Pylint. La mantenibilidad, modularidad, los olores de código y los errores funcionales mostraron señales positivas que no conservaron evidencia suficiente después del ajuste por multiplicidad.

La ausencia de beneficio concluyente en vulnerabilidades y duplicación impide resumir la intervención como una mejora general de la calidad. El hallazgo metodológico más relevante es la separación entre velocidad, corrección y seguridad. Un sistema puede ayudar a producir una solución más rápida y funcional sin reducir los riesgos que requieren análisis especializado. Por ello, la adopción responsable debe integrar pruebas, análisis estático, revisión humana y controles de seguridad.

Referencias

Al Madi, N. (2022). How readable is model-generated code? Examining readability and visual inspection of GitHub Copilot. *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 1–5. https://doi.org/10.1145/3551349.3560438

Asare, O., Nagappan, M., & Asokan, N. (2023). Is GitHub's Copilot as bad as humans at introducing vulnerabilities in code? *Empirical Software Engineering, 28*(6), Article 129. https://doi.org/10.1007/s10664-023-10380-1

Banh, L., Holldack, F., & Strobel, G. (2025). Copiloting the future: How generative AI transforms software engineering. *Information and Software Technology, 183*, 107751. https://doi.org/10.1016/j.infsof.2025.107751

Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded Copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages, 7*(OOPSLA1), 85–111. https://doi.org/10.1145/3586030

Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., & Gazit, I. (2022). Taking flight with Copilot. *Queue, 20*(6), 35–57. https://doi.org/10.1145/3582083

Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer, 27*(8), 44–49. https://doi.org/10.1109/2.303623

Dantas, C., Rocha, A., & Maia, M. (2023). Assessing the readability of ChatGPT code snippet recommendations: A comparative study. *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*, 283–292. https://doi.org/10.1145/3613372.3613413

Denny, P., Kumar, V., & Giacaman, N. (2023). Conversing with Copilot: Exploring prompt engineering for solving CS1 problems using natural language. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 1136–1142. https://doi.org/10.1145/3545945.3569823

Ebert, C., & Louridas, P. (2023). Generative AI for software practitioners. *IEEE Software, 40*(4), 30–38. https://doi.org/10.1109/MS.2023.3265877

Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., Yu, J., & Chen, J. (2025). Security weaknesses of Copilot-generated code in GitHub projects: An empirical study. *ACM Transactions on Software Engineering and Methodology, 34*(8), 1–34. https://doi.org/10.1145/3716848

Haindl, P., & Weinberger, G. (2024). Does ChatGPT help novice programmers write better code? Results from static code analysis. *IEEE Access, 12*, 114146–114156. https://doi.org/10.1109/ACCESS.2024.3445432

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology, 33*(8), 1–79. https://doi.org/10.1145/3695988

Husein, R. A., Aburajouh, H., & Catal, C. (2025). Large language models for code completion: A systematic literature review. *Computer Standards & Interfaces, 92*, 103917. https://doi.org/10.1016/j.csi.2024.103917

International Organization for Standardization. (2023). *ISO/IEC 25010:2023 systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—Product quality model*. https://www.iso.org/standard/78176.html

Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: A practical primer for t-tests and ANOVAs. *Frontiers in Psychology, 4*, Article 863. https://doi.org/10.3389/fpsyg.2013.00863

Liang, J. T., Yang, C., & Myers, B. A. (2024). A large-scale survey on the usability of AI programming assistants: Successes and challenges. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 1–13. https://doi.org/10.1145/3597503.3608128

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, SE-2*(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. (2023). GitHub Copilot AI pair programmer: Asset or liability? *Journal of Systems and Software, 203*, 111734. https://doi.org/10.1016/j.jss.2023.111734

Mozannar, H., Bansal, G., Fourney, A., & Horvitz, E. (2024). Reading between the lines: Modeling user behavior and costs in AI-assisted programming. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–16. https://doi.org/10.1145/3613904.3641936

Nguyen, N., & Nadi, S. (2022). An empirical evaluation of GitHub Copilot's code suggestions. *Proceedings of the 19th International Conference on Mining Software Repositories*, 1–5. https://doi.org/10.1145/3524842.3528470

Oertel, J., Klünder, J., & Hebig, R. (2025). Don't settle for the first! How many GitHub Copilot solutions should you check? *Information and Software Technology, 183*, 107737. https://doi.org/10.1016/j.infsof.2025.107737

Ouyang, S., Zhang, J. M., Harman, M., & Wang, M. (2025). An empirical study of the non-determinism of ChatGPT in code generation. *ACM Transactions on Software Engineering and Methodology, 34*(2), 1–28. https://doi.org/10.1145/3697010

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. *2022 IEEE Symposium on Security and Privacy*, 754–768. https://doi.org/10.1109/SP46214.2022.9833571

Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do users write more insecure code with AI assistants? *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2785–2799. https://doi.org/10.1145/3576915.3623157

Prather, J., Reeves, B. N., Denny, P., Becker, B. A., Leinonen, J., Luxton-Reilly, A., Powell, G., Finnie-Ansley, J., & Santos, E. A. (2024). "It's weird that it knows what I want": Usability and interactions with Copilot for novice programmers. *ACM Transactions on Computer-Human Interaction, 31*(1), 1–31. https://doi.org/10.1145/3617367