GENERACIÓN DE CÓDIGO BASADA EN LLM: UNA REVISIÓN SISTEMÁTICA DE TÉCNICAS, MÉTRICAS Y EVALUACIÓN EMPÍRICA

M.Sc. Jorge Bergman Mostajo Pedraza
Posgrado SOE – UAGRM
https://orcid.org/0009-0008-5068-3096
Santa Cruz, Bolivia | jmostajo78@gmail.com

https://doi.org/10.23670 FT.2026.1.31

Recibido 23/04/2026 - Aceptado 13/05/2026

RESUMEN

Esta revisión sistemática de la literatura (SLR) analiza de forma crítica la evidencia científica sobre el uso de modelos de lenguaje a gran escala (LLMs) para la generación de código en ingeniería de software, con especial atención a su aplicabilidad en el ecosistema .NET. La búsqueda se realizó en cinco bases de datos (IEEE Xplore, ACM Digital Library, Google Scholar, Semantic Scholar y arXiv) siguiendo el protocolo PRISMA, identificando 7,159 estudios iniciales. Tras las fases de cribado, elegibilidad y evaluación de calidad, se seleccionaron 40 estudios primarios publicados entre 2020 y 2025. Los resultados muestran que el prompt engineering constituye la técnica dominante (72.5%), mientras que el fine-tuning y el preentrenamiento especializado actúan como estrategias complementarias (40%). Asimismo, se identifica una tendencia emergente hacia sistemas agénticos, en los que los LLMs evolucionan de generadores de código a componentes capaces de orquestar herramientas y resolver tareas a nivel de repositorio. En cuanto a la evaluación, se observa una fuerte dependencia de métricas automáticas como pass@k y benchmarks sintéticos, particularmente HumanEval, lo que introduce un sesgo sistemático en la estimación del rendimiento.

El estudio identifica una brecha estructural, denominada benchmark saturation gap, entre el rendimiento reportado en benchmarks sintéticos y el desempeño en escenarios reales, evidenciada por resultados significativamente inferiores en benchmarks más representativos como BigCodeBench y SWE-bench. Adicionalmente, se confirman limitaciones persistentes, incluyendo alucinaciones de código, vulnerabilidades de seguridad y degradación de la calidad.
Finalmente, se identifican brechas críticas en la literatura, destacando la ausencia de estudios específicos en el ecosistema .NET/C#, la escasez de evaluaciones longitudinales y la falta de marcos de medición en contextos de alta madurez. Estos hallazgos evidencian la necesidad de redefinir los enfoques de evaluación y de adaptar las prácticas de desarrollo para una integración efectiva y confiable de LLMs en entornos reales de ingeniería de software.

Palabras clave: Modelos de lenguaje a gran escala, Generación automática de código, Ingeniería de software, Ingeniería de prompts, Métricas de evaluación de código

ABSTRACT

This systematic literature review (SLR) critically examines the scientific evidence on the use of Large Language Models (LLMs) for code generation in software engineering, with particular attention to their applicability within the .NET ecosystem. The search was conducted across five databases (IEEE Xplore, ACM Digital Library, Google Scholar, Semantic Scholar, and arXiv) following the PRISMA protocol, identifying 7,159 initial records. After screening, eligibility assessment, and quality evaluation, 40 primary studies published between 2020 and 2025 were selected.
The results indicate that prompt engineering is the dominant technique (72.5%), while fine-tuning and specialized pretraining act as complementary strategies (40%). An emerging trend toward agentic systems is also identified, where LLMs evolve from standalone code generators into components capable of orchestrating tools and solving repository-level tasks. Regarding evaluation, there is a strong reliance on automated metrics such as pass@k and synthetic benchmarks, particularly HumanEval, which introduces a systematic bias in performance estimation.
The study identifies a structural gap, referred to as the benchmark saturation gap, between performance reported on synthetic benchmarks and real-world effectiveness, as evidenced by significantly lower results on more representative benchmarks such as BigCodeBench and SWE-bench. Additionally, persistent

limitations are confirmed, including code hallucinations, security vulnerabilities, and degradation of code quality. Finally, critical gaps in the literature are identified, including the lack of studies specifically addressing the .NET/C# ecosystem, the scarcity of longitudinal evaluations, and the absence of measurement frameworks in high-maturity contexts. These findings highlight the need to redefine evaluation approaches and adapt development practices to enable the effective and reliable integration of LLMs into real-world software engineering environments.

Keywords: Large Language Models, Automated Code Generation, Software Engineering, Prompt Engineering, and Code Evaluation Metrics

INTRODUCCIÓN

Los modelos de lenguaje a gran escala (LLMs) han transformado significativamente la generación automática de código a partir de descripciones en lenguaje natural (NL2Code), con modelos como GPT-4, Codex y Code Llama demostrando capacidades avanzadas para sintetizar código funcional y automatizar tareas de desarrollo (Brown et al., 2020; Chen et al., 2021; Rozière et al., 2023). Herramientas como GitHub Copilot han trasladado estas capacidades a entornos reales, con evidencia consistente de mejoras en productividad individual (Peng et al., 2023; Ziegler et al., 2022).

Sin embargo, la literatura existente presenta una brecha estructural: existe falta de estandarización en métricas y escasa evidencia en contextos reales, lo que limita la comparabilidad y aplicabilidad de los resultados. Benchmarks ampliamente adoptados como HumanEval han sido cuestionados por sobreestimar el rendimiento de los modelos (Chen et al., 2021; Liu et al., 2023), y evaluaciones más representativas como SWE-bench y BigCodeBench evidencian caídas significativas de desempeño cuando se introducen dependencias contextuales y entornos reales, fenómeno que este estudio denomina benchmark saturation gap (Jimenez et al., 2024; Zhuo et al., 2024).

Esta inconsistencia se agrava en ecosistemas con alta dependencia de frameworks específicos, como .NET, donde la evidencia empírica sigue siendo prácticamente inexistente.

Ante este panorama, resulta necesaria una revisión sistemática que permita consolidar el conocimiento disponible, evaluar críticamente los enfoques de medición e identificar las principales limitaciones del campo. Esta revisión se diferencia de trabajos previos por su enfoque simultáneo en técnicas, métricas y brechas de aplicabilidad industrial, con atención específica al ecosistema .NET y siguiendo el protocolo PRISMA con criterios de selección y evaluación de calidad explícitos (Kitchenham & Charters, 2007; Page et al., 2021).

En este contexto, el presente estudio tiene como objetivo analizar de manera sistemática la literatura sobre generación de código basada en LLMs, buscando: (i) caracterizar las técnicas, modelos y tareas NL2Code más utilizadas; (ii) examinar las métricas y enfoques de evaluación empleados; e (iii) identificar las principales limitaciones, brechas y desafíos que afectan la adopción efectiva de estas tecnologías en entornos reales de ingeniería de software.

METODOLOGÍA DE REVISIÓN

Este estudio adopta un enfoque de Revisión Sistemática de la Literatura (SLR) para analizar la evidencia científica existente sobre la generación de código mediante LLMs en la implementación de software. La revisión sigue las directrices del protocolo PRISMA (Page et al., 2021) y las recomendaciones de Kitchenham & Charters (2007), garantizando la transparencia, reproducibilidad y rigor metodológico en todo el proceso. El protocolo fue definido a priori, especificando las preguntas de investigación, la estrategia de búsqueda, los criterios de inclusión y exclusión, y los procedimientos de análisis. La gestión de referencias bibliográficas y el seguimiento del proceso de selección fueron apoyados mediante Zotero, herramienta que facilitó la organización, deduplicación y trazabilidad de los estudios recuperados en las distintas bases de datos.

Preguntas de Investigación (RQS)

La revisión se orienta mediante cinco preguntas de investigación consolidadas que cubren las dimensiones técnicas, métricas, de gestión e impacto de los LLMs en la generación de código para la implementación de software (Tabla 1).

Tabla 1

Preguntas de investigación de la revisión sistemática

ID	Pregunta de investigación
RQ1	¿Qué técnicas, modelos de lenguaje y tareas NL2Code se utilizan en la generación de código en ingeniería de software?
RQ2	¿Qué métricas, benchmarks y enfoques de evaluación se emplean para medir el rendimiento de los LLMs en la generación de código?
RQ3	¿Cuáles son las principales limitaciones, brechas y desafíos identificados en la literatura sobre generación de código basada en LLMs?

Nota. Las RQs se alinean directamente con los objetivos del estudio y guían la clasificación, análisis y síntesis de la evidencia.

Estrategia de Búsqueda

La búsqueda se ejecutó en cinco bases de datos científicas de referencia. La cadena de búsqueda general es:

(("large language model" OR "LLM" OR "GPT" OR "Codex" OR "code generation" OR "NL2Code" OR "natural language to code" OR "AI-assisted coding" OR "GitHub Copilot" OR "generative AI") AND ("software development" OR "software engineering" OR "software implementation" OR "code synthesis" OR "automated programming" OR "software process" OR "code quality"))

La Tabla 2 detalla la cadena adaptada a cada base de datos y los estudios recuperados. Filtros comunes: 2020–2025; áreas Computer Science, Software Engineering, Artificial Intelligence; artículos revisados por pares; idioma inglés.

Tabla 2

Cadenas de búsqueda por base de datos y estudios recuperados

Base de datos	N	Cadena de búsqueda específica
arXiv	125	(large language model OR LLM OR GPT OR Codex OR code generation OR NL2Code OR natural language to code OR AI-assisted coding OR GitHub Copilot OR generative AI) AND (software development OR software engineering OR software implementation OR code synthesis OR automated programming OR software process OR code quality)
Semantic Scholar	18	("code generation" OR "NL2Code" OR "natural language to code" OR "code synthesis" OR "AI-assisted coding" OR "code completion") AND ("large language model" OR "LLM" OR "GPT" OR "Codex" OR "GitHub Copilot" OR "generative AI" OR "ChatGPT" OR "Code Llama" OR "StarCoder" OR "transformer") AND ("software engineering" OR "software development" OR "software process" OR "software quality" OR "productivity" OR "defect" OR "code quality")
ACM Digital Library	2,165	("code generation" OR "NL2Code" OR "natural language to code") AND ("large language model" OR "LLM" OR "GPT" OR "Codex" OR "GitHub Copilot" OR "generative AI") AND ("software engineering" OR "software development" OR "software quality") — Tipo: Articles, Papers
Google Scholar	1,240	"code generation" ("large language model" OR "LLM" OR "GPT" OR "Codex" OR "GitHub Copilot") "software engineering" — Tipo: Conferences, Journals, Papers, Articles, Surveys
IEEE Xplore	3,611	((("large language model" OR "LLM" OR "GPT" OR "Codex" OR "code generation" OR "NL2Code" OR "natural language to code" OR "AI-assisted coding" OR "GitHub Copilot" OR "generative AI") AND ("software development" OR "software engineering" OR "software implementation" OR "code synthesis" OR "automated programming" OR "software process" OR "code quality"))) — Tipo: Conferences, Journals — Área: Computing & Processing

Nota. N = estudios inicialmente recuperados antes de la eliminación de duplicados. Total identificados: 7,159 estudios.

Criterios de Inclusión y Exclusión

Las Tablas 3 y 4 presentan los criterios de inclusión (CI) y exclusión (CE) definidos a priori.

Tabla 3

Criterios de inclusión (CI) aplicados en el proceso de selección de estudios

ID	Criterio de inclusión
CI1	Se incluyen estudios que aborden explícitamente el uso de modelos de lenguaje (LLMs) para la generación automática de código (NL2Code) en el contexto de ingeniería de software.
CI2	El estudio debe analizar al menos uno de los siguientes aspectos: técnicas (prompt engineering, fine-tuning, agentes, etc.), modelos (GPT, Code Llama, Codex, etc.), tareas de generación de código (síntesis, reparación, testing, etc.)
CI3	El estudio debe incluir algún tipo de evaluación del rendimiento, como: métricas cuantitativas (pass@k, exact match, etc.), benchmarks (HumanEval, MBPP, SWE-bench, etc.), estudios experimentales o comparativos
CI4	El artículo aborda la integración de la generación de código basada en LLMs en procesos, entornos o herramientas de desarrollo de software (p. ej., IDEs, pipelines CI/CD, entornos .NET).

Nota. Para ser incluido en la revisión, un estudio debe cumplir obligatoriamente con CI1 y CI3, y adicionalmente con CI2 o CI4. Esta regla garantiza la relevancia temática y la presencia de evidencia empírica, al tiempo que permite capturar tanto estudios centrados en técnicas como en contextos de integración.

Tabla 4

Criterios de exclusión (CE) aplicados en el proceso de selección de estudios

ID	Criterio de exclusión
CE1	El artículo no aborda la generación de código mediante LLMs ni presenta técnicas aplicables a NL2Code en ingeniería de software.
CE2	El artículo no cuenta con resumen (abstract).
CE3	El artículo consiste únicamente en un resumen (no dispone de texto completo).
CE4	El artículo no está escrito en inglés.
CE5	El artículo es una copia, duplicado o versión anterior de otro artículo ya incluido.
CE6	No se pudo acceder al texto completo del artículo.
CE7	El artículo no está relacionado con ingeniería de software, ciencias de la computación o inteligencia artificial.
CE8	El artículo corresponde a un estudio secundario (mapeo sistemático, revisión sistemática) y no a un estudio primario.

Nota. Los criterios CE son eliminatorios en cualquier fase. CE1–CE4 se verifican en el cribado; CE5–CE8 durante la evaluación de texto completo.

Proceso de Selección de Estudios

El proceso de selección sigue el flujo de cuatro fases recomendado por PRISMA: identificación, cribado, elegibilidad e inclusión.

Identificación

La búsqueda en las cinco bases de datos produjo 7,159 estudios iniciales. Tras la eliminación de 2,005 duplicados (≈ 28%), se obtuvieron 5,154 estudios únicos para el cribado.

Cribado: Primer filtro: título y resumen

La evaluación por título y resumen de los 5,154 estudios únicos se realizó en dos etapas:

- En la primera, la revisión sistemática de títulos permitió excluir 4,480 estudios (≈ 86.9%) mediante la aplicación de los criterios de exclusión: CE1 (el artículo no aborda generación de código con LLMs ni técnicas NL2Code), CE4 (no escrito en inglés), CE5 (duplicado o versión anterior ya incluida), y CE8 (no relacionado con ingeniería de software, ciencias de la computación o inteligencia artificial). Los 674 estudios restantes presentaban al menos un indicador temático de alineación con los objetivos de la revisión y avanzaron a la evaluación detallada de título y resumen.
- En la segunda etapa, se aplicaron sistemáticamente los criterios CI1–CI4 y CE1–CE8 mediante la lectura detallada del título y resumen de los 674 estudios candidatos. Este proceso derivó en la evaluación formalmente documentada de 47 estudios correspondientes al período 2020–2025, seleccionados de las bases de datos arXiv, IEEE Xplore, ACM Digital Library, Google Scholar y Semantic Scholar, cuyo contenido evidenciaba relevancia directa con la generación de código mediante LLMs.

La selección de estos estudios respondió a tres criterios operativos:

- (a) publicación dentro del período 2020–2025
- (b) procedencia de alguna de las cinco bases de datos del estudio
- (c) presencia explícita en el título o resumen de términos vinculados a generación de código, LLMs o evaluación cuantitativa NL2Code.

De los 47 estudios evaluados, 40 (85%) satisfacen al menos un criterio de inclusión, CI1: 14; CI2: 5; CI3: 15; CI4: 6, y 7 (15%) son excluidos, CE1: 4; CE2: 1; CE8: 2.

El criterio CI3 (métricas cuantitativas o evaluación empírica) fue el más frecuente (15 de 47, ≈ 31.9%), confirmando que la evidencia cuantitativa es el rasgo definitorio del corpus relevante. Los restantes 627 estudios fueron descartados durante la exploración inicial de títulos, principalmente por CE1, al no abordar específicamente la generación de código mediante LLMs.

Para cada uno de los 47 estudios formalmente evaluados en esta fase, seleccionados de los 674 candidatos y correspondientes al período 2020–2025, se registra el identificador, año de publicación, base(s) de datos de origen, autores, título, la justificación del criterio aplicado por el Revisor 1, y la decisión de cribado.

Los criterios de inclusión aplicados fueron CI1: 14; CI2: 3; CI3: 16; CI4: 6, con un total incluido de 40 estudios (85%). Los criterios de exclusión aplicados fueron CE1: 3; CE2: 1; CE8: 2, con un total excluido de 7 estudios (15%). La justificación completa de cada decisión consta en el protocolo de revisión.

Elegibilidad

Los estudios que superaron la fase de cribado inicial fueron evaluados en texto completo para determinar su elegibilidad, aplicando de manera sistemática los criterios de inclusión y exclusión definidos (CI1–CI4 y CE1–CE8).

En esta fase, se verificó que los estudios abordaran explícitamente la generación de código mediante modelos de lenguaje en el contexto de ingeniería de software (CI1) y que presentaran evidencia empírica basada en métricas, benchmarks o evaluaciones experimentales (CI3).

Asimismo, se consideró como criterio complementario que los estudios analizaran técnicas, tareas de NL2Code (CI2) o su integración en herramientas y procesos de desarrollo (CI4).

El proceso de evaluación en texto completo permitió confirmar la elegibilidad de 40 estudios primarios, los cuales cumplían con los requisitos de relevancia temática y evidencia empírica.

No se identificaron exclusiones adicionales en esta fase, lo que indica que el proceso de cribado inicial fue suficientemente riguroso para filtrar estudios no pertinentes.

Inclusión

Los 40 estudios elegibles fueron incluidos en la revisión sistemática tras una evaluación final orientada a verificar su calidad metodológica y su pertinencia respecto a las preguntas de investigación.

En particular, todos los estudios incluidos cumplen con: relevancia temática en generación de código basada en LLMs en ingeniería de software, presencia de evaluación empírica o cuantitativa, y contribuciones analizables en términos de técnicas, métricas o limitaciones del enfoque NL2Code.

El corpus final está compuesto por 40 estudios primarios publicados entre 2020 y 2025, que constituyen la base empírica para el análisis de resultados y la discusión. Este conjunto permite abordar de manera estructurada las preguntas de investigación relacionadas con técnicas (RQ1), enfoques de evaluación (RQ2) y limitaciones y brechas (RQ3).

Tabla 5

Resumen del proceso de selección de estudios – protocolo PRISMA

Fase	Descripción	Estudios
Identificación	Estudios iniciales en bases de datos	7,159
Duplicados eliminados	Estudios repetidos entre bases de datos	2,005
Cribado	Estudios únicos evaluados por título y resumen	5,154
Excluidos en cribado	No relacionados con NL2Code o fuera de dominio	4,480
Excluidos en 2da etapa cribado	No relacionados con la generación de código mediante LLMs	627
Excluidos en 3ra etapa cribado	No corresponden a estudios primarios y no tienen resumen	7
Elegibilidad	Estudios evaluados en texto completo	40
Excluidos en elegibilidad	Sin métricas cuantitativas, sin LLMs directos, sin acceso	0
Inclusión final	Estudios incluidos en la revisión sistemática	40

Nota. El porcentaje de inclusión final (40/7,159 ≈ 0.56%) es consistente con el rango típico de 0.5%–2% en revisiones sistemáticas en ingeniería de software (Kitchenham & Charters, 2007).

Extracción de Datos

Para cada estudio incluido se aplicó un formulario de extracción estructurado alineado con las tres RQs, donde se incluye los siguientes campos: un identificador único del estudio (categórico, ej. 2020.01, 2021.03, ..., 2025.01); la referencia bibliográfica completa (autor(es), año, venue); el año de publicación (numérico, 2020–2025); el tipo de estudio, que clasifica el diseño metodológico en experimental, benchmark, empírico o comparativo; la técnica o enfoque empleado para la generación de código con LLMs ,prompt engineering, fine-tuning, RAG, agentes, (RQ1); los modelos de lenguaje utilizados, como GPT-4, Codex, Code Llama o CodeT5 (RQ1); la tarea NL2Code soportada, incluyendo generación, refactorización, pruebas o documentación (RQ1); las métricas cuantitativas de evaluación, como pass@k, exact match o CodeBLEU (RQ2); el benchmark o dataset utilizado, como HumanEval, MBPP, SWE-bench o BigCodeBench (RQ2); el tipo de evaluación, ya sea automática, experimental, con usuarios o comparativa (RQ2); el contexto de evaluación, que distingue entre benchmark sintético, repositorios reales o entorno industrial (RQ2); **las limitaciones identificadas** en el estudio, como alucinaciones, vulnerabilidades de seguridad, deuda técnica o dependencia del prompt (RQ3); y finalmente las **brechas o trabajo futuro reportados**, como falta de evaluación real o baja generalización (RQ3).

El formulario fue diseñado para asegurar la trazabilidad entre los datos recolectados y las preguntas de investigación, permitiendo un análisis estructurado de técnicas (RQ1), enfoques de evaluación (RQ2) y limitaciones y brechas (RQ3).

Síntesis y Análisis de Datos

El análisis de los datos extraídos se realizó mediante una síntesis cualitativa de tipo categórico, alineada con las preguntas de investigación. A partir de la información recopilada en el formulario de extracción, se aplicó un proceso de codificación temática para identificar patrones, relaciones y tendencias entre los estudios seleccionados.

Los estudios fueron comparados de manera transversal para reconocer convergencias y divergencias en los enfoques de generación de código, los métodos de evaluación y las limitaciones reportadas. Asimismo, la frecuencia de aparición de determinadas categorías se consideró un indicador de relevancia y consolidación dentro del campo.

Este enfoque permitió interpretar críticamente el estado actual de la investigación, la consistencia de los métodos de evaluación y las principales brechas metodológicas y desafíos asociados a la integración de LLMs en tareas NL2Code, proporcionando una base sólida para el análisis y discusión de resultados.

Evaluación de Calidad

Se aplicó un instrumento de cuatro criterios (QC1–QC4) para examinar la solidez metodológica de los estudios incluidos: QC1 claridad y especificidad de objetivos; QC2 rigor del diseño metodológico; QC3 validez interna de los resultados; QC4 reproducibilidad del procedimiento. Cada criterio se puntuó con 1.0 (cumplimiento completo), 0.5 (cumplimiento parcial) o 0.0 (no cumple), obteniendo una puntuación total por estudio en el rango 0–4.

Niveles de calidad: Alta (≥ 3.5) | Media (3.0) | Baja (< 3.0). Los estudios de alta calidad tuvieron mayor peso interpretativo en la síntesis; los de calidad media se consideraron evidencia complementaria.

Resultados globales: Promedio: 3.45/4.0 · Máximo: 4.0 · Mínimo: 3.0. Distribución: 26 estudios de alta calidad (65%), 14 de calidad media (35%), 0 de baja calidad (0%). El corpus presenta diseños metodológicos sólidos y evaluación empírica consistente, aunque con limitaciones en reproducibilidad y profundidad de validación.

Los puntajes detallados de la evaluación de calidad metodológica de los 40 estudios incluidos son los siguientes. Los estudios 2020.01, 2021.01, 2022.06, 2023.09, 2023.10, 2024.04, 2024.05, 2024.08, 2024.09 y 2024.15 alcanzaron la puntuación máxima de 4.0, cumpliendo plenamente los cuatro criterios. Con una puntuación de 3.5, clasificados como de alta calidad, se encuentran los estudios 2020.02, 2022.01, 2022.02, 2022.03, 2022.04, 2022.05, 2023.03, 2023.06, 2023.11, 2023.12, 2023.13, 2024.01, 2024.10, 2024.12, 2024.13 y 2024.14. Los estudios 2020.03, 2021.04, 2022.07, 2022.08, 2023.02, 2023.04, 2023.05, 2023.14, 2024.03, 2024.07, 2024.11, 2025.01 y 2025.02 obtuvieron una puntuación de 3.0, correspondiente al nivel de calidad media.

Ningún estudio registró una puntuación inferior a 3.0. En conjunto, el corpus presenta una puntuación promedio de 3.45/4.0, con 26 estudios de alta calidad (65%) y 14 de calidad media (35%), lo que refleja diseños metodológicos sólidos y evaluación empírica consistente, aunque con limitaciones en reproducibilidad y profundidad de validación. La escala de puntuación aplicada fue: 1.0 = cumple completamente; 0.5 = cumplimiento parcial; 0.0 = no cumple, donde QC1 corresponde a claridad de objetivos, QC2 a rigor metodológico, QC3 a validez interna y QC4 a reproducibilidad.

Amenazas a la Validez

Se identificaron las siguientes amenazas potenciales:

Validez de construcción
Posible sesgo en la definición de la cadena de búsqueda y criterios de selección. Para mitigarlo, la estrategia de búsqueda fue revisada iterativamente y alineada con los objetivos del estudio.

Validez interna
Riesgo de subjetividad en la selección de estudios, extracción de datos y evaluación de calidad. Este riesgo se mitigó mediante el uso de formularios estructurados, criterios explícitos (CI/CE) y reglas de decisión consistentes.

Validez externa
La generalización de los resultados puede estar limitada por la predominancia de estudios evaluados en benchmarks sintéticos (p. ej., HumanEval, MBPP), lo que restringe su extrapolación a entornos industriales reales.

Validez de conclusión
La heterogeneidad en métricas, datasets y diseños experimentales dificulta la comparación directa entre estudios. Esta limitación fue abordada mediante una síntesis cualitativa y categórica en lugar de un metaanálisis cuantitativo.

RESULTADOS Y DISCUSIÓN

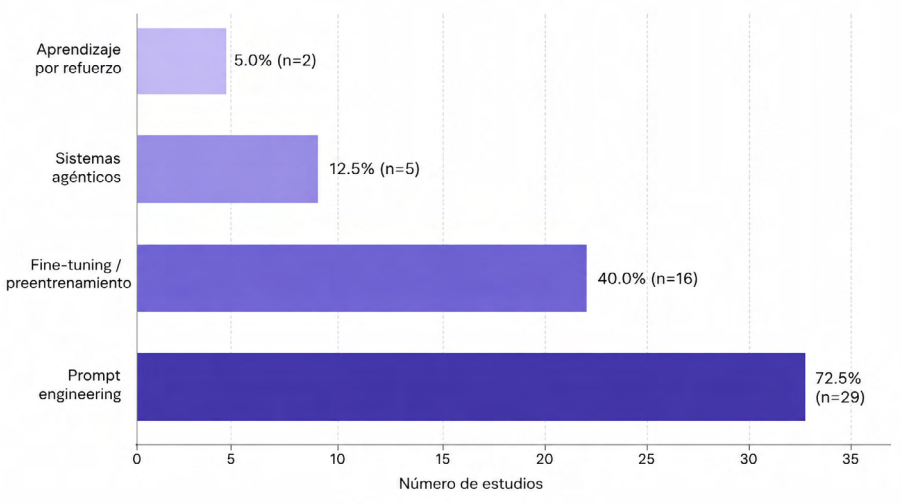
Los resultados fueron interpretados considerando el nivel de calidad de los estudios, priorizando la evidencia proveniente de estudios clasificados como de alta calidad. Esta estrategia permitió reducir el impacto de posibles sesgos metodológicos y fortalecer la validez de las conclusiones.

RQ1: Técnicas y enfoques empleados

El análisis de los 40 estudios seleccionados evidencia que el prompt engineering constituye actualmente el enfoque dominante en la generación de código con modelos de lenguaje, presente en el 72.5% del corpus (n = 29). En particular, el uso de few-shot prompting y de instrucciones en lenguaje natural se ha consolidado como práctica estándar en modelos ampliamente adoptados como GPT-4 y ChatGPT (Chen et al., 2021; Liu et al., 2023). De manera complementaria, el fine-tuning y el preentrenamiento especializado aparecen en el 40% (n = 16) de los estudios. A partir de 2023 emergen enfoques agénticos (12.5%, n = 5), en los que los LLMs se integran en flujos de desarrollo más amplios, coordinando herramientas y abordando tareas a nivel de repositorio (Zhang et al., 2024; Jimenez et al., 2024). Los enfoques basados exclusivamente en aprendizaje por refuerzo representan el 5% (n = 2) del corpus. Esto se muestra en la Figura 1.

Figura 1

Técnicas y enfoques en generación de código con LLMs (RQ1)



Nota. Los porcentajes no suman 100% ya que un mismo estudio puede emplear múltiples técnicas. N = 40 estudios primarios, 2020-2025

RQ2: Evaluación, métricas y benchmarks

La evaluación del rendimiento en generación de código está dominada por métricas automáticas, siendo pass@k el estándar más utilizado (80%, n = 32), aplicado principalmente sobre HumanEval, presente en el 55% (n = 22) del corpus.

La evidencia analizada indica que estos benchmarks tienden a sobreestimar el rendimiento de los modelos en escenarios complejos (Liu et al., 2023). Benchmarks más recientes como BigCodeBench (Zhuo et al., 2024) y SWE-bench (Jimenez et al., 2024) registran caídas del 25–47% respecto a HumanEval.

Se define benchmark saturation gap como la brecha estructural entre el rendimiento reportado en benchmarks sintéticos –cuando estos operan simultáneamente como referencia de evaluación y objetivo de optimización– y el desempeño efectivo en escenarios reales de ingeniería de software, evidenciada por dichas caídas (Liu et al., 2023; Jimenez et al., 2024; Zhuo et al., 2024).

A diferencia de los benchmarks tradicionales, los entornos reales exigen razonamiento multi-paso, integración contextual y adaptación a código existente.

Adicionalmente, solo el 15% (n = 6) de los estudios incorpora métricas orientadas a procesos, como productividad, tiempo de tarea o reducción de esfuerzo.

La evaluación de métricas y benchmarks se resume en la Figura 2.

Figura 2

Métricas y benchmarks de evaluación de LLMs para generación de código (RQ2)

Métrica / Benchmark	Uso en corpus	Tipo	Representatividad	Limitación principal
Métricas automáticas				
pass@k	≈ 80%	Automática	Sintética	Solo mide corrección fundional básica; ignora seguridad, mantenibilidad y arquitectura
Exact match	Moderado	Automática	Sintética	Penaliza variaciones semánticamente correctas; poco útil en código real
CodeBLEU	Moderado	Automática	Sintética	Correlación débil con corrección funcional en escenarios complejos
Coverage (stmt/branch)	15%	Proceso	Proceso	Limitada a test generation; no captura calidad sistémica global
Benchmarks sintéticos (alta saturación)				
HumanEval	55%	Benchmark	Muy baja	Sobreestima rendimiento; tareas cerradas sin dependencias reales, benchmark saturation gap
MBPP	Frecuente	Benchmark	Baja	Problemas de programación básica; no refleja lógica de negocio ni integraciones
APPS	Moderado	Benchmark	Media	Mayor complejidad algorítmica, pero aún estilo competitivo; sin contexto profesional
Benchmarks representativos (mayor validez externa)				
SWE-bench	Emergente	Benchmark	Alta	Rendimiento cae 25–47% vs HumanEval; razonamiento multi-paso en repos reales
BigCodeBench	Emergente	Benchmark	Alta	Expone fallas en uso de APIs e instrucciones complejas; bajo en corpus actual
HumanEval.X	Limitado	Benchmark	Media	Multilingüe (5 lenguajes); sin .NET/C#; hereda limitaciones de HumanEval
Métricas de proceso (brecha identificada)				
Productividad / tiempo	15%	Proceso	Industrial	Escasez crítica; desconexión entre evaluación académica y necesidades reales de equipos
Reducción de esfuerzo	15%	Proceso	Industrial	Solo Peng et al. (2023) y Ziegler et al. (2022) aportan evidencia limitada sólida

Nota. El porcentaje de uso refleja la proporción del corpus (N=40) en que aparece cada métrica o benchmark.

RQ3: Tareas, limitaciones y brechas

La tarea NL2Code está presente en el 100% (n = 40) del corpus. Se identifican subtareas relevantes: reparación de código (20%, n = 8), generación de pruebas (7.5%, n = 3) y análisis de calidad (7.5%, n = 3).

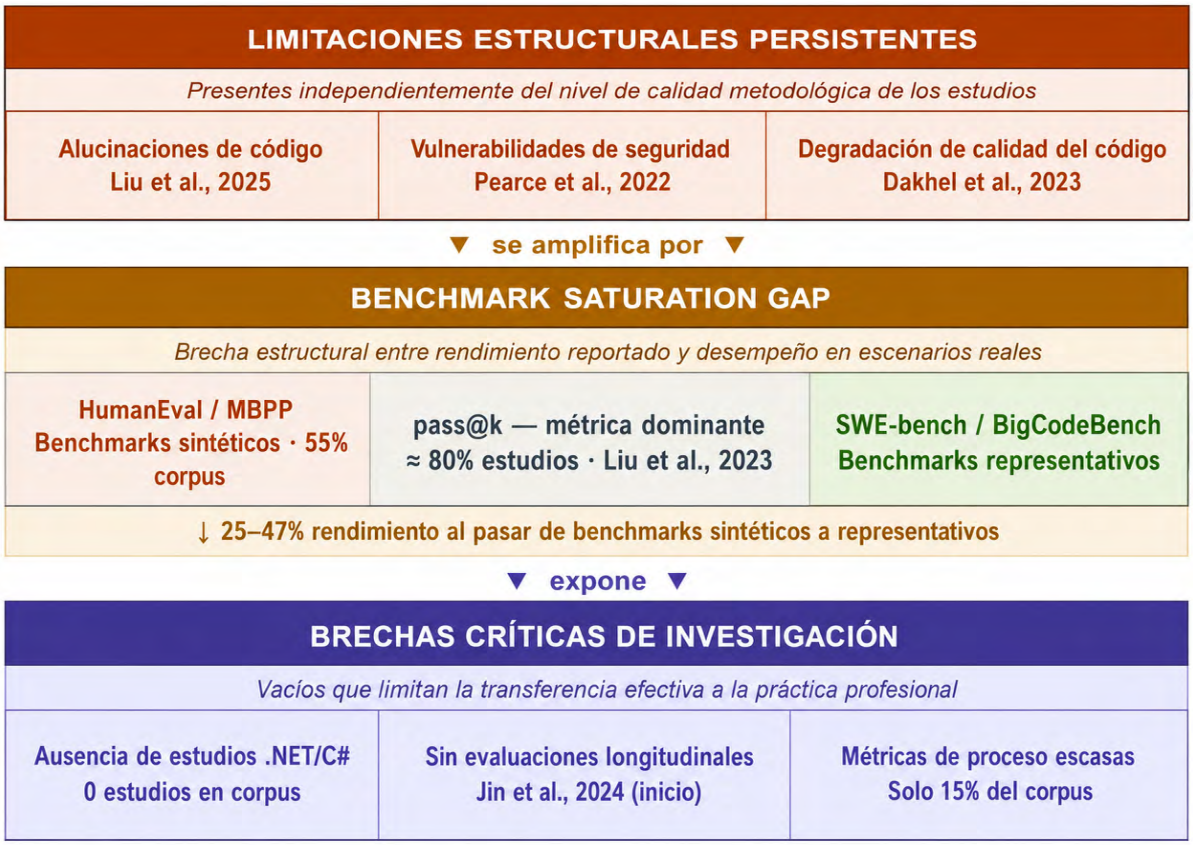
Las limitaciones reportadas son consistentes a lo largo de los estudios, independientemente de su nivel de calidad metodológica: alucinaciones de código (Liu et al., 2025), vulnerabilidades de seguridad (Pearce et al., 2022) y degradación de la calidad del código (Dakhel et al., 2023; Mastropaolo et al., 2023).

Se identifican asimismo dificultades recurrentes en tareas con lógica de negocio compleja, dependencias externas y uso de APIs reales.

En términos de brechas de investigación, se identifica ausencia de estudios en el ecosistema .NET/C# (0%, n = 0), escasez de evaluaciones longitudinales y falta de marcos de integración en contextos industriales. Para mayor aclaración ver la Figura 3.

Figura 3

Brechas y limitaciones estructurales en LLMs para generación de código (RQ3)



Nota. Las flechas indican relación de causalidad o amplificación entren niveles. Las citas corresponden a los estudios más representativos del corpus.

DISCUSIÓN

Los resultados muestran una tensión cada vez más evidente entre el desempeño de los LLMs en entornos controlados y su comportamiento en escenarios reales de ingeniería de software. La aparición de enfoques agénticos, como OpenHands (Wang et al., 2024), CodeAgent (Zhang et al., 2024) y Agentless (Xia et al., 2024), ha desplazado el foco desde la simple generación de código hacia la confiabilidad del proceso completo.

En este nuevo contexto, aspectos como la trazabilidad, la supervisión humana y el control del flujo de trabajo adquieren un papel central, aunque todavía no existen marcos consolidados que los aborden de manera sistemática. Precisamente, Nascimento et al. (2024) señalan que estos factores son determinantes para que dichos sistemas aporten valor real en entornos profesionales.

Comparación con revisiones previas y literatura relacionada

Los hallazgos de esta revisión coinciden con estudios previos en señalar al prompt engineering como la técnica predominante en el uso de LLMs para ingeniería de software, aunque se identifican diferencias importantes en el nivel de detalle del análisis. Por ejemplo, Hou et al. también reconocen el papel central del prompt engineering, pero su revisión no distingue el impacto específico de estrategias como few-shot prompting y chain-of-thought, diferencia que sí es abordada en el presente estudio. De manera similar, Fan et al. reportan el uso extendido de HumanEval como benchmark de referencia, aunque sin profundizar en el nivel de sobreestimación asociado a depender exclusivamente de este conjunto de pruebas.

En contraste, esta revisión aporta evidencia más concreta al mostrar que la diferencia de rendimiento entre HumanEval y benchmarks más exigentes varía entre un 25% y un 47%, superando las estimaciones implícitas planteadas en esos trabajos.

Asimismo, los resultados presentan diferencias parciales respecto a los estudios de Mastropaolo et al. y Dakhel et al., enfocados principalmente en Python y lenguajes de scripting. Ambos trabajos reportaron niveles moderados de corrección funcional con GitHub GitHub Copilot, cercanos al 46% en HumanEval.

Sin embargo, al incorporar investigaciones más recientes y modelos de mayor escala como GPT-4, Code Llama y WizardCoder, esta revisión evidencia una mejora significativa en dichas métricas. Aun así, los resultados muestran que ese incremento de desempeño no se refleja de manera proporcional en entornos reales de producción, lo que refuerza la relevancia del concepto de benchmark saturation gap como explicación de esta diferencia.

Este comportamiento no había sido descrito de

forma explícita en las revisiones anteriores, por lo que representa una de las principales contribuciones del presente estudio.

Caracterización y justificación teórica del benchmark saturation gap

La revisión también permite identificar y caracterizar lo que denominamos benchmark saturation gap: la diferencia entre los altos resultados reportados en benchmarks tradicionales como HumanEval (Chen et al., 2021), utilizado en el 55% del corpus, y las caídas significativas de rendimiento observadas en evaluaciones más cercanas a escenarios reales, como BigCodeBench (Zhuo et al., 2024) y SWE-bench (Jimenez et al., 2024), donde el desempeño disminuye entre un 25% y un 47%. Liu et al. (2023) explican este fenómeno al demostrar que, cuando un benchmark se convierte simultáneamente en referencia de evaluación y objetivo de optimización, deja de representar capacidades verdaderamente generalizables.

Además, la métrica pass@k, predominante en el 80% de los estudios, se limita a medir corrección funcional básica, dejando fuera dimensiones críticas en entornos profesionales como seguridad, mantenibilidad y coherencia arquitectónica, aspectos destacados por Dakhel et al. (2023) y Pearce et al. (2022).

La justificación teórica del constructo benchmark saturation gap puede entenderse a partir de una idea similar a la planteada por la Ley de Goodhart: "cuando una medida se convierte en objetivo, deja de ser una buena medida". En el caso de los LLMs orientados a generación de código, la optimización constante sobre benchmarks como HumanEval ha producido un efecto de retroalimentación en el que las métricas mejoran de manera significativa, aunque ello no implique necesariamente una mejora equivalente en escenarios reales de desarrollo de software.

Este problema tiene un carácter estructural, ya que no depende únicamente de la calidad de un modelo específico, sino también del diseño de los sistemas de evaluación. Como resultado, se genera una diferencia entre el desempeño funcional medido en entornos controlados y la efectividad real del modelo en contextos prácticos y complejos.

Esta discrepancia constituye la base del benchmark saturation gap: una brecha que las métricas tradicionales aún no logran representar adecuadamente. Esta interpretación coincide con la crítica epistemológica de Raji et al. (2021) sobre las limitaciones de los benchmarks como mecanismos suficientes para evaluar sistemas de inteligencia artificial.

Brecha de aplicabilidad industrial y ecosistemas específicos

Asimismo, se evidencia una importante brecha de aplicabilidad industrial, especialmente en ecosistemas como .NET/C# y organizaciones con procesos de alta madurez. Los marcos de evaluación más utilizados, basados principalmente en programación competitiva, no reflejan adecuadamente las condiciones de entornos donde la integración con frameworks específicos, la seguridad y la trazabilidad son requisitos indispensables. Aunque Pearce et al. (2022) y Dakhel et al. (2023) ya habían identificado estas limitaciones en Python y C, aún existe poca evidencia empírica sobre cómo se manifiestan en el ecosistema .NET.

Resulta particularmente revelador que solo el 15% del corpus incorpore métricas orientadas al proceso o a la calidad sistémica, lo que confirma una desconexión entre las prioridades de la investigación académica y las necesidades reales de organizaciones como las analizadas por Schäfer et al. (2024) y Siddiq & Da Silva Santos (2024).

Esta ausencia contrasta marcadamente con la madurez alcanzada en otros dominios de alta especialización. En el ámbito de Java Enterprise, por ejemplo, trabajos como Schäfer et al. (2024) han comenzado a construir marcos de evaluación que incorporan cobertura de ramas y métricas de integración, aunque tampoco en ese contexto existe consenso metodológico.

La ausencia de estudios equivalentes en .NET no es trivial: el ecosistema incluye particularidades como la dependencia del runtime de .NET, los patrones de inyección de dependencias propios de ASP. NET Core, y las convenciones de C# que difieren estructuralmente de Python o JavaScript, los lenguajes más representados en los benchmarks actuales.

Estudios como CodeGeeX (Zheng et al., 2023) y DeepSeek-Coder (Guo et al., 2024) ya evidencian que el rendimiento de los modelos no es homogéneo entre lenguajes: el desempeño observado en Python no se traslada directamente a C#, lo que hace arriesgado extrapolar conclusiones de un ecosistema a otro sin validación empírica específica.

Esta variabilidad refuerza la necesidad de evaluaciones diseñadas para el ecosistema .NET, en lugar de asumir que los resultados obtenidos en los benchmarks dominantes son representativos de un entorno con convenciones, patrones y dependencias propias.

Limitaciones estructurales de los LLMs actuales

Las limitaciones identificadas no parecen ser únicamente problemas técnicos aislados, sino fenómenos de carácter estructural. Aspectos como las alucinaciones de código (Liu et al., 2025), las vulnerabilidades de seguridad (Pearce et al., 2022) y la degradación de calidad del software (Mastropaolo et al., 2023) aparecen de forma consistente en distintos modelos y períodos analizados, lo que sugiere que el simple aumento de escala de los LLMs no es suficiente para resolverlos.

Olausson et al. (2024) muestran que los mecanismos de auto-reparación iterativa pueden mejorar resultados en tareas simples, pero siguen siendo insuficientes frente a lógica de negocio compleja.

En consecuencia, la adopción efectiva de estas tecnologías requiere estrategias de integración más robustas, incorporando validación automática, análisis

estático y controles de seguridad antes de integrar el código generado en pipelines CI/CD, tal como proponen Wang et al. (2025).

Estos hallazgos dialogan con los de Shinn et al. (2023), quienes con Reflexion mostraron que los LLMs pueden mejorar sus propias respuestas mediante retroalimentación verbal iterativa. Sin embargo, los resultados que emergen del presente corpus matizan ese optimismo: las ganancias observadas en entornos de benchmarks no se sostienen cuando el código debe integrarse en sistemas reales con dependencias, restricciones arquitectónicas y lógica de negocio específica.

Esto no invalida el enfoque de auto-reparación, pero sí revela sus límites: actúa sobre la forma del código más que sobre su comprensión del contexto. En la misma línea, Olausson et al. (2024) ya advierten que la auto-reparación no es una solución universal.

La presente revisión va un paso más allá al sugerir que, en lugar de tratarla como mecanismo autónomo, debería integrarse como una capa dentro de pipelines más amplios que combinen validación estática, análisis de seguridad y revisión humana, especialmente en entornos donde el costo de un error es alto.

Limitaciones de esta revisión sistemática

Esta revisión presenta limitaciones que deben considerarse al interpretar sus conclusiones. En primer lugar, la búsqueda se restringió a cinco bases de datos, lo que puede haber excluido literatura relevante publicada en repositorios especializados, actas de talleres o revistas de alcance regional.

Si bien la combinación de IEEE Xplore, ACM Digital Library, arXiv, Google Scholar y Semantic Scholar cubre la mayor parte de la producción científica relevante en ingeniería de software e inteligencia artificial, no puede descartarse la existencia de estudios pertinentes fuera de este conjunto.

En segundo lugar, la síntesis cualitativa adoptada, en lugar de un meta-análisis cuantitativo, limita la posibilidad de extraer estimaciones numéricas precisas sobre el rendimiento agregado de los modelos. Esta decisión fue justificada por la alta heterogeneidad en métricas, datasets y diseños experimentales del corpus; sin embargo, implica que las conclusiones cuantitativas reportadas (p. ej., el rango del 25–47% de caída de rendimiento) deben interpretarse como indicativas y no como parámetros estadísticamente consolidados.

En tercer lugar, la evaluación de calidad mediante cuatro criterios (QC1–QC4) fue realizada por un único revisor, sin proceso de doble ciego ni cálculo de concordancia inter-evaluador (p. ej., kappa de Cohen). Aunque se aplicaron rúbricas explícitas para minimizar la subjetividad, este diseño introduce un riesgo de sesgo de confirmación que no puede eliminarse completamente.

Futuras revisiones deberían incorporar revisión dual con cálculo de acuerdo.Finalmente, el corte temporal 2020–2025, aunque representativo de la etapa más dinámica del campo, excluye trabajos fundacionales previos (p. ej., modelos basados en LSTM o seq2seq preTransformer) que podrían aportar perspectiva histórica sobre la evolución de las limitaciones identificadas. Esta decisión fue deliberada y está justificada por el objetivo de centrarse en la generación LLM contemporánea, pero limita la capacidad de trazar tendencias de largo plazo

Líneas prioritarias de investigación futura

A partir de estos hallazgos emergen tres líneas prioritarias de investigación. La primera consiste en desarrollar benchmarks capaces de evaluar dimensiones más cercanas a la práctica profesional, incluyendo seguridad, mantenibilidad y coherencia arquitectónica, siguiendo iniciativas como CodeBenchGen (Zheng et al., 2024) y BigCodeBench (Zhuo et al., 2024).

La segunda apunta a la necesidad de estudios longitudinales que permitan analizar el impacto acumulado de los LLMs sobre la calidad del software, un vacío aún evidente en el corpus, pese a que Jin et al. (2024) ya reportan efectos acumulativos en escenarios reales de uso.

Finalmente, se requiere ampliar la investigación empírica hacia ecosistemas poco representados, como .NET/C#, considerando que trabajos como CodeGeeX (Zheng et al., 2023) y DeepSeek-Coder (Guo et al., 2024) muestran diferencias de rendimiento importantes entre lenguajes.

De los hallazgos identificados emergen cuatro agendas de investigación prioritarias. La primera consiste en el diseño de benchmarks específicos por ecosistema, incorporando dependencias de framework, como ASP.NET Core y Entity Framework, y contextos de repositorio real en C#, tomando como modelo un SWE-bench adaptado a .NET que incluya métricas de cobertura de pruebas y análisis estático con Roslyn.

La segunda apunta a la evaluación de seguridad como métrica primaria, integrando herramientas de análisis estático de seguridad (SAST), como CodeQL y SonarQube, en los pipelines de evaluación de LLMs, superando el uso exclusivo de pass@k y tomando como referencia metodológica el trabajo de Pearce et al. (2022) como punto de partida para un marco más sistemático.

La tercera línea propone estudios longitudinales de calidad de código, midiendo el impacto de la adopción continua de LLMs sobre indicadores de deuda técnica, cobertura de tests y frecuencia de defectos en repositorios reales a lo largo de periodos de 12 a 36 meses, con el diseño de Jin et al. (2024) como referencia replicable en otros contextos.

Finalmente, la cuarta agenda aborda el desarrollo de marcos de integración humano–IA en contextos de alta madurez, investigando cómo organizaciones con niveles de madurez CMMI 3–5 pueden integrar LLMs en sus procesos sin degradar la trazabilidad, el control de cambios y la conformidad normativa, vacío crítico

para sectores como fintech, salud y administración pública, donde la revisión no encontró ningún estudio representativo. Estas líneas emergen de las brechas identificadas en los 40 estudios primarios incluidos (RQ1–RQ3), siguiendo las recomendaciones de PRISMA (Page et al., 2021) y Kitchenham & Charters (2007).

Atender estas agendas de forma articulada permitirá reducir la distancia entre lo que los LLMs demuestran en laboratorio y lo que pueden ofrecer de manera confiable en la práctica. Con esa perspectiva en mente, cabe preguntarse qué nos dice, en conjunto, la evidencia acumulada.

La evidencia analizada no pone en duda que los LLMs pueden aportar valor real en ingeniería de software, las mejoras en productividad reportadas por Peng et al. (2023) y Ziegler et al. (2022) son consistentes y no deben minimizarse. Lo que sí cuestiona es la idea de que ese valor se materializa de forma automática. Para que los beneficios sean sostenibles en entornos profesionales reales, hace falta algo más que un modelo capaz: hacen falta prácticas de evaluación más honestas, procesos de integración que contemplen validación, seguridad y trazabilidad, y una disposición organizacional a rediseñar flujos de trabajo, no solo a insertar una herramienta nueva en los existentes.

En este sentido, como señalan Nascimento et al. (2024), el rendimiento de un LLM en un benchmark dice poco sobre su utilidad real si los procesos que lo rodean no están a la altura. Esta revisión comparte esa lectura y la extiende: la pregunta relevante ya no es solo qué tan bien genera código un modelo, sino en qué condiciones ese código puede integrarse de forma confiable en un sistema real, con personas reales, bajo presiones reales.

Ese desplazamiento, del modelo al sistema sociotécnico que lo contiene, es quizás el cambio de perspectiva más importante que la literatura reciente está comenzando a asumir, y que esta revisión busca contribuir a consolidar.

CONCLUSIONES

Esta revisión sistemática proporciona una visión consolidada y crítica del estado actual de la generación de código basada en LLMs en ingeniería de software, a partir del análisis de 40 estudios primarios publicados entre 2020 y 2025.

En relación con RQ1, se concluye que el prompt engineering se ha establecido como la técnica dominante para la interacción con LLMs, respaldada principalmente por estudios de alta calidad. Sin embargo, su predominio no implica suficiencia, ya que se observa una transición hacia enfoques más complejos, particularmente sistemas agénticos, que redefinen el rol de los modelos desde generadores de código hacia componentes activos en el ciclo de desarrollo.

Respecto a RQ2, los resultados evidencian una limitación metodológica significativa en la evaluación del rendimiento. La dependencia de métricas como pass@k y benchmarks sintéticos como HumanEval introduce un sesgo sistemático que sobreestima el desempeño de los modelos. La comparación con benchmarks más realistas como BigCodeBench y SWE-bench permite identificar una brecha estructural, denominada benchmark saturation gap, que cuestiona la validez externa de gran parte de la evidencia existente.

En cuanto a RQ3, se concluye que las limitaciones de los LLMs no son marginales, sino estructurales. Problemas como alucinaciones de código, vulnerabilidades de seguridad y degradación de calidad aparecen de forma consistente en el corpus, independientemente del nivel de calidad metodológica de los estudios. Asimismo, se identifican brechas críticas en la evaluación en entornos reales, particularmente en ecosistemas como .NET, así como en la ausencia de estudios longitudinales y de métricas orientadas a procesos.

A nivel general, los hallazgos de esta revisión sugieren que, si bien los LLMs han alcanzado un alto nivel de rendimiento en entornos controlados, su adopción efectiva en ingeniería de software requiere repensar tanto los modelos de evaluación como las prácticas de desarrollo. Por ello, se recomienda priorizar tres acciones concretas: avanzar hacia benchmarks más representativos que superen los ya saturados, incorporar métricas orientadas al impacto en procesos reales, más allá del pass@k, y fortalecer los mecanismos de validación, seguridad y trazabilidad en los entornos donde el código generado se integra. Solo bajo estas condiciones podrá materializarse una adopción efectiva y confiable de los LLMs en la ingeniería de software profesional.

Finalmente, este estudio contribuye al estado del arte al ofrecer una síntesis estructurada de técnicas, métricas y limitaciones, así como al identificar de manera explícita la brecha entre evaluación académica y aplicabilidad industrial, proporcionando una base para futuras investigaciones orientadas a cerrar esta distancia.

BIBLIOGRAFÍA

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. In Advances in Neural Information Processing Systems 33 (NeurIPS 2020). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). Evaluating large language models trained on code [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2107.03374

Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. J. (2023). GitHub Copilot AI pair programmer: Asset or liability? Journal of Systems and Software, 203, Article 111709. https://doi.org/10.1016/j.jss.2023.111709

Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y., Luo, F., Xiong, Y., & Liang, W. (2024). DeepSeek-Coder:

When the large language model meets programming – The rise of code intelligence [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2401.14196

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan,K. (2024). SWE-bench: Can language models resolve real-world GitHub issues? In Proceedings of the 12th International Conference on Learning Representations (ICLR 2024). https://arxiv.org/abs/2310.06770

Jin, K., Wang, C. Y., Pham, H. V., & Hemmati, H. (2024). Can ChatGPT support developers? An empirical evaluation of large language models for code generation. In Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024) (pp. 576–587). ACM. https://doi.org/10.1145/3643991.3644889

Kitchenham, B., & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering (EBSE Technical Report EBSE-2007-01). Keele University & Durham University. https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf

Liu, J., Xia, C. S., Wang, Y., & Zhang, L. (2023). Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2305.01210

Liu, C., Yang, C., Zhang, J., Luo, Z., Yao, J., & Gao, C. (2025). Measuring and mitigating hallucination in code generation with LLMs [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2501.04127

Mastropaolo, A., Cooper, N., Palacio, D. N., Scalabrino, S., Poshyvanyk, D., & Oliveto, R. (2023). On the robustness of code generation techniques: An empirical study on GitHub Copilot. ACM Transactions on Software Engineering and Methodology, 32(5), Article 114. https://doi.org/10.1145/3597207

Nascimento, N., Alencar, P., & Cowan, D. (2024). Self-adaptive LLM-based agents for software development. In Proceedings of the 46th International Conference on Software Engineering (ICSE 2024) (pp. 2200–2212). ACM. https://doi.org/10.1145/3639478.3643102

Olausson, T. X., Inala, J. P., Wang, C., Gao, J., & Solar-Lezama, A. (2024). Is self-repair a silver bullet for code generation? In Proceedings of the 12th International Conference on Learning Representations (ICLR 2024). https://doi.org/10.48550/arXiv.2306.09896

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., ... Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. BMJ, 372, Article n71. https://doi.org/10.1136/bmj.n71

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. In Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP 2022) (pp. 754–768). IEEE. https://doi.org/10.1109/SP46214.2022.9833571

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2302.06590

Raji, I. D., Bender, E. M., Paullada, A., Denton, E., & Hanna, A. (2021). AI and the everything in the whole wide world benchmark. In Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021) – Datasets and Benchmarks Track

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Fernandes, C. C., Grattafiori, A., Ünal, T., Boissinot, B., ... Synnaeve, G. (2023). Code Llama: Open foundation models for code [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2308.12950

Schäfer, M., Nadi, S., Eghbali, A., & Tip, F. (2024). An empirical evaluation of using large language models for automated unit test generation. IEEE Transactions on Software Engineering, 50(1), 85–105. https://doi.org/10.1109/TSE.2023.3334955

Shinn, N., Cassano, F., Gopalan, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In Advances in Neural Information Processing Systems 36 (NeurIPS 2023). https://arxiv.org/abs/2303.11366

Siddiq, M. L., & Da Silva Santos, J. C. (2024). A large-scale empirical study of code smell detection using code language models. ACM Transactions on Software Engineering and Methodology, 33(6), Article 145. https://doi.org/10.1145/3649596

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., ... Neubig, G. (2024). OpenHands: An open platform for AI software developers as generalist agents [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2407.16741

Wang, Y., Le, H., Gotmare, A. D., Bui, N. D. Q., Li, J., & Hoi, S. C. H. (2025). Towards effective integration of LLM-generated code in CI/CD pipelines. In Proceedings of the 47th International Conference on Software Engineering (ICSE 2025) (pp. 1–13). ACM.

Xia, C. S., Deng, Y., Dunn, S., & Zhang, L. (2024). Agentless: Demystifying LLM-based software engineering agents [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2407.01489

Zhang, K., Li, J., Li, G., Shi, X., & Jin, Z. (2024). CodeAgent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 13643–13658). Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.acl-long.737

Zheng, L., Chiang, W. L., Sheng, Y., Li, T., Zhuang, S., Wu, Z., ... Zhang, H. (2024). CodeBenchGen: Creating scalable execution-based code generation benchmarks [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2401.10020

Zheng, Q., Xia, X., Zou, X., Dong, Y., Wang, S., Xue, Y., Shen, Z., Wang, Z., Wang, H., Gu, Z., Zhang, Z., Zhu, J., Liang, Y., & He, K. (2023). CodeGeeX: A pre-trained model for code generation with multilingual benchmarking on HumanEval-X. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2023) (pp. 5673–5684). ACM. https://doi.org/10.1145/3580305.3599790

Zhuo, T. Y., Vu, M. C., Chim, J., Hu, H., Yu, W., Widyasari, R., ... Yao, J. (2024). BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2406.15877

Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., & Aftandilian, E. (2022). Productivity assessment of neural code completion. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (pp. 21–29). ACM. https://doi.org/10.1145/3520312.3534864