UNIVERSIDAD TÉCNICA DE MANABÍ
Fundada en 1952

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Estudio Comparativo de Herramientas de Inteligencia Artificial y su Incidencia en el Desarrollo Web: Un Enfoque Basado en Angular y Node.js

Comparative Study of Artificial Intelligence Tools and Their Impact on Web Development: An Approach Based on Angular and Node.js

Autores

* ***Dustin Adrian Cabrera Lavayen***
dcabrera4@utmachala.edu.ec

Ricardo Josue Cabrera Calderón
rcabrera4@utmachala.edu.ec

Joofre Antonio Honores Tapia
jhonores@utmachala.edu.ec

John Patricio Orellana Preciado
jporellana@utmachala.edu.ec

Universidad Técnica de Machala, Facultad de Ingeniería Civil, Machala, El Oro, Ecuador.
*Autor para correspondencia

Comó citar el artículo:
Cabrera Lavayen, D.A., Cabrera Calderón, R.J., Honores Tapia, J.A. & Orellana Preciado, J.P. 2025. Estudio Comparativo de Herramientas de Inteligencia Artificial y su Incidencia en el Desarrollo Web: Un Enfoque Basado en Angular y Node. js. *Informática y Sistemas, 9*(1), 30-40. https://doi.org/10.33936/isrtic.v9i1.7345

Enviado: 18/02/2025
Aceptado: 31/03/2025
Publicado: 04/04/2025

Resumen

Este artículo presenta un estudio comparativo de herramientas de inteligencia artificial (IA) para el desarrollo web, enfocándose en Angular y Node.js. Se analizaron ChatGPT, Gemini, GitHub Copilot y DeepSeek, evaluando su capacidad para generar código funcional mediante SonarQube. Se implementó un caso práctico basado en un sistema de facturación, donde se evaluó métricas clave como mantenibilidad, fiabilidad y seguridad. Los resultados indican que todas las herramientas a nivel de backend generan un código mantenible y fiable, aunque se lograron identificar vulnerabilidades de seguridad menores. La generación del frontend presentó más inconvenientes en términos de fiabilidad y mantenibilidad, identificando errores comunes como la declaración de variables sin reasignación y presencia de archivos vacíos. ChatGPT y DeepSeek destacaron en la usabilidad y en la resolución de errores, mientras que GitHub Copilot y Gemini mostraron limitaciones en la etapa del desarrollo. El estudio concluye que, si bien estas herramientas mejoran la productividad y reducen la carga de codificación manual, dependen en gran medida de la precisión y detalle de las instrucciones proporcionadas por el desarrollador y de la supervisión humana para garantizar calidad y seguridad del software.

Palabras clave: Inteligencia artificial; generación de código; Angular; Node.js

Abstract

This article presents a comparative study of artificial intelligence (AI) tools for web development, focusing on Angular and Node.js. ChatGPT, Gemini, GitHub Copilot, and DeepSeek were analyzed, evaluating their ability to generate functional code using SonarQube. A practical case study based on a billing system was implemented, assessing key metrics such as maintainability, reliability, and security. The results indicate that all backend tools generate maintainable and reliable code, although minor security vulnerabilities were identified. Frontend generation posed more challenges in terms of reliability and maintainability, with common issues such as variable declarations without reassignment and the presence of empty files. ChatGPT and DeepSeek excelled in usability and error resolution, while GitHub Copilot and Gemini showed limitations during the development stage. The study concludes that while these tools enhance productivity and reduce the manual coding workload, they heavily depend on the precision and detail of the instructions provided by the developer, as well as human supervision to ensure software quality and security.

Keywords: Artificial intelligence; code generation; Angular; Node.js; case study.

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

1. Introducción

La inteligencia artificial (IA) ha evolucionado notablemente desde sus inicios, posicionándose como una herramienta indispensable en la sociedad moderna; partiendo de sus primeras aplicaciones en sistemas expertos en el siglo XX, hasta los recientes avances en Machine Learning (ML) y Aprendizaje Profundo (Deep Learning, DL) (Abeliuk & Gutiérrez, 2021; Colther & Doussoulin, 2024). Este crecimiento ha sido posible gracias al aumento en la capacidad de procesos, la disponibilidad de grandes volúmenes de datos y la mejora en los algoritmos, lo que ha permitido que la IA no solo resuelva problemas específicos, sino que también se convierta en una ayuda clave para la automatización y optimización de procesos en diversos sectores del campo informático (France, 2024; Roman Gallardo et al., 2024; Tao et al., 2019). Además, según Usman Hadi et al. (2024) los modelos de lenguaje han sido fundamentales para superar las barreras tradicionales en la interacción humano-máquina, ofreciendo niveles sin precedentes de precisión y contextualización en tareas complejas.

En el ámbito tecnológico, los Modelos de Lenguaje de Gran Escala (LLM, por sus siglas en inglés, Large Language Models) han redefinido las capacidades de la IA al posibilitar la generación de texto, código y contenido visual de manera autónoma; herramientas como ChatGPT y Gemini se destacan como ejemplos sobresalientes de estas tecnologías, ya que han sido diseñadas para interactuar con los usuarios en tareas complejas como la programación, la creación de contenido y el análisis de datos (Kuhail et al., 2024; Muthumanikandan & Ram, 2024). En este contexto, DeepSeek emerge como un LLM de código de abierto que ha demostrado un alto rendimiento en tareas de razonamiento, pero con la ventaja de ser más eficiente en termino de costos computacionales (Mercer et al., 2025).

ChatGPT basado en la arquitectura Generative Pre-trained Transformer (GPT), ha demostrado una notable capacidad para comprender el lenguaje natural y generar respuestas coherentes y detalladas, mientras que Gemini se posiciona como un fuerte competidor en la generación de código y en el análisis avanzado de datos, ampliando significativamente el alcance de los LLM en aplicaciones prácticas (Cho et al., 2025; Muthumanikandan & Ram, 2024; Siam et al., 2024). Al igual que ChatGPT, DeepSeek ha demostrado un alto rendimiento en la generación de código con la capacidad de producir resúmenes de código de alta calidad (Afrin et al., 2025). La arquitectura de DeepSeek basada en Mixture-of-Experts (MoE) y Reinforcement Learning (RL), le ha permitido alcanzar resultados notables en tareas de matemáticas y codificación (DeepSeek-AI et al., 2024).

Por otra parte, GitHub Copilot ha revolucionado la experiencia de los desarrolladores al integrarse directamente en los entornos de desarrollo y ofrecer funcionalidades avanzadas como autoempleado inteligente, generación de fragmentos de código y sugerencias de mejora, lo que ha permitido acelerar los ciclos de desarrollo y reducir errores humanos incrementando la productividad en un 28% en tareas repetitivas y complejas, sin embargo, es importante tener en cuenta que la efectividad depende de la calidad de las indicaciones proporcionadas por el desarrollador y que una confianza excesiva en la herramienta puede generar código incorrecto o inseguro (France, 2024; Ng et al., 2024).

En este contexto, el impacto de la IA en el desarrollo de software ha trascendido su rol inicial como herramienta de automatización para convertirse en un componente importante en procesos como la detección de errores, la generación de pruebas y la gestión de proyectos, lo que ha permitido no solo reducir los tiempos de desarrollo, sino también mejorar la calidad de los productos finales (Hegde & G, 2024; Leung & Murphy, 2023; Roman Gallardo et al., 2024).

En el área del desarrollo web, la IA ha demostrado ser una herramienta clave para mejorar la escalabilidad y la optimización de aplicaciones, ya que tecnologías como los chatbots impulsados por LLM y herramientas de diseño asistido han permitido a los desarrolladores enfrentar retos complejos, como la gestión de grandes volúmenes de datos y la mejora de la experiencia del usuario; asimismo, estas tecnologías han facilitado la implementación de metodologías ágiles en entornos multicapas, donde la interacción eficiente entre el servidor y el cliente resulta esencial (Balsam & Mishra, 2025; Muthumanikandan & Ram, 2024; Savani, 2023). En este aspecto, frameworks modernos como Angular y Node.js han ocupado un papel importante en la construcción de aplicaciones web robustas y escalables.

Por una parte; Angular, con su enfoque basado en componentes, facilita la creación de aplicaciones estructuradas y mantenibles; mientras que, por otro lado, Node.js con su arquitectura orientada a eventos, permite manejar grandes volúmenes de solicitudes simultáneas con un alto rendimiento, garantizando así una experiencia de usuario más eficiente y fluida (Ollila et al., 2022; Rahikainen, 2021).

Además, investigaciones recientes han señalado que estos modelos no solo han mejorado tareas existentes, sino que también han abierto nuevas posibilidades en la comunicación y el procesamiento de datos a gran escala, así como en sectores como la educación y la industria, al proporcionar retroalimentación personalizada, aumentar la productividad y reducir significativamente los tiempos de desarrollo (Chandramouli et al., 2022; Mao & Li, 2024;

e-ISNN 2550-6730 | Vol. 9, Núm. 1 (30-40): Enero-Junio, 2025 | DOI: 10.33936/isrtic.v9i1.7345

Rahikainen, 2021; Usman Hadi et al., 2024). Sin embargo, esta integración enfrenta desafíos significativos relacionados con la confiabilidad del código generado, la seguridad del software y el manejo de sesgos en los modelos entrenados, lo que exige una supervisión adecuada para mitigar riesgos inherentes al uso de estas herramientas (Chandramouli et al., 2022; Tao et al., 2019).

En consecuencia, a partir de los trabajos relacionados, se plantea la hipótesis de que, el uso de herramientas de IA como ChatGPT, Gemini, GitHub Copilot y DeepSeek mejoran la calidad del código en el campo de seguridad, mantenibilidad y Fiabilidad, al tiempo que facilitan la escritura y optimización del código, reduciendo la memorización del desarrollador bajo las evaluaciones de SonarQube. Por esta razón resulta fundamental desarrollar fases sistemáticas que permitan evaluar de manera objetiva el impacto de estas herramientas en el desarrollo de software moderno.

Este artículo tiene como objetivo desarrollar un caso práctico con requerimientos específicos para Angular y Node.js a fin de que las herramientas de IA generen código para cumplir cada uno de los requisitos y así evaluar la calidad del código generado, utilizando SonarQube como base para realizar una evaluación estructurada. Este análisis considerara métricas clave que permitirán obtener una visión integral del desempeño de estas tecnologías, facilitando la identificación tanto de fortalezas como de limitaciones y proporcionando una base sólida para comprender su impacto y utilidad en el contexto del desarrollo moderno de software.

2. Materiales y Métodos

Se adoptó un enfoque experimental y comparativo, estructurado en cuatro fases que permitieron analizar las capacidades de herramientas seleccionadas en entornos controlados. Este diseño buscó proporcionar una visión integral del desempeño de estas tecnologías mediante métricas estandarizadas.

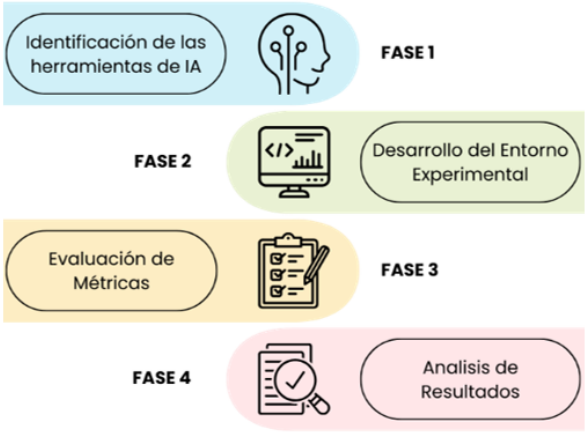


Figura 1. Fases Metodológicas.
Fuente: Los autores

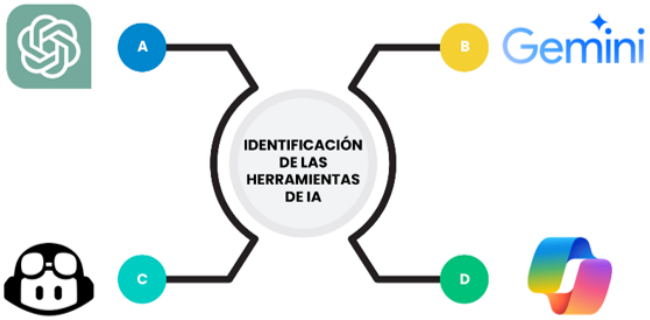


Figura 2. Herramientas de IA identificadas.
Fuente: Los autores

Fase 1: Identificación de las herramientas de IA

Se realizó una investigación sobre diversos estudios para seleccionar las herramientas más relevantes en el desarrollo web. Los criterios incluyeron su relevancia, popularidad y capacidad documentada para tareas específicas como generación de código y análisis avanzado de datos. Las herramientas seleccionadas fueron:

1. **ChatGPT 4o**, reconocido por generar texto y código de forma contextualizada y coherente (Muthumanikandan & Ram, 2024; Usman Hadi et al., 2024).

2. **Gemini**, especializado en generación de código avanzado y análisis de datos complejos (Cho et al., 2025; Siam et al., 2024).

3. **GitHub Copilot**, impulsado por OpenAI Codex, conocido por sus sugerencias de código en tiempo real en entornos de desarrollo integrados (IDEs) (Ng et al., 2024; Ollila et al., 2022).

4. **DeepSeek** por mostrar un rendimiento competitivo en comparación con otros LLM de código abierto, tanto en la generación de código y razonamiento matemático (DeepSeek-AI et al., 2024).

Fase 2: Desarrollo del entorno experimental

Figura 3. Herramientas para el entorno experimental.
Fuente: Los autores

El entorno experimental consistió en desarrollar un Sistema de Facturación para Pequeñas Empresas utilizando Angular 17 (standalone) en frontend y Node.js con Express en backend, utilizando MongoDB como base de datos. Las funcionalidades implementadas incluyeron:

Fase 3: Evaluación de métricas

En esta fase se evaluó la calidad del código generado por las herramientas de IA seleccionadas, usando SonarQube que es una herramienta ampliamente utilizada en la industria del software.

e-ISNN 2550-6730 | **Vol. 9, Núm. 1 (30-40): Enero-Junio, 2025** | **DOI: 10.33936/isrtic.v9i1.7345**

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

Tabla 1. Requerimientos del Caso Práctico.
Fuente: Los autores

Módulo	Frontend (Angular 17 - Standalone, Bootstrap 5)	Backend (Node.js + MongoDB)
Autenticación y Usuarios	- Formulario de inicio de sesión con validaciones. - Guards para proteger rutas según el rol. - Gestión de sesiones con JWT y LocalStorage.	- API REST para registro e inicio de sesión. - JWT para autenticación y middleware para proteger rutas. - Encriptación de contraseñas con bcrypt.
Gestión de Productos (CRUD)	- Formulario para crear, editar y eliminar productos. -	- API REST para CRUD de productos. - Express Validator para validaciones. - Stock reducido automáticamente al generar una factura.
Gestión de Clientes (CRUD)	- Formulario para registrar y editar clientes. -	- API REST para CRUD de clientes. - Validaciones de email y teléfono.
Gestión de Facturación	- Formulario de facturación con selección de cliente y productos. - Cálculo de subtotal, impuestos y total. - Descarga de factura en PDF.	- API REST para crear y listar facturas. - Stock reducido automáticamente al registrar la venta. - Generación de PDF en backend para descarga.
Reportes y Estadísticas	- Dashboard con gráficos de ventas y productos más vendidos. - Opción para exportar reportes en Excel.	- API REST para reportes de ventas y estadísticas. - Exportación de datos en Excel.

Tabla 2. Modelo Usuario.
Fuente: Los autores

Atributo	Tipo de Dato	Descripción
_id	ObjectId	Identificador único (generado por MongoDB).
nombre	String	Nombre completo del usuario.
email	String	Correo electrónico del usuario (único).
password	String	Contraseña encriptada con bcrypt.
rol	String	Rol del usuario (admin o usuario).

Tabla 3. Modelo Producto.
Fuente: Los autores

Atributo	Tipo de Dato	Descripción
_id	ObjectId	Identificador único (generado por MongoDB).
nombre	String	Nombre del producto.
descripcion	String	Descripción corta del producto.
precio	Number	Precio del producto.
stock	Number	Cantidad disponible en inventario.
createdAt	Date	Fecha de creación (generada automáticamente).

Tabla 4. Modelo Cliente.
Fuente: Los autores

Atributo	Tipo de Dato	Descripción
_id	ObjectId	Identificador único (generado por MongoDB).
nombre	String	Nombre completo del cliente.
email	String	Correo electrónico del cliente.
telefono	String	Número de teléfono del cliente.
direccion	String	Dirección del cliente.
createdAt	Date	Fecha de creación (generada automáticamente).

Según Software Qualities | SonarQube Docs (n.d.), un código limpio debe ser mantenible, confiable y seguro, ya que dichas cualidades garantizan su sostenibilidad a largo plazo y reducen los costos de mantenimiento y corrección de errores.

Para esta evaluación, el estudio se centra en tres métricas claves que SonarQube utiliza para determinar la calidad del software:

1. **Mantenibilidad**: Se refiere a la facilidad con la que se puede reparar, mejorar y comprender el código fuente. Un código mantenible permite una evolución más sencilla y económica del software.

2. **Fiabilidad**: Mide la capacidad del software de mantener su

e-ISNN 2550-6730 | Vol. 9, Núm. 1 (30-40): Enero-Junio, 2025 | DOI: 10.33936/isrtic.v9i1.7345

Tabla 5. Modelo Factura.
Fuente: Los autores

Atributo	Tipo de Dato	Descripción
_id	ObjectId	Identificador único (generado por MongoDB).
cliente	ObjectId	Referencia al cliente asociado a la factura.
productos	Array	Lista de productos vendidos.
productos[*]._id	ObjectId	ID del producto vendido.
productos[*].nombre	String	Nombre del producto.
productos[*].cantidad	Number	Cantidad vendida del producto.
productos[*].precio_unitario	Number	Precio unitario del producto.
subtotal	Number	Suma del costo de los productos antes de impuestos.
impuesto	Number	Monto del impuesto aplicado a la venta.
total	Number	Monto total de la factura.
createdAt	Date	Fecha de emisión de la factura (generada automáticamente).

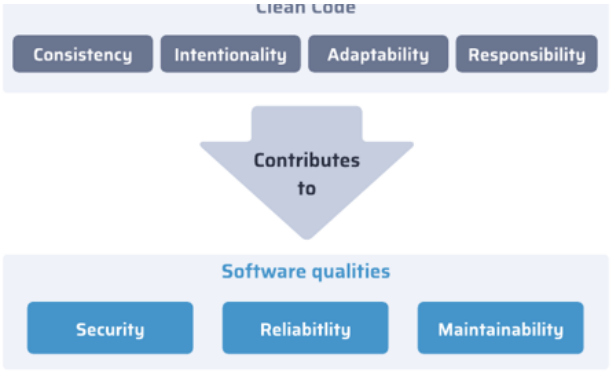


Figura 4. Cualidades del software.
Fuente: (Software Qualities | SonarQube Docs, n.d.)

nivel de rendimiento bajo condiciones establecidas durante un periodo de tiempo determinado. Un software confiable funciona sin fallos críticos que puedan interrumpir su operación.

3. **Seguridad**: Evalúa la protección del software contra accesos no autorizados, uso indebido o destrucción. Un código seguro previene vulnerabilidades que podrían ser explotadas, asegurando la integridad y confidencialidad de la aplicación.

La Figura 5 presenta el flujo de evaluación que SonarQube utiliza para determinar la calidad del código basado en el concepto de código limpio.

El proceso comienza con la evaluación de atributos de código limpio, los cuales se analizan mediante reglas de codificación diseñadas para un lenguaje especifico.

Cuando la regla de codificación identifica un problema se genera un problema de código que hereda los atributos del código limpio afectados. Posteriormente, el impacto de ese problema es evaluado en una o varias métricas de calidad del software (mantenibilidad, fiabilidad y seguridad).

Finalmente, a cada problema identificado se le asigna un nivel de severidad, indicando su impacto en la calidad general del software.

A continuación, las métricas específicas que se utilizaron para evaluar el código se presentan en las tablas correspondientes, adaptadas de la documentación oficial de SonarQube.

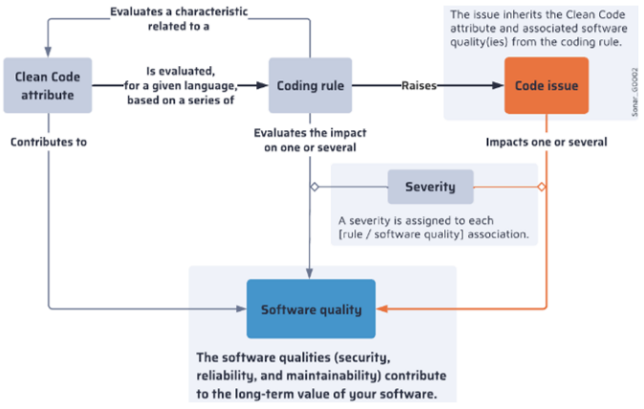


Figura 5. Proceso de análisis de código en SonarQube.
Fuente: (Clean-Code-Based Analysis | SonarQube Docs, n.d.)

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

Fase 4: Análisis de resultados

Los resultados obtenidos se basaron en los datos proporcionados por SonarQube.

A partir de estos datos se comparó el desempeño de ChatGPT, Gemini, GitHub Copilot y DeepSeek evaluando su capacidad de generar código funcional.

Tabla 6. Mantenibilidad.
Fuente: (Code Metrics & SonarQube, n.d.)

Métrica	Definición
Problemas	El número total de problemas que afectan la mantenibilidad (problemas de mantenibilidad).
Deuda técnica	Una medida del esfuerzo necesario para solucionar todos los problemas de mantenibilidad.
Ratio de deuda técnica	Relación entre el costo de desarrollo del software y el costo de reparación.
Calificación de mantenibilidad	La calificación relacionada con el valor del ratio de deuda técnica.

Tabla 7. Fiabilidad.
Fuente: (Code Metrics & SonarQube, n.d.)

Métrica	Definición
Problemas	El número total de problemas que afectan la confiabilidad (problemas de confiabilidad).
Calificación de confiabilidad	Calificación relacionada con la confiabilidad. La escala de calificación es la siguiente: A = 0 error B = al menos un error menor C = al menos un error mayor D = al menos un error crítico E = al menos un error bloqueador
Esfuerzo de remediación de confiabilidad	El esfuerzo necesario para solucionar todos los problemas de confiabilidad. El costo de remediación de un problema se basa en el esfuerzo (en minutos) asignado a la regla que generó el problema. Se asume una jornada de 8 horas cuando los valores se muestran en días.

Los valores obtenidos fueron organizados en tablas y gráficos, facilitando la interpretación y permitiendo una discusión detallada sobre las fortalezas y debilidades en relación con la generación de código y uso de estas herramientas de IA.

3. Resultados y Discusión

3.1. Análisis de la Evaluación de Métricas con SonarQube

Los resultados obtenidos en la evaluación de SonarQube en el backend reflejan que todas las herramientas de IA generar un código mantenible y confiable, sin embargo, se observa que la seguridad del código está comprometida, debido a 2 errores encontrados en la evaluación.

Tabla 8. Seguridad.
Fuente: (Code Metrics & SonarQube, n.d.)

Métrica	Definición
Problemas	El número total de problemas de seguridad (también llamados vulnerabilidades).
Calificación de seguridad	Calificación relacionada con la seguridad. La escala de calificación es la siguiente: A = 0 vulnerabilidad B = al menos una vulnerabilidad menor C = al menos una vulnerabilidad mayor D = al menos una vulnerabilidad crítica E = al menos una vulnerabilidad bloqueadora
Esfuerzo de remediación de seguridad	El esfuerzo necesario para solucionar todas las vulnerabilidades. El costo de remediación de un problema se basa en el esfuerzo (en minutos) asignado a la regla que generó el problema. Se asume una jornada de 8 horas cuando los valores se muestran en días.
Puntos críticos de seguridad	El número de puntos críticos de seguridad.
Puntos críticos de seguridad revisados	El porcentaje de puntos críticos de seguridad revisados en relación con el número total de puntos críticos de seguridad.
Calificación de revisión de seguridad	La calificación de revisión de seguridad es una letra basada en el porcentaje de puntos críticos de seguridad revisados. Se consideran revisados si están marcados como Reconocidos, Corregidos o Seguros. A = >= 80% B = >= 70% y <80% C = >= 50% y <70% D = >= 30% y <50% E = < 30%

Tabla 9. Ajustes de SonarQube.
Fuente: Los Autores.

Característica	Descripción
Versión	10.7
Configuración	Se uso la configuración global que viene por defecto y se selecciono el tipo de lenguaje que se indicaba una opción donde salían varios lenguajes entre ellos el TypeScript y JavaScript.
Extras	SonarQube detecta que se usa angular por lo cual evita analizar los archivos generados por Angular como las configuraciones y otros que sean necesarios para el uso del framework. Para el caso del backend se agrega que no analice la carpeta de los node modules, debido que son bibliotecas externas, que no fueron generadas por IA.

Estos problemas a simple vista pueden parecer inofensivos, de hecho, así lo dice SonarQube indicando que no son de prioridad alta, pero que no se deben pasar por alto. El primer error es debido al uso de la librería llamada “CORS” (Cross-Origin Resource Sharing) y la configuración que indica acceso desde cualquier origen al API, lo cual para la etapa del desarrollo es

e-ISNN 2550-6730 | Vol. 9, Núm. 1 (30-40): Enero-Junio, 2025 | DOI: 10.33936/isrtic.v9i1.7345

factible, pero, es muy peligroso en un entorno de producción. Y el segundo error indica que Express.js envía un encabezado en las respuestas HTTP que revelan la versión de la librería y esto

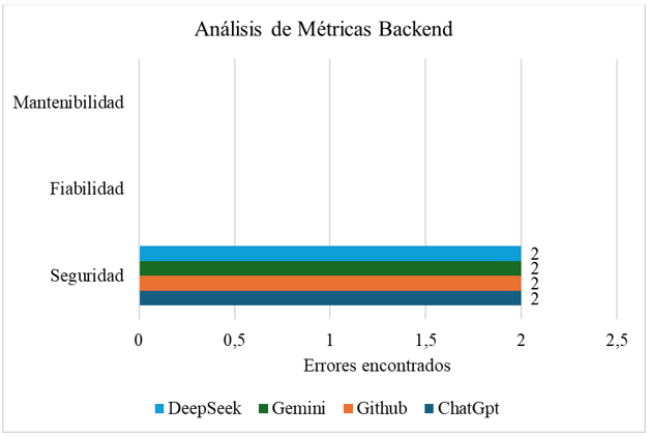


Figura 6. Evaluación de SonarQube en el backend.
Fuente: Los Autores.

puede ser un riesgo de seguridad, dado que, un atacante podría buscar las vulnerabilidades de esa versión y atacarlas.

La evaluación de SonarQube para el frontend revela una cantidad considerable de problemas generados por las herramientas de IA, como se detalla en la Figura 7.

Un aspecto positivo común es la solidez en la seguridad que demuestran todas las herramientas, ya que no presentan error alguno en la métrica de seguridad, lo cual indica una generación de código libre de vulnerabilidades.

No obstante, solo ChatGPT muestra una deficiencia en la confiablidad con una puntuación de 15 errores. Esto indica que, a pesar de su buena seguridad, el código generado podría tener inconsistencias que podrían afectar su estabilidad y funcionamiento, sin embargo, SonarQube nos da una clasificación de "A" en fiabilidad, entendiendo que, los problemas detectados

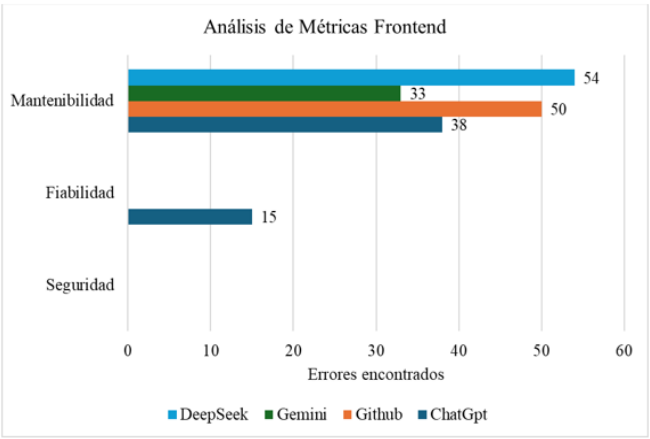


Figura 7. Evaluación de SonarQube del frontend.
Fuente: Los Autores.

no influyen de manera significativa en la calidad del código.

En cuestión de mantenibilidad se presentan resultados variables de errores, demostrando que ninguna herramienta alcanza un nivel óptimo en esta métrica, sugiriendo que generan un código complicado de entender y modificar en el futuro.

Entre los problemas expuestos por SonarQube: la declaración de variables que posteriormente no se les asigna un valor, archivos de CSS vacíos e importación de módulos que no se utilizan. Cabe recalcar que "A" es la clasificación obtenida en esta métrica para todas las herramientas de IA, lo que sugiere que, el código generado es mantenible a pesar de la cantidad de errores encontrados en la evaluación.

3.2. Análisis de la Severidad evaluada por SonarQube

Analizando la severidad de los problemas encontrados en cuanto a Fiabilidad en ChatGPT, los problemas que presenta son de severidad media, y todos relacionados con la falta de asociación entre etiquetas de formulario y sus controles correspondientes, este tipo de problema, aunque no es crítico para la funcionalidad, siempre se recomienda cumplir con las mejores prácticas de accesibilidad y usabilidad.

Asimismo, en los problemas encontrados en la mantenibilidad del código, se observa que la mayoría son de severidad media, lo que indica que, si bien existen oportunidades de mejora, el código

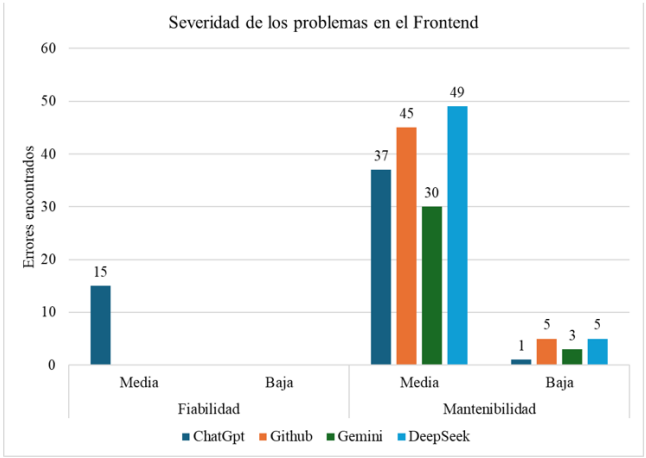


Figura 8. Severidad de los problemas encontrados.
Fuente: Los Autores.

no presenta problemas críticos que dificulten su mantenibilidad. Los problemas obtenidos en fiabilidad se generan dado que Angular maneja la asociación entre etiquetas y campos de formularios a través de directivas como "ngModel", por lo tanto, la IA al generar el código omite atributos que no son necesarios, sin embargo, SonarQube analiza usando reglas generales de accesibilidad basadas en HTML estándar.

Por esta razón el impacto en la funcionalidad es inexistente y obtiene una clasifican positiva por parte de SonarQube.

En cuanto a la cantidad de problemas de mantenibilidad presentados por las herramientas son: DeepSeek (54), GitHub

e-ISNN 2550-6730 | **Vol. 9, Núm. 1 (30-40): Enero-Junio, 2025** | **DOI: 10.33936/isrtic.v9i1.7345**

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

Copilot (50), ChatGPT (38) y Gemini (33). Es importante mencionar que; aunque DeepSeek presenta un mayor número de problemas, la diferencia con respecto a las otras IA no es muy significativa.

3.3. Problemas Comunes Detectados

Como se observa en la Tabla 10, los problemas de mantenibilidad más comunes detectados por SonarQube se relacionan con la declaración de variables, la organización de archivos e importaciones, y la presencia de código no utilizado.La frecuencia del problema "Member '...' is never reassigned, mark it as 'readonly'" sugiere que las IA tienden a generar código con variables o miembros de clase que no se modifican después de su declaración inicial; si bien esto no afecta la funcionalidad del código, marcar estas variables como "readonly" puede mejorar la calidad del código.

Este mensaje se refiere a las variables que almacenan instancias

Tabla 10. Problemas Comunes de Mantenibilidad.
Fuente: Los Autores.

Problema	Descripción
"Member '...' is never reassigned, mark it as 'readonly'"	Este problema es el más frecuente en todas las IA. Indica que hay variables o miembros de clase que se declaran, pero nunca se les reasigna un valor, por lo que podrían marcarse como "readonly" para mejorar la legibilidad y prevenir modificaciones accidentales.
"Unexpected empty source"	Este problema se refiere a archivos CSS vacíos, lo que puede indicar que se crearon archivos innecesarios o que falta código.
"Remove this unused import of..."	Este problema señala que hay importaciones de módulos o componentes que no se utilizan en el archivo, lo que puede afectar la organización y el rendimiento.

de servicios, las cuales, al ser clases, solo se utilizan para acceder a sus métodos sin reasignarles nuevos valores.

Respecto a la presencia de archivos CSS vacíos ("Unexpected empty source") se debe a que en este caso se usó Boostrap para el diseño y al usar sus clases, componentes y demás utilidades, se dejó vacíos varios archivos CSS que Angular generó con la creación de los componentes.

Por último, el problema de "Remove this unused import of..." señala que las IA a veces incluyen importaciones de módulos o componentes que no se utilizan en el archivo, sin embargo, esto puede suceder debido a que la IA en algunas ocasiones generó código que no ejecutaba y se tuve que volver a generar para que pueda ejecutar de manera correcta, entonces en esa nueva generación se quedaron algunas importaciones que pertenecían a la versión anterior.

3.4. Discusión

Los resultados obtenidos en este estudio respaldan parcialmente la hipótesis que se planteó en este artículo, con base en la seguridad se observó que las IA generan código con pocos problemas de seguridad, incluso ningún problema en el caso del frontend, lo que sugiere que las IA pueden ser herramientas para generar código seguro, pero sin descartar la necesidad de revisión manual para garantizar la seguridad de la aplicación.

En cuanto a la mantenibilidad, si bien la mayoría de los problemas fueron de severidad media o baja, la cantidad fue considerable, especialmente en el caso de DeepSeek y GitHub Copilot, esto indica que las IA aún tienen margen de mejora en la generación de código limpio y mantenible.

Respecto a la fiabilidad, solo ChatGPT presento problemas en el frontend, esto sugiere que la fiabilidad del código generado puede variar significativamente.

Todo esto revelando que las herramientas pueden generar un código de calidad a pesar de los problemas que puedan manifestarse, debido a que, si se presentan se podrían solucionar de una manera fácil sin afectar la funcionalidad del software.

Comparando con estudios anteriores se registraron similitudes a los trabajos realizados por Siam et al. (2024) y Afrin et al. (2025), quienes exponen que ChatGpt, Gemini y DeepSeek ayudan a mejorar la eficiencia y productividad del desarrollo de software, también generan código de calidad aceptable sin descartar la revisión humana para corrección de errores pequeños; lo cual afirma lo evidenciado en este estudio sobre el impacto positivo que tienen las herramientas de IA en la generación de código; considerando incluso los problemas registrados por SonarQube para el caso de Angular y Node.js.

Paralelamente Ng et al. (2024) señala que, GitHub Copilot genera código de calidad y reduce el tiempo de codificación, lo cual se asemeja con los resultados obtenidos en este estudio comparativo donde se reflejó la eficiencia de las herramientas de IA en la generación de código.

La influencia de estas herramientas en un entorno real, como en el desarrollo de un sistema que requiere varios meses de trabajo, es un aspecto clave a considerar.

Es fundamental analizar las diferencias significativas en su uso a lo largo del tiempo, según Siam et al. (2024), aunque Gemini es una IA poderosa en la generación de código, presenta una limitación importante al no recordar el código generado días anteriores, lo que dificulta la continuidad del desarrollo.

De manera similar GitHub Copilot olvida fragmentos de código casi de inmediato, lo que puede generar errores, redundancias o la necesidad de volver a generar partes del trabajo.

e-ISNN 2550-6730 | Vol. 9, Núm. 1 (30-40): Enero-Junio, 2025 | DOI: 10.33936/isrtic.v9i1.7345

DeepSeek enfrenta un problema relacionado, ya que impone un límite en la cantidad de mensajes por chat, obligando a reenviar el código previamente generado en nuevas conversaciones para mantener la coherencia del desarrollo.

En contraste, ChatGPT se posiciona como la opción más adecuada para proyectos de gran tamaño, ya que su capacidad de recordar el código generado durante varios días facilita la continuidad y eficiencia del proceso de desarrollo.

Las posibles causas de los problemas encontrados durante la generación de código pueden deberse a factores relacionados con el uso del prompt o el contenido de estos, si bien se usó el mismo para cada una de las herramientas, no todas lograron interpretarlo de la misma manera, por lo cual al querer generar una respuesta se mostraron errores para entregar lo solicitado a cada IA.

Esto tiene relación con el modelo que usa cada IA para generar las respuestas, por esa razón las indicaciones otorgadas no son relevantes, porque cada herramienta la interpreta de una manera distinta y realiza la entrega del código según su entrenamiento específico; se evidencia que esta sería la razón fundamental de que Gemini y GitHub Copilot mostraran errores al inicio del desarrollo.

4. Conclusiones

Este estudio evaluó la capacidad de distintas herramientas de IA para generar código funcional, evidenciando diferencias significativas en la calidad y fiabilidad de los resultados. En términos de seguridad, se identificó que se presentaron pocos problemas críticos, pero que no hubo ausencia de los mismo, lo que sugiere que pueden ser utilizadas como apoyo en la generación de código seguro, aunque siempre con una supervisión y validación humana.

La experiencia práctica con estas herramientas evidenció diferencias claves en su desempeño. ChatGPT demostró ser una de las más eficientes en la generación de backend funcional con un prompt detallado con los requerimientos del caso práctico, el flujo de uso y los modelos de la base de datos.

En contraste, Gemini presentó más errores en su primera respuesta y requirió mayor interacción y desgloses más detallados de los pasos a seguir. GitHub Copilot, por su parte, generó un backend bastante funcional con pocos errores, aunque mostro una limitación al olvidar código generado previamente, especialmente cuando el trabajo se retomaba al día siguiente.

DeepSeek también generó un backend con pocas fallas, pero su uso estuvo afectado por un límite de mensajes por chat, lo que obligó a reenviar información que ya había generado previamente para mantener la continuidad del desarrollo.

Un punto clave es que la generación del frontend presentó bastantes dificultades recurrentes en todas las IA, mencionando especialmente que en GitHub Copilot fue propenso a confundir el trabajo con módulos y no de manera standalone como lo requería el caso práctico planteado.

Los resultados obtenidos en este estudio comparativo establecen que las herramientas de IA dependen en gran medida de la precisión y profundidad de las indicaciones proporcionadas por el desarrollador, un prompt detallado y estructurado mejora significativamente la funcionalidad del código generado, reduciendo correcciones posteriores.

Mediante la evaluación con SonarQube se demostró que la generación de código es de calidad, a pesar de todos los errores descritos en esta investigación, todas las herramientas son capaces de generar un código claro y funcional, sin presentar cuestiones de gran impacto en la ejecución final.

ChatGPT y DeepSeek fueron las herramientas que destacaron en cuanto a la usabilidad e interacción humano-maquina durante las etapas del desarrollo, mostrando un comportamiento amigable y ordenado en las respuestas generadas, asimismo, siendo capaces de resolver cada uno de los errores que se emitían cuando una parte del código generado no funcionaba de la manera correcta.

Contribución de los autores

Dustin Adrian Cabrera Lavayen: Conceptualización, Investigación, Metodología, Software. **Ricardo Josue Cabrera Calderón:** Conceptualización, Investigación, Redacción – borrador original, Recursos. **Joofre Antonio Honores Tapia:** Supervisión, Administración del proyecto. **John Patricio Orellana Preciado:** Validación, Redacción – revisión y edición del artículo.

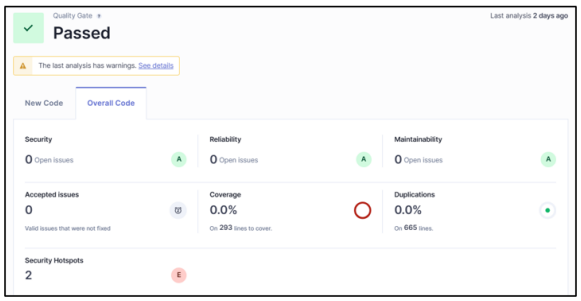
Conflictos de interés

Los autores declaran no tener ningún conflicto de interés.

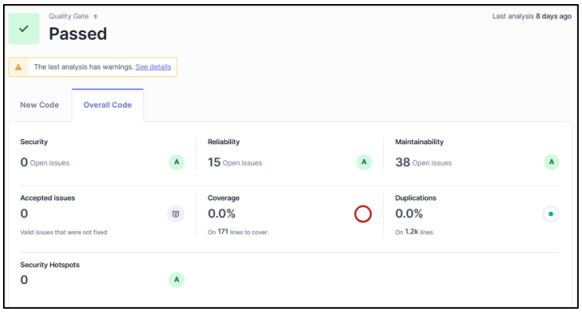
Anexos

Anexo A: Evaluación de ChatGPT mediante SonarQube.

A.1. Backend.



A.1. Backend.



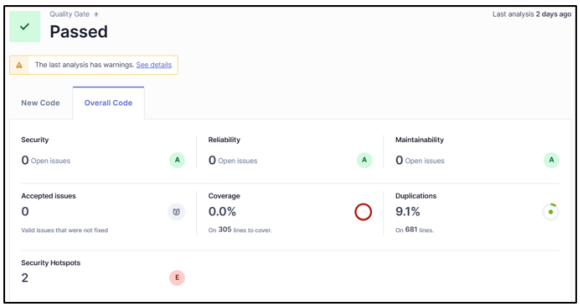
e-ISNN 2550-6730 | **Vol. 9, Núm. 1 (30-40): Enero-Junio, 2025** | **DOI: 10.33936/isrtic.v9i1.7345**

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

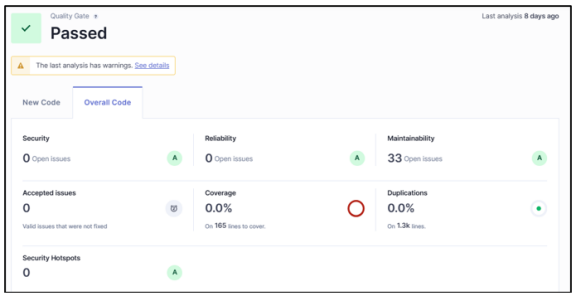
Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

Anexo B: Evaluación de Gemini mediante SonarQube.

B.1. Backend.

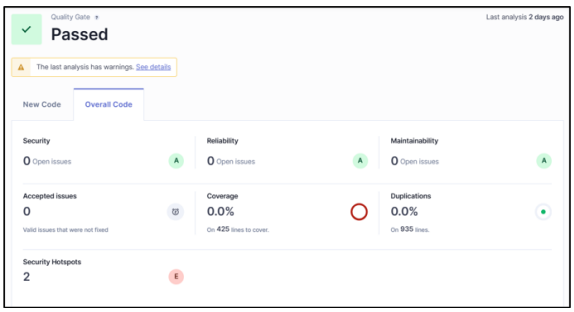


B.2. Frontend.

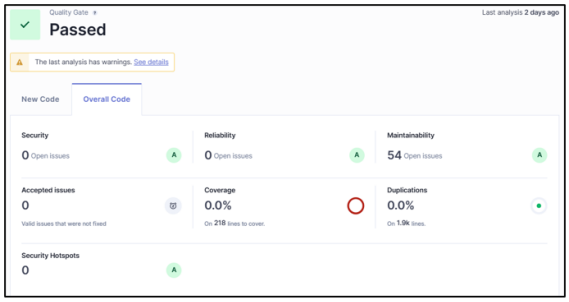


Anexo C: Evaluación de DeepSeek mediante SonarQube.

C.1. Backend.

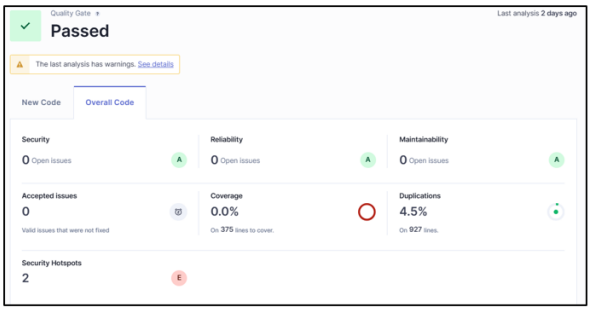


C.2. Frontend.

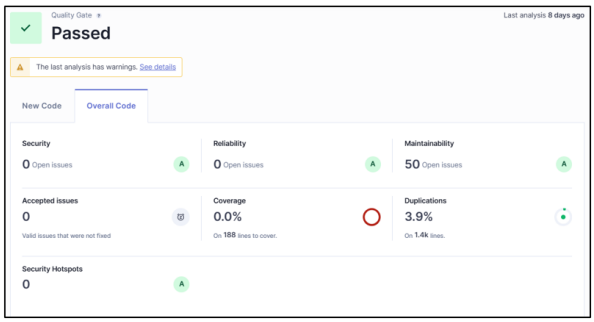


Anexo D: Evaluación de GitHub Copilot mediante SonarQube.

D.1. Backend.



D.2. Frontend.



Referencias bibliográficas

Abeliuk, A., & Gutiérrez, C. (2021). Historia y evoluación de la inteligencia artificial. *Revista Bits de Ciencia, 21*, 14–21. https://doi.org/10.71904/BITS.VI21.2767

Afrin, S., Call, J., Nguyen, K.-N., Chaparro, O., & Mastropaolo, A. (2025). Resource-efficient & effective code summarization. arXiv. https://arxiv.org/abs/2502.03617v1

Balsam, S., & Mishra, D. (2025). Web application testing—Challenges and opportunities. *Journal of Systems and Software, 219*, 112186. https://doi.org/10.1016/J.JSS.2024.112186

Chandramouli, P., Codabux, Z., & Vidoni, M. (2022). analyzeR: A SonarQube plugin for analyzing object-oriented R Packages. *SoftwareX, 19*, 101113. https://doi.org/10.1016/J.SOFTX.2022.101113

Cho, K., Park, Y., Kim, J., Kim, B., & Jeong, D. (2025). Conversational AI forensics: A case study on ChatGPT,

e-ISNN 2550-6730 | Vol. 9, Núm. 1 (30-40): Enero-Junio, 2025 | DOI: 10.33936/isrtic.v9i1.7345

Gemini, Copilot, and Claude. *Forensic Science International: Digital Investigation, 52*, 301855. https://doi.org/10.1016/J.FSIDI.2024.301855

Clean-code-based analysis | SonarQube Docs. (n.d.). Retrieved February 10, 2025, from https://docs.sonarsource.com/sonarqube-server/10.7/core-concepts/clean-code/code-analysis/

Code metrics & SonarQube. (n.d.). Retrieved February 10, 2025, from https://docs.sonarsource.com/sonarqube-server/10.7/user-guide/code-metrics/metrics-definition/

Colther, C., & Doussoulin, J. P. (2024). Artificial intelligence: Driving force in the evolution of human knowledge. *Journal of Innovation & Knowledge, 9*(4), 100625. https://doi.org/10.1016/J.JIK.2024.100625

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., ... Pan, Z. (2024). DeepSeek-V3 Technical Report. arXiv. https://arxiv.org/abs/2412.19437v1

France, S. L. (2024). Navigating software development in the ChatGPT and GitHub Copilot era. *Business Horizons, 67*(5), 649–661. https://doi.org/10.1016/J.BUSHOR.2024.05.009

Hegde, N. N., & G, M. (2024). Prototype pollution detection for Node.Js applications: A review. *Journal of Cyber Security, Privacy Issues and Challenges, 3*(2), 23–32. https://matjournals.net/engineering/index.php/JCSPIC/article/view/682

Kuhail, M. A., Mathew, S. S., Khalil, A., Berengueres, J., & Shah, S. J. H. (2024). "Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools. *Science of Computer Programming, 235*, 103111. https://doi.org/10.1016/J.SCICO.2024.103111

Leung, M., & Murphy, G. (2023). On automated assistants for software development: The role of LLMs. Proceedings - 2023 38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, 1737–1741. https://doi.org/10.1109/ASE56229.2023.00035

Mao, Q., & Li, Y. (2024). Blockchain evolution, artificial intelligence and ferrous metal trade. *Resources Policy, 98*, 105369. https://doi.org/10.1016/J.RESOURPOL.2024.105369

Mercer, S., Spillard, S., & Martin, D. P. (2025). Brief analysis of DeepSeek R1 and its implications for generative AI. arXiv. https://arxiv.org/abs/2502.02523v3

Muthumanikandan, V., & Ram, S. (2024). Visistant: A conversational chatbot for natural language to visualizations with Gemini large language models. IEEE Access. https://doi.org/10.1109/ACCESS.2024.346554199

Ng, K. K., Fauzi, L., Leow, L., & Ng, J. (2024). Harnessing the potential of Gen-AI coding assistants in public sector software development. arXiv. https://arxiv.org/abs/2409.17434v1

Ollila, R., Mäkitalo, N., & Mikkonen, T. (2022). Modern web frameworks: A comparison of rendering performance. *Journal of Web Engineering, 21*(3), 789–813. https://doi.org/10.13052/JWE1540-9589.21311

Rahikainen, M. (2021). Web application in Angular and Ionic. https://www.theseus.fi/handle/10024/702876

Roman Gallardo, A., Roman Herrera Morales, J., Sandoval Carrillo, S., & Alvarez Cardenas, O. (2024). Uso de herramientas de IA generativa para la automatización del desarrollo de software: Un caso de estudio. Ingeniantes, 3. https://citt.itsm.edu.mx/ingeniantes/articulos/ingeniantes11no1vol3/19.pdf

Savani, N. (2023). The future of web development: An in-depth analysis of micro-frontend approaches. International Journal of Computer Trends and Technology. https://www.researchgate.net/publication/376477757

Siam, M. K., Gu, H., & Cheng, J. Q. (2024). Programming with AI: Evaluating ChatGPT, Gemini, AlphaCode, and GitHub Copilot for programmers. arXiv. https://arxiv.org/abs/2411.09224

Software qualities | SonarQube Docs. (n.d.). Retrieved February 10, 2025, from https://docs.sonarsource.com/sonarqube-server/10.7/core-concepts/clean-code/software-qualities/

Tao, C., Gao, J., & Wang, T. (2019). Testing and quality validation for AI software—Perspectives, issues, and practices. *IEEE Access, 7*, 120164–120175. https://doi.org/10.1109/ACCESS.2019.2937107

Usman Hadi, M., Al Tashi, Q., Qureshi, R., Shah, A., Muneer, A., Irfan, M., Zafar, A., Bilal Shaikh, M., Akhtar, N., Zohaib Hassan, S., Shoman, M., Wu, J., Mirjalili, S., Shah, M., Al-Tashi, Q., & Ali Al-Garadi, M. (2024). Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects. Authorea Preprints. https://doi.org/10.36227/TECHRXIV.23589741.V740