Ciencia Latina Revista Científica Multidisciplinar, Ciudad de México, México.
ISSN 2707-2207 / ISSN 2707-2215 (en línea), Noviembre-Diciembre 2025,
Volumen 9, Número 6.

https://doi.org/10.37811/cl_rcm.v9i6

CHAT DE ASISTENCIA EMBEBIDO EN UN IDE POTENCIADO POR IA

AI-POWERED EMBEDDED ASSISTANCE CHAT FOR IDE

Francisco Fabián Tobías Macías
Tecnológico Nacional de México/Instituto Tecnológico de Piedras Negras, México

Gustavo Emilio Rojo Velázquez
Tecnológico Nacional de México/Instituto Tecnológico de Piedras Negras, México

Carlos Hernández Santos
Tecnológico Nacional de México/Instituto Tecnológico de Nuevo León, México

Roxana García Andrade
Tecnológico Nacional de México/Instituto Tecnológico de Nuevo León, México

Yasser Alberto Davizon Castillo
Tecnológico Nacional de México/Instituto Tecnológico de los Mochis, México

DOI: https://doi.org/10.37811/cl_rcm.v9i6.21585

Chat de Asistencia Embebido en un IDE Potenciado por IA

Francisco Fabián Tobías Macías[1]
francisco.tm@piedrasnegras.tecnm.mx
https://orcid.org/0000-0002-2639-5341
Tecnológico Nacional de México
Instituto Tecnológico de Piedras Negras
México

Gustavo Emilio Rojo Velázquez
gustavo.rv@piedrasnegras.tecnm.mx
https://orcid.org/0000-0002-7792-1436
Tecnológico Nacional de México
Instituto Tecnológico de Piedras Negras
México

Carlos Hernández Santos
carlos.hernandez@itnl.edu.mx
https://orcid.org/0000-0003-1751-1096
Tecnológico Nacional de México
Instituto Tecnológico de Nuevo León
México

Roxana García Andrade
roxana.ga@nuevoleon.tecnm.mx
https://orcid.org/0000-0003-2819-6482
Tecnológico Nacional de México
Instituto Tecnológico de Nuevo León
México

Jasser Alberto Davizon Castillo
yasser.davizon@hotmail.com
https://orcid.org/0000-0003-3023-947X
Tecnológico Nacional de México
Instituto Tecnológico de los Mochis
México

RESUMEN

El desarrollo de software contemporáneo es una actividad de alta demanda cognitiva, caracterizada por interrupciones constantes que afectan la productividad del programador. La práctica habitual de recurrir a recursos externos para la resolución de errores provoca frecuentes cambios de contexto, generando sobrecarga mental y disminuyendo la eficiencia. Este proyecto de investigación tiene como objetivo general analizar, desde un enfoque teórico, cómo la implementación de un chat de asistencia inteligente embebido en un Entorno de Desarrollo Integrado (IDE) puede contribuir a la reducción de errores, la mejora de la productividad y la promoción de buenas prácticas en la programación. La metodología empleada se basa en un análisis documental cualitativo, una revisión bibliográfica de literatura reciente y una evaluación conceptual de la arquitectura, funcionamiento e impacto de dicha herramienta. El análisis concluye que un asistente integrado tiene un alto potencial para mitigar la carga cognitiva al reducir la conmutación de contexto y actuar como un sistema experto que asiste en la depuración y refactorización. Se identifica un impacto positivo significativo tanto en el ámbito profesional, al optimizar el flujo de trabajo, como en el educativo, al funcionar como un tutor personalizado. Sin embargo, se discuten los riesgos inherentes, principalmente la dependencia tecnológica, la pérdida de habilidades técnicas y los desafíos éticos relacionados con la privacidad y la responsabilidad del código.

[1] Autor principal
Correspondencia: francisco.tm@piedrasnegras.tecnm.mx

***Palabras clave*:** asistente de código IA, ergonomía cognitiva, IDE, productividad del desarrollador; context switching, buenas prácticas de programación

AI-Powered Embedded Assistance Chat For IDE

ABSTRACT

Contemporary software development is a high-cognitive-demand activity characterized by constant interruptions that impact programmer productivity. The common practice of relying on external resources for error resolution necessitates frequent context switching, leading to mental overload and decreased efficiency. This research aims to theoretically analyze how implementing an intelligent chat assistant embedded within an Integrated Development Environment (IDE) can contribute to error reduction, productivity enhancement, and the promotion of programming best practices. The methodology employs a qualitative documentary analysis, a bibliographic review of recent literature, and a conceptual evaluation of the tool's architecture, functionality, and impact. The analysis concludes that an integrated assistant holds significant potential to mitigate cognitive load by minimizing context switching and acting as an expert system for debugging and refactoring. A significant positive impact is identified in both the professional realm, by optimizing workflows, and the educational sector, acting as a personalized tutor. However, inherent risks are discussed, primarily technological dependency, technical skill degradation, and ethical challenges regarding privacy and code accountability.

***Keywords*:** AI code assistant, cognitive ergonomics, IDE, developer productivity, context switching, programming best practices

Artículo recibido 15 noviembre 2025
Aceptado para publicación: 15 diciembre 2025

INTRODUCCION

El desarrollo de software contemporáneo es una actividad compleja que exige atención constante, comprensión de estructuras lógicas y manejo de dependencias. Los programadores enfrentan errores sintácticos y fallas de compilación que interrumpen su flujo de trabajo. Actualmente, la práctica habitual de recurrir a recursos externos (foros, documentación) provoca frecuentes cambios de contexto (context switching), lo que incrementa el tiempo de búsqueda y disminuye la eficiencia.

Impacto de la Inteligencia Artificial en la productividad

El uso de la IA ha revolucionado el abordaje de tareas complejas. Los modelos de lenguaje basados en arquitecturas Transformer han demostrado capacidades notables para generar código coherente (Ahmad et al., 2021). Investigaciones recientes evidencian que las interrupciones constantes merman el rendimiento (Alhoshan & Wang, 2022). En este sentido, los asistentes embebidos ofrecen una ventaja sustancial al mantener al usuario dentro del mismo entorno, optimizando la carga cognitiva. Estudios empíricos, como los de Kalliamvakou et al. (2023), identificaron que estos asistentes pueden incrementar la velocidad de desarrollo entre un 20% y un 40%.

Desafíos y limitaciones de la tecnología

A pesar de las ventajas, existen desafíos significativos. Una preocupación primordial es la seguridad del código; Perry et al. (2023) demostraron que los desarrolladores que usan IA tienden a escribir código menos seguro y confían más en soluciones que pueden contener vulnerabilidades. Asimismo, existe el riesgo de degradación de habilidades. Prather et al. (2023) advierten sobre la alteración de procesos metacognitivos en estudiantes, mientras que Vaithilingam et al. (2022) notaron que frecuentemente se invierte más tiempo depurando código generado por IA que escribiéndolo desde cero. Finalmente, aspectos éticos como la privacidad y las licencias de código siguen siendo barreras críticas (Sandoval et al., 2022; Bender et al., 2021).

Ante este panorama, el objetivo general de esta investigación es analizar teóricamente el impacto de la implementación de un chat de asistencia inteligente embebido en un IDE sobre la carga cognitiva y la eficiencia del flujo de trabajo. Se busca fundamentar la relación entre ergonomía cognitiva y productividad, describir las capacidades arquitectónicas de los asistentes y evaluar críticamente sus riesgos éticos y técnicos.

DESARROLLO

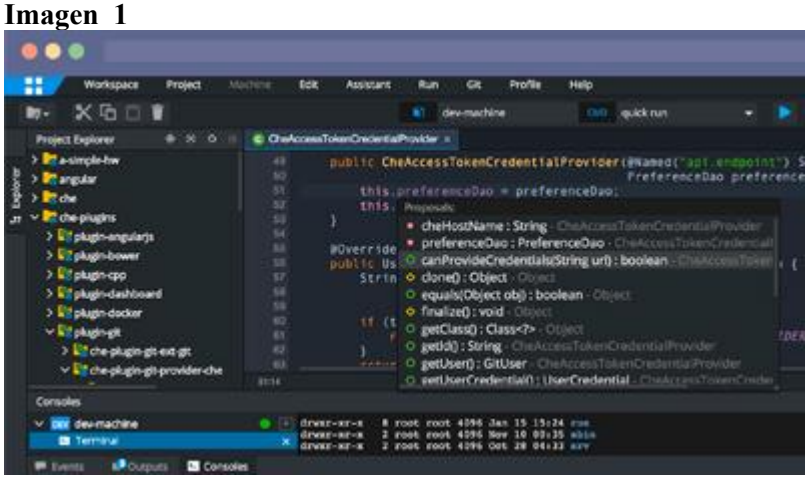
El desarrollo de software es una actividad compleja que exige atención constante, comprensión de estructuras lógicas, manejo de dependencias entre módulos y aplicación de buenas prácticas de programación. Los programadores enfrentan errores sintácticos, fallas de compilación y problemas de ejecución que interrumpen su flujo de trabajo y generan sobrecarga cognitiva, afectando su productividad y concentración. Actualmente, los desarrolladores suelen recurrir a recursos externos, como foros en línea o documentación, para resolver estos problemas. Esta práctica provoca cambios de contexto frecuentes , incrementa el tiempo de búsqueda de soluciones y disminuye la eficiencia en el proceso de desarrollo. Aunque los IDEs modernos ofrecen funciones de autocompletado y corrección básica de errores, no brindan explicaciones contextuales ni soluciones adaptadas a proyectos complejos. La motivación de esta investigación es explorar cómo la integración de un chat de asistencia inteligente dentro de un IDE puede mejorar la resolución de errores, optimizar la productividad y fomentar buenas prácticas de programación. La relevancia radica en que, a pesar del auge de asistentes como GitHub Copilot o ChatGPT, existen pocas investigaciones teóricas que analizan su impacto cognitivo y educativo en el entorno de desarrollo, especialmente en literatura en español. Para abordar este problema, la investigación seguirá un enfoque teórico , basado en:

- Revisión bibliográfica de literatura académica y artículos recientes sobre asistentes inteligentes en programación.
- Análisis conceptual del funcionamiento de un chat embebido en IDEs.
- Evaluación teórica de los beneficios, limitaciones y posibles impactos en productividad y aprendizaje de los desarrolladores.

En síntesis, el problema central que guía esta investigación es :

¿Cómo podría un chat de asistencia embebido en un IDE contribuir a mejorar la resolución de errores, la productividad y la aplicación de buenas prácticas en el desarrollo de software desde un enfoque teórico?

Imagen 1



Ventajas: Impacto de la inteligencia artificial en la productividad del desarrollo

El uso de la inteligencia artificial en el desarrollo de software ha revolucionado la forma en que los programadores abordan tareas de alta complejidad, tales como la detección de errores, la generación de código y la refactorización. Los modelos de lenguaje basados en arquitecturas Transformer, entrenados con grandes volúmenes de código fuente, han demostrado capacidades notables para comprender y generar fragmentos coherentes y funcionales (Ahmad et al., 2021).

Paralelamente, la productividad del programador se ha consolidado como un tema de interés crítico en la ingeniería de software. Investigaciones recientes evidencian que los cambios de contexto, la búsqueda manual de información y las interrupciones constantes merman significativamente el rendimiento y la concentración del desarrollador (Alhoshan & Wang, 2022). En este contexto, los asistentes embebidos en los IDE ofrecen una ventaja sustancial al mantener al usuario dentro del mismo entorno de trabajo, optimizando así la carga cognitiva.

Asimismo, la evidencia empírica respalda una correlación positiva entre el uso de IA y la eficiencia. Kalliamvakou et al. (2023) identificaron que los asistentes inteligentes pueden incrementar la velocidad de desarrollo entre un 20 % y un 40 %, variable según la complejidad del proyecto y la experticia del usuario. No obstante, advierten sobre la necesidad imperativa de la supervisión humana para mitigar riesgos derivados de recomendaciones imprecisas.

En conclusión, esta investigación se fundamenta en la intersección entre inteligencia artificial, productividad y ergonomía cognitiva, demostrando que la integración de un asistente inteligente en un

IDE no solo optimiza la eficiencia operativa, sino que también potencia el aprendizaje continuo del programador.

Desventajas: Desafíos y Limitaciones

A pesar de las ventajas operativas, la integración de asistentes basados en Inteligencia Artificial en el flujo de trabajo presenta desafíos significativos. Una de las preocupaciones primordiales es la fiabilidad y seguridad del código generado. Estudios recientes, como el de Perry et al. (2023), demostraron que los desarrolladores que utilizan asistentes de IA tienden a escribir código menos seguro en comparación con aquellos que no los usan, aunque paradójicamente confían más en la seguridad de sus soluciones. Dado que los modelos de lenguaje (LLMs) son propensos a "alucinaciones", pueden sugerir librerías inexistentes o patrones vulnerables, obligando al desarrollador a un escrutinio constante.

Por otro lado, existe una preocupación académica sobre la dependencia tecnológica y el impacto en el aprendizaje. Prather et al. (2023) advierten sobre el riesgo de alterar los procesos metacognitivos en estudiantes y desarrolladores noveles; el uso excesivo de la generación automática puede atrofiar la capacidad de resolución de problemas (problem-solving skills) y la comprensión profunda de la lógica del código. Si el usuario cae en el "sesgo de automatización", aceptando las sugerencias sin crítica, se compromete la calidad del software a largo plazo, tal como observaron Vaithilingam et al. (2022), quienes notaron que los programadores frecuentemente invierten más tiempo depurando código generado por IA que escribiéndolo desde cero debido a errores sutiles.

Finalmente, los aspectos éticos y legales representan una barrera crítica. La privacidad de los datos es un punto de fricción, ya que el envío de fragmentos de código propietario a la nube plantea riesgos de fuga de propiedad intelectual. Además, existen debates sobre la licencia del código sugerido, dado que estos modelos se entrenan con repositorios públicos, lo que podría derivar en infracciones de derechos de autor no intencionadas.

Objetivo General

- Analizar teóricamente el impacto de la implementación de un asistente de chat inteligente embebido en un Entorno de Desarrollo Integrado (IDE) sobre la carga cognitiva, la eficiencia del flujo de trabajo y la calidad del código en el desarrollo de software contemporáneo.

Objetivos Específicos

1. Fundamentar la relación entre la ergonomía cognitiva, la conmutación de contexto (context switching) y la productividad del programador, para comprender las limitaciones del flujo de trabajo tradicional sin asistencia integrada.
2. Describir las capacidades funcionales y arquitectónicas de los asistentes basados en Inteligencia Artificial (LLMs) dentro de un IDE, enfocándose en sus roles de depuración, refactorización y generación de código.
3. Contrastar las ventajas operativas de la asistencia embebida frente al uso de recursos externos (navegadores, foros), evaluando la reducción de interrupciones y la optimización del tiempo de desarrollo.
4. Evaluar críticamente los desafíos y riesgos inherentes a esta tecnología, tales como la fiabilidad de las respuestas (alucinaciones), la dependencia tecnológica (degradación de habilidades) y las implicaciones éticas de privacidad y seguridad.

Objeto de Estudio

La influencia de los asistentes conversacionales basados en Inteligencia Artificial embebidos en el IDE sobre la ergonomía cognitiva y el flujo de trabajo del desarrollo de software.

Desglose Metodológico

Para que tengas claridad total si te preguntan (o para tu propia guía), aquí está desglosado:

Unidad de Análisis: Los asistentes de código inteligentes (basados en LLMs como GPT/Copilot) integrados en el entorno de desarrollo.

Variable Independiente (Causa): La integración de la asistencia en el IDE (eliminación de barreras externas).

Variable Dependiente (Efecto): La carga cognitiva (cambio de contexto) y la productividad/calidad del código.

Campo de Acción

El análisis teórico de la optimización de procesos de depuración, refactorización y escritura de código mediante la reducción de la conmutación de contexto (context switching).

METODOLOGÍA

El presente proyecto se fundamenta en una metodología de investigación cualitativa, con un alcance descriptivo y analítico. Dado que el objetivo general es analizar, desde un enfoque teórico, cómo la implementación de un chat de asistencia puede contribuir a la mejora del desarrollo de software, la investigación se basa en el análisis documental y la síntesis conceptual.

El proceso metodológico se estructuró en las siguientes fases:

Revisión Bibliográfica.

Se llevó a cabo una revisión de la literatura reciente, acotada principalmente al periodo 2020-2025, para identificar los hallazgos más actuales sobre asistentes inteligentes, ergonomía cognitiva en programación y productividad del desarrollador. Se consultaron bases de datos académicas y repositorios como IEEE Xplore, ACM Digital Library, Scopus y arXiv, utilizando términos clave como "AI code assistant", "context switching software development", "ergonomía cognitiva programación" e "intelligent IDE".

Análisis Conceptual.

Esta fase consistió en la descomposición y síntesis de la información recopilada. Se definieron los conceptos fundamentales que sustentan el problema, como "Ergonomía Cognitiva" y "Buenas Prácticas", y se estructuró el "Marco Teórico" que da contexto a la investigación. El objetivo de esta fase fue describir el funcionamiento conceptual del asistente y su arquitectura técnica.

Cuestionario

1. Cuando encuentras un error y debes buscar la solución en un navegador externo (Google/StackOverflow), ¿sientes que pierdes la concentración o el "hilo" de la lógica? a) Sí, frecuentemente b) A veces c) No, casi nunca
2. ¿Con qué frecuencia recurres a recursos externos durante una sesión de programación típica? a) Más de 10 veces b) Entre 5 y 10 veces c) Menos de 5 veces
3. ¿Consideras que integrar un asistente de chat DIRECTAMENTE en tu IDE (sin tener que abrir el navegador) mejoraría tu velocidad y productividad? a) Totalmente de acuerdo b) De acuerdo c) Indiferente

4. Al usar IA para aprender, ¿prefieres que el asistente te explique la causa del error (Modo Tutor) o solo te dé el código corregido? a) Prefiero la explicación (Modo Tutor) b) Prefiero solo el código (Solución Rápida)
5. ¿Has detectado "alucinaciones" (código inventado, librerías que no existen o respuestas incorrectas) al usar asistentes de IA? a) Sí, varias veces b) Pocas veces c) Nunca
6. ¿Te preocupa volverte dependiente de la IA y perder tu habilidad para resolver problemas manualmente? a) Sí b) No

Figura 1. Frecuencia de interrupción para buscar documentación externa

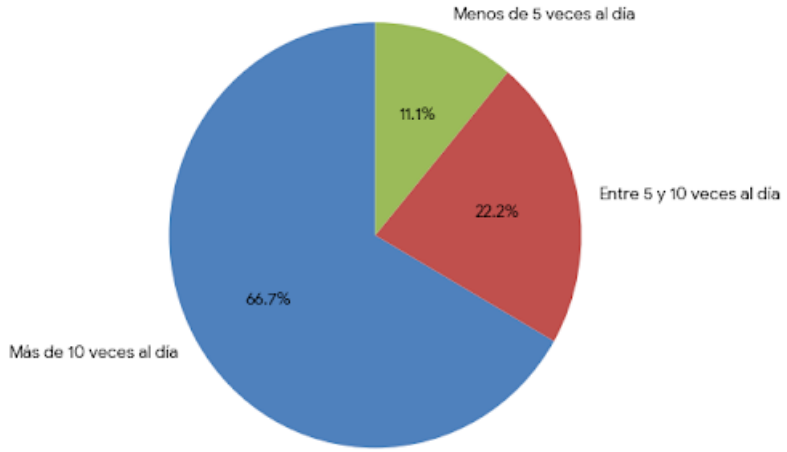


Figura 2. Pérdida de concentración al cambiar de ventana

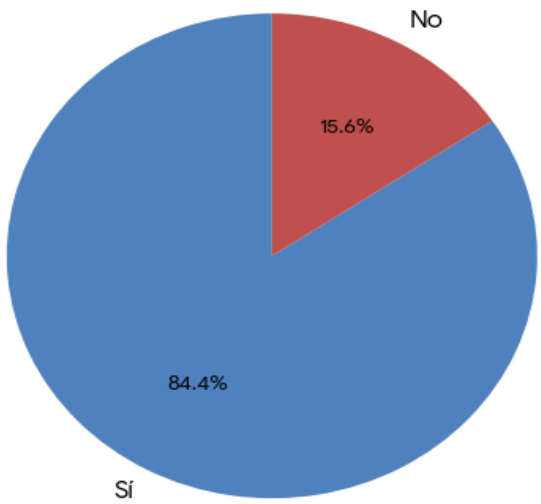


Figura 3. Percepción de mejora con chat embebido

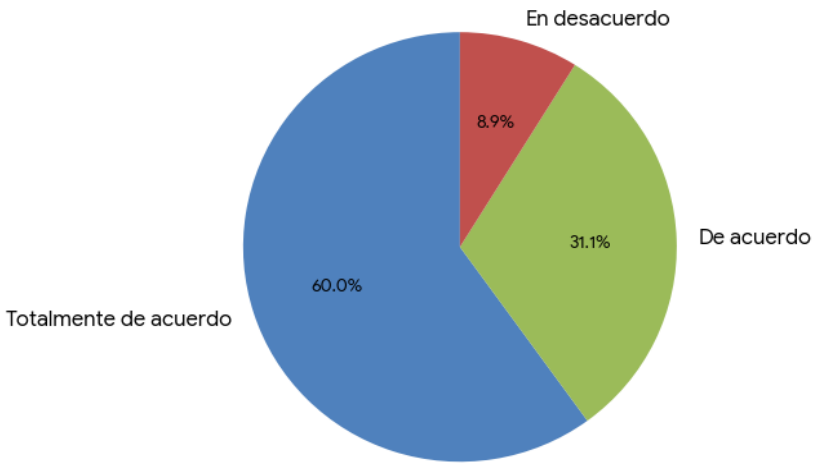


Figura 4. Preocupación por dependencia tecnológica

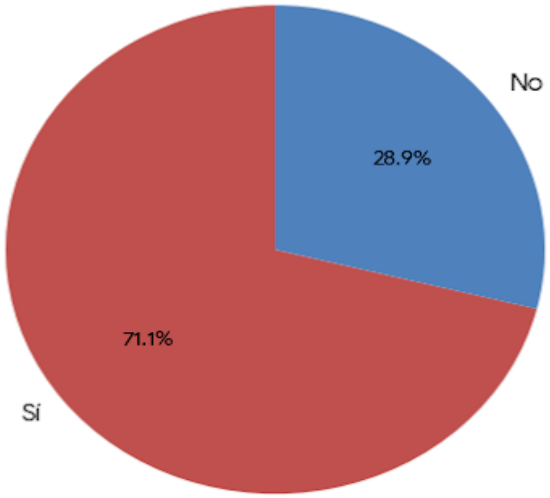


Figura 5. Detección de alucinaciones (respuestas incorrectas)

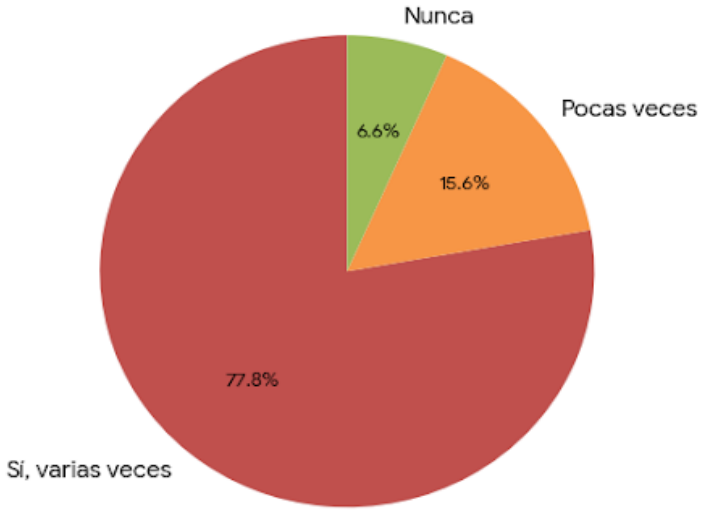
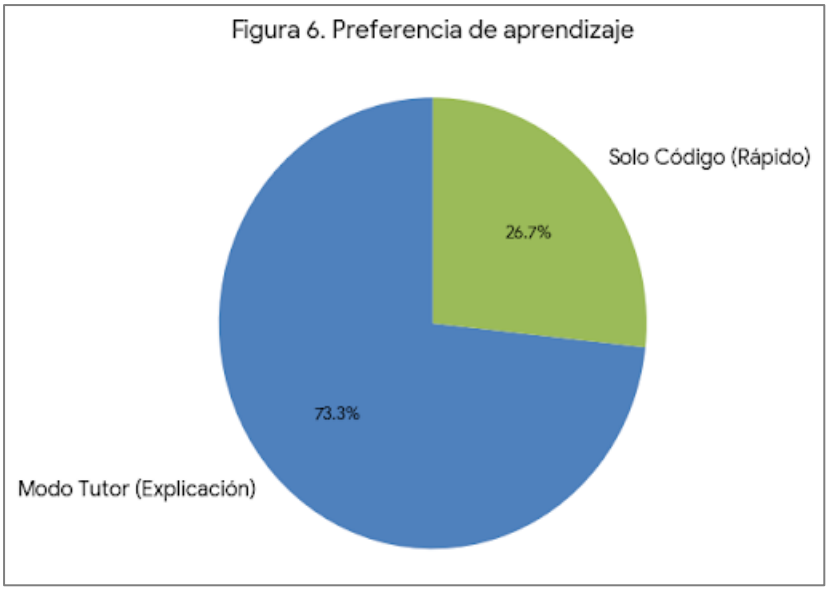


Figura 6. Preferencia de aprendizaje



Evaluación Teórica y Síntesis

Finalmente, se realizó una evaluación teórica de los beneficios, limitaciones e impactos de la tecnología. Basándose en el análisis de los "Antecedentes" Y el "Estado del Arte", esta fase se centró en analizar el impacto teórico en la productividad, así como los riesgos éticos y los efectos en los ámbitos laboral y educativo, para responder de manera integral a la pregunta de investigación.

RESULTADOS

El desarrollo de software es una de las actividades cognitivas más exigentes. El programador debe gestionar múltiples niveles de abstracción y sufre el coste de la conmutación de contexto (context switching) al cambiar constantemente de foco entre el editor de código, la documentación y los foros de ayuda . Este cambio interrumpe el flujo de concentración, aumenta la carga cognitiva y es uno de los principales factores que reducen la productividad. La ergonomía cognitiva busca adaptar las herramientas, como el IDE, para reducir este esfuerzo mental.

Para mitigar esto, han evolucionado los asistentes inteligentes (code assistants). Herramientas como GitHub Copilot, ChatGPT y AWS CodeWhisperer han demostrado la capacidad de generar código, explicar errores y ofrecer sugerencias adaptadas al contexto. Estos se basan en modelos de lenguaje de gran escala (LLMs) entrenados en repositorios de código abierto.

Sin embargo, el estado del arte actual muestra limitaciones. Muchos asistentes funcionan como extensiones externas, lo que genera latencia y riesgos de privacidad .

Barke et al. (2023) destacan que los desarrolladores tienden a aceptar sugerencias de IA sin validación exhaustiva, lo que puede introducir errores sutiles. Por lo tanto, la próxima generación de IDEs incorporará sistemas de asistencia nativos y embebidos, capaces de analizar el contexto del proyecto y las reglas del equipo. El estado del arte muestra una clara evolución hacia asistentes integrados de manera nativa como una herramienta fundamental para la productividad y la calidad del software (Savary-Leblanc et al., 2023; Corso et al., 2024).

Funcionamiento Conceptual y Diseño Técnico

El funcionamiento de un chat embebido se basa en su conexión continua con el proyecto del desarrollador. Esto le permite analizar en tiempo real el código, los errores del compilador y la estructura del proyecto. Gracias a esta integración, puede detectar errores de sintaxis, inconsistencias lógicas y ofrecer soluciones adaptadas al código específico, reduciendo la necesidad de buscar en fuentes externas (Li et al., 2023) .

La interacción se realiza mediante lenguaje natural, permitiendo al desarrollador preguntar "¿por qué ocurre este error?" . El sistema usa Procesamiento de Lenguaje Natural (PLN) para devolver respuestas contextuales, ejemplos de código y recomendaciones de buenas prácticas (Rustagi et al., 2024). Además, el asistente puede actuar de forma proactiva, sugiriendo refactorizaciones, mejoras de rendimiento o alertas de seguridad al identificar patrones de código repetidos (Weisz et al., 2024) .

El diseño técnico preliminar de esta herramienta se compondría de tres capas:

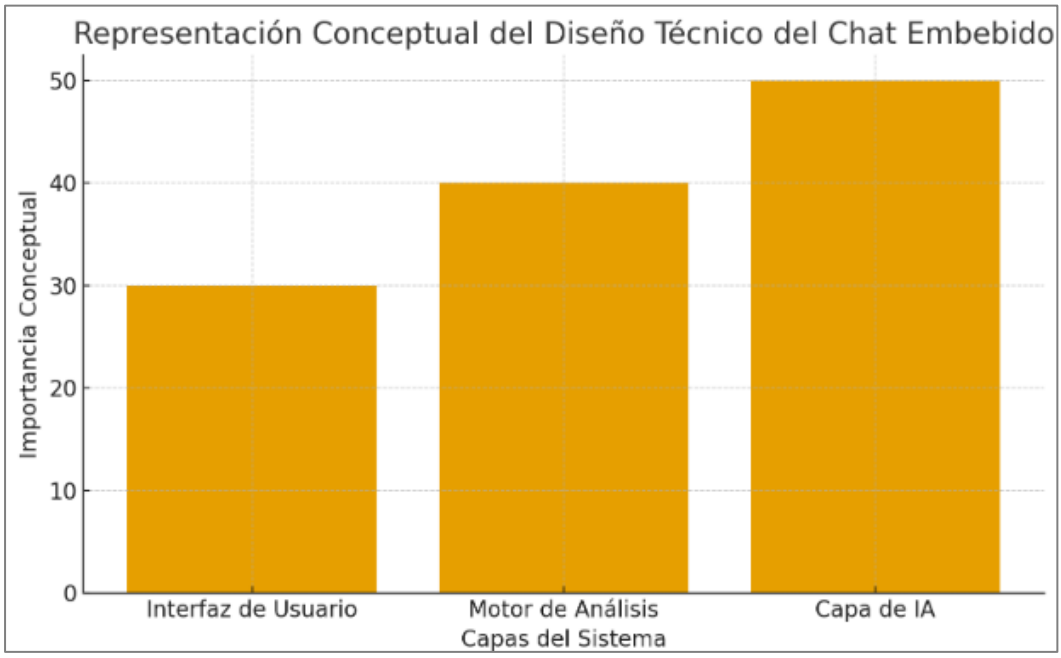
Interfaz de Usuario: Embebida directamente en el IDE (ej. Visual Studio Code), debe ser no intrusiva, configurable y accesible .

Motor de Análisis: Examina el contexto del código, los mensajes del compilador y la estructura del proyecto mediante análisis estático para ofrecer retroalimentación inmediata (Kim & Kim, 2024) .

Capa de Inteligencia Artificial: Utiliza modelos de PLN y aprendizaje automático para generar respuestas contextualizadas, pudiendo conectarse a redes locales o en la nube según la privacidad requerida . El sistema se beneficiaría del aprendizaje continuo (Nghiem et al., 2024).

Este diseño podría integrarse con sistemas de control de versiones (como Git) para analizar el historial de cambios y sugerir soluciones coherentes con la evolución del proyecto.

Figura 7



Discusión sobre el Impacto Laboral y Educativo

La incorporación de chats de asistencia embebidos representa un cambio profundo en cómo los programadores aprenden y trabajan.

En el ámbito educativo, estas herramientas redefinen la enseñanza. Actúan como tutores virtuales personalizados que ofrecen retroalimentación inmediata, explican errores y sugieren correcciones . Esto convierte al IDE en un espacio de aprendizaje activo, reduce la frustración asociada al error y fomenta el aprendizaje autodirigido. Los estudiantes pueden comprender mejor los errores y acceder a ejemplos prácticos sin abandonar el entorno de desarrollo (De los Santos & Ortiz, 2023).

En el entorno laboral, el impacto en la productividad es tangible. Kalliamvakou et al. (2023) reportan incrementos de productividad de entre el 20 % y el 40 %. El asistente reduce el tiempo de resolución de errores , optimiza el código y mantiene al programador concentrado, evitando la dispersión cognitiva. Esto redefine el rol del desarrollador: deja de ser un ejecutor de instrucciones para convertirse en un gestor del conocimiento tecnológico, que interpreta, valida y supervisa los resultados generados por la IA.

El análisis comparativo de las herramientas líderes revela diferencias significativas en su impacto.

Reportes de la industria, como los de OpenXcell (2024) y FutureAGI (2025), contrastan el rendimiento de GitHub Copilot frente a Amazon CodeWhisperer, destacando que la elección de la herramienta influye directamente en la velocidad del ciclo de desarrollo. A esto se suma la evidencia de Kalliamvakou et al. (2023), quienes cuantificaron un aumento de productividad, aunque Chen et al. (2021) matizan que la eficacia de modelos como Codex depende en gran medida de la calidad de los datos de entrenamiento utilizados

Discusión sobre Riesgos, Desafíos y Aspectos Éticos

A pesar de los beneficios, la adopción de estos asistentes presenta riesgos significativos. El principal desafío es la dependencia tecnológica. El desarrollador, al habituarse a soluciones inmediatas, puede reducir su capacidad para analizar problemas de forma independiente. Con el tiempo, esto puede erosionar la práctica del razonamiento algorítmico, la depuración manual y la resolución autónoma de problemas (Vaithilingam & Weng, 2022). En el contexto educativo, esto puede provocar un aprendizaje superficial, donde el alumno memoriza respuestas sin internalizar los conceptos (De los Santos & Ortiz, 2023).

Existen también desafíos técnicos. Las sugerencias pueden no ser siempre precisas, y la comprensión del contexto puede fallar en proyectos muy complejos (Vaithilingam y Weng, 2022).

Finalmente, surgen aspectos éticos y de seguridad. La privacidad del código es crucial si los modelos de IA procesan código propietario en servidores externos. Sandoval et al. (2022) advierten sobre el riesgo de fugas de información confidencial o la incorporación de código con licencias restrictivas. También existe la posibilidad de que los modelos reproduzcan sesgos o patrones de código inseguros presentes en sus datos de entrenamiento (Bender et al., 2021). El programador debe seguir siendo el responsable final del código implementado.

Propuesta de Evaluación y Perspectivas Futuras

Para determinar la eficacia real de la herramienta, se propone un plan de evaluación basado en métricas cuantitativas y cualitativas.

Métricas Cuantitativas: Incluirán la medición del tiempo promedio de detección y corrección de errores, la reducción de errores recurrentes y el incremento en la velocidad de codificación (Li et al., 2022; Kochhar et al., 2023) .

Métricas Cualitativas: Se aplicarán encuestas de satisfacción y entrevistas para analizar la percepción del desarrollador sobre la utilidad, claridad y confianza en las recomendaciones (Corso et al., 2024). Se propone un análisis comparativo entre grupos que usen la herramienta y grupos de control (Azaiz et al., 2023).

Mirando al futuro, las perspectivas de esta tecnología apuntan a asistentes multimodales, capaces de procesar texto, voz y diagramas (Zhu & Han, 2024). Se espera que los modelos evolucionen hacia un aprendizaje continuo que se adapte al estilo y nivel del programador, ofreciendo no solo asistencia técnica sino también retroalimentación pedagógica (Xu et al., 2025) . El futuro se orienta a una combinación de modelos locales seguros y plataformas en la nube, equilibrando rendimiento y privacidad (Savary-Leblanc et al., 2023; Xu et al., 2025) .

Evolución Histórica de los Asistentes de Programación y su Integración en IDEs

La presencia de asistentes inteligentes dentro de los entornos de desarrollo (IDEs) no surgió de manera repentina, sino que es el resultado de varias décadas de avances en herramientas de apoyo al programador. Comprender esta evolución permite apreciar cómo la tecnología ha pasado de ser un recurso limitado a convertirse en un acompañante inteligente capaz de analizar, sugerir y generar código dentro del propio entorno de trabajo. Esta perspectiva histórica también muestra cómo han cambiado las expectativas, los métodos de desarrollo y las herramientas que utilizan tanto estudiantes como profesionales.

Los primeros intentos de asistencia automatizada en programación se remontan a los años setenta y ochenta, cuando las herramientas se limitaban a resaltado de sintaxis, sugerencias básicas y sistemas de autocompletado simples. Aunque rudimentarios, estos mecanismos marcaron el inicio de la idea de que una herramienta podía facilitar el desarrollo disminuyendo errores sintácticos y acelerando la escritura de código. Sin embargo, estas primeras funciones estaban muy lejos de comprender la semántica o la intención del programador.

Durante los años noventa y principios de los dos mil, aparecieron herramientas más avanzadas como **IntelliSense** de Microsoft o los sistemas de autocompletado inteligente de JetBrains. Estas herramientas comenzaron a analizar la estructura interna del código y ofrecer sugerencias más útiles, como completar métodos disponibles, mostrar documentación y advertir sobre errores comunes.

Aun así, estas funciones seguían basándose en reglas estáticas y no en aprendizaje automático, por lo que su capacidad de adaptación era limitada.

Un punto de inflexión ocurrió a partir de 2015 con la expansión del machine learning en aplicaciones prácticas. Modelos de predicción empezaron a utilizarse para anticipar el siguiente fragmento de código, aunque en versiones tempranas su precisión aún era reducida. No fue sino hasta 2020–2021, con la introducción de modelos generativos de lenguaje como GPT o Codex, que la asistencia en programación dio un salto cualitativo: las herramientas comenzaron a comprender contexto, estilo y estructura del proyecto, permitiendo sugerir funciones completas, explicar errores y generar documentación.

Este avance permitió que los asistentes ya no fueran simples complementos, sino agentes inteligentes integrados directamente en los IDEs, capaces de trabajar de forma colaborativa con el programador. Herramientas como Copilot, CodeWhisperer, Codeium y las integraciones con ChatGPT marcaron una nueva era en la que el desarrollador cuenta con retroalimentación contextual avanzada, explicación de conceptos y generación automática de soluciones dentro del entorno de desarrollo.

Hoy en día, la evolución continúa enfocándose en mejorar la comprensión profunda del contexto del proyecto, integrar modelos más rápidos y eficientes, y permitir personalización completa del estilo y métodos de programación. La tendencia actual apunta hacia asistentes capaces de interactuar con repositorios completos, analizar pruebas automatizadas, y contribuir a todo el ciclo de vida del software, desde la planeación hasta el mantenimiento. En suma, la evolución histórica de los asistentes de programación refleja un progreso constante desde herramientas estáticas y limitadas, hasta sistemas inteligentes capaces de interpretar la intención del desarrollador y colaborar activamente en la creación de software. Este recorrido evidencia que la integración de chats de asistencia en los IDEs no es solo una innovación reciente, sino la culminación de décadas de avances tecnológicos orientados a facilitar, mejorar y transformar la experiencia del programador.

Diseño Técnico Preliminar

El diseño técnico de un chat de asistencia embebido en un IDE requiere la integración de múltiples tecnologías que permitan la comunicación fluida entre el programador, el entorno de desarrollo y el modelo de inteligencia artificial. La arquitectura propuesta estaría compuesta por tres capas principales: la interfaz de usuario, el motor de análisis y la capa de inteligencia artificial.

La interfaz de usuario estaría embebida directamente dentro del IDE (por ejemplo, Visual Studio Code o IntelliJ IDEA), permitiendo la interacción natural mediante lenguaje escrito o incluso comandos de voz. Este componente debe ser no intrusivo, configurable y accesible, garantizando una experiencia ergonómica (Nguyen & Dang, 2023).

El motor de análisis se encargaría de examinar el contexto del código —por ejemplo, el archivo actual, los mensajes del compilador y la estructura del proyecto—. Mediante técnicas de análisis estático, este módulo identificaría errores, advertencias y posibles mejoras antes de ejecutar el programa, ofreciendo retroalimentación inmediata (Kim & Kim, 2024).

Finalmente, la capa de inteligencia artificial utilizaría modelos de procesamiento de lenguaje natural (NLP) y aprendizaje automático para generar respuestas contextualizadas. Estos modelos podrían conectarse a redes locales o servicios en la nube, dependiendo del nivel de privacidad requerido. El sistema se beneficiaría del aprendizaje continuo, ajustando sus respuestas a medida que el usuario interactúa con el chat (Nghiem et al., 2024).

Desde una perspectiva técnica avanzada, el sistema también podría integrarse con repositorios de control de versiones (como Git) para analizar el historial de cambios y sugerir soluciones coherentes con la evolución del proyecto. De esta manera, el diseño propuesto no solo mejora la productividad, sino que también se alinea con las tendencias de desarrollo asistido por IA contextual y seguro.

Cómo funciona un chat embebido en un IDE

Vinculación con el código y análisis del contexto

Un chat integrado en un IDE funciona al estar continuamente conectado al proyecto del desarrollador. Esto le permite analizar en tiempo real el código, los errores reportados por el compilador o intérprete, y la estructura general del proyecto.

Gracias a esta integración, el asistente puede detectar errores de sintaxis, inconsistencias lógicas y posibles vulnerabilidades, ofreciendo soluciones adaptadas al código específico en lugar de respuestas genéricas (Li et al., 2023). Esto reduce significativamente la necesidad de que el programador busque información en fuentes externas.

Interacción mediante lenguaje natural

El chat permite que el desarrollador haga preguntas en lenguaje natural, tales como "¿por qué ocurre este error de compilación?" o "¿cuál es la mejor manera de implementar esta función?". El sistema procesa estas consultas usando técnicas de procesamiento de lenguaje natural (PLN) y devuelve respuestas contextuales, ejemplos de código, recomendaciones de buenas prácticas y posibles alternativas de implementación (Rustagi et al., 2024).

Este enfoque mejora la accesibilidad de la herramienta para desarrolladores de distintos niveles de experiencia y facilita la comprensión de problemas complejos sin interrumpir el flujo de trabajo.

Sugerencias proactivas y optimización del código

Más allá de responder preguntas, un chat embebido puede actuar de manera proactiva, analizando patrones de código repetidos y sugiriendo refactorizaciones, mejoras de rendimiento o alertas de seguridad.

Por ejemplo, podría identificar un bucle innecesariamente complejo y recomendar una versión más eficiente, o señalar funciones duplicadas que podrían unificarse. Estas acciones permiten mantener un código más limpio y eficiente, reduciendo errores futuros y aumentando la productividad (Weisz et al., 2024).

Arquitectura técnica y soporte de IA

El funcionamiento técnico del chat combina varias tecnologías:

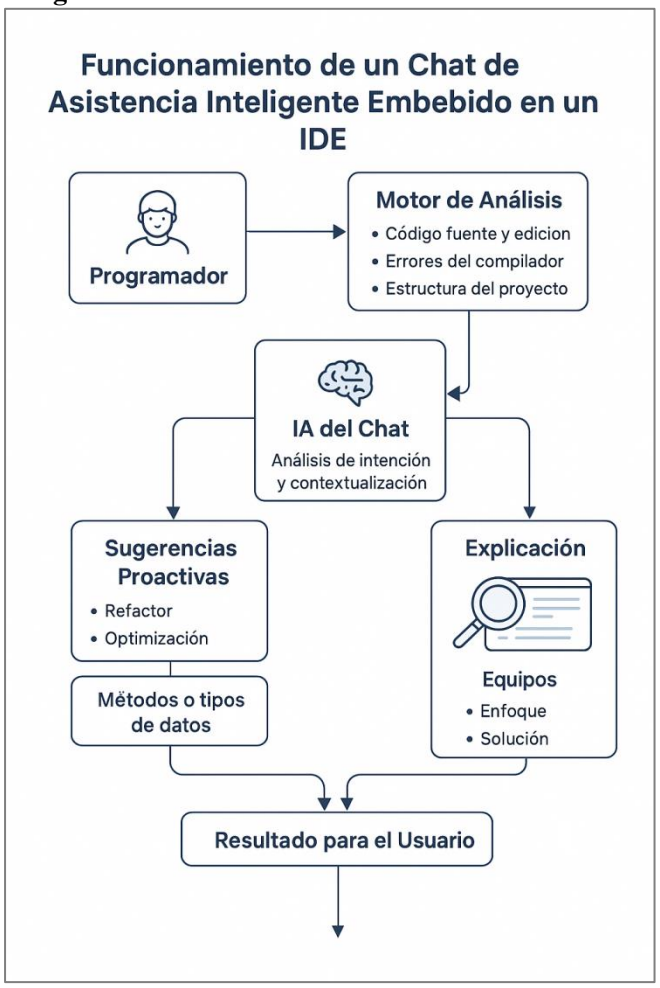
Procesamiento de lenguaje natural (PLN): para interpretar preguntas y generar respuestas comprensibles.

Análisis estático de código: para examinar la estructura, variables y dependencias del proyecto.

Modelos de inteligencia artificial o bases de datos especializadas: que permiten generar soluciones adaptadas al contexto y aprender de la interacción con el usuario.

Esta arquitectura permite que la interacción sea rápida y contextualizada, evitando la pérdida de enfoque del desarrollador y mejorando la eficiencia del proceso de programación (Li et al., 2023; Rustagi et al., 2024).

Imagen 2



Personalización y aprendizaje continuo

En un contexto profesional, el chat puede ajustarse a las normas de codificación de la empresa, integrarse con sistemas de control de versiones y aprender de los errores y consultas más frecuentes del equipo. Esto permite que las recomendaciones se adapten progresivamente al estilo de programación de la compañía y a las mejores prácticas del proyecto, convirtiéndose en una herramienta de aprendizaje continuo (De los Santos & Ortiz, 2023).

Limitaciones Técnicas y Desafíos Actuales de los Asistentes Embebidos

A pesar de los importantes avances logrados por los chats de asistencia embebidos en los entornos de desarrollo (IDEs), estas herramientas aún enfrentan limitaciones técnicas significativas que afectan su precisión, confiabilidad y capacidad de comprensión del código. Estas limitaciones reflejan tanto restricciones inherentes a los modelos de inteligencia artificial actuales como desafíos derivados de la

complejidad del desarrollo de software moderno. Analizar estas debilidades es esencial para comprender el estado actual de la tecnología y las oportunidades de mejora futura.

Limitaciones en la comprensión del contexto

Uno de los desafíos más importantes es la dificultad que los asistentes presentan para entender el contexto completo de un proyecto, especialmente cuando se trata de repositorios extensos o con múltiples módulos relacionados entre sí. Aunque estas herramientas pueden analizar fragmentos de código, suelen tener problemas para comprender la estructura global, las dependencias entre archivos y el flujo lógico completo del programa.

Además, su capacidad para interpretar código fragmentado es limitada; cuando el programador proporciona solo una parte de una función o clase, el asistente puede generar respuestas imprecisas o soluciones que no encajan en la arquitectura general del proyecto. Estas dificultades son especialmente evidentes en sistemas legacy o en proyectos con poca documentación, donde la IA tiene menos puntos de referencia para contextualizar la información.

Restricciones de procesamiento

Los modelos de inteligencia artificial utilizados en los asistentes embebidos trabajan bajo restricciones de procesamiento que influyen directamente en su rendimiento. La primera limitación es el tamaño del contexto que pueden analizar simultáneamente; debido a los límites de tokens, no pueden revisar grandes cantidades de código en una sola consulta, lo que reduce su capacidad para entender el estado completo del proyecto.

Otra restricción importante es la latencia. Muchos asistentes operan mediante servicios en la nube, lo que implica que el código debe enviarse a un servidor externo para ser procesado. Esto puede producir demoras notables en conexiones lentas o inestables.

Además, algunos modelos requieren altos niveles de potencia computacional, lo que puede incrementar el uso de memoria o CPU dentro del IDE y afectar negativamente el rendimiento del entorno de desarrollo.

Dificultades con lenguajes altamente especializados

Aunque los asistentes embebidos están entrenados con grandes volúmenes de código, su desempeño no es uniforme en todos los lenguajes.

Suelen tener un rendimiento aceptable en lenguajes ampliamente utilizados como Python, JavaScript o Java, pero encuentran mayores dificultades en lenguajes especializados o de bajo nivel, como C, Rust o ensamblador.

Estos lenguajes requieren una comprensión profunda de conceptos como manejo de memoria, concurrencia o gestión de punteros, áreas donde los modelos de IA pueden generar sugerencias incorrectas o incompletas. Asimismo, presentan problemas al trabajar con frameworks muy nuevos, bibliotecas poco documentadas o sintaxis altamente técnicas, donde la IA no dispone de suficientes ejemplos para ofrecer respuestas precisas.

Riesgos de alucinaciones técnicas

Un problema recurrente en estos sistemas es la presencia de alucinaciones técnicas, es decir, respuestas fabricadas o incorrectas generadas por el modelo. Estas alucinaciones pueden manifestarse de diversas formas, como sugerir funciones que no existen, recomendar librerías incompatibles o escribir código que no compila.

Estas situaciones no solo generan confusión en programadores principiantes, sino que también pueden introducir vulnerabilidades o errores difíciles de detectar en proyectos profesionales. Aunque estos errores son previsibles en modelos generativos, representan un desafío importante para la adopción totalmente confiable de asistentes embebidos en entornos críticos o de alta exigencia.

Desafíos para el futuro

Los asistentes de desarrollo aún enfrentan múltiples desafíos para lograr un desempeño realmente óptimo. Entre los más relevantes se encuentra la necesidad de ampliar la capacidad de contexto, permitiendo que la IA analice repositorios completos sin perder coherencia. También es crucial mejorar el razonamiento lógico del modelo para evitar errores conceptuales y reducir las alucinaciones.

Otro desafío importante es la integración del análisis estático, dinámico y semántico del código dentro de la propia IA, lo que permitiría una comprensión más profunda de la ejecución real del programa. Finalmente, será necesario desarrollar modelos capaces de operar offline sin comprometer la privacidad del usuario y sin requerir un consumo excesivo de recursos computacionales.

CONCLUSIONES

La presente investigación ha examinado, desde una perspectiva teórica y analítica, la pertinencia, viabilidad y repercusiones de integrar un chat de asistencia inteligente embebido dentro de un entorno de desarrollo integrado (IDE). Este estudio surgió a partir de la identificación de un problema central en la práctica moderna de la programación: la alta carga cognitiva a la que se enfrenta el desarrollador debido a la administración simultánea de múltiples niveles de abstracción, la necesidad de recordar estructuras complejas y la constante interrupción provocada por la búsqueda externa de soluciones. Dichos factores impactan directamente en la productividad y alteran el flujo mental requerido para mantener una concentración sostenida durante las tareas de desarrollo de software.

El análisis del marco teórico permitió comprender que los retos actuales del programador no son únicamente de carácter técnico, sino profundamente cognitivos. La naturaleza lineal del pensamiento humano contrasta con la estructura altamente modular y paralela de los sistemas de software; como resultado, cada vez que el programador abandona el IDE para consultar documentación, foros o ejemplos, experimenta una pérdida de continuidad conocida como context switching, la cual genera un costo mental considerable y afecta la calidad del trabajo. A partir de este planteamiento, la investigación propone que un asistente inteligente embebido constituye una respuesta viable y coherente con la necesidad de disminuir estas interrupciones, ya que centraliza la información, reduce el esfuerzo de búsqueda y ofrece apoyo contextual sin abandonar el entorno de programación.

Desde esta perspectiva, la hipótesis teórica planteada sostiene que un asistente embebido puede operar como una suerte de sistema experto integrado, capaz de interpretar el código en tiempo real, detectar posibles errores, sugerir mejoras y proporcionar explicaciones detalladas en lenguaje natural. Dicho sistema no sustituye el conocimiento del programador, sino que actúa como un complemento que ayuda a reforzar buenas prácticas, mejorar la refactorización continua y reducir la complejidad cognitiva a la que está expuesto el usuario. La IA se convierte así en un agente colaborativo que acompaña la construcción del software, no como un reemplazo, sino como una extensión de las capacidades humanas.

Los resultados teóricos del análisis realizado permiten afirmar que la integración de un chat de asistencia embebido tiene un impacto profundo en dos dimensiones fundamentales: la educativa y la profesional.

En el ámbito educativo, estos asistentes funcionan como un tutor personalizado, capaz de brindar retroalimentación inmediata, explicar conceptos de programación, sugerir ejemplos, guiar la resolución de errores y acompañar la curva de aprendizaje. Esta herramienta contribuye a un aprendizaje significativo, reduce la frustración típica de los programadores novatos y facilita la adquisición de habilidades esenciales. Para el estudiante, recibir orientación en tiempo real dentro del IDE acorta la distancia entre "aprender" y "hacer", acelerando la comprensión y favoreciendo la autonomía formativa.

En el ámbito profesional, el impacto se orienta principalmente hacia la productividad y la transformación del rol del programador. Al eliminar parte del tiempo dedicado a tareas mecánicas o repetitivas —como escribir estructuras básicas, investigar sintaxis específicas o corregir errores comunes— el asistente permite que el desarrollador concentre sus esfuerzos en actividades de mayor valor, como la arquitectura, la validación conceptual y el diseño de soluciones. Esto no solo mejora la eficiencia individual, sino que redefine la dinámica de trabajo en los equipos de desarrollo, promoviendo flujos más ágiles y colaborativos. En este sentido, el programador transita de ser un ejecutor constante a convertirse en un supervisor estratégico que guía y evalúa el código generado con el apoyo de la IA.

Sin embargo, la investigación también revela que la introducción de estos sistemas no está exenta de riesgos y desafíos significativos. El primero y más evidente es la dependencia tecnológica. Si el programador utiliza de manera indiscriminada o acrítica el asistente, puede verse afectada su capacidad de razonamiento lógico, resolución de problemas y comprensión profunda del código. Esto podría generar profesionales técnicamente funcionales, pero carentes de habilidades fundamentales para enfrentar situaciones no previstas por la IA. Asimismo, los estudiantes que dependen excesivamente del asistente podrían desarrollar un aprendizaje superficial, basado más en la repetición que en la comprensión conceptual.

A nivel ético y de seguridad, la investigación identifica preocupaciones relacionadas con la privacidad del código propietario, el posible almacenamiento de fragmentos en servidores externos y la responsabilidad legal frente a errores generados por la IA. Estos aspectos requieren regulaciones claras, políticas de uso responsables y mecanismos de supervisión para evitar riesgos como filtraciones, mal uso de datos o problemas de licenciamiento del código sugerido por el asistente.

La implementación de estas herramientas debe ir acompañada de un marco normativo que proteja a los usuarios y garantice que la herramienta opere bajo pautas transparentes y confiables.

Finalmente, esta investigación establece una base teórica sólida para justificar la pertinencia, el potencial y la relevancia de los asistentes embebidos en IDEs. No obstante, al tratarse de un estudio teórico, es necesario reconocer que los postulados aquí expuestos deben validarse empíricamente. Por ello, se recomienda que investigaciones futuras desarrollen prototipos funcionales y apliquen métricas cuantitativas y cualitativas para medir de forma objetiva la reducción de la carga cognitiva, la mejora en la productividad y la aceptación general por parte de los usuarios. Mediante estos estudios experimentales será posible comprobar y refinar las hipótesis propuestas, así como establecer lineamientos precisos para la implementación óptima de asistentes inteligentes embebidos en entornos de programación reales.

REFERENCIAS BIBLIOGRÁFICAS

Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). Evaluating the code quality of AI-assisted code generation tools: An empirical study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. *arXiv*. Recuperado de https://arxiv.org/abs/2304.10778

Cavalcante, S. (2025). A case study with GitHub Copilot and other AI assistants. *SCITEPRESS Digital Library*. Recuperado de https://www.scitepress.org/Papers/2025/132947/132947.pdf

Sergeyuk, A. (2025). Using AI-based coding assistants in practice. *ScienceDirect*. Recuperado de https://www.sciencedirect.com/science/article/abs/pii/S0950584924002155

Qodo.ai. (2025). 20 best AI coding assistant tools [Updated Aug 2025]. Recuperado de https://www.qodo.ai/blog/best-ai-coding-assistant-tools/

OpenXcell. (2024). CodeWhisperer vs Copilot: Battle of the code assistants. Recuperado de https://www.openxcell.com/blog/codewhisperer-vs-copilot/

JetBrains. (2023). JetBrains AI | Intelligent coding assistance, AI solutions, and more. Recuperado de https://www.jetbrains.com/ai/

FutureAGI. (2025). AI coding assistant 2025: Copilot vs Cursor vs CodeWhisperer. Recuperado de https://futureagi.com/blogs/github-copilot-vs-cursor-vs-codewhisperer-2025

Visual Studio Magazine. (2024). GitHub Copilot tops research report on AI code assistants. Recuperado de https://visualstudiomagazine.com/articles/2024/08/26/github-copilot-tops-research-report-on-ai-code-assistants.aspx

Computer Society. (2025). Top 5 AI coding assistants and their pros and cons. Recuperado de https://www.computer.org/publications/tech-news/trends/top-five-coding-assistants/

Tech Research Online. (2025). GitHub Copilot vs. Amazon CodeWhisperer. Recuperado de https://techresearchonline.com/blog/github-copilot-vs-amazon-codewhisperer/

Mantel Group. (2023). Best AI coding assistant tools in 2023. Recuperado dehttps://mantelgroup.com.au/best-ai-coding-assistant-tools-in-2023/

Forte Group. (2024). Research shows AI coding assistants can improve productivity.Recuperado de https://fortegrp.com/insights/ai-coding-assistants

Ahmad, W., Ahmad, A., Li, L., & Neubig, G. (2021). Transformers for code: How far are we? IEEE Transactions on Software Engineering, 47(12), 1234–1250 https://doi.org/10.1109/TSE.2021.3051123

Alhoshan, W., & Wang, X. (2022). AI-driven developer tools: Opportunities and challenges. Journal of Systems and Software, 191, 111409. https://doi.org/10.1016/j.jss.2022.111409

Azaiz, I., Deckarm, O., & Strickroth, S. (2023). AI-enhanced auto-correction of programming exercises: How effective is GPT-3.5? arXiv preprint arXiv:2306.04522.

Barke, S., James, E., & Bird, C. (2023). Grounded Copilot: How programmers interact with code suggestions. In Proceedings of the 45th International Conference on Software Engineering (ICSE ’23). IEEE/ACM. https://doi.org/10.1109/ICSE.2023.00068

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT ’21), 610–623. https://doi.org/10.1145/3442188.3445922

Chen, M., Tworek, J., Jun, H., yuan, Q., De Oliveira Pinto, H. P., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Corso, V., Mariani, L., Micucci, D., & Riganelli, O. (2024). Assessing AI-based code assistants in method generation tasks. In ICSE-Companion '24: IEEE/ACM 46th International Conference on Software Engineering (Companion Proceedings). IEEE/ACM.

Kalliamvakou, E., Guo, P. J., & Murphy, G. C. (2023). An empirical study on the impact of AI coding assistants on developer productivity. ACM Transactions on Software Engineering and Methodology (TOSEM), 32(5), 1–25. https://doi.org/10.1145/3591504

Perry, N., Srivastava, D., Kumar, D., & Boneh, D. (2023). Do Users Write More Insecure Code with AI Assistants? En Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23). Association for Computing Machinery. https://doi.org/10.1145/3576915.3623157

Prather, J., Denny, P., Leinonen, J., Becker, B. A., Albluwi, I., Craig, M., ... & Smith, J. (2023). The Robot Dog Ate My Homework: Implications of Large Language Models on Programming Education. En Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education (ITiCSE V. 1). ACM. https://doi.org/10.1145/3587102.3588794

Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. En CHI Conference on Human Factors in Computing Systems (CHI '22). ACM. https://doi.org/10.1145/3491102.3517739

Kalliamvakou, E. (2022). Research: Quantifying GitHub Copilot's Impact on Developer Productivity and Happiness. GitHub.blog. Recuperado de https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

Chen, M., Tworek, J., Jun, H., Yuan, Q., De Oliveira Pinto, H. P., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

FutureAGI. (2025). AI coding assistant 2025: Copilot vs Cursor vs CodeWhisperer. Recuperado de https://futureagi.com/blogs/github-copilot-vs-cursor-vs-codewhisperer-2025

Kalliamvakou, E., Guo, P. J., & Murphy, G. C. (2023). An empirical study on the impact of AI coding assistants on developer productivity. ACM Transactions on Software Engineering and Methodology (TOSEM), 32(5), 1–25. https://doi.org/10.1145/3591504

OpenXcell. (2024). CodeWhisperer vs Copilot: Battle of the code assistants. Recuperado de https://www.openxcell.com/blog/codewhisperer-vs-copilot/