Revisión sistemática de la literatura: Sistemas de tutoría inteligente

Systematic literature review: Intelligent tutoring systems

Mauricio Aburto Lara
Universidad Veracruzana, Xalapa, México
mauricioaburtol@outlook.com

Lorena Alonso Ramírez
Universidad Veracruzana, Xalapa, México
lalonso@uv.mx

Carlos Alberto Ochoa Rivera
Universidad Veracruzana, Xalapa, México
cochoa@uv.mx

https://doi.org/10.36825/RITI.13.31.004

Recibido: Junio 22, 2025
Aceptado: Septiembre 30, 2025

Resumen: Aprender a programar implica una curva de aprendizaje elevada, especialmente en las primeras etapas, debido a la carga cognitiva, la abstracción de conceptos y las dificultades propias del lenguaje de programación. Este estudio tiene como objetivo identificar enfoques tecnológicos recientes que integren Sistemas de Tutoría Inteligente (ITS) y Modelos de Lenguaje de Gran Escala (LLM) para apoyar el proceso de enseñanza de programación. Se realizó una revisión sistemática siguiendo las directrices de Kitchenham, consultando bases de datos especializadas y aplicando criterios de inclusión y exclusión en tres fases para seleccionar trabajos relevantes al área. Los estudios analizados incluyen propuestas con retroalimentación conversacional, aprendizaje adaptativo y análisis automatizado de código, mostrando mejoras en la comprensión de conceptos, aumento de la confianza y mayor finalización de tareas en estudiantes principiantes. Sin embargo, se identificaron desafíos como la dificultad para mantener el contexto en interacciones prolongadas y la presencia de respuestas erróneas o "alucinaciones" en los modelos. Se concluye que la integración multimodal, el diseño centrado en el usuario y un manejo optimizado de datos representan áreas clave para potenciar la personalización y efectividad de estos sistemas en entornos educativos, favoreciendo tanto el desarrollo de habilidades como el seguimiento continuo del aprendizaje.

Palabras clave: *Sistemas de Tutoría Inteligente, Aprendizaje Adaptativo, Enseñanza de la Programación, Retroalimentación Conversacional, Modelos de Lenguaje de Gran Escala*.

Abstract: Learning to program involves a steep learning curve, especially in the early stages, due to cognitive load, concept abstraction, and the inherent difficulties of programming languages. This study aims to identify recent technological approaches that integrate Intelligent Tutoring Systems (ITS) and Large Language Models (LLM) to support the programming teaching process. A systematic review was conducted following Kitchenham's guidelines, consulting specialized databases, and applying inclusion and exclusion criteria in three phases to select relevant works in the field. The analyzed studies include proposals featuring conversational feedback, adaptive

learning, and automated code analysis, showing improvements in concept comprehension, increased confidence, and higher task completion rates among beginner students. However, challenges were identified, such as the difficulty in maintaining context during prolonged interactions and the presence of erroneous responses or "hallucinations" in the models. It is concluded that multimodal integration, user-centered design, and optimized data management represent key areas to enhance the personalization and effectiveness of these systems in educational environments, fostering both skill development and continuous learning monitoring.

Keywords: *Intelligent Tutoring Systems, Programming Education, Adaptive Learning, Conversational Feedback, Large Language Models.*

1. Introducción

El aprendizaje de la programación siempre ha sido un desafío para aquellos novatos en el área, ya que este proceso trae consigo diversos retos, como la carga cognitiva involucrada, la abstracción de los conceptos y la falta de retroalimentación inmediata [1], [2]. A esto se suman los desafíos que conlleva el comprender la sintaxis del lenguaje de programación elegido para desarrollar dichas habilidades [3], [4]. Tomando todo esto en cuenta, es fácil perder claridad sobre cuál es la mejor estrategia para adquirir estos conocimientos y desarrollar estas habilidades [5].

Por lo tanto, la enseñanza guiada de un profesor experto en el área es importante para guiar el proceso de aprendizaje de manera estructurada [6]. Sin embargo, cada alumno tiene diferentes ritmos de aprendizaje, dudas específicas y dificultades al momento de abordar los temas, lo que convierte a la enseñanza personalizada en una necesidad [7]. Esta necesidad, sin embargo, resulta difícil de satisfacer debido a la cantidad de alumnos que el docente debe atender [8], [9].

Debido a esta limitación, se ha buscado incluir nuevas soluciones tecnológicas que puedan apoyar al docente en ciertos aspectos de la enseñanza. Entre estas soluciones se encuentran los modelos de lenguaje de gran escala *Large Language Models* (LLMs), sistemas de inteligencia artificial entrenados con grandes volúmenes de datos [11]. Estos modelos permiten generar y comprender el lenguaje natural, por lo que se facilita el uso en entornos conversacionales [9]. Esto los hace útiles para poder brindar apoyo en el aprendizaje, ya que pueden ofrecer explicaciones, resolver dudas y adaptar respuestas al nivel del estudiante [12], [13].

Es con este contexto que, durante los últimos años, los LLMs se han aplicado en diferentes ámbitos, siendo uno de ellos el área de la enseñanza, empleándose como guías capaces de ofrecer retroalimentación [12], [13]. Estos modelos han emergido como una herramienta prometedora para apoyar esta problemática, gracias a su capacidad de proporcionar respuestas en lenguaje natural [5] y adaptarse a diversas necesidades del estudiante [11], [9]. Sin embargo, a pesar del desarrollo de estas herramientas con el objetivo de brindar un apoyo personalizado, se presentan desafíos al momento de integrar estas tecnologías en un entorno educativo controlado para apoyar al estudiante [2].

Algunos estudios señalan que una integración deficiente de estas tecnologías podría impedir que el estudiante obtenga beneficios reales o provocar una dependencia que limite el desarrollo de las habilidades planteadas [6], [9], [8]. Por ello, se considera crucial identificar las técnicas más adecuadas para el desarrollo de estos sistemas, así como definir qué estructuras resultan más efectivas para construir Sistemas de Tutoría Inteligente que integren un LLM [18], [19], [5].

Este trabajo presenta una revisión sistemática centrada en identificar enfoques tecnológicos recientes que integren tutores inteligentes, inteligencia artificial generativa, retroalimentación conversacional y herramientas de visualización para facilitar la enseñanza de programación. Para ello, se consultaron bases de datos como IEEE Xplore, ACM Digital Library, ScienceDirect y Wiley Online Library.

2. Metodología

La revisión siguió una estrategia sistemática basada en los lineamientos propuestos por Kitchenham [20] para desarrollar revisiones sistemáticas en el área de la computación. El proceso se estructuró en las siguientes tres fases:

1. Organizar la revisión: Durante la primera fase se definieron las preguntas de investigación, enfocadas en identificar elementos clave relacionados con los sistemas de tutoría inteligente en la enseñanza de la

programación, las áreas de trabajo y el estado actual del desarrollo de estas tecnologías en entornos educativos.

2. Realizar la revisión sistemática: En la segunda fase se aplicó la cadena de búsqueda en los repositorios seleccionados, junto con los criterios de inclusión y exclusión definidos previamente, con el fin de acotar la investigación.
3. Reportar los hallazgos: En la tercera fase, se organizó la información recopilada para dar respuesta a las preguntas de investigación planteadas. En las siguientes secciones se detalla dicho proceso.

2.1. Preguntas de investigación

Con este contexto, se definieron 4 preguntas de investigación con el objetivo de identificar qué aspectos son importantes al construir un sistema inteligente, qué factores influyen en lograr una personalización más detallada, cuál es el impacto de estos sistemas y qué características son las que más influyen. Las preguntas seleccionadas, que mejor cubren el área de investigación, fueron las siguientes:

- PI1 ¿Cuál es el impacto del uso de sistemas de tutoría inteligente en el aprendizaje de la programación?
- PI2 ¿Qué factores influyen en la personalización del aprendizaje en los sistemas de tutoría inteligente para programación?
- PI3 ¿Qué mecanismos permiten la personalización del aprendizaje dentro de los sistemas de tutoría inteligente aplicados a programación?
- PI4 ¿Qué desafíos y limitaciones se han identificado en el diseño e implementación de los sistemas de tutoría inteligente?

2.2. Criterios de inclusión y Exclusión

Para asegurar que los resultados obtenidos fueran lo más precisos respecto al contexto de investigación, y que los estudios analizados tuvieran la relevancia necesaria para llegar a conclusiones concretas se definieron criterios de inclusión y exclusión que se aplicarían a los estudios recopilados. Los criterios de inclusión y exclusión definidos se presentan en la Tabla 1.

Tabla 1. Criterios de inclusión y exclusión.

Identificador	Criterio de inclusión
CI1	Documentos publicados entre los años 2020 y 2025.
CI2	Artículos escritos en español o inglés.
CI3	Documentos de acceso abierto (Open Access).
CI4	Estudios que presenten o analicen soluciones computacionales.
CI5	Artículos que involucren el uso de Inteligencia Artificial Generativa (IA Generativa) o sistemas de tutoría inteligente (ITS).
CI6	Investigaciones que integren mecanismos de evaluación de código, análisis o retroalimentación automatizados para la enseñanza de programación.
CI7	Estudios que aborden el aprendizaje adaptativo o personalizado aplicado al proceso de enseñanza de programación.
Identificador	**Criterio de inclusión**
CE1	Estudios que se enfoquen en áreas fuera de la enseñanza de programación, como matemáticas, álgebra, química o biología.

CE2	Artículos puramente teóricos que no presenten soluciones prácticas o aplicaciones computacionales.
CE3	Trabajos que no presenten resultados cuantitativos derivados de la investigación.
CE4	Artículos repetidos.

Fuente: Elaboración propia.

2.3. Cadena de búsqueda

Para asegurar la obtención de investigaciones afines y poder aplicar los criterios previamente mencionados, se requerían palabras clave que permitieran obtener resultados precisos al momento de consultar las bases de datos seleccionadas. Por esta razón, se utilizaron los siguientes términos: AI *Assistant Systems, Programming Language Learning, Learning Environments y Computational Thinking*. Se utilizó lógica booleana mediante los conectores "AND" y "OR" para vincular trabajos relacionados, lo cual derivó en la siguiente cadena de búsqueda:

(("AI *Assistant S*ystems" OR "*intelligent tutoring* systems")
AND ("*programming language learning*" OR "*teaching programming*")
AND ("*learning environments*" OR "*educational platforms*")
AND ("*computational* thinking")
AND ("*user experience*" OR "UX" OR "*adaptive interfaces*")
AND ("*large language* model"))

Las bases de datos consultadas para esta investigación fueron: IEEE Xplore, Wiley Online Library, ACM Digital Library y ScienceDirect. Estas bases fueron seleccionadas debido a su relevancia en el ámbito de la computación.

2.4. Aplicación de criterios de inclusión y exclusión

Se aplicaron los criterios de inclusión y exclusión de forma estructurada, con el objetivo de obtener los mejores resultados posibles al descartar investigaciones irrelevantes. Este proceso se realizó de lo general a lo particular, asegurando la calidad de los estudios seleccionados (Tabla 2).

Tabla 2. Criterios de inclusión y exclusión por fase.

Fase1	
Identificador	**Criterio**
CI1	Documentos publicados entre los años 2020 y 2025
CI3	Documentos de acceso abierto (*Open Access*)
CI2	Artículos escritos en español o inglés.
CI5	Artículos que involucren el uso de Inteligencia Artificial Generativa (IA Generativa) o sistemas de tutoría inteligente (ITS).
Fase 2	
CE1	Estudios que se enfoquen en áreas fuera de la enseñanza de programación, como matemáticas, álgebra, química o biología
CE2	Artículos puramente teóricos que no presenten soluciones prácticas o aplicaciones computacionales
CI4	Estudios que presenten o analicen soluciones computacionales

CI7	Estudios que aborden el aprendizaje adaptativo o personalizado aplicado al proceso de enseñanza de programación
CI6	Investigaciones que integren mecanismos de evaluación de código, análisis o retroalimentación automatizados para la enseñanza de programación
	Fase 3
CE4	Artículos repetidos
CE3	Trabajos que no presenten resultados cuantitativos derivados de la investigación

Fuente: Elaboración propia.

Durante la primera fase se aplicaron cuatro criterios de inclusión (CI1, CI2, CI3 y CI5). Se filtraron los resultados seleccionando documentos publicados en los últimos cinco años (CI1), debido a que este rango de tiempo permite obtener una visión más actualizada del estado del arte, escritos en inglés o español (CI2), y disponibles en acceso abierto (CI3). Además, se contemplaron únicamente documentos que hicieran uso de Inteligencia Artificial Generativa o sistemas de tutoría inteligente (CI5), con el objetivo de identificar la mayor cantidad de sistemas desarrollados con estas tecnologías. Esta fase redujo el total de documentos de 670 a 91 (Tabla 3)

Tabla 3. Resultados con la cadena de búsqueda.

Fuente	Búsqueda sin filtros	CI1	CI3	CI2	CI5	Total	CE1	CE2	CI4	CI7	CI6	Total	CI1	CI1	Total
IEEE	120	120	40	40	40	40	16	12	8	8	8	8	8	8	8
WOL	160	160	50	50	50	50	25	17	17	12	5	5	5	5	5
ACM	445	389	217	23	23	23	2	2	2	2	2	2	2	2	2
SD	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Total	481	670	362	114	114	91	42	29	25	20	16	16	16	16	16

Fuente: Elaboración propia.

IEEE, **WOL**: Wiley Online Library, **ACM**:ACM Digital library, **SD**: Science Direct

En la segunda fase se aplicaron cinco criterios (CE1, CE2, CI4, CI6 y CI7), en la cual se incorporaron la mayor cantidad de filtros. Se eliminaron estudios que no se enfocaran en un área de la computación o en la enseñanza de la programación (CE1), con el propósito de enfocar la investigación en sistemas desarrollados específicamente para la enseñanza de programación.

Asimismo, se descartaron los trabajos teóricos que no presentaban resultados prácticos (CE2). Se conservaron los artículos que proponían soluciones computacionales (CI4), incorporaban retroalimentación automatizada o análisis de código (CI6), y abordaban el aprendizaje adaptativo o personalizado en la enseñanza de la programación (CI7). Esta fase dio como resultado un total de 20 artículos (véase Tabla 3)

Por último, durante la tercera fase se excluyeron artículos duplicados encontrados en diferentes bases de datos (CE4) y trabajos que no presentaban resultados precisos o cuyos hallazgos fueran derivados exclusivamente de investigaciones aplicadas (CE3). Esta tercera y última fase dejó un total de 16 artículos que cumplieron con los criterios establecidos (Tabla 3).

2.5. Resultados de criterios aplicados

Como se muestra en la Tabla 3, se presentan los resultados obtenidos tras aplicar los criterios de inclusión y exclusión definidos para esta revisión sistemática. Aunque se reporta en la Tabla 3 la identificación de 16 estudios relevantes según los criterios establecidos solo 13 de ellos describen sistemas implementados con un nivel de detalle suficiente para su sistematización comparativa. No obstante, uno de ellos fue descartado posteriormente por no ofrecer información técnica estructurable, por lo que la Tabla 4 incluye únicamente 12 sistemas. Los tres estudios restantes se conservaron como referencias conceptuales complementarias dentro del análisis y discusión, pero no fueron considerados en la tabla comparativa principal.

Tabla 4. Resumen de sistemas analizados con tipo de integración y métricas reportadas.

Autor/Sistema	Leng	Fuente	Tipo de Integración	Métricas/Resultados
Alasmari *et al.* (Python OCTS) [15]	Python	IEEE	LLM + interfaz visual	Mejora de comprensión
Frankford *et al.* (AI-Tutoring SE) [11]	Multi	ACM	Feedback LLM	Evaluación automática, rendimiento
Pirvulescu y Al-Sudani (Java Starter) [7]	Java	SD	Ontologías	Iniciación en programación
Alshaikh y Hewahi (ITS con DRL) [18]	N/E	IEEE	Tutor adaptativo con DRL	Tiempo de respuesta, errores comunes
Ross *et al.* (Programmer's Assistant) [7]	Multi	ACM	LLM conversacional	Seguimiento del contexto
Yang *et al.* (PyTutor) [1]	Python	SD	Pistas jerárquicas	Mejora inicial, comprensión
Wang *et al.* (ChatGPT API Rec.) [17]	Java	Wiley	Recomendador LLM	Precisión en API sugeridas
Cheah *et al.* (Feedback Copilot) [8]	Multi	SD	LLM + feedback escalable	Adaptación a nivel del estudiante
Park *et al.* (ChatGPT vs SO) [13]	Pyhon	Wiley	Comparativa LLM vs StackOverflow	Mayor rendimiento
Mattalo *et al.* (AI en CT y Mot.) [2]	Python	SD	Apoyo reflexivo con IA	Mejora en pensamiento computacional
Ahmed *et al.* (Knowledge Build.) [19]	Multi	SD	ChatGPT como herramienta reflexiva	Apoyo contextualizado
Chen *et al.* (Predicción Rend.) [15]	N/E	Wiley	SVM	Precisión 96% en predicción académica

Fuente: Elaboración propia.

3. Resultados

La siguiente sección presenta los resultados obtenidos de la revisión sistemática y de las investigaciones seleccionadas. Estos resultados responden a las preguntas planteadas al inicio del documento, con el objetivo de identificar oportunidades de mejora, aspectos clave para el desarrollo de sistemas de tutoría inteligente, posibles problemáticas asociadas y criterios para evaluar la calidad de los sistemas encontrados. A continuación, se responde a cada una de las preguntas de investigación:

PI1 ¿Cuál es el impacto del uso de sistemas de tutoría inteligente en el aprendizaje de la programación?

Diversos estudios sobre sistemas de tutoría inteligente han demostrado mejoras en estudiantes principiantes, debido a que, según las métricas evaluadas, la tasa de finalización de ejercicios incrementa en comparación con los estudiantes que no hicieron uso de esta herramienta [2]. También se identificó una mejor comprensión de conceptos, así como un aumento en la confianza del estudiante, logrando una mayor tasa de éxito al momento de estar trabajando con estos sistemas [2], [1]. Por otro lado, se observó que los estudiantes que utilizaron un LLM, como ChatGPT-3.5 obtuvieron un mejor desempeño en comparación con quienes utilizaron Stack Overflow o no contaron con guía alguna [10], lo cual sugiere que una guía personalizada, junto con retroalimentación inmediata, mejora la velocidad de aprendizaje y ofrece ventajas en la adquisición del conocimiento [13].

PI2 ¿Qué factores influyen en la personalización del aprendizaje en los sistemas de tutoría inteligente para programación?

Se identificaron cuatro características clave para construir sistemas capaces de ofrecer una enseñanza más personalizada. Una de ellas es la construcción de un modelo de estudiante, el cual permite identificar errores comunes, nivel de conocimiento y estilos de aprendizaje para adaptar la instrucción [16]. Otro punto clave es el modelo de lenguaje utilizado, ya que herramientas como GPT-3.5 adaptan de manera más efectiva la retroalimentación brindada según las necesidades del usuario [12]. También se encontró que la personalización se ve mejorada si está diseñada para adaptarse a un curso previamente construido, con materiales estructurados y métricas de evaluación definidas [1]. Por último, se evidenció que el diseño centrado en el usuario influye en su desempeño, pues aspectos como el orden de los contenidos, el tipo de retroalimentación o la claridad de las sugerencias afectan la experiencia del usuario [8], [7].

PI3 ¿Qué mecanismos permiten la personalización del aprendizaje dentro de los sistemas de tutoría inteligente aplicados a programación?

Los mecanismos identificados varían según el enfoque del sistema analizado. Por ejemplo, un mecanismo progresivo de pistas basado en *Deep Reinforcement Learning* (DRL) permite analizar el rendimiento del estudiante y brindar respuestas relacionadas con su concepto teórico correspondiente [18]. Algunos sistemas detectan errores comunes y ofrecen sugerencias sobre fragmentos específicos de código, proporcionando desde pseudocódigo hasta soluciones completas. Esto permite al estudiante construir su conocimiento paso a paso [12], [1]. Otros sistemas integran LLMs como asistentes conversacionales para mantener un canal de comunicación natural y continuo que simula la interacción con un tutor humano. Este mecanismo permite brindar retroalimentación con base en errores previos y mantener el contexto de la sesión [7]. Asimismo, algunos sistemas como Python OCTS [14] y AI-Tutoring [12] monitorean el tiempo de respuesta, errores frecuentes y número de intentos, adaptando así la dificultad de los ejercicios al dominio actual del estudiante [2], [1].

PI4 ¿Qué desafíos y limitaciones se han identificado en el diseño e implementación de los sistemas de tutoría?

Además de los desafíos éticos y pedagógicos asociados con la integración de LLMs en sistemas de tutoría inteligente (ITS), se identificaron limitaciones técnicas, como la dificultad para mantener el contexto en interacciones prolongadas [7]. Muchos sistemas están diseñados para consultas breves, lo que limita su capacidad de adaptación continua al progreso del estudiante [8]. Esto se ve agravado por la falta de acceso a datos educativos reales o a rúbricas de evaluación, lo cual reduce la efectividad tanto de la personalización como de la evaluación sistemática [12], [19]. Asimismo, aunque los LLMs pueden generar código, aún presentan limitaciones para detectar errores sintácticos o evaluar la lógica subyacente en las soluciones propuestas por los estudiantes [2], [17].

4. Discusión

En esta sección se exponen las oportunidades y retos identificados en los proyectos recuperados de la revisión, así como las soluciones propuestas por los autores. El objetivo es identificar áreas con oportunidades de mejora, así como potenciales avances que puedan implementarse para optimizar el desarrollo de estos sistemas inteligentes.

4.1. Resumen de sistemas identificados y sus características

La Tabla 4 presenta un resumen de los sistemas identificados en la literatura, incluyendo los lenguajes de programación utilizados, las tecnologías implementadas y los resultados reportados. Se observa que la mayoría de estos sistemas emplean Java y Python como lenguajes principales, además de integrar modelos de lenguaje (LLM) como GPT-3.

Algunos sistemas, como Python OCTS [14] y PyTutor [1] incorporan entornos que facilitan la visualización del estado de los algoritmos. Otros como el Programmer's Assistant [7], facilitan la interacción gracias a su integración con GPT-3, lo que mejora la comprensión del contexto durante la codificación.

Por otra parte, sistemas como AI-Tutoring SE [12] y Feedback Copilot [8] buscan proporcionar retroalimentación basada en LLMs, con el objetivo de evaluar el desempeño del estudiante en contextos educativos. Asimismo, algunas propuestas están orientadas al desarrollo del pensamiento computacional [2], la alfabetización en IA escolar [19] y la predicción de rendimiento académico [15].

Como resumen de los sistemas se identifica una tendencia hacia la personalización de cada usuario, incluyendo el uso de inteligencia artificial para buscar reducir errores y se busca generar interfaces más comprensibles para los estudiantes. Sin embargo, aún persisten retos en la integración de todas estas tecnologías para lograr una mejor sinergia entre las tecnologías, la mitigación de alucinaciones generadas por los LLMs y la generación de una interfaz amigable con el usuario.

4.1.1. Intelligent Tutoring Systems

La integración efectiva de sistemas inteligentes representa un desafío constante, debido a la diversidad de tecnologías que deben considerarse para su correcto funcionamiento [9]. Sin embargo, a lo largo de los años se ha logrado un estándar para estos sistemas, los cuales incluyen retroalimentación inmediata y generan asistencia para el estudiante durante el desarrollo de código [18], [14].

Estos sistemas, al integrar inteligencias artificiales generativas, ofrecen retroalimentación que simula la función de un tutor humano, proporcionando pistas durante el proceso iterativo de resolución de problemas [12], [1]. Algunas investigaciones recientes han demostrado que esta forma de asistencia mejora la comprensión del estudiante [13], [2] y promueve su autonomía, gracias a una interacción continua adaptada a su progreso y errores previos [7], [8].

4.1.2. Adaptive Learning

De forma complementaria, otra estrategia consiste en adaptar la retroalimentación según las respuestas del usuario, lo que permite minimizar tanto el tiempo como los errores potenciales cometidos, facilitando así la comprensión del tema abordado [1]. Esta adaptación resulta más efectiva cuando se considera el nivel de conocimiento del estudiante, permitiendo una retroalimentación personalizada y contextualizada [8].

Combinar estas características en un sistema de tutoría inteligente resulta en una mejora sustancial para los estudiantes, debido a que al momento de integrar un mecanismo que permita recabar la información sobre los errores que ha cometido y procesarla, se puede brindar una retroalimentación específica para la forma de entender del estudiante [2]. Algunos sistemas, como *AI-Tutoring*, implementan estos mecanismos para ofrecer asistencia paso a paso en función del historial de desempeño del usuario [12].

Comúnmente, estas respuestas se integran mediante LLMs como GPT-3.5, los cuales permiten mantener el contexto y simular la interacción con un tutor humano [7]. Estos modelos también permiten adaptar la retroalimentación al estilo de razonamiento del estudiante, mejorando así la experiencia personalizada de aprendizaje [13].

La combinación de este tipo de características es beneficiosa, ya que permite abordar temas de una manera más estructurada y reducir el tiempo necesario para identificar los puntos de error, los cuales frecuentemente pasan desapercibidos para el propio estudiante [19], [18].

4.1.3. Limitaciones y desafíos de los sistemas basados en LLMs

Adicionalmente, estos sistemas integran intérpretes de código que cumplen la función de compilador sin la necesidad de que el usuario tenga que estar interactuando con diversos sistemas a la vez, lo que hace más fluida la interacción del usuario [14].

Aunque la integración de un LLM permite en automático brindar, sugerir o generar fragmentos de código, hay dos puntos importantes a tomar en cuenta. El primer aspecto corresponde a la forma en que se presenta esta integración en la interfaz, ya que solo genera líneas de código como parte de la retroalimentación textual, si bien entiende el código que se está modificando, el modelo entrega líneas de código sin explicar la lógica que sustenta dicha funcionalidad [7]. Si bien algunos modelos comprenden el código que se está modificando, su retroalimentación carece de una estructura pedagógica adaptada al nivel del estudiante [13].

En segundo lugar, existe la posibilidad de que el LLM cometa errores o genere respuestas incorrectas, fenómeno conocido como “alucinaciones”, lo que puede causar confusión o errores mayores en el proceso de aprendizaje [8]. Es por esta razón que algunos sistemas, para mitigar esto, en vez de optar por dejar que el modelo de inteligencia artificial genere las respuestas, se integran con documentación externa respectiva al lenguaje de programación utilizado, dando propuestas de métodos. Esto convierte al sistema en una herramienta inteligente

centrada en la documentación del lenguaje en uso [17]. Aunque esto no representa un impedimento, los sistemas que optan por esta estrategia no aprovechan plenamente el potencial de las demás funcionalidades del LLM [12].

4.1.4. Apoyo visual e interacción fluida con el sistema

Una de las áreas con mayor potencial de mejora en los tutores inteligentes es la incorporación de elementos gráficos que funcionen como complemento a las retroalimentaciones textuales, buscando mejorar la experiencia del usuario durante la navegación, la edición y la ejecución de código sean más fluidas [8], [7]. Incorporar este tipo de interacciones podría ayudar a tener una mejor sinergia entre tecnologías, ya que esta representación visual podría hacer evidentes las respuestas erróneas o alucinaciones del LLM [12]. De esta forma, una interfaz bien integrada complementa la respuesta, refuerce la comprensión conceptual, identifique errores y apoye en que el estudiante pueda desarrollar su lógica de resolución de una forma más efectiva [2], [19].

Como preámbulo a la siguiente sección, la Figura 1 presenta un cuadro comparativo en el que se identifican las características clave de los sistemas analizados. Este análisis visual permite contextualizar las oportunidades de mejora discutidas a continuación.

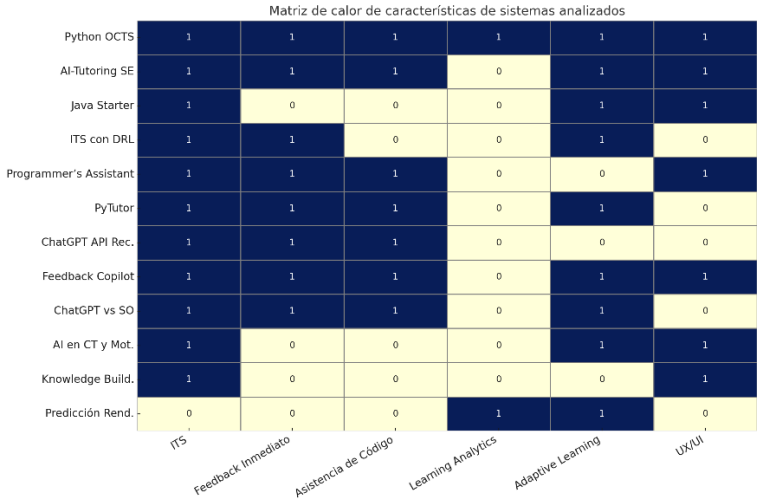

Figura 1. Características presentes en los sistemas analizados.

4.1.5. Oportunidades de mejora y direcciones futuras

A partir del análisis de los sistemas identificados y sus características, se destacan varias áreas con potencial de mejora en el desarrollo de tutores inteligentes basados en inteligencia artificial.

- Integración Multimodal: Este aspecto permitiría desarrollar sistemas que, al integrar diversas formas de retroalimentación (visual, textual e interactiva), faciliten la comprensión de conceptos abstractos mediante representaciones dinámicas. Diversos estudios destacan que la retroalimentación visual generada por modelos de IA puede apoyar la representación de estructuras abstractas, como árboles de sintaxis o diagramas de flujo, mejorando así la comprensión del estudiante [1], [8]. Además, los tutores inteligentes que adaptan el tipo de retroalimentación según el nivel del estudiante han demostrado ser efectivos para aumentar la claridad conceptual y el ritmo de aprendizaje [14]. Finalmente, algunos sistemas que integran IA generativa logran aumentar el compromiso del estudiante al brindar explicaciones más dinámicas, útiles para mantener la atención y facilitar la comprensión contextualizada [12].
- Mejor manejo de datos: Muchos sistemas dependen de modelos de lenguaje como GPT-3.5, los cuales pueden presentar errores de predicción conocidos como “alucinaciones” generando respuestas incorrectas o poco confiables [8]. En investigaciones futuras, sería clave lograr una mejor integración que permita verificar la calidad de las respuestas generadas, utilizando bases de datos como contexto de validación [17].

- Mayor Personalización del Aprendizaje: Aunque varios sistemas ya contemplan algún nivel de adaptabilidad, aún es limitada la personalización profunda basada en estilos cognitivos, ritmo de aprendizaje y tipos de errores específicos. Incorporar sistemas de *learning analytics* que analicen patrones de comportamiento podría mejorar significativamente esta dimensión [1]. Unificar múltiples herramientas en un solo sistema facilitaría el apoyo al aprendizaje, al proporcionar mayor operabilidad y datos suficientes para personalizar la experiencia educativa [12].
- Diseño Centrado en el Usuario (UX): Una constante en los sistemas analizados es que la experiencia de usuario representa un reto en el diseño de tutores inteligentes [2]. Las interfaces deben complementar el proceso de codificación y contribuir a reducir la curva de aprendizaje en estudiantes principiantes [19].

Estas oportunidades marcan el rumbo y los desafíos para desarrollar sistemas más robustos y centrados en el estudiante. El diseño de tutores inteligentes requiere un enfoque interdisciplinario que combine diversas áreas para facilitar su implementación como herramienta complementaria en el aula.

5. Conclusión

El trabajo futuro orientado a integrar sistemas inteligentes para la enseñanza de programación deberá centrarse en lograr una integración tecnológica más complementaria. Aunque algunos enfoques actuales incorporan funciones visuales durante la iteración del código o integran modelos de lenguaje de gran escala (LLMs) como módulos generales, aún persisten problemas como errores de interpretación o la falta de continuidad entre sesiones.

Ante este panorama, una línea de mejora relevante consiste en adoptar un enfoque de Diseño Centrado en el Usuario (DCU), que permita desarrollar interfaces adaptadas a las necesidades reales del estudiante, mejorando la usabilidad, la retroalimentación y la personalización de la experiencia de aprendizaje. Además, visualizar el flujo del programa, los cambios en las variables o los errores detectados facilita la comprensión del código, especialmente en estudiantes principiantes.

Por tanto, las investigaciones futuras deberían considerar que estos sistemas inteligentes, en lugar de ser soluciones autónomas para resolver problemas, se conciban como herramientas educativas que generen evidencia útil para que docentes e investigadores puedan identificar áreas de mejora en sus estudiantes. Esto favorecería una integración más efectiva entre el usuario y la inteligencia artificial, fortaleciendo tanto el desarrollo de habilidades como el seguimiento personalizado del progreso académico.

6. Agradecimientos

El autor expresa su sincera gratitud a SECIHTI por su valioso apoyo financiero, el cual ha sido fundamental para llevar a cabo y completar esta investigación. Su contribución ha permitido avanzar en la generación de conocimiento y enriquecer el campo de la ciencia y la tecnología.

7. Referencias

[1] Yang, A. C. M., Lin, J.-Y., Lin, C.-Y., Ogata, H. (2024). Enhancing Python learning with PyTutor: Efficacy of a ChatGPT-based intelligent tutoring system in programming education. *Computers and Education: Artificial Intelligence, 7*, 1-15. https://doi.org/10.1016/j.caeai.2024.100309

[2] Yilmaz, R., Karaoglan Yilmaz, F. G. (2023). The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence, 4*, 1-14. https://doi.org/10.1016/j.caeai.2023.100147

[3] Mekouar, L. (2022). *The art of teaching programming languages: Challenges and accomplishments*. In *IEEE World Engineering Education Conference (EDUNINE)*. Santos, Brazil. https://doi.org/10.1109/EDUNINE53672.2022.9782372

[4] Senanayake, S., Karunanayaka, K., Ekanayake, K. V. J. P. (2024). *Review on AI assistant systems for programming language learning in learning environments*. *IEEE 8th SLAAI-ICAI International Conference on Artificial Intelligence*. Ratmalana, Sri Lanka. https://doi.org/10.1109/SLAAI-ICAI63667.2024.10844969

[5] Liu, C., Wang, G.-C., Wang, H.-F. (2025). The application of artificial intelligence in engineering education: A systematic review. *IEEE Access, 13*, 17895–17910. https://doi.org/10.1109/ACCESS.2025.3360423F

[6] Paladines, J., Ramírez, J. (2020). A systematic literature review of intelligent tutoring systems with dialogue in natural language. *IEEE Access, 8*, 164246–164267. https://doi.org/10.1109/ACCESS.2020.3021383

[7] Ross, S. I., Martinez, F., Houde, S., Muller, M., Weisz, J. D. (2023). *The programmer's assistant: Conversational interaction with a large language model for software development*. ACM 28th International Conference on Intelligent User Interfaces. Sydney, Australia. https://doi.org/10.1145/3581641.3584037

[8] Pozdniakov, S., Brazil, J., Abdi, S., Bakharia, A., Sadiq, S., Gašević, D., Denny, P., Khosravi, H. (2024). Large language models meet user interfaces: The case of provisioning feedback. *Computers and Education: Artificial Intelligence, 7*, 1-20. https://doi.org/10.1016/j.caeai.2024.100289

[9] Chen, L., Chen, P., Lin, Z. (2020). Artificial intelligence in education: A review. *IEEE Access, 8*, 75264–75278. https://doi.org/10.1109/ACCESS.2020.2988510

[10] Cheah, Y. H., Lu, J., Kim, J. (2025). Integrating generative artificial intelligence in K-12 education: Examining teachers' preparedness, practices, and barriers. *Computers and Education: Artificial Intelligence, 8*, 1-12. https://doi.org/10.1016/j.caeai.2025.100363

[11] Kim, T.-S., Ignacio, M. J., Yu, S., Jin, H., Kim, Y.-G. (2024). UI/UX for generative AI: Taxonomy, trend, and challenge. *IEEE Access, 12*, 179891–179904. https://doi.org/10.1109/ACCESS.2024.3502628

[12] Frankford, E., Sauerwein, C., Bassner, P., Krusche, S., Breu, R. (2024). *AI-tutoring in software engineering education: Experiences with large language models in programming assessments*. ACM 46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). Lisbon, Portugal. https://doi.org/10.1145/3639474.3640061

[13] Park, A., Kim, T. (2025). Code suggestions and explanations in programming learning: Use of ChatGPT and performance. *The International Journal of Management Education, 23* (2). https://doi.org/10.1016/j.ijme.2024.101119

[14] Alasmari, O. A. F., Singer, J., Bikanga Ada, M. (2024). *Python OCTS: Design, implementation, and evaluation of an online coding tutorial system prototype*. IEEE World Engineering Education Conference (EDUNINE). Guatemala City, Guatemala. https://doi.org/10.1109/EDUNINE60625.2024.10500548

[15] Ahmed, E. (2024). Student performance prediction using machine learning algorithms. *Applied Computational Intelligence and Soft Computing, 2024* (1). https://doi.org/10.1155/2024/4067721

[16] Pirvulescu, M., Al-Sudani, S. (2023). *The Java Starter: An ontology-based tutoring system for Java beginners*. IEEE 16th International Conference on Developments in eSystems Engineering (DeSE). Istanbul, Turkiye. https://doi.org/10.1109/DeSE60595.2023.10469264

[17] Wang, Y., Xue, W., Huang, Q., Jiang, B., Zhang, H. (2025). Exploring ChatGPT's potential in Java API method recommendation: An empirical study. *Journal of Software: Evolution and Process, 37* (1). https://doi.org/10.1002/smr.2765

[18] Alshaikh, F., Hewahi, N. (2025). Intelligent tutoring system: A pedagogical model approach based on deep reinforcement learning. *IAENG International Journal of Computer Science, 52* (4), 1196–1212.

[19] Chen, B., Zhu, X., Díaz del Castillo H., F. (2023). Integrating generative AI in knowledge building. *Computers and Education: Artificial Intelligence, 5*, 1-12. https://doi.org/10.1016/j.caeai.2023.100184

[20] Kitchenham, B., Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (EBSE Technical Report No. EBSE-2007-01). Keele University & University of Durham. https://docs.edtechhub.org/lib/EDAG684W

