Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

Asistente de IA para Visual Studio Code para el aprendizaje de programación

Trabajo fin de estudio presentado por:	Daniel Prol Pérez
Línea de investigación:	Ingeniería del software
Director/a:	D. Luis Pedraza Gómara
Fecha:	Marzo 2025
Repositorio:	https://github.com/dprol/Buddy
Vídeo de demostración:	Funcional: https://youtu.be/Wqar0u3LG50

Resumen

Comprender el problema es fundamental para los estudiantes que aprenden programación. Los modelos de lenguaje como ChatGPT ofrecen ayuda instantánea, pero proporcionan respuestas directas con código, lo que puede dificultar una comprensión profunda del problema.

Tras realizar un análisis en profundidad de los distintos asistentes copilotos en el mercado y observar que no existen soluciones con un objetivo pedagógico, se desarrolla Buddy, un asistente de aprendizaje basado en un modelo de lenguaje que acompaña a los estudiantes en el proceso de resolver problemas de programación sin revelar directamente soluciones de código y sin que el estudiante tenga que formular *prompts*.

Se implementa el asistente sobre una interfaz interactiva integrada en Visual Studio Code. El asistente cuenta con herramientas para proporcionar explicaciones de conceptos esenciales de la programación, generar ejemplos en pseudocódigo y diagramas de flujo, sugerir pistas iniciales, recomendar próximos pasos basados en el código actual del estudiante, explicar soluciones posibles y generar preguntas de seguimiento para profundizar la comprensión.

Se lleva a cabo una evaluación de la usabilidad del asistente mediante el protocolo *think-aloud*. Además, se realiza un cuestionario final para obtener una perspectiva más cuantitativa.

Se concluye que el asistente cumple el objetivo marcado en el trabajo permitiendo a los estudiantes resolver los problemas de programación y mejorar la experiencia en el proceso de aprendizaje.

Palabras clave: Enseñanza de la programación, Large Language Models, aprendizaje en IDE, IA generativa, Educación en Informática

Abstract

Understanding the problem is critical for students learning programming. Language models such as ChatGPT offer instant help, but provide direct answers with code, which can make it difficult to truly understand the problem.

After performing an in-depth analysis of the various copilot assistants on the market and observing that there are no solutions with a pedagogical purpose, Buddy is developed, a language model-based learning assistant that accompanies students in the process of solving programming problems without directly revealing code solutions and without the student having to formulate prompts.

The assistant is implemented on an interactive interface integrated into Visual Studio Code. The assistant has tools to provide explanations of fundamental programming concepts, generate examples in pseudocode and flowcharts, suggest initial hints, recommend next steps based on the student's current code, explain possible solutions, and generate follow-up questions to deepen understanding.

An evaluation of the usability of the assistant is carried out using the *think-aloud* protocol. In addition, a final questionnaire is conducted to obtain a more quantitative perspective.

The assistant fulfills the objective set out in the project by allowing students to solve programming problems and improve the experience in the learning process.

Keywords: Programming education, Large Language Models, in-IDE learning, generative AI, Computing Education

Índice de contenidos

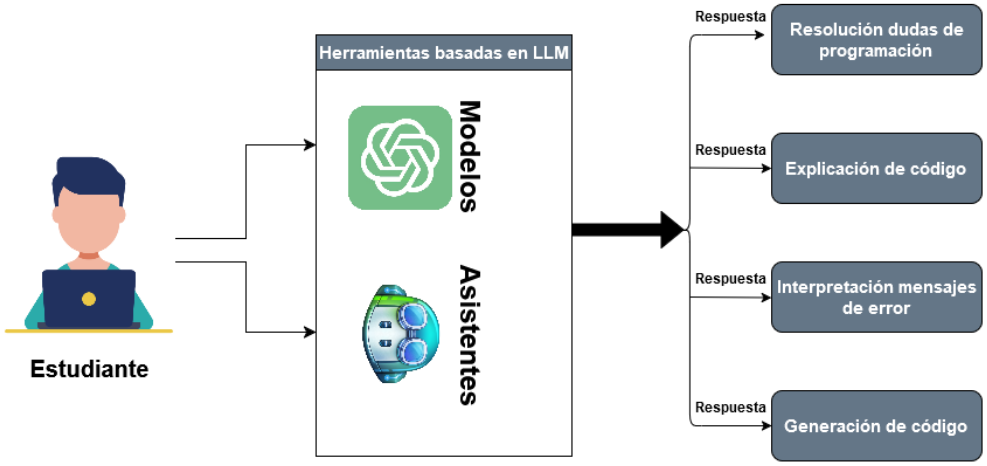
Índice de figuras

Índice de tablas

1. Introducción

En el área de Educación en Computación (CED, por sus siglas en inglés: *Computing Education*), la programación es una actividad fundamental para la resolución de problemas. Sin embargo, los estudiantes principiantes encuentran dificultades para aprender a programar bien que incluyen la comprensión del problema y la falta de confianza en su capacidad para resolver problemas. Estos desafíos se suman a los problemas tradicionales del aprendizaje de programación, como las barreras relacionadas con la sintaxis y la pronunciada curva de aprendizaje inicial. Aunque los LLMs y asistentes copilotos como GitHub Copilot (figura 1) pueden ayudar, también plantean retos adicionales, como la dificultad de los estudiantes para describir con precisión sus necesidades y su limitada capacidad para evaluar la calidad de las respuestas generadas por la IA. En este contexto, el presente trabajo se centra en el desarrollo de Buddy, un asistente de IA diseñado para Visual Studio Code, que proporciona apoyo a estudiantes principiantes en programación mediante herramientas que simplifican la comprensión de los problemas de programación. Buddy está diseñado para ofrecer sugerencias sin necesidad de solicitudes explícitas por parte del usuario. De esta forma, este Trabajo de Fin de Grado busca contribuir a la mejora del aprendizaje de la programación de los estudiantes en sus primeras etapas.

Figura 1. *LLMs en educación de la informática*



Fuente: Elaboración propia

1.1. Motivación

La idea de Buddy surge de la observación de los retos comunes que tienen los estudiantes en las asignaturas de programación, especialmente aquellos que carecen de experiencia previa. A través del análisis del **Estado del Arte**, se ha identificado la necesidad de una herramienta que no solo facilite la escritura de código, sino que también promueva la autonomía de los estudiantes en un entorno guiado. Aunque existen numerosas herramientas que ofrecen funcionalidades avanzadas (como se pueden apreciar en la **figura 2**), muchas de ellas carecen de un enfoque pedagógico específicamente diseñado para principiantes, lo que puede fomentar una dependencia excesiva por parte de los estudiantes. Por otro lado, aunque las herramientas existentes en IDE, como los depuradores, ofrecen cierto grado de apoyo, no siempre son comprensibles para los que están dando sus primeros pasos en programación. La posibilidad de desarrollar un producto que aborde estas limitaciones, integrando los conocimientos adquiridos durante el grado y aprovechando tecnologías emergentes como los LLMs, constituye una de las principales motivaciones para este trabajo.

Figura 2. *Buddy en el mercado de los asistentes*

Fuente: Elaboración propia

1.2. Planteamiento del trabajo

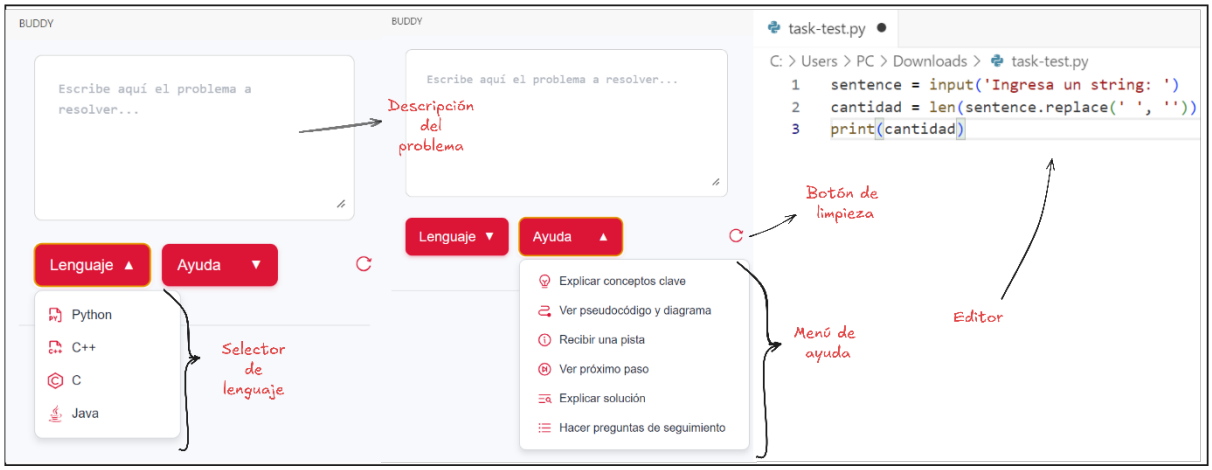
El aprendizaje de la programación está lleno de barreras que incluyen la falta de familiaridad con la sintaxis, la interpretación de errores y la dificultad para comprender conceptos. Estas barreras, si no se abordan adecuadamente, pueden producir frustración en los estudiantes.

Muchas de las herramientas del mercado requieren que el usuario identifique y comunique sus necesidades, lo que puede ser desafiante para estudiantes sin experiencia.

En respuesta a esta problemática, el presente trabajo propone un entorno guiado (**figura 3**) que ofrece sugerencias y proporciona explicaciones claras del problema, con el objetivo de facilitar el aprendizaje y reducir las barreras inherentes al programar.

El propósito del asistente es guiar al estudiante en el proceso de resolver problemas, explicar cómo abordarlos, proponer enfoques adecuados que permitan desglosar los problemas en partes manejables y orientar sobre las estructuras más apropiadas, tomando como base la definición del problema y el lenguaje de programación empleado.

Figura 3. *Entorno guiado de Buddy*



Fuente: Elaboración propia

Con una arquitectura basada en un stack moderno y un diseño centrado en el usuario, este asistente para Visual Studio Code pretende ser una herramienta diferenciadora en la experiencia de aprendizaje de programación.

1.3. Estructura del trabajo

A partir de este capítulo, el resto del trabajo se estructura de la siguiente manera:

En el capítulo 2 se detalla el contexto que fundamenta este trabajo. Se aborda el impacto de la IA en el desarrollo de software y los nuevos enfoques pedagógicos, tanto en el aprendizaje como en la enseñanza de programación. También se hace una breve introducción a los LLMs.

Asimismo, en el capítulo 3 del Estado del Arte se realiza un estudio de herramientas existentes en el mercado relacionadas con el aprendizaje asistido por IA, abarcando tanto asistentes desarrollados en la academia como en la industria y destacando sus principales características, así como se realiza un análisis comparativo de arquetipos de herramientas IA.

El capítulo 4 se centra en los objetivos del trabajo y describe en detalle la metodología empleada para el desarrollo del asistente. Se decide adoptar la metodología de Diseño Centrado en el Usuario, en particular, se aplica el modelo de Doble Diamante de *Design Thinking*.

El capítulo 5 presenta las funcionalidades y los casos de uso previstos para la primera versión del asistente, abarcando el escenario principal para cada caso de uso y el diagrama de secuencia asociado al flujo de las operaciones que se realizan. Además, se define el alcance de la primera versión y se detallan los requisitos no funcionales.

El capítulo 6 se centra en el diseño y la implementación del asistente, tomando como base la especificación desarrollada en el capítulo anterior. En este apartado, se explica la arquitectura del sistema y se detallan las tecnologías utilizadas junto con las razones para su elección. Además, se profundiza en los prototipos diseñados para la fase de comprensión de la metodología y en el resultado digital final, profundizando en el desarrollo de los distintos componentes del sistema, describiendo la estructura y los elementos programados tanto en la parte del servidor como en la parte del cliente.

El capítulo 7 está dedicado a la evaluación del asistente. Una vez finalizado el desarrollo, se llevan a cabo una serie de pruebas de usabilidad. En esta etapa se desglosan las diferentes entrevistas realizadas a los usuarios con sus resultados. Se seleccionan cinco participantes que cumplen con el perfil de los usuarios potenciales del producto. Se lleva a cabo un análisis de las interacciones con Buddy en estas cinco entrevistas a estudiantes. Además, se realiza un cuestionario final para obtener más perspectivas.

Finalmente, el trabajo concluye con las conclusiones y líneas futuras. En estos apartados se sintetizan las principales conclusiones obtenidas a lo largo del desarrollo del trabajo. Asimismo, se presentan posibles líneas de trabajo futuro orientadas a ampliar y mejorar las capacidades de Buddy.

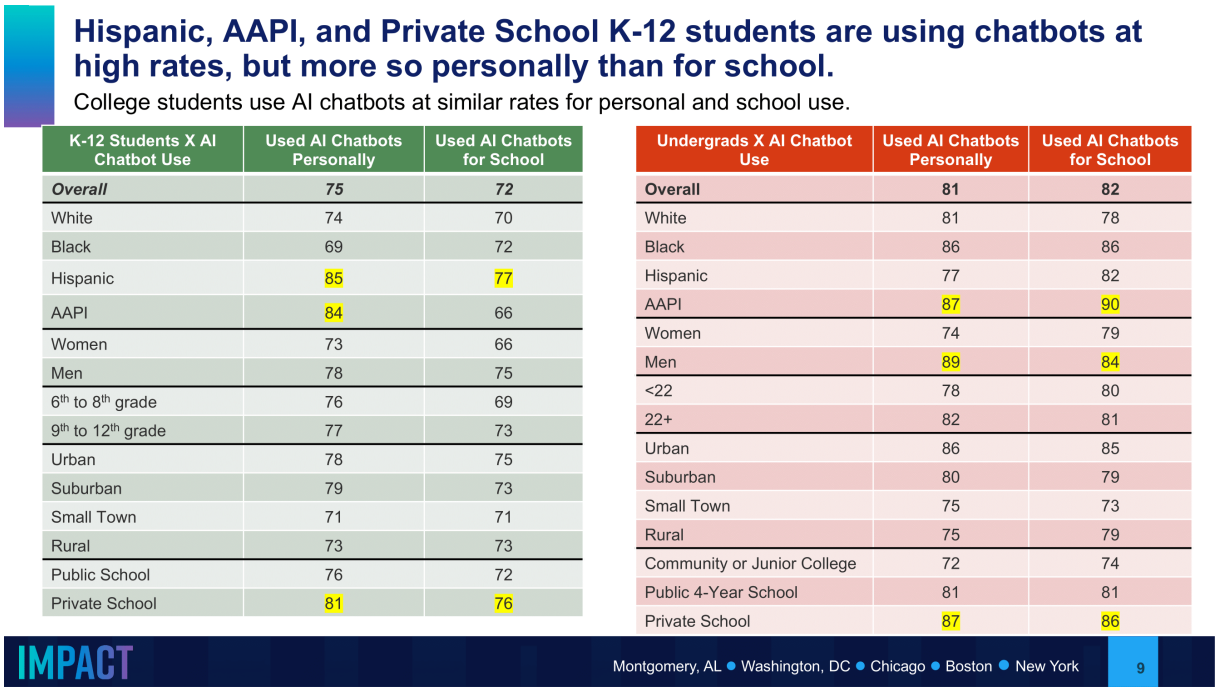
2. Contexto

En este apartado se exponen los aspectos generales que fundamentan el desarrollo de Buddy. Estos elementos constituyen la base conceptual sobre la cual se estructura este trabajo. En la siguiente sección de Análisis del contexto, se presenta una visión general sobre la influencia de los asistentes de IA en la educación en informática.

2.1. Análisis del contexto

El uso de la IA se ha convertido en una actividad prácticamente omnipresente entre los estudiantes universitarios. En la encuesta realizada por Impact Research presentada en la figura 4, se puede observar que el 82% de ellos declaró haber utilizado herramientas de IA en su vida académica, lo que demuestra que este grupo las ha adoptado muy rápido. Según Peixoto et al. (2024), el 84.8% de los estudiantes universitarios ya utiliza algún tipo de ayuda en programación, siendo los LLMs la herramienta más frecuente (empleada por 50-70 de los 92 estudiantes encuestados).

Figura 4. *Uso de chatbots de IA entre estudiantes universitarios*



Hispanic, AAPI, and Private School K-12 students are using chatbots at high rates, but more so personally than for school.

College students use AI chatbots at similar rates for personal and school use.

K-12 Students X AI Chatbot Use	Used AI Chatbots Personally	Used AI Chatbots for School
Overall	***75***	***72***
White	74	70
Black	69	72
Hispanic	85	77
AAPI	84	66
Women	73	66
Men	78	75
6th to 8th grade	76	69
9th to 12th grade	77	73
Urban	78	75
Suburban	79	73
Small Town	71	71
Rural	73	73
Public School	76	72
Private School	81	76

Undergrads X AI Chatbot Use	Used AI Chatbots Personally	Used AI Chatbots for School
Overall	**81**	**82**
White	81	78
Black	86	86
Hispanic	77	82
AAPI	87	90
Women	74	79
Men	89	84
<22	78	80
22+	82	81
Urban	86	85
Suburban	80	79
Small Town	75	73
Rural	75	79
Community or Junior College	72	74
Public 4-Year School	81	81
Private School	87	86

Fuente: Impact Research. (2024, p. 9)

Este fenómeno plantea preguntas fundamentales: ¿sigue siendo relevante aprender a programar cuando la IA es capaz de generar código? ¿estarán atrofiando la capacidad de programar los asistentes de IA o ayudan a programar mejor? ¿cómo lograr un equilibrio que permita aprovechar los beneficios de la IA en la educación sin comprometer el desarrollo integral del alumnado? En este contexto, resulta crucial analizar cómo los estudiantes están integrando la IA en sus procesos de aprendizaje y, asimismo, explorar cómo pueden formarse como usuarios con un enfoque crítico y responsable en una era marcada por la influencia de la IA.

En los diferentes niveles educativos, la integración de la IA representa tanto una oportunidad como un reto. Como plantea el profesor de Wharton Ethan Mollick en su libro *Co-Intelligence: Living and Working with AI* "ahora contamos con otra cointeligencia: la IA puede ayudarnos como un compañero de pensamiento para mejorar nuestra toma de decisiones y reflexionar acerca de nuestras elecciones, en lugar de limitarse a elegir por nosotros" *(Mollick, 2024)*. Al mismo tiempo, Mollick también subraya que "resulta crucial mantener un ojo crítico y tratar a la IA como una herramienta que funciona para ti. Definiendo su personalidad, entablando un proceso de edición colaborativa y proporcionándole orientación continuamente, puedes aprovecharla como forma de cointeligencia colaborativa" *(Mollick, 2024)*. En este sentido, la incorporación de la IA en el aprendizaje de programación no debe considerarse un sustituto del pensamiento crítico, sino una oportunidad para ampliar y potenciar las capacidades únicas que solo los humanos pueden aportar.

En la misma línea, Porter y Zingaro, autores de *Learn AI-Assisted Python Programming*, destacan que estas herramientas funcionan como "copilotos" y no "autopilotos". Es decir, permiten expresar en lenguaje natural una expresión y que la máquina traduzca estas intenciones a lenguaje máquina (Porter y Zingaro, 2024).

Por su parte, Salman Khan, en su libro *Brave New Words*, señala que "los estudiantes más exitosos serán aquellos que utilicen la IA para establecer conexiones conceptuales que les ayuden a desarrollar ideas. Aquellos que aprendan a usar la IA de forma ética y productiva no solo podrán aprender a un ritmo exponencialmente más rápido que los demás, sino también de una manera que les permita mantenerse competitivos a lo largo de sus carreras. Además, tendrán una comprensión más profunda de las materias que estudien, porque sabrán cómo

encontrar respuestas a sus preguntas. Lejos de atrofiarse, su curiosidad se verá fortalecida" (Khan, 2024).

Por lo tanto, es evidente que la educación, a todos los niveles, afronta un reto mayúsculo en "la era de ChatGPT". Uno de los debates más recurrentes gira en torno a cómo convivir con estos modelos. La integración de herramientas como ChatGPT en la universidad exige una comprensión profunda de sus capacidades y limitaciones. Según Ribera y Montesdeoca, en su libro *ChatGPT y educación universitaria*, aunque "tardaremos unos años en aprovechar estas herramientas con plena productividad, podemos recorrer este camino con un poco más de calma y conocimiento para conseguir una implementación más satisfactoria de esta tecnología" (Ribera & Montesdeoca, 2024). Los autores destacan que "la IA va a ser una realidad en el futuro de nuestro alumnado" (Ribera y Montesdeoca, 2024), lo que hace esencial que el profesorado desarrolle estrategias pedagógicas orientadas a maximizar las oportunidades que esta tecnología ofrece, al mismo tiempo que se minimizan los riesgos asociados a su uso, como las complicaciones para diferenciar entre un trabajo humano y uno producido por ChatGPT, la pérdida de memoria o las preocupaciones por la integridad académica.

2.1.1. Impacto en los desarrolladores

La aparición de asistentes de IA está redefiniendo el papel del programador. Lejos de sentirse amenazada, la comunidad desarrolladora está adoptando estos asistentes como potenciadores de productividad, entendiendo que el futuro no consiste en su sustitución completa, sino en la evolución del proceso de desarrollo. Los programadores están transitando desde la generación de código hacia un papel más estratégico que implica habilidades como:

- **Evaluación crítica de las soluciones generadas por IA:** revisar cuidadosamente las respuestas de la IA en busca de errores, inconsistencias o alucinaciones. Este nuevo modo de interacción humano-máquina subraya la importancia de que los programadores desarrollen habilidades fundamentales, como la comprensión y la validación de los resultados generados por la IA.
- **Formulación efectiva de consultas** (*prompt engineering*): el *prompting* es el proceso de formular preguntas de calidad o consultas de manera que la IA pueda entender y

generar respuestas útiles. Cuanto más claro y específico sea el *prompt*, mejor será la respuesta recibida.

- **Gestión de la calidad**: aunque la IA puede generar código funcional, este no siempre es fácilmente reutilizable o compatible con el código existente.

Durante los últimos meses, diversas voces de la industria tecnológica han enriquecido esta conversación. A continuación, se destacan algunas de sus contribuciones:

Jodie Burchell, científica de datos en JetBrains[1], afirma que "la función de un desarrollador no es programar, es resolver problemas complejos en un contexto de negocio, independiente de la herramienta de turno" (Software Engineering Daily, 2024). Por su parte, Quinn Slack, CEO de Sourcegraph[2], señala que los desarrolladores deben asumir roles más estratégicos mientras la IA se encarga de tareas rutinarias y actúa como apoyo en decisiones complejas. Slack enfatiza que "el desarrollo futuro de la IA en programación depende de su capacidad para comprender y utilizar el contexto completo del código, tal como lo haría un humano al tomar decisiones de arquitectura" *(Changelog Interviews*, 2024).

Freddy Vega, en el canal de Nico Orellana, destaca que "la ingeniería de software no se limita a escribir código, ya que su esencia radica en comprender los requisitos del cliente y traducirlos en soluciones claras. Esto requiere un nivel de interpretación y contexto que las herramientas actuales, como Devin[3], aún no alcanzan" (Nico Orellana, 2024). En la misma línea, Antonio Feregrino subraya que "la programación es un proceso creativo que no solo implica lógica y automatización, sino también un toque de intuición, creatividad, empatía, comprensión de los requisitos y visión a largo plazo en entornos cambiantes. Hay aspectos intrínsecamente humanos, como la colaboración en equipo, el debate para llegar a consensos o la comunicación con el cliente, que estas herramientas todavía no dominan" (Feregrino, 2024).

En este nuevo contexto, otro de los retos de esta convivencia está relacionado con garantizar que se proporcione suficiente contexto para que las herramientas sean efectivas. Un caso práctico que ilustra esta situación es el experimento realizado por Héctor de León en su canal

[1] JetBrains: https://www.jetbrains.com/es-es/
[2] Sourcegraph: https://sourcegraph.com
[3] Devin: https://app.devin.ai/

de YouTube, donde evaluó a GPT-4 en la creación de un *backend* completo con SQL Server y .NET. Aunque la IA pudo generar código funcional rápidamente, su efectividad mejoró significativamente cuando se especificó explícitamente el uso de .NET 8 en lugar de versiones anteriores. Sin embargo, incluso con este contexto, se identificaron inconsistencias y prácticas obsoletas, lo que subraya la importancia del conocimiento del desarrollador para evaluar y adaptar las sugerencias generadas por la IA (Héctor de León, 2024).

Además, surge la necesidad de adoptar enfoques responsables en el diseño y la implementación de herramientas de IA. Como enfatizó Scott Hanselman en la *Epic Web Conference de 2024*: "No es trabajo del modelo decidir qué hacer; es nuestro trabajo como diseñadores de interfaz, como personas de producto, decidir cómo debería comportarse" (*Epic Web Conference, 2024)*. Esta reflexión destaca que la responsabilidad de las acciones de la IA recae en los desarrolladores e investigadores que diseñan estos sistemas, quienes deben definir cuidadosamente su comportamiento para minimizar riesgos y garantizar un uso ético y seguro. Así lo advierten también Alex Rayón, CEO de Brain & Code, y Alberto Núñez, profesor de ESADE, en el artículo "Los dilemas (éticos) del ChatGPT", publicado en Ethic (Núñez, A., & Rayón, A., 2023).

Finalmente, Kent C. Dodds añade que estos asistentes no reemplazan al desarrollador, sino que lo amplifican: "Me gusta pensar en los asistentes de IA como un multiplicador de fuerza. Si no hay fuerza, entonces no hay nada que multiplicar. Necesitas ser tú quien sostenga el martillo y clave el clavo" (Dodds, 2024).

2.1.2. Impacto en la enseñanza-aprendizaje de la programación

La integración de la IA generativa en la educación en informática exige una revisión de las prácticas docentes tradicionales. Un estudio reciente sobre la implementación de IA generativa en cursos de programación propone un marco estructurado para los docentes, que incluye desde la familiarización con estas herramientas hasta el establecimiento de políticas claras sobre su uso (Alpizar-Chacon, 2024). Además, Lau & Guo plantean estrategias específicas para aprovechar la IA generativa en la enseñanza, como la revisión del código generado por estas herramientas para fomentar habilidades de juicio crítico y reflexión en los estudiantes (Lau & Guo, 2023).

La IEEE, en su revista *Spectrum*[4], destaca que este cambio está llevando a los profesores a enfocarse menos en la sintaxis y más en el desarrollo de habilidades de alto nivel, como el pensamiento crítico y la resolución de problemas *(Caballar, 2024).*

En agosto de 2023, un grupo de 15 profesores de informática se reunió en Denver como parte de un taller patrocinado por la NSF (*National Science Foundation*). De esta reunión surgió el artículo académico titulado *AI in CS Education: Opportunities, Challenges, and Pitfalls to Avoid*, que identifica varias oportunidades. Entre ellas, destaca la capacidad de la IA para actuar como un tutor sin prejuicios, disponible las 24 horas del día, los 7 días de la semana, para tareas como la depuración de código (*debugging*) y el análisis de problemas (*reasoning*). Asimismo, resalta su utilidad para autocalificar programas y preguntas cortas de exámenes, además de ofrecer instrucción personalizada que puede ayudar a reducir brechas de equidad relacionadas con recursos o tiempo disponible.

Sin embargo, el estudio también advierte sobre retos que podrían obstaculizar el aprendizaje. Entre estos, se menciona el riesgo de que el acceso rápido a soluciones desincentive el esfuerzo y reduzca el desarrollo del pensamiento crítico en los estudiantes. Además, se subrayan las dificultades asociadas al diseño de experiencias educativas que minimicen la dependencia excesiva de la IA (Vahid, 2024).

Estos asistentes presentan más desafíos en su adopción en el ámbito educativo. Kazemitabaar, Ye, et al. (2024) destacan en su artículo otras barreras, como:

- **Percepción de falta de necesidad:** Algunos estudiantes consideran que los recursos actuales son suficientes y no perciben el valor añadido que ofrecen los asistentes de IA.
- **Preferencia por herramientas tradicionales:** Los estudiantes tienden a inclinarse por comunidades online como Stack Overflow, otros foros técnicos como GeeksforGeeks e incluso recursos de aprendizaje en formato de vídeo a través de YouTube. Estas opciones, al estar ya integradas en sus hábitos de estudio, suelen ser las más utilizadas y valoradas.

[4] IEEE Spectrum: https://spectrum.ieee.org/

- **Resistencia al aprendizaje asistido:** Algunos estudiantes valoran el proceso de aprendizaje autónomo y prefieren resolver los problemas por sí mismos. Esto mismo piensa Tim Ruscica[5]: "Programar realmente consiste en encontrarse con una gran cantidad de diferentes errores y *bugs*, y averiguar cómo solucionarlos. Estos momentos de enfrentarse a material "difícil" son fundamentales para desarrollar habilidades críticas como el pensamiento crítico y la resolución de problemas" (freeCodeCamp, 2024). Un caso práctico reciente, realizado por el mismo Ruscica, evidencia las limitaciones de la IA en el ámbito educativo. En su experimento, intentó aprender C utilizando exclusivamente ChatGPT como herramienta de aprendizaje. Aunque logró avances gracias a la rapidez de la IA, se encontraron importantes limitaciones: el asistente asumió conocimientos previos, ofreció explicaciones fuera de contexto y careció de un enfoque pedagógico estructurado (Tech With Tim, 2024).
- **Desconfianza y cierto escepticismo en el contenido generado por IA:** La falta de credibilidad percibida en las sugerencias de la IA puede limitar su aceptación. Según la la encuesta anual de Stack Overflow, un 31 % de los encuestados desconfía de los resultados generados por IA debido a problemas de precisión y al riesgo de producir información errónea (Stack Overflow Developer Survey, 2024).

2.1.3. La era de los LLMs

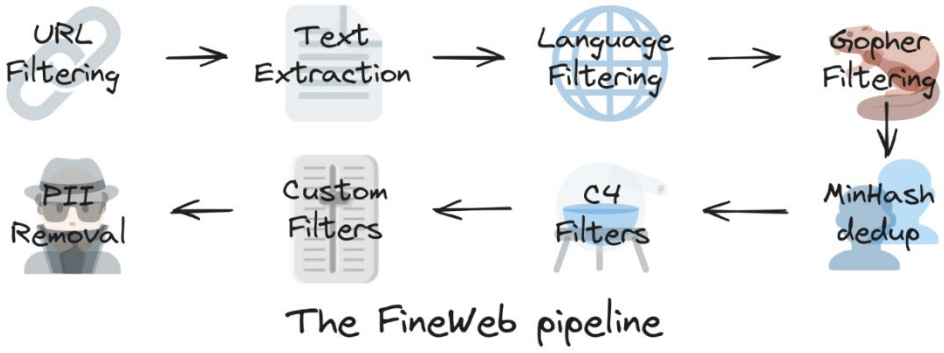
"Los modelos de lenguaje grande (LLMs) son sistemas avanzados de IA **que entienden y generan lenguaje natural, o texto similar al humano, con los datos en los que se han entrenado mediante técnicas de aprendizaje automático"** (Microsoft Azure, 2024). Esta popularización de LLMs marca un cambio en la interacción humano-máquina mediante el procesamiento del lenguaje natural. Estos modelos, como el caso de Claude utilizado para este trabajo, ofrecen capacidades sorprendentes en la generación de texto, aunque también tienen limitaciones inherentes a su arquitectura y entrenamiento.

Como se puede apreciar en la **figura 5**, el desarrollo de un *dataset* para entrenar LLMs involucra varias etapas. La primera es el preentrenamiento, donde se recopila y procesa un volumen masivo de texto de internet. Empresas como OpenAI, Google y Anthropic utilizan

[5] Tim Ruscica: https://www.techwithtim.net

conjuntos de datos equivalentes a *FineWeb*[6], un corpus curado que garantiza diversidad y calidad en los textos recopilados. Para ello, se filtran fuentes irrelevantes, se extraen textos de páginas web y se eliminan elementos no textuales como etiquetas HTML. Esta curación de datos es clave para garantizar la calidad de los corpus de entrenamiento y ha sido detallado en un estudio reciente (Palacio et al., 2023).

Figura 5. *Pipeline de procesamiento de datos de FineWeb*



Fuente: Página Web https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1

Los LLMs han encontrado aplicación en diversos dominios, desde la asistencia en la redacción y traducción de textos hasta la **generación de código y el soporte en tareas académicas**. Sin embargo, su desempeño está condicionado por los datos de entrenamiento y la estructura de su arquitectura neuronal. Problemas como la incapacidad de razonar a nivel abstracto, la falta de una memoria persistente y la dependencia en datos previos son solo algunos de los muchos retos que aún presentan estos modelos (Wong et al., 2023). Para mitigar estos problemas, se han desarrollado técnicas como el aprendizaje por refuerzo con retroalimentación humana (RLHF), que "permiten imitar las respuestas, los comportamientos y la toma de decisiones de los humanos" (AWS, 2024). Esta técnica consiste en recolectar datos de evaluaciones humanas y utilizarlos para ajustar el comportamiento del modelo, optimizando la calidad de sus respuestas.

[6] FineWeb: https://huggingface.co/datasets/HuggingFaceFW/fineweb

3. Estado del Arte

En este apartado se analizan herramientas y asistentes, tanto en el ámbito académico como en la industria, con el objetivo de evaluar sus funcionalidades. Este análisis sirve como base para justificar y diseñar la solución propuesta en este trabajo.

3.1. Asistentes desarrollados en la academia

En este apartado se presentan algunos asistentes desarrollados en el ámbito académico, junto con una descripción de sus principales funcionalidades.

3.1.1. CodeAid

CodeAid[7] es un asistente de programación en la nube basado en GPT-3.5, pensado especialmente para estudiantes universitarios. La herramienta se puso a prueba en un curso de programación en C de 700 estudiantes durante un semestre como un recurso opcional más, similar al de una tutoría o a consultar los foros de la plataforma virtual de la asignatura. Las funcionalidades que hacen especial a *CodeAid* se muestran en la **figura 6** y son seis:

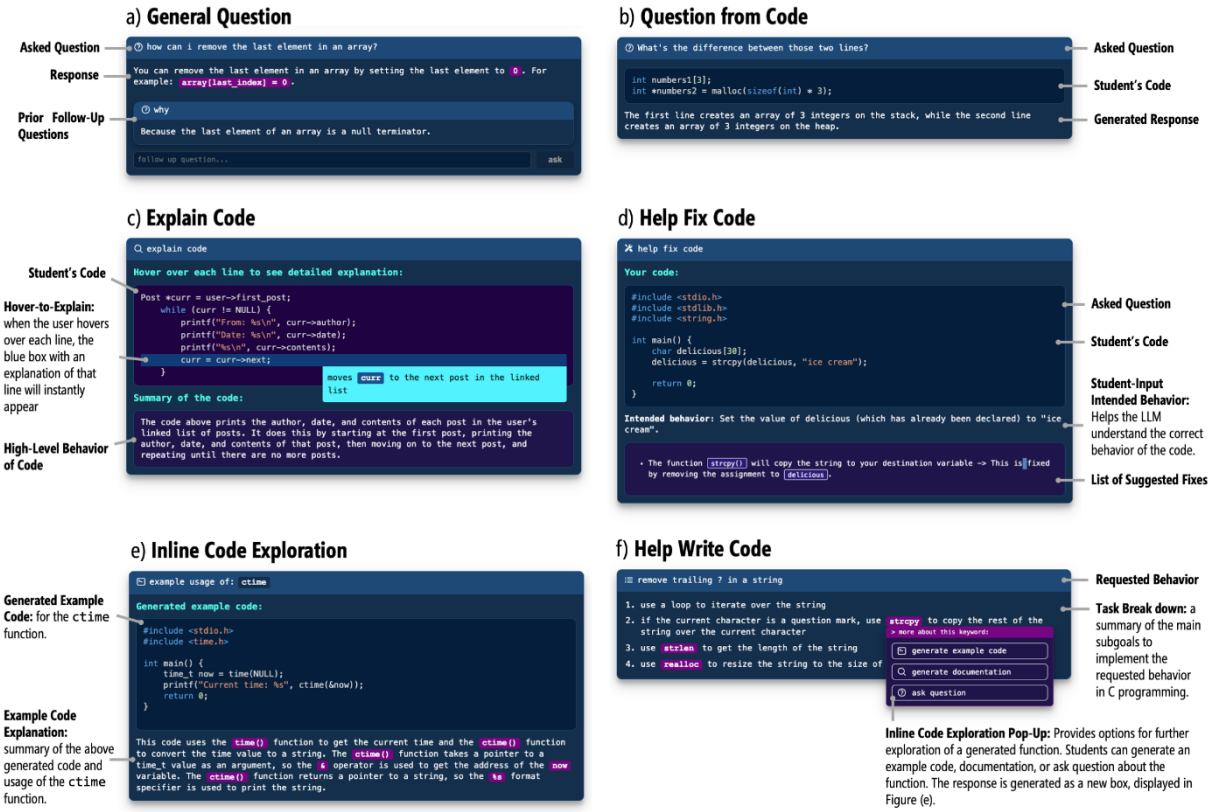
a) **Preguntas generales**: Cuando el usuario selecciona "General Question", la IA tiene en cuenta el enunciado de la pregunta y genera respuestas cortas con explicaciones.
b) **Explicaciones de código:** Los usuarios pueden pasar el ratón sobre un elemento del código para explorarlo más en profundidad.
c) **Preguntas sobre el código:** Ayuda a los estudiantes con tareas de depuración o preguntas conceptuales en un contexto específico. A diferencia de "General Question", tiene la capacidad añadida de proporcionar algo de código como contexto.
d) **Ayuda para arreglar el código:** Ayuda a los estudiantes en sus tareas de depuración de código y realiza dos tareas en el *backend*: intenta generar la versión correcta del código basándose en la descripción del problema, y explica con viñetas qué se ha cambiado y por qué. La interfaz de respuesta no revela soluciones directas de código.
e) **Exploraciones del código en línea:** Para ayudar a los estudiantes en casos de uso como la comprensión del código utilizado en las tareas o el código enseñado durante las

[7] code-aid: https://github.com/MajeedKazemi/code-aid

clases, se diseñó la funcionalidad "Explain Code". Al seleccionarla, los estudiantes pueden pegar el código a explicar en el editor. El resultado generado es una interfaz que muestra el código de los usuarios y les permite pasar el ratón por encima de cada línea para ver la explicación. Se condiciona el modelo para producir una salida sencilla de entender.

f) **Ayuda a escribir código:** La función de "Help Write Code" sirve para ayudar a los estudiantes con los problemas sin mostrar ningún código. Para ello pide a los usuarios que introduzcan el comportamiento previsto del programa y genera una estructura de alto nivel del código con pseudocódigo.

Figura 6. *Funcionalidades de CodeAid*



Fuente: Kazemitabaar, Ye, et al. (2024)

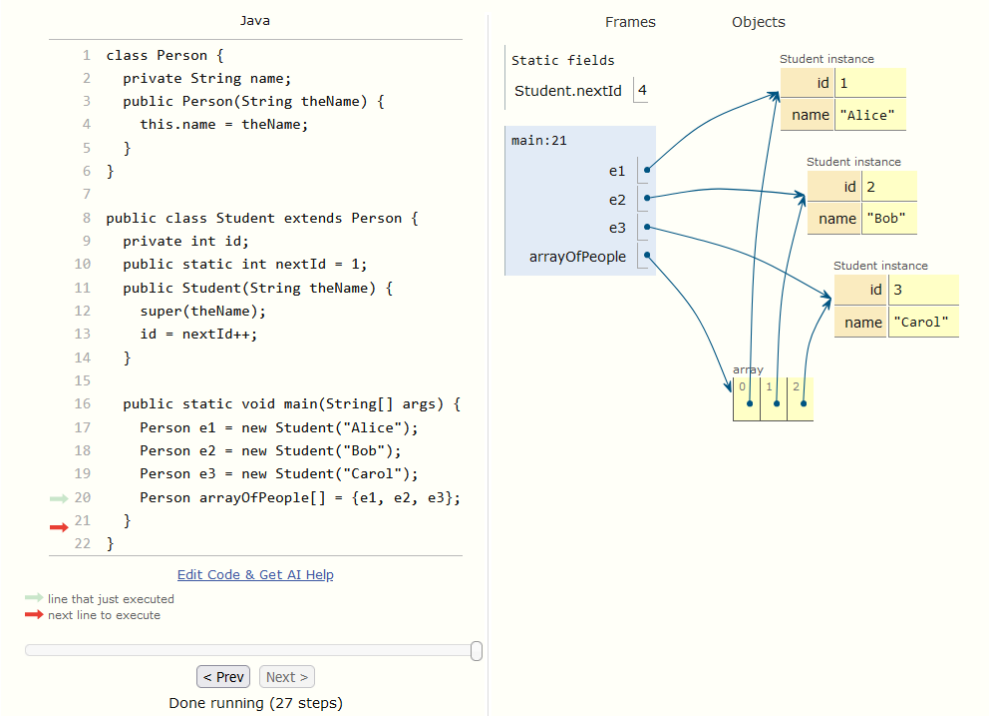
3.1.2. Python Tutor

Otra herramienta interesante y didáctica es el asistente web *Python Tutor*[8] que, a pesar de su nombre, soporta más lenguajes aparte de Python, incluyendo Java, C, C++ y JavaScript. Python

[8] Python Tutor: https://pythontutor.com

Tutor se diferencia por explicar el código línea a línea, describiendo el funcionamiento y mostrando gráficamente todo lo que está ocurriendo con el código en cada momento. Esta es la característica que lo hace un asistente especialmente útil a la hora de entender el código. Está diseñado para trabajar con ejemplos concisos y es ideal para código que "quepa en una pizarra o diapositiva", más que para proyectos de software completos. La figura 7 muestra cómo Python Tutor permite visualizar las interacciones entre clases en un programa que crea varias instancias de la clase *Student* (que hereda de *Person*) y las almacena en un array. En particular, permite observar cómo cada objeto ocupa una posición única en memoria y cómo el uso de *super()* inicializa el campo *name* en la clase base. Además, se ilustra cómo se actualizan los campos estáticos, como el contador *nextId*.

Figura 7. *Caso de uso en Java con Python Tutor*



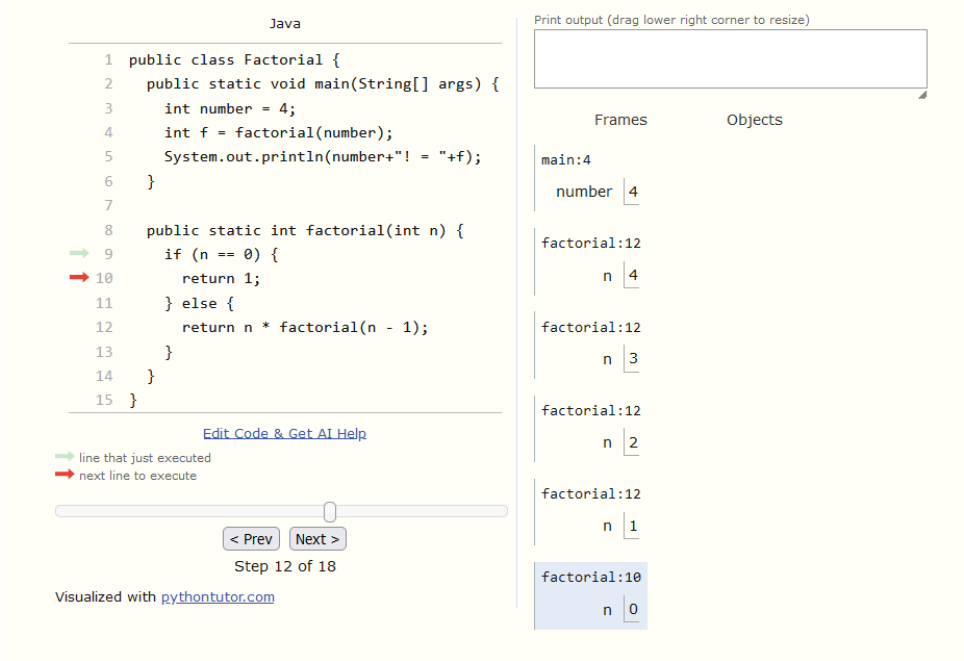
Fuente: Página Web https://pythontutor.com/articles/java-visualizer.html

Esto ayuda a ilustrar dinámicamente conceptos como el uso de *super()* para invocar el constructor de una clase padre y la actualización de campos estáticos como un contador de identificadores.

Otro caso de uso es el manejo de recursión. Por ejemplo, la figura 8 ilustra el cálculo del factorial utilizando recursión en Java. Python Tutor permite observar cómo cada llamada recursiva genera un nuevo marco en la pila de ejecución y cómo los parámetros cambian en

cada iteración hasta alcanzar el caso base (n == 0). Una vez alcanzado, el control vuelve hacia atrás en la pila, completando el cálculo.

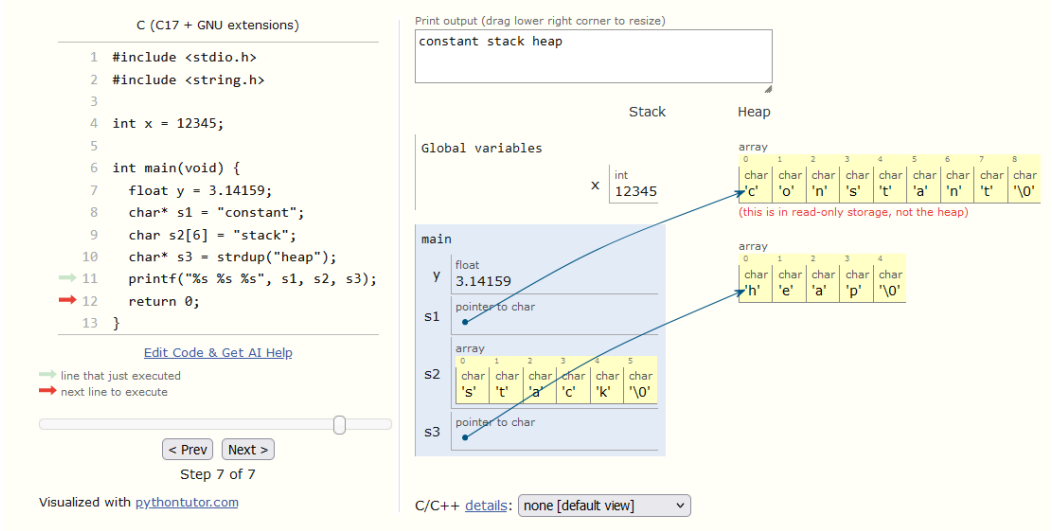
Figura 8. *Calcular factorial en Java con Python Tutor*



Fuente: Página Web https://pythontutor.com/articles/java-visualizer.html

Finalmente, la figura 9 muestra cómo Python Tutor permite visualizar la gestión de memoria en C. En este caso, se observa cómo se asignan bloques de memoria en el heap mediante strdup().

Figura 9. *Gestión de memoria en C con Python Tutor*



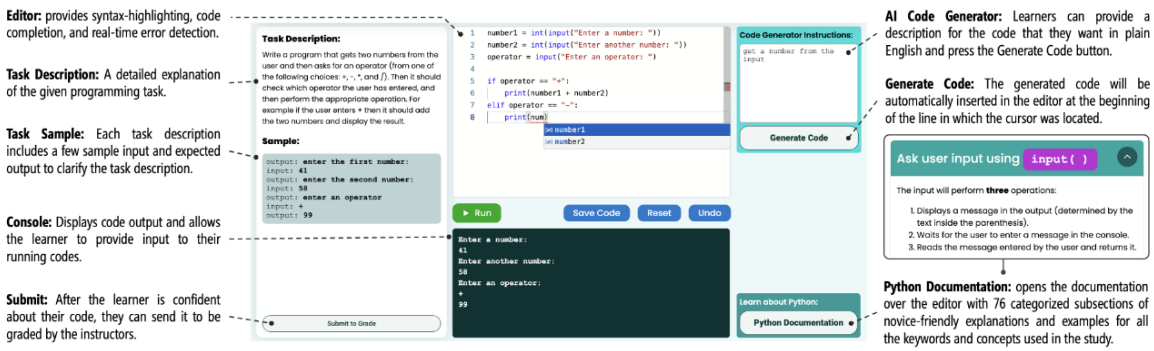
Fuente: Página Web https://pythontutor.com/articles/c-cpp-visualizer.html

3.1.3. Coding Steps

Otro asistente desarrollado por la comunidad académica es Coding Steps[9], una herramienta web para estudiar el impacto de los asistentes en estudiantes principiantes de programación en Python. A través de un experimento con 69 estudiantes sin experiencia previa en programación, se encontró cómo el uso del asistente aumentó considerablemente el rendimiento en las tareas de código (1.15x de crecimiento en el ratio de finalización y un 1.8x de mejores puntuaciones) sin empeorar la eficiencia en tareas manuales de modificación de código. La figura 10 muestra el entorno de programación de Coding Steps que incluye:

1. Una descripción de la tarea y un conjunto de ejemplos.
2. Un editor de código con resaltado de sintaxis, detección de errores en tiempo real y autocompletado.
3. Una consola para mostrar el código.
4. Una documentación en Python con miniejemplos escritos específicamente para principiantes.
5. Un generador de código para insertar código generado por IA en el editor de código.

Figura 10. *Entorno de Coding Steps*



Fuente: Kazemitabaar et al. (2023)

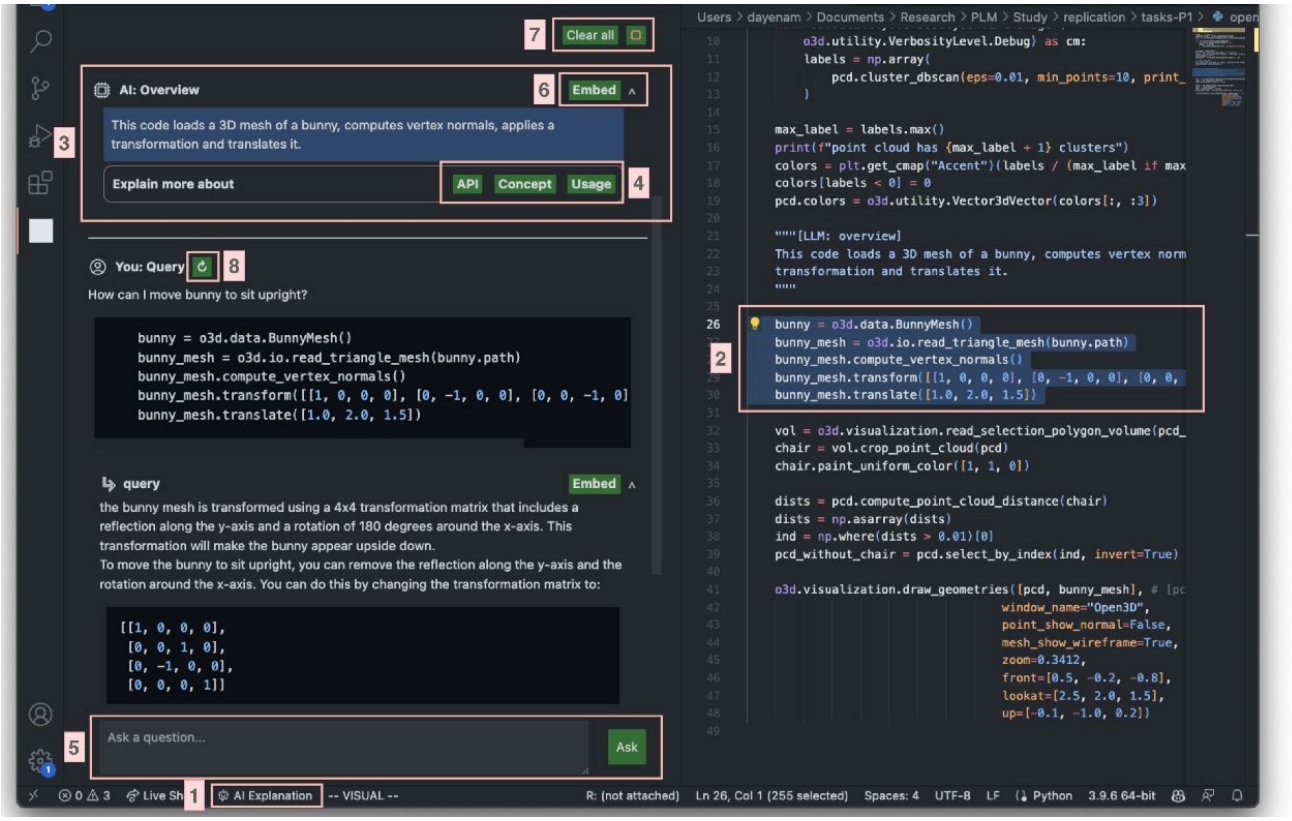
La integración de todas estas funcionalidades en una única herramienta permite a los alumnos progresar de forma independiente en las tareas de formación, a la vez que facilita la recopilación de datos de los alumnos para el estudio.

[9] coding-steps: https://github.com/MajeedKazemi/coding-steps/

3.1.4. GILT

GILT[10] representa otra gran contribución a este dominio. Su diseño ayudar a entender el código a través de una interfaz conversacional dentro del IDE. La figura 11 presenta sus funcionalidades que incluyen: 1) Un botón de activación, 2) El código usado como contexto cuando se hace *prompting* con el LLM, 3) Un resumen del código (activación sin solicitud), 4) Unos botones para obtener más detalles, 5) Un campo de texto para ingresar solicitudes del usuario, 6) Opciones para incrustar información en el código (incrustar) y un botón de ocultar/mostrar, 7) Opciones para limpiar el panel (limpiar todo) y un botón para abortar el modelo de lenguaje, 8) Un botón de actualización.

Figura 11. *Prototipo de GILT*



Fuente: Nam et al. (2024)

El sistema utiliza GPT-3.5 Turbo para generar cuatro tipos de explicaciones sin necesidad de *prompts* explícitos:

[10] GILT: https://marketplace.visualstudio.com/items?itemName=dayen.gilt

1. Explicaciones de fragmentos de código resaltados.
2. Explicaciones detalladas de llamadas a la API.
3. Explicación de conceptos concretos del problema.
4. Ejemplos de uso de las API.

En un estudio con 32 participantes, GILT demostró mejorar los índices de finalización de tareas en comparación con las búsquedas web tradicionales, aunque su efectividad varió entre estudiantes y profesionales. Uno de los resultados más interesantes fue que los estudiantes tendieron a utilizar más las interacciones sin *prompts*, mientras que los profesionales se beneficiaron más de la capacidad de formular consultas directas al modelo a través del chat.

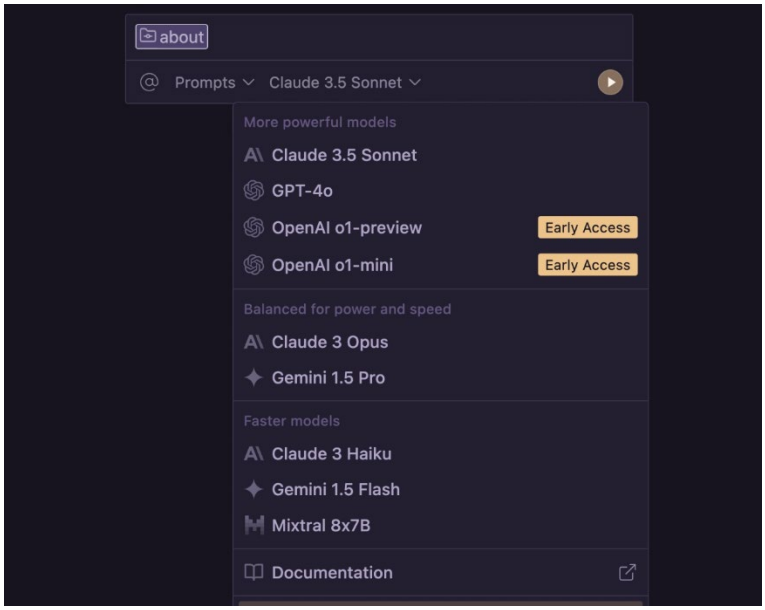
3.2. Asistentes desarrollados en la industria

En este apartado se presentan algunos asistentes desarrollados en el mundo empresarial, junto con una descripción de sus principales funcionalidades.

3.2.1. Cody

Cody[11], desarrollado por Sourcegraph, destaca por su flexibilidad en la selección de LLMs (**figura 12**), lo que permite elegir entre diferentes proveedores como Anthropic y OpenAI.

Figura 12. *Flexibilidad de modelos en Cody*

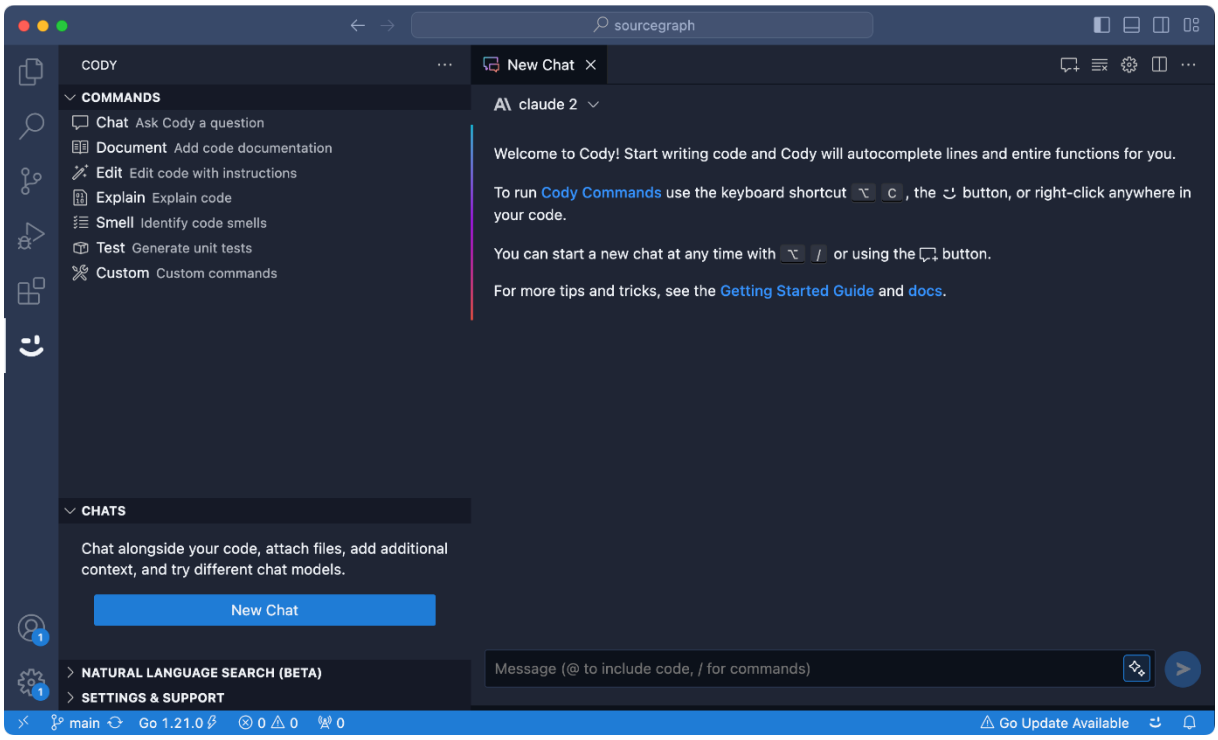


Fuente: Página Web https://sourcegraph.com/cody

[11] Cody: https://sourcegraph.com/cody

Cody también se integra con múltiples IDEs incluyendo IntelliJ, NeoVim y Visual Studio Code. Otra funcionalidad interesante que se puede observar en la figura 13 es el sistema de comandos personalizados, que permite crear y guardar instrucciones específicas para el flujo de trabajo. Además, Cody tiene la capacidad de indexar y comprender el contexto completo del *codebase*, no solo del archivo que se está editando.

Figura 13. *Comandos de Cody en VSCode*



Fuente: Página Web https://itproger.com/news/realno-poleznie-ii-instrumenti-dlya-razrabotki

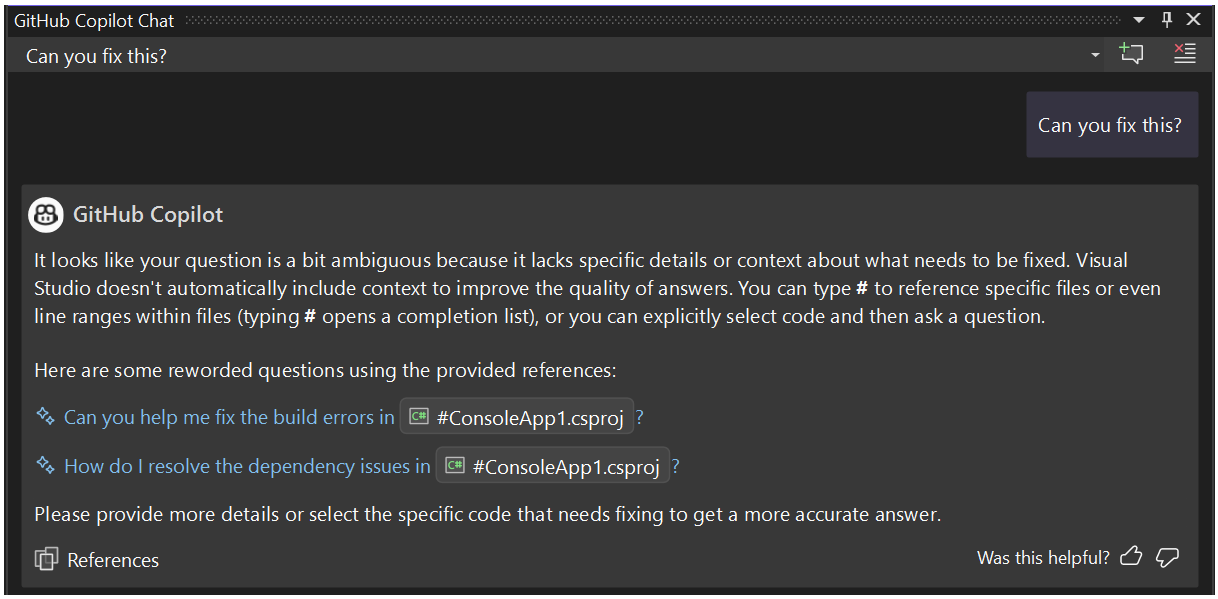
3.2.2. GitHub Copilot

GitHub Copilot[12] es, quizás, el asistente más maduro y popular entre los de asistentes de IA para programación en la industria. Después de más de tres años de evolución desde su lanzamiento en octubre de 2021, la herramienta ha alcanzado un nivel de sofisticación que la hace útil para desarrolladores de todos los niveles.

[12] GitHub Copilot: https://github.com/features/copilot

GitHub Copilot se integra con los principales IDEs y destaca especialmente por la incorporación de Copilot Chat. Como se puede observar en la **figura 14**, se trata de una interfaz conversacional diseñada específicamente para programación, accesible directamente desde el IDE.

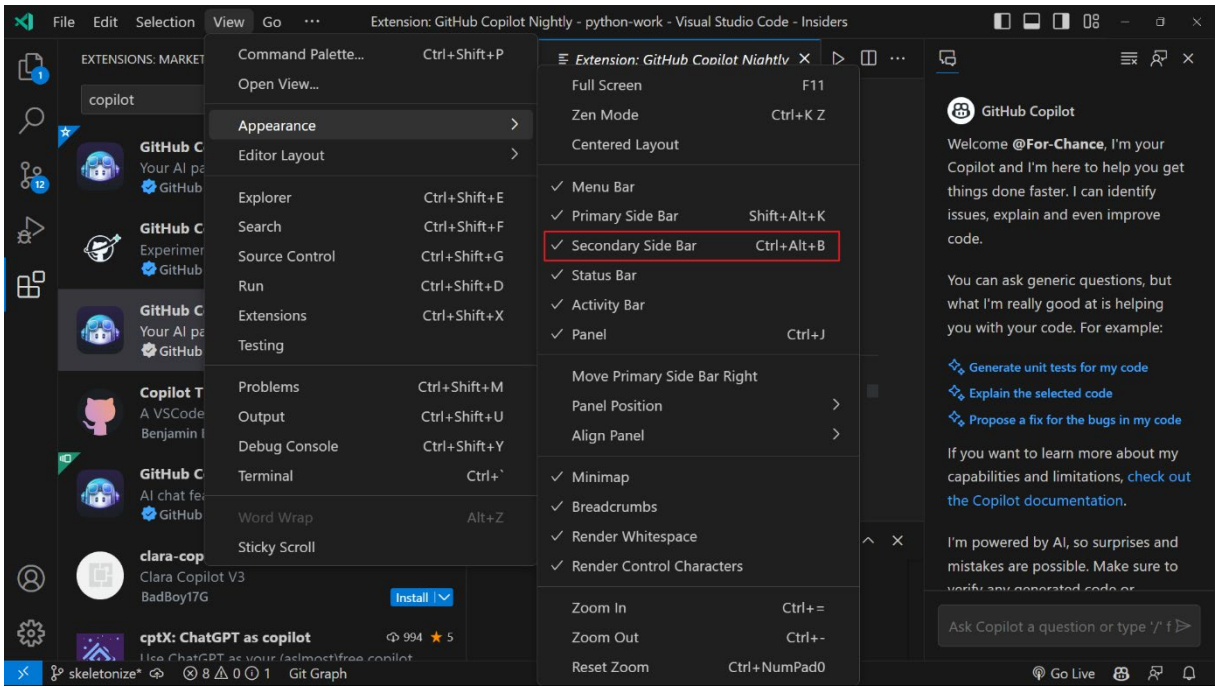
Figura 14. *Asistencia guiada en GitHub Copilot Chat*



Fuente: Página Web https://learn.microsoft.com/en-us/visualstudio/ide/copilot-chat-context?view=vs-2022

Otra funcionalidad destacada de GitHub Copilot es la barra lateral secundaria, como se puede ver en la **figura 15**. Se puede tener el chat abierto en cualquier momento, mientras no se pierde el contexto de las otras vistas disponibles como el explorador de archivos o el control de código fuente. Esto proporciona una experiencia de IA más integrada en Visual Studio Code.

Figura 15. *Barra lateral secundaria de GitHub Copilot Chat*

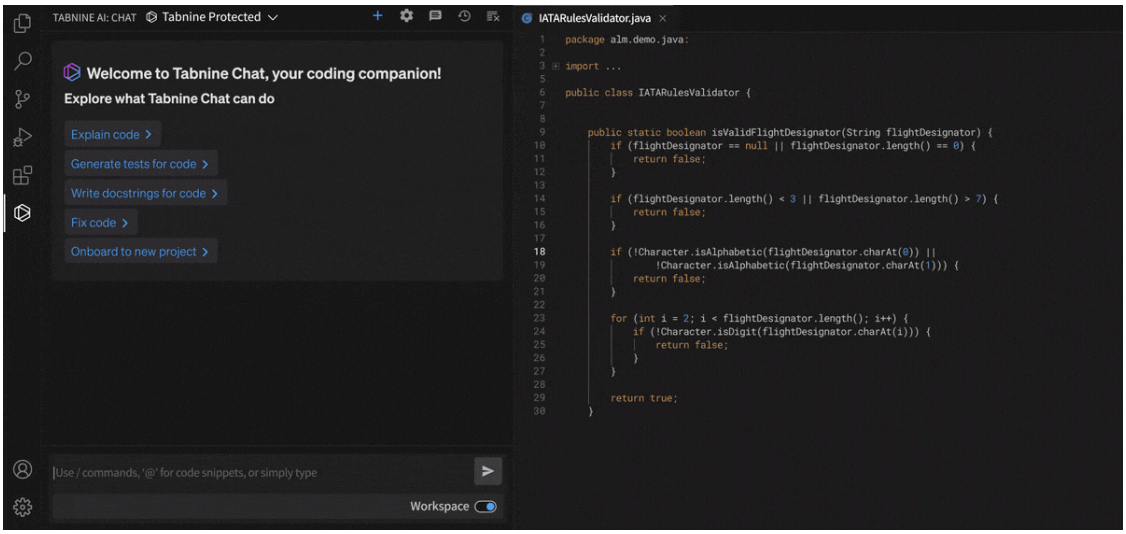


Fuente: Página Web https://github.com/orgs/community/discussions/50851

3.2.3. Tabnine

Tabnine[13] ofrece las características que se ven en la **figura 16**:

Figura 16. *Tabnine Chat*



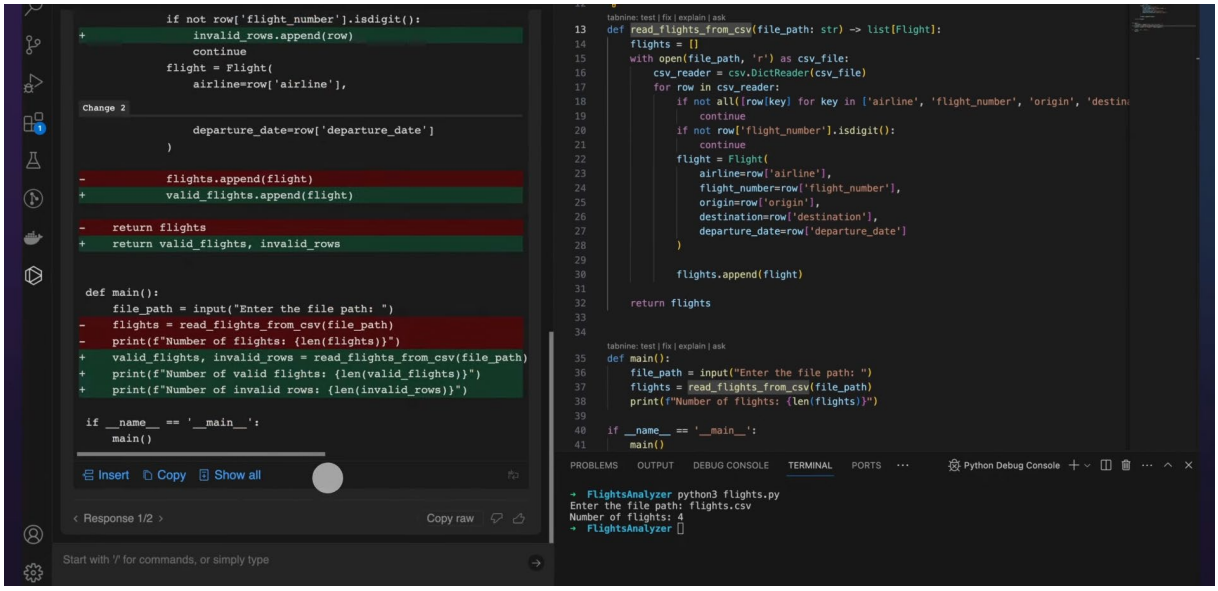
Fuente: Página Web https://marketplace.eclipse.org/content/tabnine-ai-assistant-chat-software-developers

[13] Tabnine: https://www.tabnine.com

Una característica que comparte con **Cody** es su enfoque en el contexto, que le permite entender no solo el archivo actual, sino también elementos de bibliotecas, APIs y estructura de proyectos. La herramienta opera mediante modelos personalizados que pueden desplegarse en el entorno local, respetando las necesidades de privacidad y seguridad de las empresas. Además, Tabnine permite a los equipos mantener un estilo de código armonizado, facilitando la colaboración y reduciendo la deuda técnica, algo que los desarrolladores más experimentados seguramente van a tener que afrontar si los menos experimentados contribuyen con sus programas con código generado por IA.

Como se puede ver en la **figura 17**, Tabnine también está siendo adoptado para tareas de refactorización y, en algunos casos, se explora para migraciones de código de lenguajes legados.

Figura 17. *Refactorización de código con Tabnine*



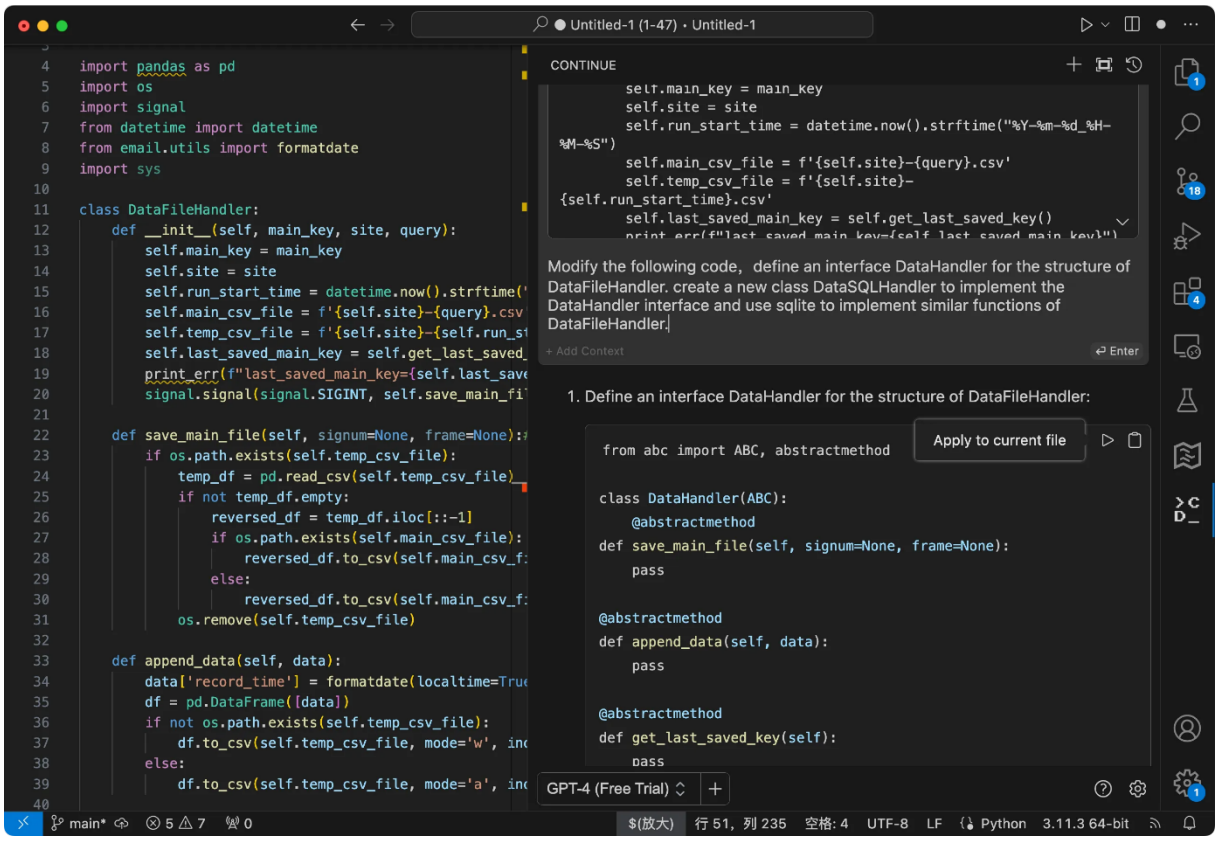
Fuente: Página Web https://www.tabnine.com/blog/code-refactoring-with-generative-ai/

3.2.4. Continue

El factor diferencial de *Continue* está en su enfoque en código abierto y en ofrecer a los desarrolladores transparencia, personalización y control sobre su flujo de trabajo, como puede ser la opción de conectarse a modelos que se ejecutan en local.

En la figura 18 se puede observar una funcionalidad interesante que ofrece Continue para que, a través de un modo diálogo, los usuarios puedan discernir y juzgar manualmente si el código devuelto por el modelo es aplicable. Tras seleccionar el botón “apply to current file”, el programa intenta fusionar y luego confirma con el usuario si la solución de fusión de cada fragmento es correcta. Sin embargo, este proceso suele estar plagado de errores y puede resultar largo y laborioso para el usuario.

Figura 18. *UI de Continue*



Fuente: Página Web https://medium.com/@GenerationAI/ai-code-assistant-internals-a0792de699dc

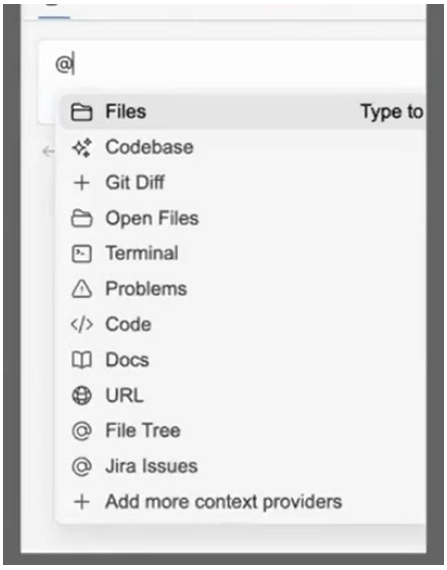
Su naturaleza de código abierto también permite a los desarrolladores conectar cualquier modelo de IA, ya sea mediante servicios alojados en la nube o modelos locales, garantizando la privacidad de los datos. Esta personalización se extiende a políticas y plantillas, lo que habilita la creación de experiencias adaptadas para cada equipo.

Continue se estructura en cuatro componentes principales:

1. **Sistema de chat integrado:** Facilita consultas directas a LLMs sin necesidad de abandonar el editor. Esto permite una interacción fluida y en tiempo real.

2. **Motor de autocompletado contextual:** Si hay información de la *codebase*, documentación u otra consulta, se puede escribir el símbolo @ para seleccionarla e incluirla como contexto, como se puede ver en la figura 19. El chat de Continue proporciona sugerencias contextuales inteligentes basadas en el análisis del código. Por ejemplo, puede ofrecer en el chat integrado recomendaciones relacionadas con bibliotecas, SDKs, preguntas sobre *tickets* en Jira, estructuras de árboles de archivos para localizar directorios o interacciones con la terminal.

Figura 19. *Contexto de referencia en Continue*



Fuente: Seminario de Thoughtworks https://thoughtworks.wistia.com/medias/f2ziwflq9v

3. **Sistema de edición avanzado:** Permite realizar modificaciones directamente en el editor sin cambiar entre archivos.
4. **Framework de acciones personalizables:** Ofrece la posibilidad de automatizar tareas frecuentes mediante comandos personalizados. Ejemplos de esto son generar resúmenes de *commits*, actualizar documentación o crear *tickets* detallados.

3.3. Arquetipos de herramientas IA

A pesar del cambiante panorama de herramientas en constante mejora, durante el último año se han asentado tres arquetipos comunes de herramientas IA, como señala el analista Tanay Jaipuria (Jaipuria, 2024). A continuación, se muestra la figura 20 donde se definen estos tres grandes arquetipos con sus casos de uso y se mencionan algunos ejemplos:

- **Copilotos:** La IA se presenta como una funcionalidad más que apoya a los profesionales de alguna forma en sus tareas diarias. En esta categoría entrarían todas las herramientas presentadas en el Estado del Arte.
- **Agentes:** La IA se ocupa de forma autónoma de una gran parte de las tareas que antes hacían los profesionales, aunque estos agentes siguen sin ser infalibles ni funcionar de manera aislada, y siguen siendo supervisados por posibles errores.
- **Servicios:** Empresas basadas en IA ofrecen su servicio de principio a fin (*end-to-end*) a otras empresas.

Figura 20. *Arquetipos de herramientas de IA*

Fuente: Elaboración propia con Canva

Mientras agentes como Devin intentan posicionarse como "ingenieros autónomos" que pueden realizar tareas complejas de desarrollo de forma independiente, copilotos como Codeium[14] tienen como propósito servir de asistentes que mejoran la productividad y optimizan el flujo de trabajo mientras el programador mantiene el control.

[14] Codeium: https://codeium.com

Al igual que Devin, otros proyectos como Sierra[15] o Decagon[16] también están replanteando la experiencia de otras industrias como la del servicio al cliente al crear entornos de IA conversacional y ser capaces de resolver de manera autónoma una gran parte de las consultas. De forma similar, empresas de servicios como Alma[17] (inmigración), Pilot[18] (contabilidad) y Gelt[19] (impuestos) destacan por su capacidad de transformar industrias enteras al ofrecer servicios nativos de IA que gestionan procesos completos de principio a fin.

3.4. Conclusión del análisis

Según lo expuesto en el apartado de contexto y estado del arte sobre los distintos asistentes copilotos, se observa que no existe ninguna solución para Visual Studio Code dirigida exclusivamente a estudiantes de programación que proporcione apoyo contextual para facilitar la comprensión de los problemas. Si bien algunos copilotos **12** ofrecen funcionalidades similares a Buddy, estos asistentes suelen generar dependencia, revelar directamente las respuestas y necesitar de *prompts* especializados, los cuales pueden ser difíciles de escribir sin experiencia previa. En contraste, Buddy busca fomentar una comprensión profunda del problema sin prescindir completamente de la IA y se plantea como una solución que permite:

- Enfocar la resolución del problema en el propio alumno, alentándolo a explorar diferentes enfoques.
- Utilizar la IA para reducir el alcance del problema, resolviéndolo de manera progresiva.
- Facilitar la comprensión de las soluciones sugeridas por la IA mediante explicaciones y preguntas que ayuden al alumno a reflexionar sobre si realmente ha entendido el problema.

[15] Sierra: https://sierra.ai/
[16] Decagon: https://decagon.ai/
[17] Alma: https://www.tryalma.ai/
[18] Pilot: https://pilot.com/
[19] Gelt: https://www.joingelt.com/

4. Objetivos y metodología de trabajo

A continuación, se detallan los objetivos del trabajo, así como las etapas metodológicas que han guiado su desarrollo.

4.1. Objetivo general

El objetivo general es **construir un asistente que guíe a los estudiantes para una comprensión profunda de los problemas de programación**.

Esta solución **acompaña a los estudiantes en el proceso de resolver problemas**, orientándolos hacia la respuesta correcta sin revelarla directamente y sin que el estudiante tenga que formular *prompts*.

4.2. Objetivos específicos

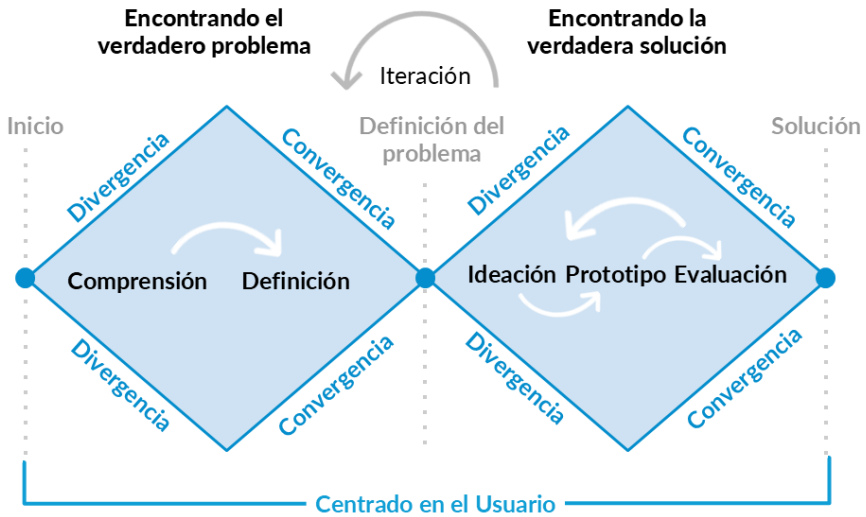
Los objetivos específicos que permiten alcanzar el objetivo general están vinculados a las diferentes fases del desarrollo del asistente:

1. **Analizar el Contexto y Estado del Arte de los asistentes.** En esta etapa se describe el impacto de la IA en el desarrollo de software y, específicamente, en el aprendizaje y la enseñanza de la programación. Además, se identifican las principales necesidades de los estudiantes y se analizan las soluciones existentes en el mercado de los asistentes y modelos de IA para programadores.
2. **Desarrollar las especificaciones del asistente.** Se especifican los casos de uso, los requisitos no funcionales y el alcance de la primera versión.
3. **Diseñar el asistente.** Se diseña la arquitectura del sistema y se detallan las tecnologías utilizadas, justificando por qué son las más adecuadas para implementar la solución.
4. **Implementar el asistente.** Durante esta fase se desarrolla el paquete de funcionalidades iniciales del asistente, teniendo en cuenta cada una de sus etapas.
5. **Evaluar el asistente.** En esta etapa se llevan a cabo pruebas con usuarios reales, con el objetivo de evaluar la usabilidad y que la experiencia en el proceso de aprendizaje sea satisfactoria.
6. **Documentar el asistente.** Se crea una guía del usuario y se publica en el *marketplace* de Visual Studio Code para que el asistente esté accesible para todos los públicos.

4.3. Metodología de trabajo

La metodología adoptada para este trabajo se encuentra en la **figura *21*** y se fundamenta en el Diseño Centrado en el Usuario (Montero, 2015). En particular, se aplicará el modelo de Doble Diamante de *Design Thinking* (Brown, 2008), una metodología que prioriza al usuario en el proceso de diseño. Este enfoque se estructura en las siguientes fases: comprensión, definición, ideación, prototipado y evaluación.

Figura 21. *Modelo Doble Diamante de Design Thinking*



Fuente: Elaboración propia basada en el artículo web *Visualizing the 4 Essentials of Design Thinking*. (Liu, 2016)

Estas fases se llevarán a cabo de manera iterativa:

4.3.1. Fase 1: Comprensión

El proceso comenzará con una investigación exhaustiva de las necesidades de los usuarios implicados en la solución desarrollada, así como de su entorno. El objetivo principal de esta fase es comprender profundamente la perspectiva de los estudiantes para generar una solución alineada con su realidad.

En esta etapa de comprensión, se emplearán técnicas como la observación y las entrevistas, que permitirán recopilar información clave sobre los usuarios y el contexto en el que se utilizará el asistente.

- **Investigación contextual:** Se recopilarán datos del sector educativo y estudios empíricos para comprender las necesidades, motivaciones, modelos mentales y actividades relacionadas con el aprendizaje y la enseñanza de la programación.
- **Análisis del dominio:** Se identificarán y analizarán asistentes similares en la industria y en el ámbito académico para entender sus funcionalidades y audiencias. Además, se estudiarán encuestas que proporcionen datos estructurados, útiles para validar o evaluar propuestas de solución y diseñar estrategias más efectivas para el público objetivo.
- **Entrevistas:** Se realizarán preguntas a usuarios potenciales con el fin de profundizar en sus necesidades, preferencias y experiencias con este tipo de asistentes. Estas entrevistas se llevarán a cabo tanto en la fase inicial como en la de evaluación. Serán informales, dirigidas a una sola persona, para las cuales se preparará un guion de preguntas y se registrarán las respuestas. Al finalizar, se elaborará un informe con las conclusiones obtenidas.

4.3.2. Fase 2: Definición

Se trabajará en esta fase la información obtenida durante la fase de comprensión.

- **Customer Journey Map:** Se identificarán los puntos débiles y las oportunidades de mejora en el asistente. Además, se representarán los diferentes puntos de contacto que caracterizan la interacción del usuario con el sistema. A partir de este análisis, se pretenden lograr los siguientes objetivos:
    - Comprender mejor las expectativas del usuario sobre el funcionamiento ideal del asistente.
    - Detectar errores o carencias en el proceso que dificultan el cumplimiento de los objetivos planteados.
- **Descubriendo *Insights*:** Se extraerán inferencias clave a partir de los datos recopilados mediante las siguientes acciones:

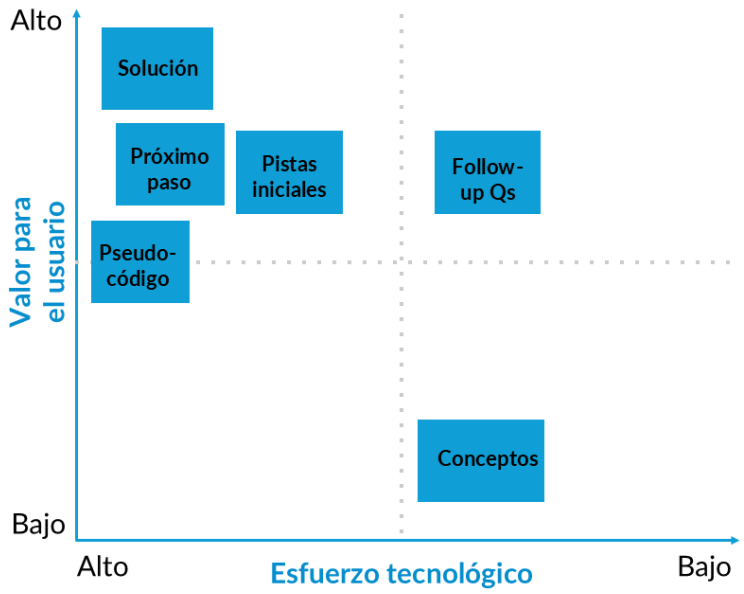
- **Establecer el contexto:** Se observará cómo se comportan los estudiantes en situaciones específicas, identificando lo que piensan, sienten y, sobre todo, explicando sus acciones y objetivos.
- **Comunicar el problema:** Se identificarán las barreras que dificultan que los usuarios logren sus objetivos con el asistente. Estas barreras suelen surgir cuando el usuario experimenta conflictos, tensiones o incomodidades, lo que ofrece oportunidades para generar nuevas ideas.
- **Articular el por qué:** Se analizarán las razones detrás del comportamiento del usuario para descubrir cómo el asistente puede ayudarles a alcanzar sus objetivos. Esto implica una síntesis clara del comportamiento observado, explicando el por qué detrás de cada acción o decisión.
- **Capturar la motivación:** Se identificarán las motivaciones que impulsan las acciones de los usuarios. Se busca en este punto comprender las frustraciones asociadas a sus experiencias para localizar los factores motivadores principales.
- **Visualizar el ideal:** Se describirá el estado final o la experiencia ideal que el usuario espera alcanzar. Este paso ayudará a definir una visión clara del objetivo que el asistente debe cumplir.

4.3.3. Fase 3: Ideación

Durante esta fase, se definirán los *insights* clave y se identificarán las problemáticas que enfrentan las personas para quienes se está diseñando la solución.

- **Crear áreas de oportunidad:** Se identificarán oportunidades de diseño que permitan abordar los problemas detectados. Para ello, se formularán preguntas del tipo "¿Cómo podríamos nosotros...?" *(How Might We...?*) con el fin de explorar múltiples soluciones potenciales, lo que resulta de gran utilidad en el proceso de ideación.
- **Matriz de priorización:** Se elaborará una matriz de priorización similar a la de la **figura 22** que permitirá identificar los problemas más relevantes a resolver, priorizando aquellos que ofrecen mayor valor al usuario.

Figura 22. *Matriz de priorización*



Fuente: Elaboración propia adaptado de R. K. Khan, 2023, Portfolio

4.3.4. Fase 4: Prototipado

El prototipado se utilizará para recopilar *feedback* y realizar experimentos rápidos con nuevas ideas. Este enfoque permitirá reducir los costos de desarrollo y evaluar la usabilidad del producto antes de avanzar a etapas más avanzadas.

- **Prototipos de baja fidelidad y wireframes:** Se desarrollarán prototipos de baja fidelidad, a papel y lápiz "que es la forma más eficaz para la elaboración de prototipos en las primeras etapas del diseño, con el esqueleto de la interfaz y la distribución de los diferentes bloques" (Montero, 2015). En este trabajo, se adoptará un enfoque iterativo en el que los componentes se diseñarán y avanzarán de manera paralela. Una vez aprobado el diseño de baja fidelidad de un componente, se procederá a trabajar en su diseño digital (**wireframes**) utilizando herramientas de ordenador.

4.3.5. Fase 5: Evaluación

Esta fase validará las decisiones de diseño, incluyendo las interfaces y los procesos de interacción, a través de un método de evaluación que involucra la participación directa de usuarios.

- **Test con usuarios:** Se realizará una evaluación estructurada que incluye los siguientes pasos.
    - Definición del alcance de la evaluación.
    - Identificación y descripción del perfil de los participantes.
    - Definición del guion de las sesiones.
    - Realización de las sesiones.
    - Análisis de resultados y elaboración de las conclusiones.
- **Resultados y hallazgos**:
    - Éxito de la tarea (Sí/No).
    - Comentarios específicos del usuario sobre la tarea y *pain points* identificados.
    - Observaciones generales realizadas por el supervisor durante la prueba.

En este enfoque, las fases del proceso no seguirán un orden estrictamente lineal. Es posible retroceder entre las etapas para corregir errores o ajustar el diseño en función de los hallazgos. De esta manera, el *Design Thinking* se aplicará de forma iterativa, permitiendo una mayor fidelidad respecto a la visión inicial planteada.

5. Casos de uso

Durante el proceso de diseño se han identificado seis características principales (figura 23) que el asistente debe incluir. Para cada funcionalidad, se detallará un caso de uso con su especificación y su diagrama de secuencia correspondiente.

Figura 23. *Funcionalidades de Buddy*

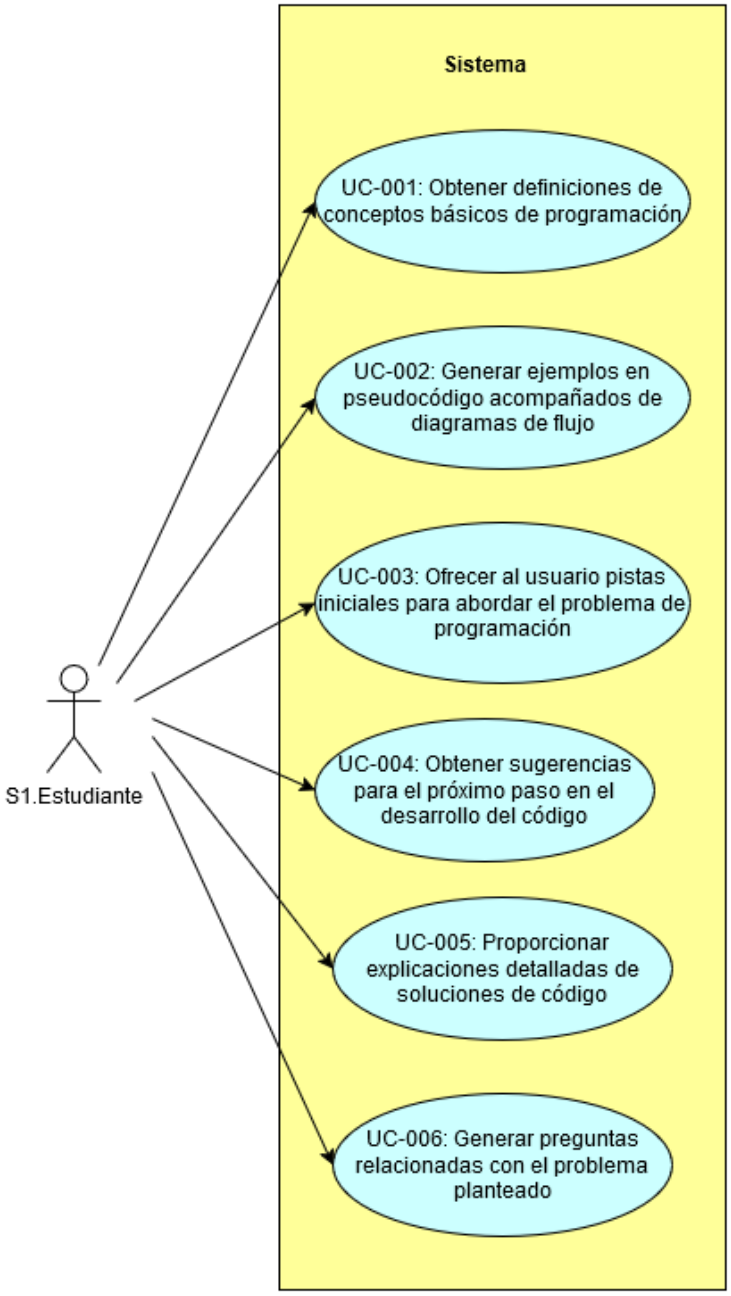
Fuente: Elaboración propia

Se han definido seis casos de uso que pueden darse en la ejecución del sistema:

- UC-001: **Obtener definiciones de conceptos básicos de programación** cuando el usuario lo solicita.
- UC-002: **Generar ejemplos en pseudocódigo acompañados de diagramas de flujo** que ilustran la solución de problemas específicos.
- UC-003: **Ofrecer al usuario pistas iniciales** para abordar el problema de programación.
- UC-004: **Obtener sugerencias para el próximo paso** en el desarrollo del código.
- UC-005: **Proporcionar explicaciones detalladas de soluciones de código**, permitiendo al usuario comprender la lógica detrás de una respuesta correcta.
- UC-006: **Generar preguntas relacionadas con el problema planteado**, fomentando una exploración más profunda del mismo.

La figura 24 muestra el diagrama de casos de uso diseñado para el desarrollo:

Figura 24. *Diagrama de casos de uso del sistema*



Fuente: Elaboración propia con draw.io[20]

[20] draw.io: https://app.diagrams.net/

Para cada caso de uso se detallarán el escenario principal y el diagrama de secuencia asociado al flujo de las operaciones que se realicen.

5.1. Caso de uso #1: Obtener definiciones de conceptos de programación

Esta funcionalidad se encarga de proporcionar definiciones breves y precisas de conceptos básicos de programación relacionados con el problema en cuestión para complementar lo explicado por el profesor y entender mejor los conceptos, facilitando el aprendizaje.

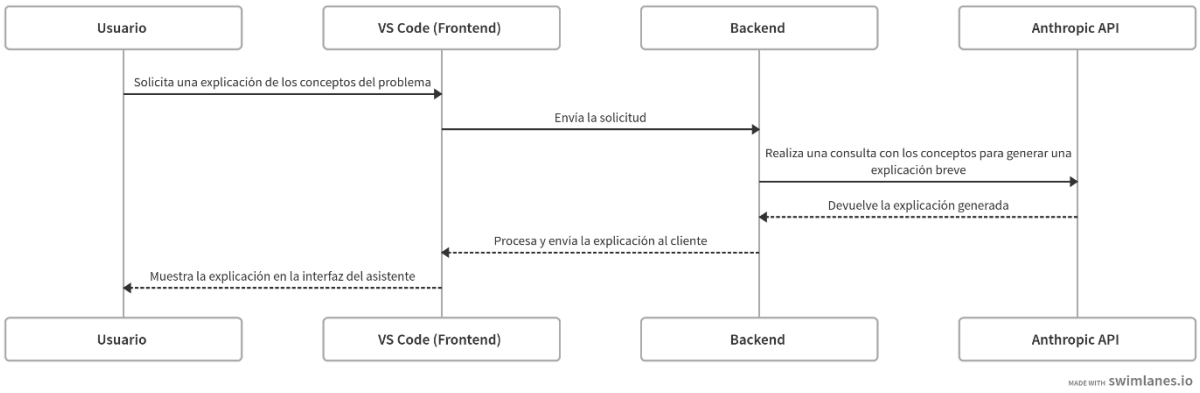
Tabla 1. *Especificación de caso de uso UC-001*

ID (identificador)
UC-001 Obtener definiciones de conceptos básicos de programación
Actor principal
S1 - Estudiante
Escenario Principal
1. El estudiante accede a la interfaz del sistema. 2. El estudiante introduce el problema a resolver. 3. El estudiante selecciona el lenguaje de programación utilizado. 4. El estudiante selecciona la opción “Explicar conceptos claves”. 5. El sistema construye una consulta apropiada para la API de Anthropic. 6. El sistema recibe y procesa la respuesta de la API. 7. El sistema formatea la definición para mejorar su legibilidad y presentación. 8. El sistema muestra al estudiante la definición de los conceptos básicos.

Fuente: Elaboración propia

La **figura 25** representa el flujo de interacción entre los componentes del sistema cuando el usuario solicita la explicación de los conceptos del problema:

Figura 25. *Diagrama de secuencia para el caso de uso UC-001*



Fuente: Elaboración propia con swimlanes.io[21]

5.2. Caso de uso #2: Generar ejemplos en pseudocódigo y diagramas de flujo

Esta funcionalidad se encarga de generar ejemplos en pseudocódigo acompañados de diagramas de flujo que ilustren la estructura del problema para comprenderlo de manera más clara y visualizar su solución paso a paso.

Tabla 2. *Especificación de caso de uso UC-002*

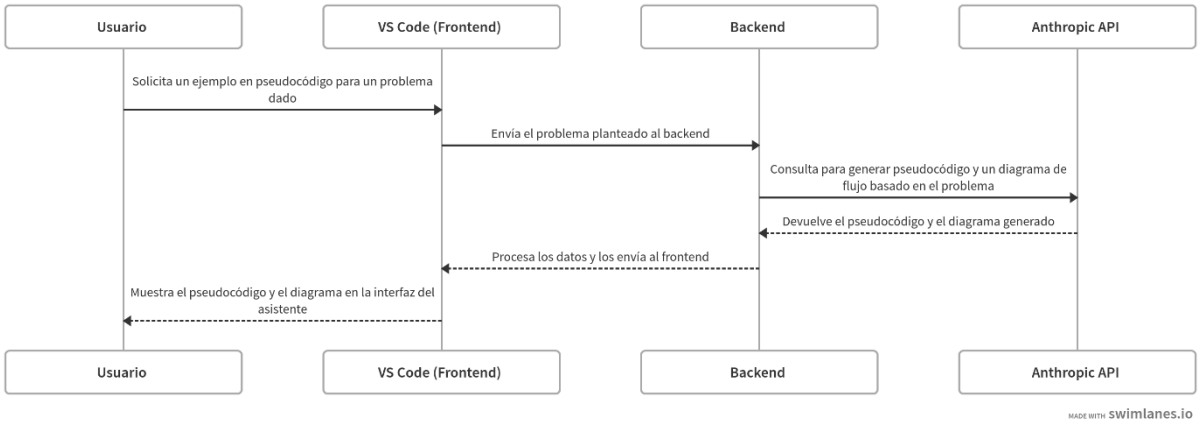
ID (identificador)
UC-002 Generar ejemplos en pseudocódigo acompañados de diagramas de flujo
Actor principal
S1 - Estudiante
Escenario Principal
1. El estudiante accede a la interfaz del sistema. 2. El estudiante introduce el problema a resolver. 3. El estudiante selecciona el lenguaje de programación utilizado. 4. El estudiante selecciona la opción “Ver pseudocódigo y diagrama”. 5. El sistema genera el pseudocódigo y diagrama correspondiente utilizando la API de Anthropic. 6. El sistema presenta al estudiante tanto el pseudocódigo como el diagrama en el entorno.

Fuente: Elaboración propia

[21] swimlanes.io: https://swimlanes.io/

La figura 26 detalla la interacción entre los componentes del sistema para generar y mostrar ejemplos en pseudocódigo y diagramas de flujo en respuesta a un problema planteado por el usuario:

Figura 26. *Diagrama de secuencia para el caso de uso UC-002*



Fuente: Elaboración propia con swimlanes.io

5.3. Caso de uso #3: Ofrecer al usuario pistas iniciales para abordar el problema

Esta funcionalidad se encarga de proporcionar una lista de pistas iniciales para abordar el problema para saber cómo comenzar la solución y estructurar el enfoque de manera efectiva.

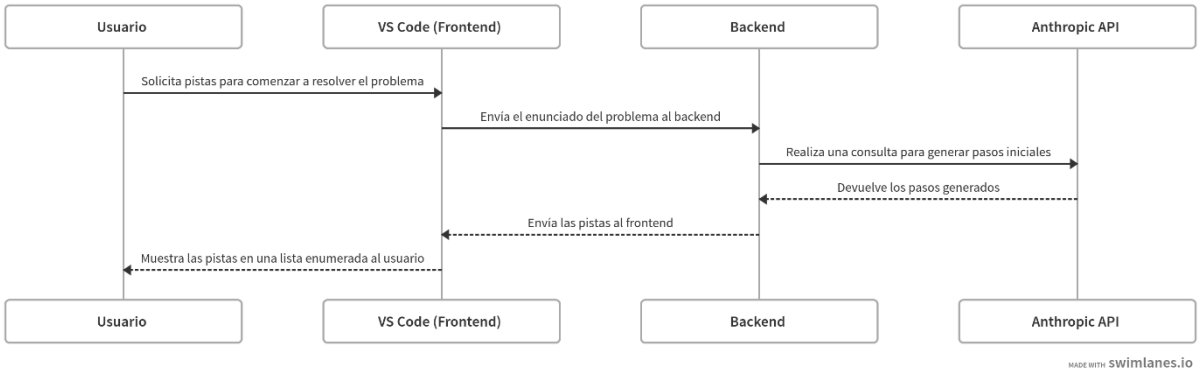
Tabla 3. *Especificación de caso de uso UC-003*

ID (identificador)
UC-003 Ofrecer al usuario pistas iniciales para abordar el problema de programación
Actor principal
S1 - Estudiante
Escenario Principal
1. El estudiante accede a la interfaz del sistema. 2. El estudiante introduce el problema a resolver. 3. El estudiante selecciona el lenguaje de programación utilizado. 4. El estudiante selecciona la opción “Recibir una pista”. 5. El sistema genera una lista ordenada de pistas utilizando la API de Anthropic. 6. El sistema presenta al estudiante las pistas de manera ordenada en la interfaz del entorno.

Fuente: Elaboración propia

La figura 27 muestra el flujo de interacción para el caso de uso UC-003, donde el sistema proporciona pistas iniciales al usuario para abordar el problema.

Figura 27. *Diagrama de secuencia para el caso de uso UC-003*



Fuente: Elaboración propia con swimlanes.io

5.4. Caso de uso #4: Proporcionar sugerencias para el próximo paso

Esta funcionalidad ofrece asistencia inmediata durante el proceso de programar. Se encarga de proporcionar sugerencias para el próximo paso en el código, basadas en el contexto existente que el usuario seleccione con el objetivo de avanzar en la resolución del problema cuando el usuario no sabe cómo avanzar o se encuentra atascado con un error. Esta retroalimentación consiste en explicaciones sobre lo que está mal en el código existente o sugerencias sobre cómo solucionar el error.

Tabla 4. *Especificación de caso de uso UC-004*

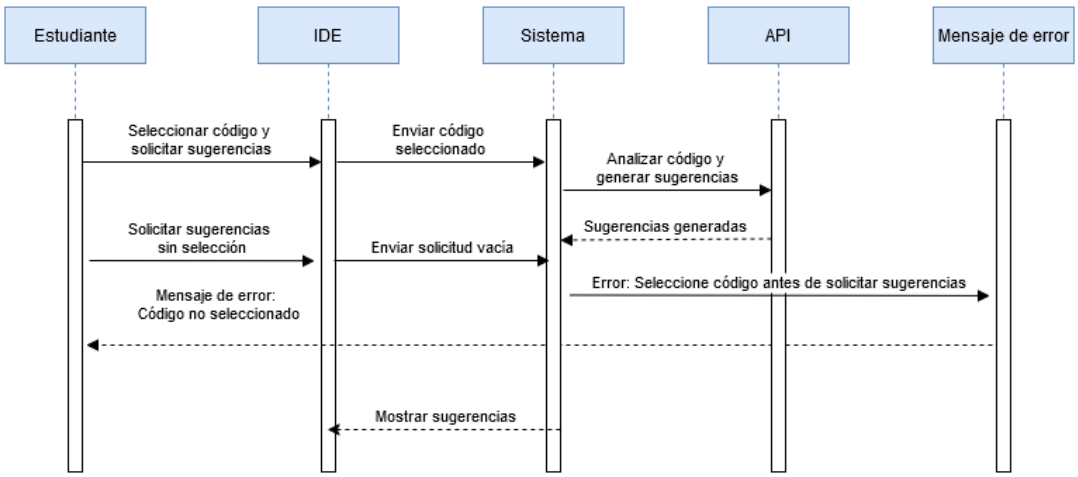
ID (identificador)
UC-004 Proporcionar sugerencias para el próximo paso en el desarrollo del código
Actor principal
S1 - Estudiante
Escenario Principal
1. El estudiante accede a la interfaz del sistema. 2. El estudiante introduce el problema a resolver. 3. El estudiante selecciona el lenguaje de programación utilizado. 4. El estudiante selecciona el código sobre el que necesita ayuda. 5. El estudiante selecciona la opción de “Ver próximo paso”.

6. El sistema verifica que se ha seleccionado código. 7. El sistema genera una sugerencia utilizando la librería highlight.js y la API de Anthropic. 8. El sistema presenta al estudiante las sugerencias con ejemplos de implementación.

Fuente: Elaboración propia

La figura 28 muestra el flujo de interacción para el caso de uso UC-004, donde el sistema proporciona sugerencias para el siguiente paso en el código basadas en el contexto actual.

Figura 28. *Diagrama de secuencia del UC-004*



Fuente: Elaboración propia con draw.io

5.5. Caso de uso #5: Proporcionar explicaciones de soluciones

Esta funcionalidad se encarga de proporcionar explicaciones detalladas de las soluciones posibles, independientemente de si el intento previo del usuario fue correcto o no, para comprender la lógica detrás de una respuesta correcta y aprender diferentes formas de enfocar el problema.

Tabla 5. *Especificación de caso de uso UC-005*

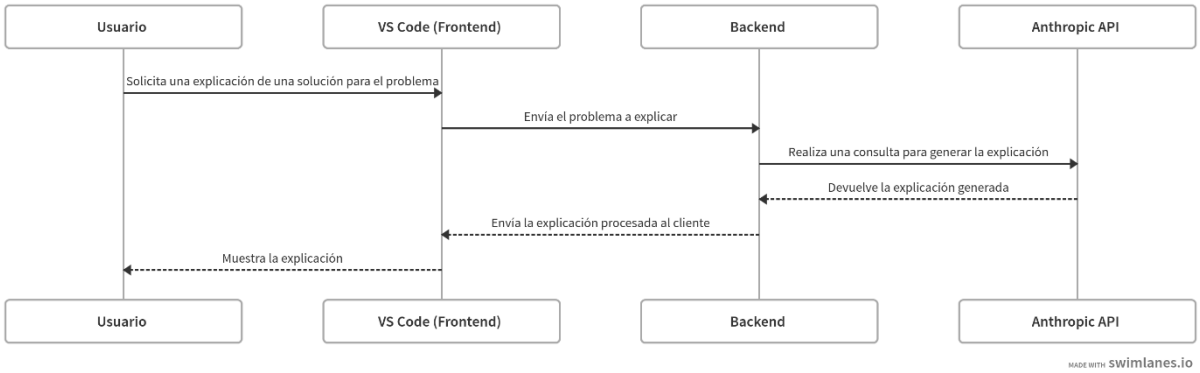
ID (identificador)
UC-005 Proporcionar explicaciones detalladas de soluciones de código
Actor principal
S1 - Estudiante

Escenario Principal
1. El estudiante accede a la interfaz del sistema. 2. El estudiante introduce el problema a resolver. 3. El estudiante selecciona el lenguaje de programación utilizado. 4. El estudiante selecciona la opción “Explicar solución”. 5. El sistema genera una solución con ejemplos utilizando la API de Anthropic. 6. El sistema presenta al estudiante las explicaciones con ejemplos.

Fuente: Elaboración propia

La **figura 29** muestra el flujo de interacción para el caso de uso UC-005, donde el sistema permite al usuario solicitar una explicación de la solución propuesta por el modelo.

Figura 29. *Diagrama de secuencia para el caso de uso UC-005*



Fuente: Elaboración propia con swimlanes.io

5.6. Caso de uso #6: Generar preguntas de seguimiento

Esta funcionalidad se encarga de generar preguntas de seguimiento para seguir profundizando en temas que conciernen al problema para ampliar los conocimientos del estudiante.

Tabla 6. *Especificación de caso de uso UC-006*

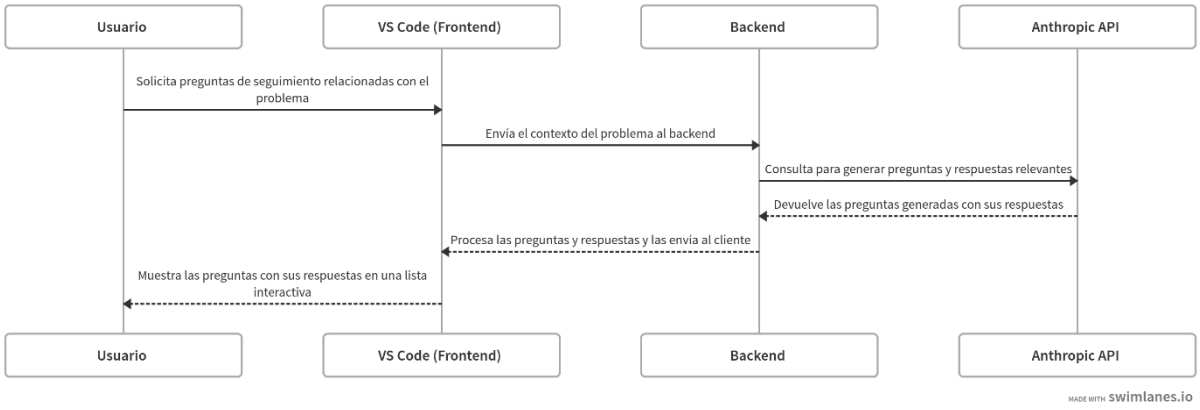
ID (identificador)
UC-006 Generar preguntas relacionadas con el problema planteado
Actor principal
S1 - Estudiante

Escenario Principal
1. El estudiante accede a la interfaz del sistema. 2. El estudiante introduce el problema a resolver. 3. El estudiante selecciona el lenguaje de programación utilizado. 4. El estudiante selecciona la opción “Hacer preguntas de seguimiento”. 5. El sistema genera unas preguntas relacionadas con el problema utilizando la API de Anthropic. 6. El sistema presenta al estudiante las preguntas con su respuesta para comprobar.

Fuente: Elaboración propia

La figura 30 ilustra el flujo de interacción para el caso de uso UC-006, donde el sistema genera preguntas de seguimiento basadas en el problema de manera que el usuario profundice en la comprensión del problema y tenga una visión más holística.

Figura 30. *Diagrama de secuencia para el caso de uso UC-006*



Fuente: Elaboración propia con swimlanes.io

5.7. Requisitos no funcionales

Los requisitos no funcionales están enfocados a los atributos de calidad y parámetros operativos del asistente. Estos requisitos buscan garantizar una experiencia de aprendizaje efectiva en condiciones reales que apoye la comprensión de problemas de programación y se definen para todos los componentes del asistente:

- **RNF001 - Disponibilidad**: El asistente debe asegurar una disponibilidad del 99.9%, con infraestructura que permita soporte continuo incluso durante actualizaciones.

- **RNF002 - Compatibilidad**: El sistema debe ser completamente funcional en Visual Studio Code v1.96+ y garantizar soporte para todos los sistemas operativos.
- **RNF003 - Usabilidad**: La interfaz debe ser intuitiva y estar integrada de manera no intrusiva en el flujo de trabajo del IDE, siguiendo las guías de diseño específicas de Visual Studio Code.
- **RNF004 - Eficiencia**: El asistente debe priorizar el uso eficiente de los tokens del modelo de lenguaje y optimizar el uso de CPU y memoria en entornos locales.
- **RNF005 - Código abierto**: El proyecto debe estar disponible en GitHub bajo una licencia permisiva (MIT, Apache 2.0) que permita la colaboración y personalización por parte de la comunidad.
- **RNF006 - Accesibilidad**: Los elementos de la interfaz deben ser completamente accesibles (correcto funcionamiento de las flechas de navegación, botones claramente visibles, etc).
- **RNF007 - Contextualización**: Las respuestas deben adaptarse al lenguaje específico, considerando las restricciones típicas de entornos educativos como limitaciones en el uso de ciertas funciones o bibliotecas.
- **RNF008 – Portabilidad del código**: El código proporcionado debe mantener su formato y estilo (incluyendo la indentación) al ser copiado y pegado.

5.8. Alcance de la primera versión

En esta primera versión, el asistente ofrece un selector para los lenguajes Python, Java, C y C++. Más allá de la experiencia de usuario, se diseña el prototipo como una extensión IDE para ofrecer más fácilmente contexto del problema y del código al LLM ya que los participantes pueden seleccionar partes del código para ver el próximo paso o pueden usar el panel de acciones para seleccionar otro tipo de ayuda.

Quedan fuera del alcance de este trabajo intencionalmente la generación de código por chat para poner el foco en cómo los estudiantes comprenden el problema. Además, se deja para futuras versiones el soporte para otros IDEs distintos a Visual Studio Code u otros modelos diferentes a Anthropic.

6. Diseño e implementación

En esta sección se aborda la fase de diseño del asistente desde un enfoque técnico. Se detalla la arquitectura elegida, las tecnologías utilizadas, los *wireframes* realizados y se profundiza en cada uno de los componentes que componen la interfaz. El asistente está desarrollado en JavaScript y utiliza una arquitectura cliente-servidor que permite la comunicación con la API de Anthropic para generar respuestas.

6.1. Tecnologías utilizadas

Con base en los casos de uso y requisitos no funcionales, se presentan a continuación las soluciones tecnológicas adoptadas para el presente trabajo, junto con una justificación detallada de su elección.

6.1.1. TypeScript

TypeScript es un superconjunto de JavaScript que incorpora tipado estático, lo cual permite detectar errores en tiempo de compilación y mejorar la calidad del código. Su uso facilita el mantenimiento y escalabilidad del asistente, proporcionando una estructura más robusta y clara al desarrollo. El archivo tsconfig.json no solo configura el compilador TypeScript, sino que también define rutas de módulos y opciones para el *target* de compilación. Este archivo es clave para garantizar compatibilidad multiplataforma.

6.1.2. JavaScript

JavaScript permite la gestión de interacciones como el envío de preguntas al modelo y el manejo de respuestas. Estos eventos se manejan en el archivo main.js. Por su parte, highlight.js se usa para destacar sintaxis en los bloques de código.

6.1.3. Hojas de estilo CSS

Los componentes interactivos principales del asistente se definen en el archivo main.css. Incluyen botones, tarjetas de contenido y menús desplegables. Con CSS se crean, por ejemplo, tarjetas para cada concepto y un contenedor para el *slider* en la funcionalidad de los conceptos. Otro ejemplo del uso de CSS es el icono en la esquina derecha de la *WebView* que permite redimensionar o ampliar la caja de respuestas verticalmente.

6.1.4. Anthropic API

Para la generación de respuestas, se ha optado por la API de Anthropic, que permite interactuar con modelos avanzados de lenguaje natural. En particular, el modelo Claude 3.5 Sonnet 20241022 ha sido seleccionado por su capacidad para comprender y generar texto de manera coherente y actualizada, alineándose con las necesidades del proyecto.

6.1.5. Visual Studio Code

Visual Studio Code ha sido elegido como el IDE para el desarrollo y despliegue del asistente. Visual Studio Code es el editor más popular a nivel mundial[22] y ofrece un amplio ecosistema de extensiones. El *frontend* se comunica con el *backend* mediante mensajes enviados a través de la API de extensiones de Visual Studio Code. *vscode* es un módulo importado que permite registrar eventos, manejar comandos, y mostrar vistas personalizadas dentro del IDE.

6.1.6. Node.js + npm

Node.js, junto con npm, se utiliza para gestionar dependencias y scripts en el proyecto. Node.js permite ejecutar código JavaScript en el servidor, mientras que npm facilita la instalación y administración de paquetes necesarios para el desarrollo, asegurando un flujo de trabajo organizado y eficiente.

6.1.7. GitHub

GitHub se emplea como plataforma para el control de versiones y la colaboración en el desarrollo del proyecto. Su uso permite un seguimiento detallado de los cambios en el código, facilita la colaboración futura entre la comunidad desarrolladora y asegura la integridad y disponibilidad del código fuente a lo largo del ciclo de vida del proyecto.

6.1.8. Prettier y ESLint

Prettier[23] (formateo de código) y ESLint[24] (análisis estático) se emplean como herramientas para mantener la calidad y consistencia del código.

[22] 2024 Stack Overflow Developer Survey: https://survey.stackoverflow.co/2024/
[23] Prettier: https://prettier.io/
[24] ESLint: https://eslint.org/

6.2. Arquitectura del sistema

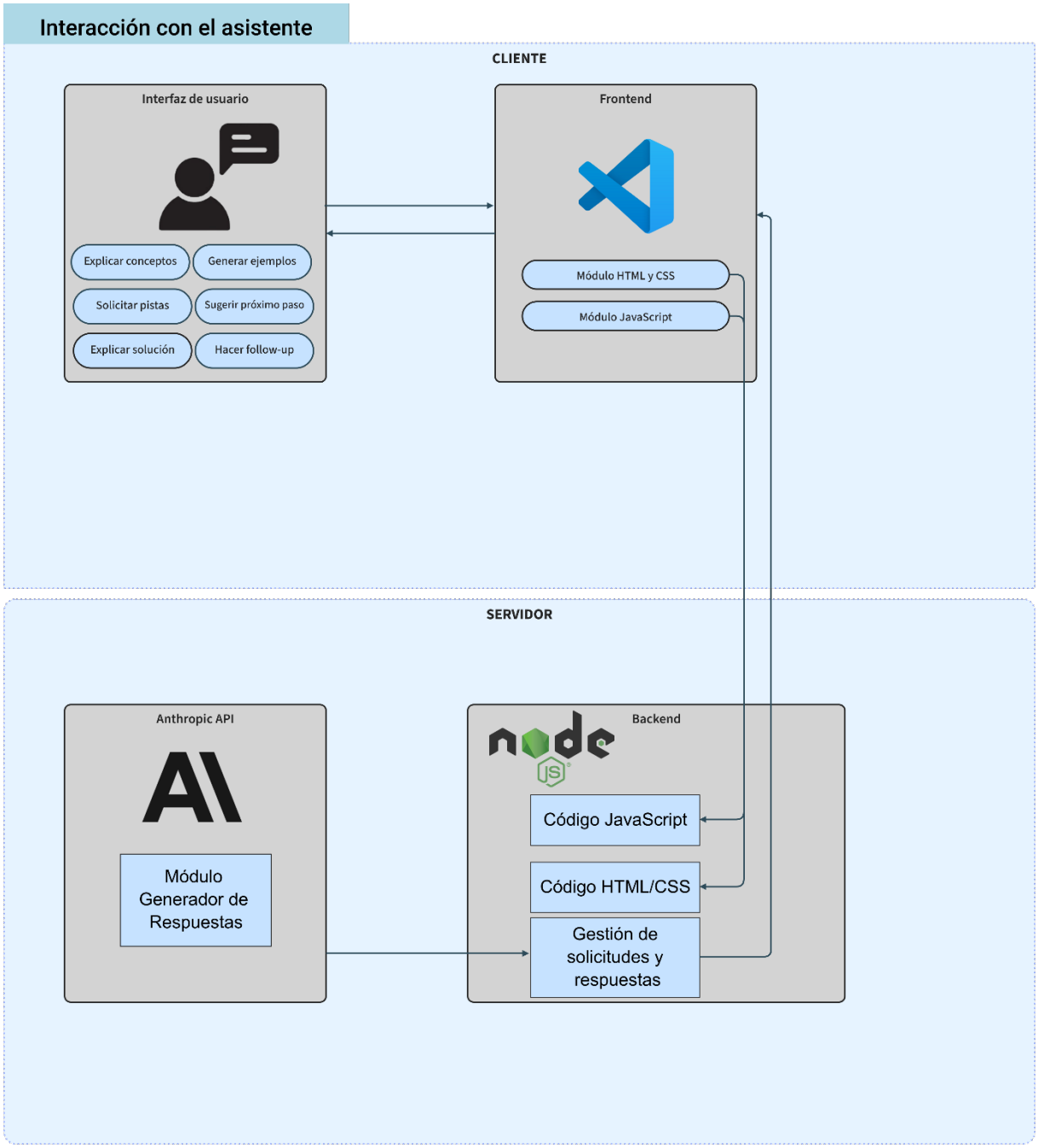
El papel de cada uno de los elementos que componen la arquitectura es el siguiente:

- **Interfaz de usuario:** La extensión Buddy para Visual Studio Code es el modo de interacción con el usuario final. A través de esta interfaz, los usuarios pueden solicitar explicaciones de conceptos, generar ejemplos o recibir sugerencias para resolver problemas. Las respuestas generadas y procesadas se muestran en esta interfaz.
- ***Frontend*:** Se utilizan módulos de HTML, CSS y JavaScript. Este componente gestiona la presentación visual y las funcionalidades interactivas de la interfaz de usuario. Además, recibe las respuestas procesadas por el *backend* y las presenta para que el usuario pueda interactuar con ellas.
- ***Backend*:** Se basa en una arquitectura modular construida con Node.js. Su responsabilidad principal es gestionar las solicitudes y respuestas del sistema, procesando las interacciones que vienen del *frontend*. Asimismo, garantiza una comunicación eficiente con la API para obtener y procesar las respuestas generadas por Claude[25].
- **Anthropic API:** Componente encargado de generar las respuestas. Este servicio de IA procesa las consultas enviadas por el usuario, generando respuestas que son devueltas al *backend* para procesarlas y posteriormente enviarlas al *frontend*.

El sistema construido consta de los componentes que se ilustran en la **figura 31**:

[25] Claude: https://claude.ai/

Figura 31. *Arquitectura cliente-servidor del asistente*



Fuente: Elaboración propia con Moqups[26]

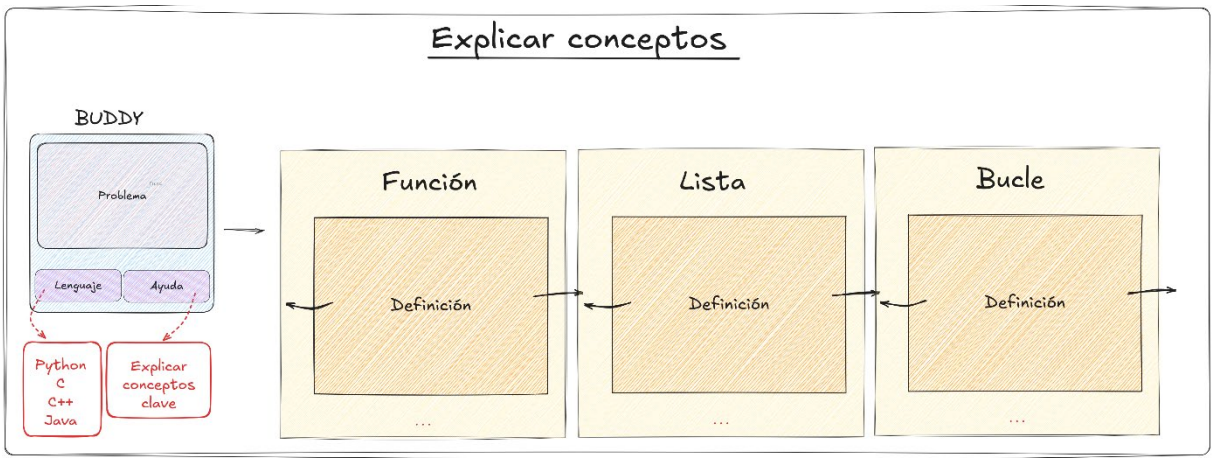
[26] Moqups: https://moqups.com/es/

6.3. Wireframes y resultados

Antes de construir el *frontend*, se diseñaron una serie de prototipos digitales que fueron validados por usuarios en entrevistas durante la fase de comprensión para entender mejor el diseño del asistente. Los wireframes se realizaron únicamente en formato desktop, ya que es donde se da el flujo de trabajo con Visual Studio Code. La decisión de utilizar wireframes se tomó para obtener un buen resultado de manera rápida, siguiendo los principios de la metodología de trabajo. Para poder apreciar los pasos de *wireframes* a digital se muestra el siguiente proceso:

La **figura 32** muestra el *wireframe* del componente "**Explicar conceptos claves"**. El diseño presenta un modelo interactivo que facilita la comprensión de los conceptos fundamentales del problema.

Figura 32. *Wireframe conceptos*



Fuente: Elaboración propia con Excalidraw[27]

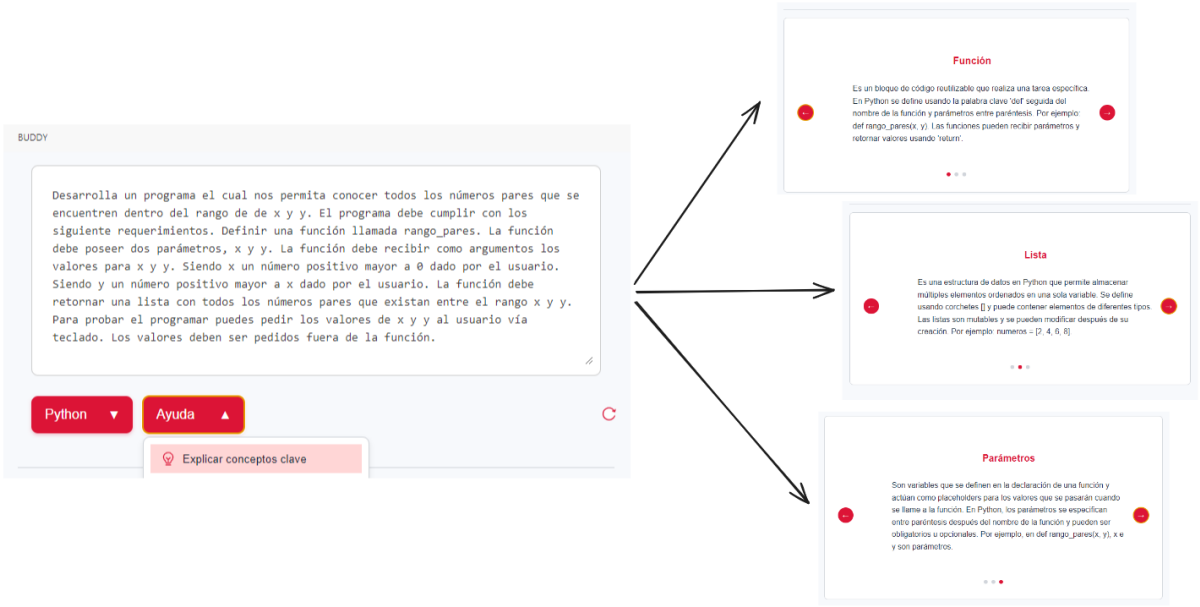
Además de la *WebView*, común en todos los componentes, **el área de visualización** muestra los tres conceptos claves para entender el problema. Este componente implementa un sistema de tarjetas deslizables que muestran definiciones y ejemplos relacionados con el problema planteado.

[27] Excalidraw: https://excalidraw.com/

En este caso concreto, se pueden observar conceptos fundamentales de programación como son **función, lista y bucle**, que se representan visualmente como tarjetas independientes pero interconectadas.

En la figura 33 se puede comprobar el resultado final en digital con el flujo de interacción y la generación de los conceptos. Cada concepto se presenta como una unidad independiente que incluye título, definición y ejemplos, todo contextualizado en el lenguaje seleccionado.

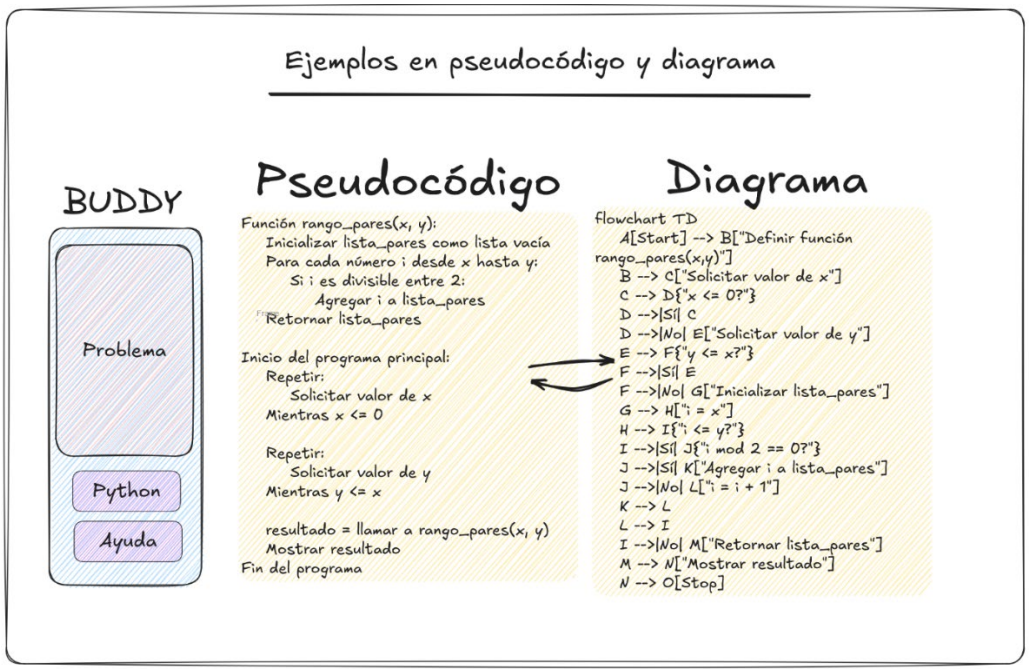
Figura 33. *Componente de conceptos*



Fuente: Elaboración propia

El wireframe de la figura *34* muestra dos secciones principales: pseudocódigo y diagrama. Ambos elementos los muestra con resaltado de sintaxis. La del pseudocódigo detalla la función y la del diagrama de flujo UML complementa la explicación mostrando el proceso paso a paso mediante notación de *flowchart*.

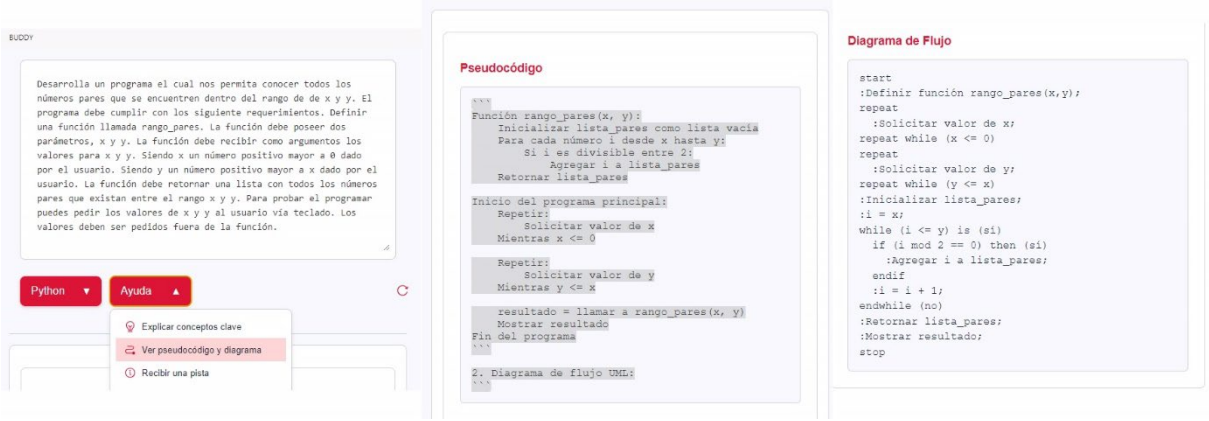
Figura 34. *Wireframe ejemplos*



Fuente: Elaboración propia

El diseño final de la figura 35 muestra el resultado con el pseudocódigo resaltado y el diagrama de flujo interactivo:

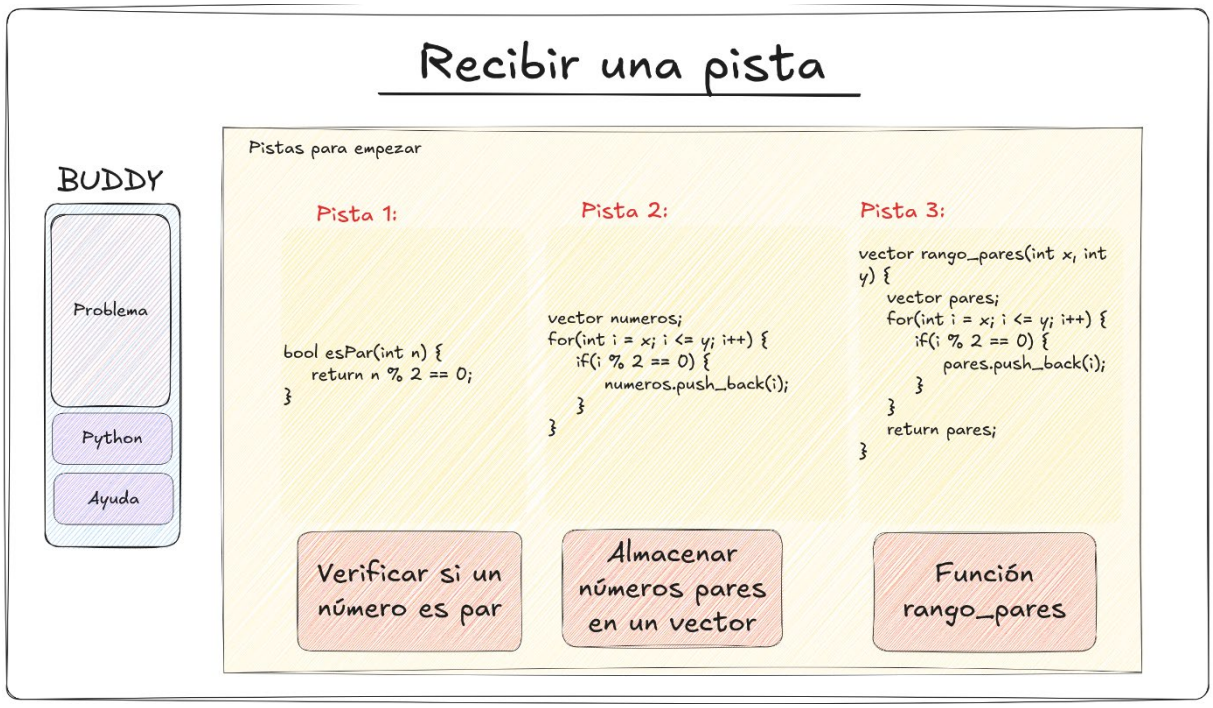
Figura 35. *Componente de ejemplos*



Fuente: Elaboración propia

El *wireframe* del componente pistas de la figura 36 presenta tres pistas diferentes para ayudar en la resolución del problema. Este componente tiene un contador de pistas secuenciales y el área de contenido donde cada pista incluye ejemplos de código con la sintaxis del lenguaje seleccionado. Cada pista incluye código de ejemplo y una breve descripción.

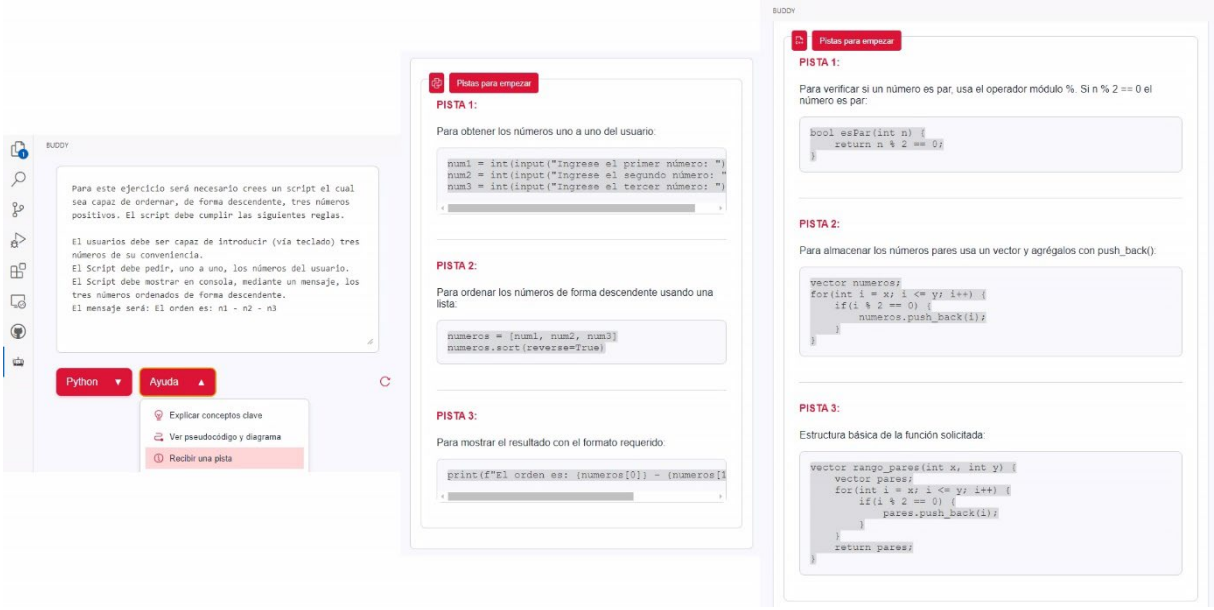
Figura 36. *Wireframe pistas*



Fuente: Elaboración propia

Por su parte, en el diseño final que se muestra en la [figura 37](#) se observa cómo se presentan estas pistas de manera interactiva, permitiendo al usuario acceder a cada una según la necesite.

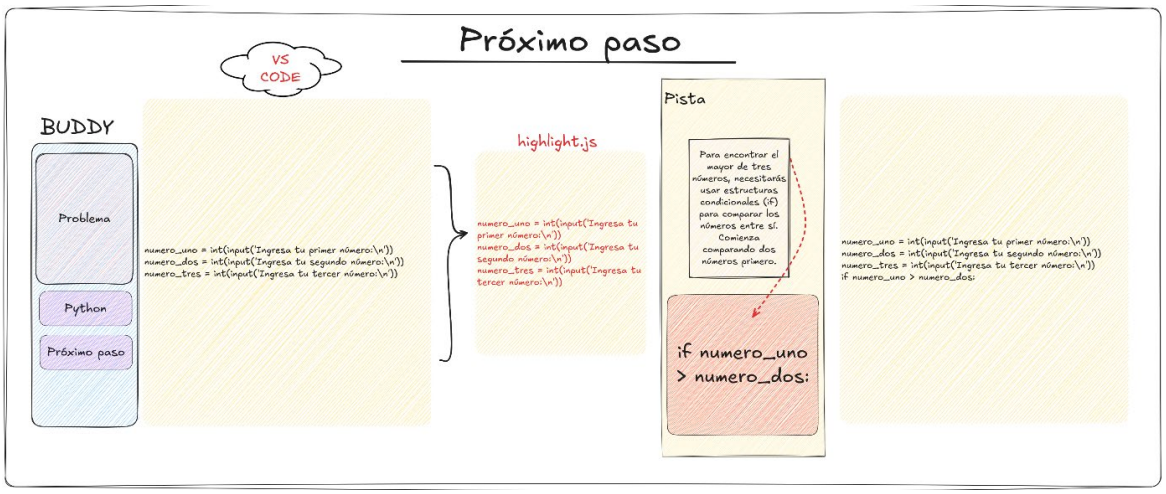
Figura 37. *Componente de pistas*



Fuente: Elaboración propia

El *wireframe* de la figura 38 muestra la interfaz con un área de código y una sección de pista. El uso de este componente está pensado para que el usuario pueda avanzar en la resolución del problema. Sugiere el siguiente paso lógico en el código, mostrando una explicación y el fragmento de código propuesto. Se incluye highlight.js para resaltar la sintaxis del código.

Figura 38. *Wireframe próximo paso*



Fuente: Elaboración propia

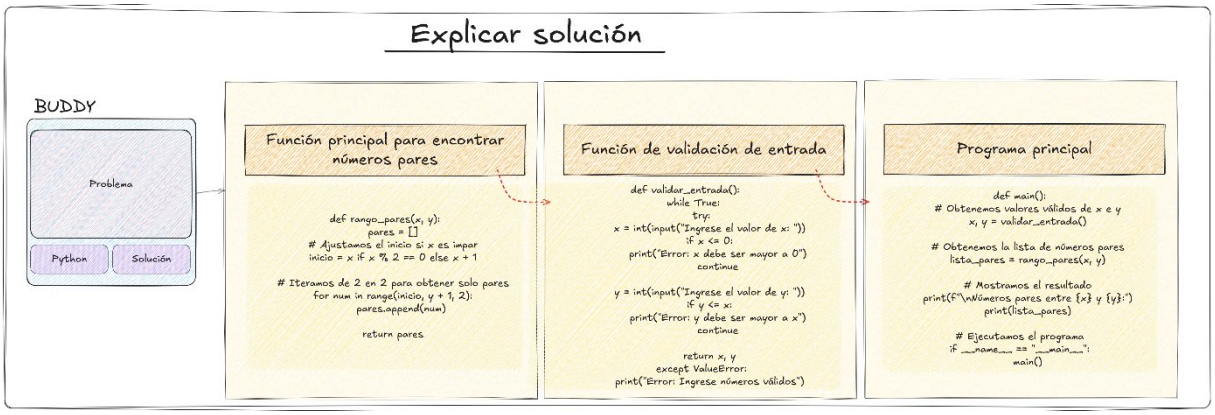
La versión digital final de la figura 39 permite entender cómo sería el flujo de interacción entre los diferentes pasos, con indicaciones basadas en el código subrayado.

Figura 39. *Componente de próximo paso*

Fuente: Elaboración propia

El siguiente *wireframe* de la funcionalidad "**Explicar solución**" divide la solución en tres pantallas. Este componente tiene un *slider* para la visualización de la solución paso a paso, tal y como se muestra en la figura 40:

Figura 40. *Wireframe solución*



Fuente: Elaboración propia

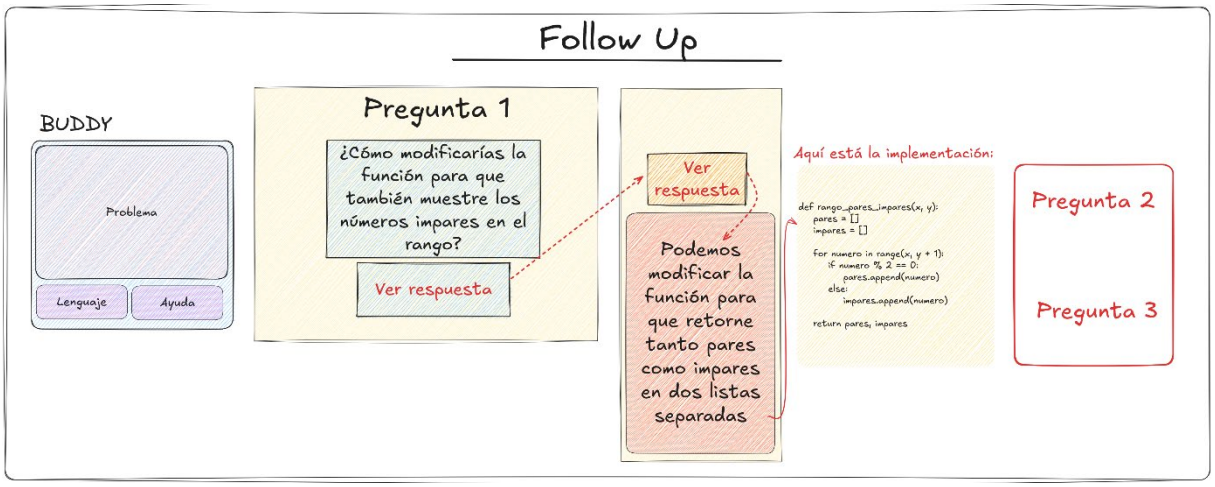
En la versión final de la figura 41 se pueden ver estas secciones con ejemplos de código y explicaciones paso a paso.

Figura 41. *Componente solución*

Fuente: Elaboración propia

El *wireframe* de la figura 42 presenta un formato de preguntas y respuestas. Este componente usa un sistema de acordeón con tarjetas expandibles que muestran preguntas y respuestas detalladas relacionadas con el problema planteado.

Figura 42. *Wireframe preguntas de seguimiento*



Fuente: Elaboración propia

La versión final representada en la [figura 43](#) permite al usuario ver las respuestas y navegar entre diferentes preguntas de seguimiento para profundizar y ahondar en el problema:

Figura 43. *Componente Follow Up*

Fuente: Elaboración propia

6.4. Desarrollo del Frontend

El *frontend* del asistente contiene los archivos principales para la interfaz de usuario: *main.js* y *main.css*. El archivo *main.js* se encarga de la lógica de interacción y los manejadores de eventos, mientras que *main.css* define los estilos y las animaciones de la interfaz.

La integración con la ***WebView*** de Visual Studio Code se implementa en el archivo ***Buddy-view-provider.js*** del *backend*, a través del método *getHtml*, que genera el HTML base para la *WebView* y establece la conexión entre el *frontend* y Visual Studio Code. A continuación, se detallan los detalles técnicos de la *WebView* y de los componentes del asistente.

6.4.1. WebView

La implementación de la interfaz de usuario del asistente se realiza mediante una única *WebView* que actúa como contenedor principal de todos los componentes. Esta arquitectura permite una interacción eficiente y una comunicación bidireccional entre el *frontend* y el *backend*. La *WebView* está compuesta por varios elementos de control:

1. Selector de lenguaje

El selector de lenguaje implementa un sistema de gestión de estado mediante la variable *currentLanguage*, que mantiene y actualiza el lenguaje de programación seleccionado, así como *closeLanguageDropdown* que cierra el menú desplegable tras iniciar la solicitud. La comunicación con el *backend* se realiza a través de mensajes tipados *languageChanged*. El siguiente fragmento de código ilustra su implementación dentro de *main.js*:

```
// Toggle del dropdown de lenguaje
languageButton?.addEventListener('click', (e) => {
    e.stopPropagation();
    const isHidden = languageOptions.classList.contains('hidden');
    const arrow = languageButton.querySelector('.dropdown-arrow');

    if (isHidden) {
        languageOptions.classList.remove('hidden');
        arrow.style.transform = 'rotate(180deg)';
    } else {
        closeLanguageDropdown();
    }
});

// Event listeners para las opciones de lenguaje
document.querySelectorAll('#language-options .dropdown-item').forEach(option => {
        option.addEventListener('click', (e) => {
            const selectedLanguage = e.currentTarget.dataset.language;
            currentLanguage = selectedLanguage;
            languageButton.querySelector('span').textContent =
e.currentTarget.textContent.trim();
            closeLanguageDropdown();
```

```
            // Notificar al backend del cambio de lenguaje
            vscode.postMessage({
                type: 'languageChanged',
                language: selectedLanguage
            });
        });
    });

// Actualizar el cierre del dropdown al hacer clic fuera
document.addEventListener('click', (e) => {
    if (!languageOptions?.contains(e.target) &&
!languageButton?.contains(e.target)) {
        closeLanguageDropdown();
    }
});
```

2. Menú de ayuda

El menú de ayuda se implementa como un componente que agrupa seis funcionalidades de ayuda en el aprendizaje. Se implementa un sistema de eventos distribuido donde cada opción de ayuda mantiene su propia lógica. Esto se logra mediante *event listeners* específicos que gestionan las interacciones del usuario y procesan las solicitudes correspondientes.

La comunicación con el *backend* se realiza mediante un sistema de mensajería tipada, que se implementa a través del API de Visual Studio Code. Para gestionar los estados asíncronos, cada acción de ayuda implementa su propio sistema de gestión de estados de carga, lo que proporciona *feedback* visual al usuario durante el procesamiento de las solicitudes.

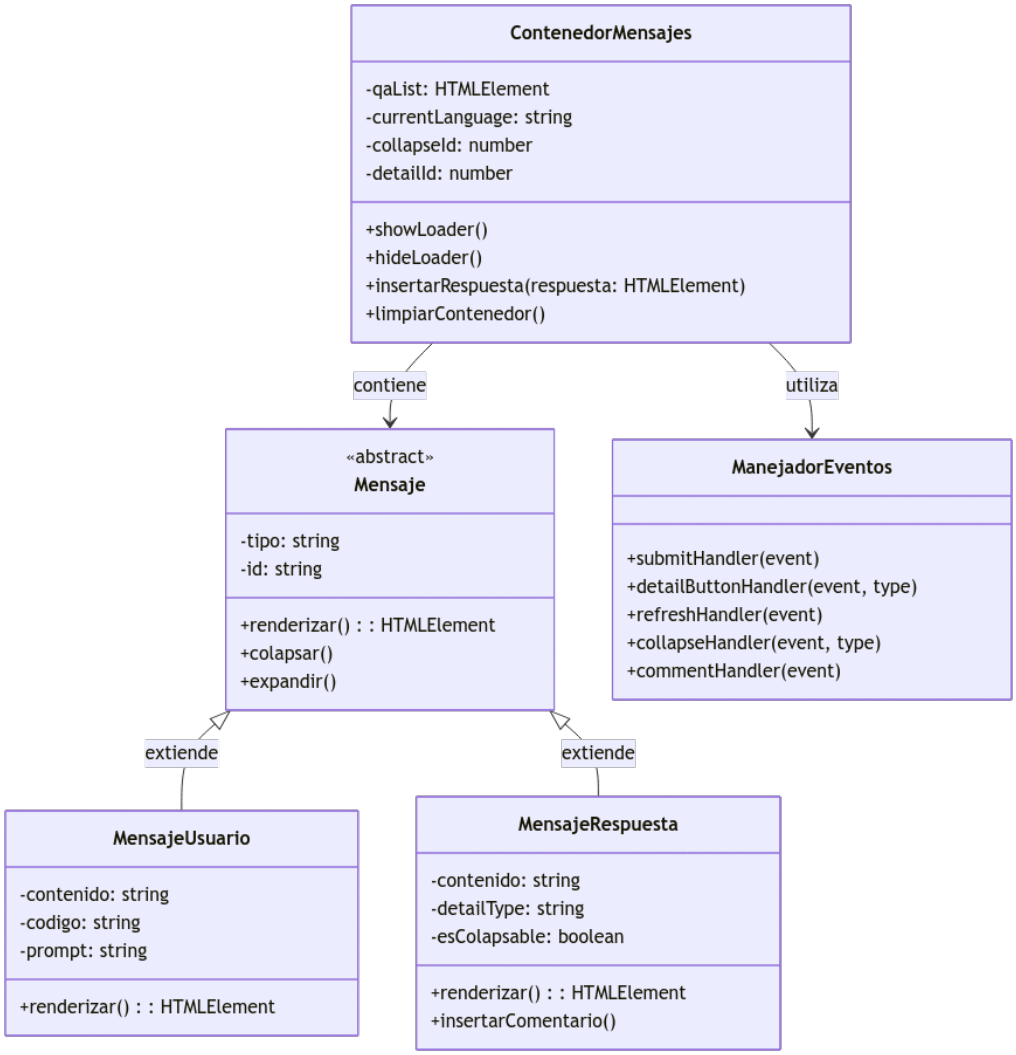
3. Contenedor de mensajes

Como se puede observar en la **figura 44**, el sistema de procesamiento y visualización de respuestas está diseñado para manejar diferentes tipos de contenido, incluyendo texto formateado y fragmentos de código, para asegurar una presentación clara y accesible de la información.

La comunicación bidireccional se gestiona de manera asíncrona. Así se facilita una experiencia de usuario fluida mientras se procesan las solicitudes en segundo plano.

Otro elemento de la interfaz que facilita la visualización de las respuestas es el botón de limpieza. Este elemento es el encargado de eliminar todo el contenido del contenedor estableciendo su propiedad *innerHTML* a una cadena vacía. De esta manera se consiguen eliminar todos los mensajes previos de la interfaz.

Figura 44. *Diagrama de Clases UML del Sistema de Contenedor de Mensajes*



Fuente: Elaboración propia con Mermaid[28]

6.4.2. Componentes

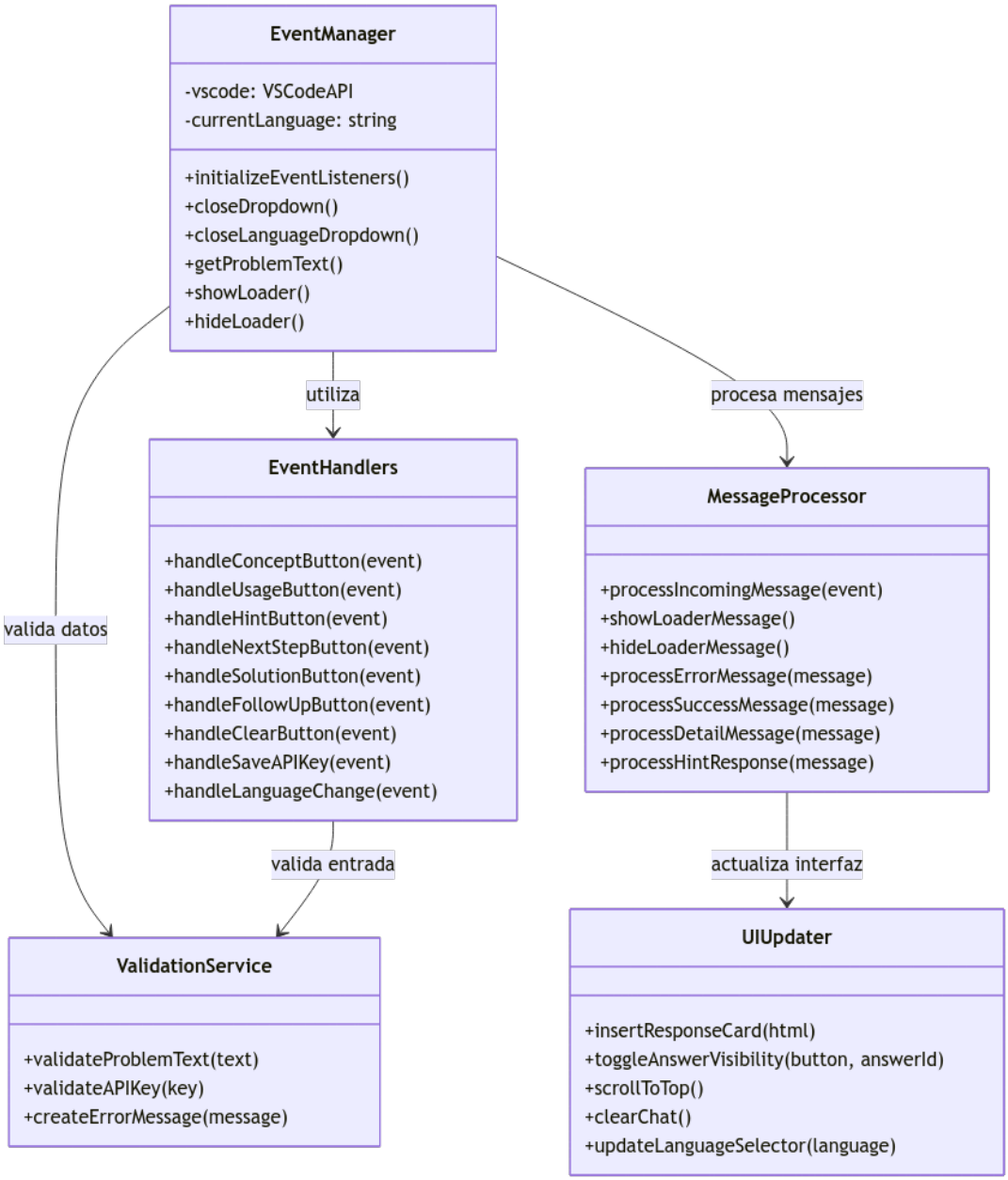
Desde el punto de vista estructural, se define un patrón común que comparten todos los componentes del *frontend*. El *main.js* se encarga de toda la interacción con el usuario y la manipulación del DOM, mientras que *Buddy-view-provider.js* maneja toda la lógica de negocio, procesamiento de datos y comunicación con la API. La estructura base de cada uno de los seis componentes contiene: Event listeners, sistema de *prompts*, procesamiento de respuestas y presentación (UI).

[28] Mermaid: https://mermaid.js.org/

6.4.2.1. Event listeners

En el *main.js* se implementa el manejador de eventos (figura 45) para cada componente que consiste en un sistema de validación y comunicación con el *backend*. Primero valida que se encuentre un problema disponible para resolver, prepara el mensaje con el contenido y el lenguaje seleccionado, gestiona el estado mostrando el *loader*, envía el mensaje al *backend* mediante el API de VS Vode y cierra el menú desplegable tras la solicitud.

Figura 45. *Diagrama UML de Event Listeners en el sistema*



Fuente: Elaboración propia con Mermaid

6.4.2.2. Sistema de *prompts*

El sistema de *prompts* establece un formato específico para facilitar el procesamiento con *regex*. Por ejemplo, en el caso de los conceptos solicita solo 3 conceptos, adapta las definiciones al lenguaje seleccionado y mantiene una estructura clara con títulos y explicaciones separadas. A continuación, se muestra una tabla resumen con todo el sistema de promps y los formatos de respuesta para cada funcionalidad:

Figura 46. *Sistema de prompts*

Funcionalidad	Tipo de Consulta	Formato de Respuesta	Propósito
Conceptos	`askAIConcept`	CONCEPTO: [nombre] EXPLICACIÓN: [detalle]	Identificar y explicar conceptos fundamentales del problema
Pseudocódigo y diagramas	`askAIUsage`	Pseudocódigo: [código] Diagrama de flujo: @startuml	Proporcionar implementación práctica y representación visual
Pista	`askAIHint`	PISTA 1: [consejo con código] PISTA 2/3: [más consejos]	Ofrecer guía inicial sin revelar la solución completa
Siguiente Paso	`askAINextStep`	EXPLICACIÓN: [razón] SIGUIENTE PASO: [código]	Asistir en el desarrollo incremental del código
Solución	`askAISolution`	PARTE1/2/3: TÍTULO + CONTENIDO	Explicación estructurada y completa de la solución
Preguntas	`Seguimiento`	PREGUNTA1/2/3: TÍTULO + RESPUESTA	Profundizar en aspectos avanzados del problema

Fuente: Elaboración propia

6.4.2.3. Procesamiento de respuestas

El sistema de *prompts* se apoya en el procesador de respuestas (*processQueryResponse*). Este método asíncrono implementa la lógica necesaria para convertir la salida en bruto en datos estructurados mediante expresiones regulares. Estas expresiones regulares parsean el texto de entrada, que sigue este formato:

```
if (queryType === "askAIConcept") {
    // Inicializa un array para almacenar los conceptos extraídos del problema
    const concepts = [];
```

```
    // Define una expresión regular para extraer conceptos y explicaciones de la
respuesta de la IA
    // Esta regex busca patrones de "CONCEPTO: [título] EXPLICACIÓN: [contenido]"
    const conceptRegex =
/CONCEPTO:\s*(.*?)\s*EXPLICACIÓN:\s*([\s\S]*?)(?=\s*CONCEPTO:|$)/g;
    let match;

    // Itera a través de todos los matches encontrados en la salida
    while ((match = conceptRegex.exec(output)) !== null) {
        // Para cada match, añade un objeto con título y contenido al array de
conceptos
        concepts.push({
            title: match[1].trim(),  // El primer grupo capturado es el título del
concepto
            content: match[2].trim() // El segundo grupo es la explicación del
concepto
        });
    }

    // Verifica si se encontraron conceptos, registra error si no hay ninguno
    if (concepts.length === 0) {
        console.error('No se pudieron extraer conceptos de la respuesta:', output);
    }

    // Genera un ID único para el slider basado en la marca de tiempo actual
    const sliderId = `concept-slider-${Date.now()}`;

    // Construye el HTML para mostrar los conceptos en un slider interactivo
    valueHtml = `
    <div class="concepts-container" id="${sliderId}">
        // Contenedor principal del slider
        <div class="concepts-slider">
            // Mapea cada concepto a una tarjeta en el slider
            ${concepts.map((concept, index) => {
                // Formatea el contenido, reemplazando bloques de código con
elementos <pre><code>
                const formattedContent = concept.content.replace(
                    /```(\w+)?\s*([\s\S]*?)```/g,
                    (_, lang, code) => `<pre><code class="language-
${this.currentLanguage}">${code.trim()}</code></pre>`
                );
                return `
                    // Tarjeta individual de concepto, solo la primera es visible
inicialmente
                    <div class="concept-card ${index === 0 ? 'active' : ''}" data-
index="${index}">
                        <h3 class="concept-title">${concept.title}</h3>
                        <div class="concept-
content">${Marked.parse(formattedContent)}</div>
                    </div>
                `;
            }).join('')}
        </div>

        // Botones de navegación para moverse entre conceptos
        <div class="concepts-navigation">
            <button class="concept-nav-button prev"
onclick="prevConcept('${sliderId}')">←</button>
            <button class="concept-nav-button next"
onclick="nextConcept('${sliderId}')">→</button>
        </div>

        // Indicadores de posición para mostrar qué concepto está activo
        <div class="concepts-indicators">
            ${concepts.map((_, index) => `
                <button class="concept-indicator ${index === 0 ? 'active' : ''}"
                        data-index="${index}"
                        onclick="goToSlide('${sliderId}', ${index})"></button>
```

```
            `).join('')}
        </div>
    </div>`;
}
```

6.4.2.4. Presentación (UI)

El siguiente fragmento de código implementa la estructura de presentación utilizando *template strings* de ES6[29]. Por ejemplo, en el caso de los conceptos, para cada *slider*, genera un identificador único basado en un *timestamp*. Además, construye una estructura *DOM* jerárquica que consta de tres componentes principales:

1. **Contenedor de *slides*** (*.concepts-slider*).
2. **Navegación** (*.concepts-navigation*).
3. **Indicadores de posición** (*.concepts-indicators*).

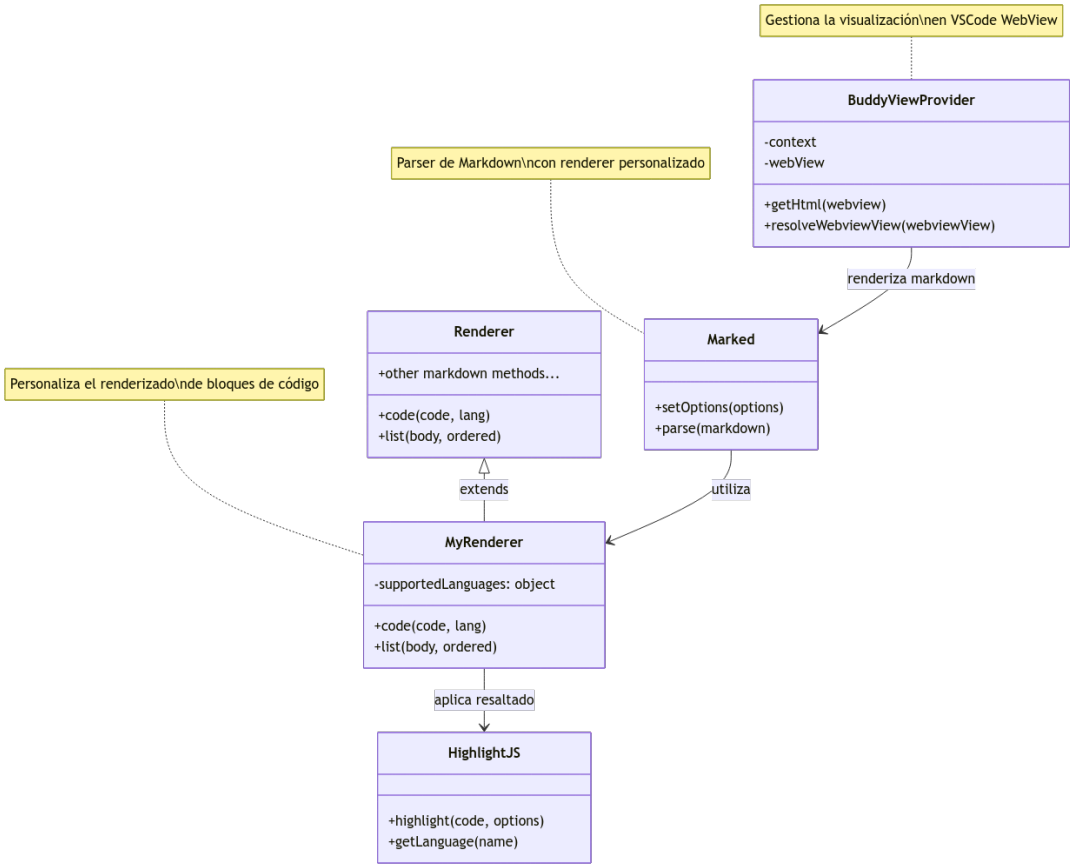
La presentación presenta un carrusel interactivo con navegación y un sistema de estados activo/inactivo que controla la visibilidad de las tarjetas. Además, incluye una funcionalidad de expansión y colapso para las respuestas.

En este otro ejemplo de pistas, el sistema implementa expresiones regulares que dividen el texto en pistas individuales. Mediante el método slice(1, 4), se asegura que el resultado final contenga exactamente tres pistas. Adicionalmente, se aplica el método trim() para eliminar espacios en blanco innecesarios y mantener el formato limpio.

En el caso de uso del próximo paso se utiliza highlight.js[30] para generar el HTML con las clases para el resaltado de la sintaxis, tal y como se explica en la **figura 47**. El asistente inicializa un renderizador para markdown y código mediante la clase MyRenderer. Esta clase extiende de Renderer y se encarga de procesar el resaltado de sintaxis para diferentes lenguajes de programación utilizando highlight.js.

[29] Template Strings en ES6: https://desarrolloweb.com/articulos/template-strings-es6.html
[30] highlight.js: https://highlightjs.org

Figura 47. *highlight.js en Buddy*



Fuente: Elaboración propia con Mermaid

La función *processSolution* en main.js procesa el texto de salida de la API para convertirlo en una sección estructurada que se puede mostrar en un *slider*:

```
async function processSolution(output) {
    // Inicializa un array para almacenar las partes de la solución
    const parts = [];

    // Define una expresión regular para extraer las partes numeradas de la solución
    // Busca patrones como "PARTE1: TÍTULO: [título] CONTENIDO: [contenido]"
    // La expresión captura: número de parte (1), título (2) y contenido (3)
    const partRegex =
/PARTE(\d+):\s*TÍTULO:\s*(.*?)\s*CONTENIDO:\s*([\s\S]*?)(?=PARTE\d+:|$)/g;
    let match;

    // Itera a través de todos los matches encontrados en la salida
    while ((match = partRegex.exec(output)) !== null) {
        // Para cada match, añade un objeto con número, título y contenido al array
        parts.push({
            number: match[1],        // El número de la parte (1, 2, 3, etc.)
            title: match[2].trim(),  // El título de esta parte de la solución
            content: match[3].trim() // El contenido explicativo de esta parte
        });
    }
```

6.5. Desarrollo del Backend

El *backend* gestiona la lógica de negocio en el servidor dentro de la arquitectura de cliente-servidor del asistente.

6.5.1. Anthropic.js

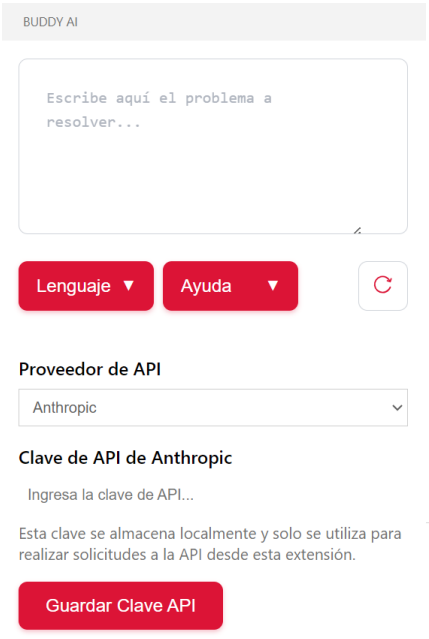
El archivo *anthropic.js* gestiona las solicitudes con el modelo Claude 3.5 mediante la API de Anthropic. El método *createPayload* es crítico en este proceso porque configura los parámetros necesarios para estas solicitudes a la API. Este método establece parámetros como la temperatura (que controla la creatividad de las respuestas), *max_tokens* (que limita la longitud de la respuesta), y el mensaje del sistema que define el comportamiento del asistente.

6.5.2. Gestión de API

La interfaz de gestión de API se divide en dos estados principales:

Como se puede ver en la Figura 48, el **estado de configuración inicial** muestra un formulario para la configuración inicial que contiene selector del proveedor con la única opción de *Anthropic*, campo para la clave API, texto informativo sobre el almacenamiento y el botón "Guardar Clave API".

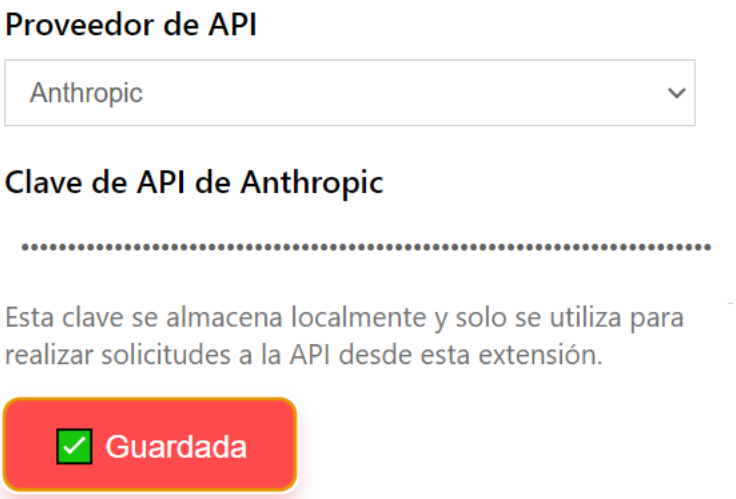
Figura 48. *Configuración inicial de la Clave API*



Fuente: Elaboración propia

El **estado de éxito,** que se muestra en la Figura *49*, se activa después de guardar la clave correctamente:

Figura 49. *Estado de éxito de API*



Fuente: Elaboración propia

6.5.3. Buddy-view-provider.js

El archivo *Buddy-view-provider.js* sirve como intermediario entre el *frontend* y el *backend*. Gestiona las solicitudes del usuario, los eventos generados en el entorno del IDE y formatea la salida para una visualización correcta. Por ejemplo, la función *queryAI* se encarga de enviar consultas al modelo y devolver las respuestas al usuario:

```
async queryAI(chatPrompt, assistantPrompt, abortController) {
    // Verifica si la conexión con Anthropic está establecida
    if (!this.anthropic) {
        await this.setUpConnection();
    }

    try {
        // Limpia y formatea los mensajes para asegurar que tengan el formato
correcto
        // Solo permite roles 'assistant' o 'user' y elimina espacios extras
        const cleanMessages = chatPrompt.map(msg => ({
            role: msg.role === 'assistant' ? 'assistant' : 'user',
            content: msg.content.trim()
        }));

        // Realiza la llamada a la API de Anthropic
        const response = await this.anthropic.messages.create({
            model: "claude-3-5-sonnet-20241022",  // Modelo a utilizar
            messages: cleanMessages,              // Mensajes formateados
```

```
            max_tokens: 1000,                    // Límite de tokens en la
respuesta
            temperature: 0.7,                    // Control de
creatividad/aleatoriedad
            system: cleanMessages[0].content.trim() // Usa el primer mensaje como
instrucción de sistema
        });

        // Devuelve el texto de la respuesta sin espacios adicionales
        return response.content[0].text.trim();
    } catch (error) {
        // Registra y propaga cualquier error que ocurra
        console.error("Error en queryAI:", error);
        throw error;
    }
}
```

El método *preparePrompt* se encarga de estructurar los mensajes que se enviarán al modelo:

```
async preparePrompt(queryType, problemText) {
    // Extrae el texto del problema, gestiona tanto strings como objetos
    const text = typeof problemText === 'string' ? problemText : problemText.text;

    // Obtiene el lenguaje de programación si existe
    const language = problemText.language;

    // Inicializa el array de mensajes con la instrucción de sistema
    let chatPrompt = [{
        "role": "system",
        "content": "Soy un asistente experto para estudiantes universitarios"
    }];

    // Variables para almacenar el mensaje principal y las instrucciones
específicas para el asistente
    let prompt = '';
    let assistantPrompt = '';
```

Dependiendo del tipo de consulta, construye diferentes formatos de *prompt*. El método *sendMessage* se utiliza para la comunicación con el *WebView*:

```
sendMessage(message) {
    // Registra en la consola el mensaje que se enviará para debugging
    console.log('Enviando mensaje a WebView:', message);

    // Verifica si existe una instancia de webView antes de intentar enviar el
mensaje
    if (this.webView) {
        // Envía el mensaje al WebView utilizando el método postMessage
        this.webView.webview.postMessage(message);
    }
}
```

Después de que la API responde, el código pasa por dos partes importantes:

Primero ***sendApiRequestWithCode*** se encarga de las solicitudes relacionadas con código seleccionado en el IDE:

```
async sendApiRequestWithCode(selectedText, queryType, abortController, overviewRef
= '', queryId = null, nlPrompt = '') {
    try {
        // Muestra un indicador de carga en la interfaz para informar al usuario
que la operación está en proceso
        this.sendMessage({ type: 'showLoader' });
```

```
        // Registra en la consola el inicio de la petición para facilitar el
debugging
        console.log('Iniciando petición API:', queryType);
```

Y luego está **processQueryResponse** que procesa la respuesta del modelo según el tipo de consulta recibida. Por ejemplo, para una consulta de conceptos:

```
async processQueryResponse(queryType, output, prompt, queryId, editorSelectedText)
{
    // Registro para debugging de la respuesta y el tipo de consulta
    console.log('Procesando respuesta:', queryType);
    console.log('Texto seleccionado en processQueryResponse:', editorSelectedText);

    // Variable para almacenar el HTML formateado
    let valueHtml = '';

    // Extrae el tipo de detalle eliminando el prefijo "askAI" y convirtiendo a
minúsculas
    // Por ejemplo: "askAIConcept" → "concept"
    const detailType = queryType.replace("askAI", "").toLowerCase();

    // Procesamiento específico para consultas de tipo concepto
    if (queryType === "askAIConcept") {
        // Array para almacenar los conceptos extraídos
        const concepts = [];

        // Expresión regular para extraer conceptos y explicaciones del formato
estructurado
        // Busca patrones como "CONCEPTO: [título] EXPLICACIÓN: [contenido]"
        const conceptRegex =
/CONCEPTO:\s*(.*?)\s*EXPLICACIÓN:\s*([\s\S]*?)(?=\s*CONCEPTO:|$)/g;
        let match;

        // Itera a través de todas las coincidencias en el output
        while ((match = conceptRegex.exec(output)) !== null) {
            // Agrega cada concepto encontrado al array, eliminando espacios extras
            concepts.push({
                title: match[1].trim(),      // El título del concepto
                content: match[2].trim()     // La explicación del concepto
            });
        }
    }
```

Finalmente, se envía la respuesta procesada al *frontend*:

```
// Envía la respuesta procesada al frontend para su visualización
    this.sendMessage({
        type: 'addDetail',
// Tipo de mensaje para el frontend
        value: output,
// Texto de salida completo
        queryId: queryType === "askAIQuery" ? this.overviewId : queryId,
// ID de la consulta o del resumen
        detailType: detailType,
// Tipo de detalle (concept, nextstep, etc.)
        valueHtml: valueHtml
// HTML formateado
    });
}
```

7. Evaluación del sistema

En esta etapa, se busca ofrecer el prototipo y observar su utilización para determinar si el asistente resulta efectivo para generar respuestas o sugerencias que ayuden a resolver los problemas planteados a los usuarios.

Se emplea el método de investigación cualitativo de la entrevista, definido como aquel que "busca información de primera mano acerca de las experiencias, opiniones, actitudes o percepciones del entrevistado" (Montero, 2015).

A través de estas entrevistas, se pretende identificar qué información valoran más los usuarios, lo que permitirá jerarquizar de manera más efectiva las funcionalidades del asistente. Además, se busca comprender con mayor profundidad las necesidades, expectativas y objetivos de los usuarios.

7.1. Participantes

El perfil objetivo de este trabajo son estudiantes universitarios en las primeras etapas de su aprendizaje de programación. Este grupo se selecciona debido a los desafíos particulares que enfrentan al interactuar con LLM mencionados en la introducción.

Para la captación de participantes en la prueba, se han identificado los cinco usuarios que se muestran en la **Figura 50** que cumplen con el perfil de los usuarios potenciales del producto. Aunque cinco participantes no constituyen una muestra representativa para determinar si el asistente es fácil de usar, sí se consideran suficientes para detectar posibles mejoras. Los cinco usuarios son estudiantes de universidades distintas, con poca experiencia en los lenguajes de programación evaluados:

Figura 50. *Participantes de las pruebas de usabilidad*

Fuente: Elaboración propia

7.2. Pruebas con usuarios

Para evaluar la usabilidad del asistente, se ha diseñado un guion que incluye **tareas** adaptadas al nivel de programación de los participantes. Durante esta prueba, se informa a los usuarios que realizarán una serie de tareas diseñadas para analizar la usabilidad del asistente. Se les proporcionan instrucciones claras sobre el uso de la herramienta y, posteriormente, se analiza su interacción con ella. El entrevistador mantiene una postura neutral en todo momento, evitando dirigir o influir en las respuestas de los entrevistados. A continuación, se presenta la plantilla entregada a los participantes junto con los resultados obtenidos en cada una de las pruebas.

Advertencias iniciales:

1. El estudio tendrá una duración aproximada de 1 hora.
2. No se evaluará tu rendimiento en las tareas de código.
3. Podrás estar presencial o virtualmente durante el experimento.

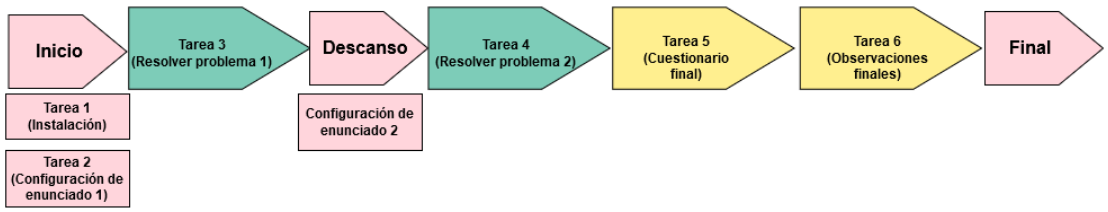
4. Este estudio está dirigido por el supervisor Daniel Prol Pérez como parte de su TFG.

El objetivo de esta prueba es comprobar el grado de usabilidad del asistente desarrollado. Este producto tiene como propósito guiar a los estudiantes para comprender mejor los problemas de programación. A continuación, se enuncian las tareas que deberás realizar. En caso de duda, no dudes en preguntar al supervisor.

Tareas a realizar:

- **Tarea 1 (10 minutos):** Instala el plugin desde el **Marketplace de Visual Studio Code.**
- **Tarea 2 (5 minutos):** Una vez instalado el asistente y la sesión iniciada, accede al campo de texto de los enunciados de problemas, introduce el enunciado del problema y selecciona el lenguaje de programación correspondiente.
- **Tarea 3 (15 minutos):** Intenta resolver el primer problema en 15 minutos. Puedes utilizar las funcionalidades del asistente que prefieras, pero está prohibido salir del entorno de desarrollo Visual Studio Code. Usa el protocolo *think-aloud* (piensa en voz alta).
- **Tarea 4 (15 minutos):** Intenta resolver el segundo problema en 15 minutos. Al igual que en la tarea anterior, utiliza las funcionalidades que desees sin salir de Visual Studio Code y piensa en voz alta.
- **Tarea 5 (5 minutos):** Completa **este cuestionario final** para recoger tus impresiones sobre el asistente.
- **Tarea 6 (10 minutos):** Participa en una conversación final de observaciones con el supervisor.

Figura 51. *Diseño del estudio*



Fuente: Elaboración propia con draw.io

Durante el estudio que se muestra en la figura anterior, el supervisor observará y anotará el comportamiento del usuario. Además, la sesión será grabada para analizar información cualitativa posteriormente. Se estudiarán el rendimiento en la prueba y las estrategias de aprendizaje. El protocolo *think-aloud* permite que los participantes expresen sus pensamientos y opiniones en tiempo real mientras completan las tareas (Jääskeläinen, 2010). Por otro lado, el cuestionario final se utiliza para recopilar datos subjetivos de forma cuantitativa.

7.3. Resultados de la evaluación

En esta sección se detallan los resultados relacionados con el uso de Buddy durante las pruebas de usabilidad. Específicamente, se tratan patrones o estrategias de uso de los estudiantes con Buddy para ver la frecuencia y elección de funcionalidades, la efectividad en cada tarea con la ayuda del asistente y las percepciones generales de los estudiantes del asistente. Se sintetizan resultados de estas fuentes: entrevistas con protocolo *think-aloud* con 5 estudiantes, interacciones con Buddy recogidas mediante la grabación y retroalimentación del cuestionario final. Para cada participante se rellenaron las siguientes tablas de resultados:

Tabla 7. *Formulario de resultados del primer usuario*

Participante nº 1
Pau González
Lenguaje
Python
Problemas / Éxito
1- Modificación de tuplas en listas / SÍ **2-** Inserción en listas con índices / SÍ
Comentarios del participante
• "El programa explica mejor que ChatGPT. Te explica los conceptos, le puedes pedir pistas, te puede explicar la solución y te plantea retos". • "Esto —explicación de preguntas de seguimiento— es de la forma en que nos enseñaban el año pasado (en clase) que teníamos que aprender". • "Las preguntas de seguimiento están muy bien, pero son un extra". • "Se entiende que, si te plantean un problema de estos, ya conoces los conceptos".

• "Con los conceptos y una pista inicial, ya puedes llegar a la solución sin el próximo paso".
Observaciones del entrevistador
• El usuario utiliza el pseudocódigo y lo encuentra útil. Mostró preferencia por entender el problema primero a través del pseudocódigo antes de empezar a programar. • Sugiere que las pistas deberían incluir la estructura inicial del programa. Al principio P1 no recordaba ni cómo empezar. No entiende de entrada la secuencia de las pistas y duda si son progresivas o si representan diferentes alternativas. • Se notó que pasaba rápidamente por algunos conceptos, lo que indica que ya los conocía. Sin embargo, reconoce que la funcionalidad de conceptos le puede servir para plantear el problema y que, si uno no se acuerda de nada, puede ser interesante. En su caso, en la encuesta la posiciona como la tercera funcionalidad más útil de seis, después del pseudocódigo y las pistas iniciales. • Cuando tuvo dudas sobre la sintaxis, acudió a las respuestas apiladas en la interfaz del asistente para resolverlas, por ejemplo, para escribir correctamente un bucle for con clave-valor. • Valora positivamente las explicaciones del asistente en comparación con ChatGPT. • Usa la sugerencia del próximo paso inicialmente, pero se da cuenta de que esa solución es más compleja y retrocede a su propia solución, tal como la había aprendido en clase. • Usa la funcionalidad de explicar solución después de resolver el problema y la valora positivamente, pero no aporta demasiado a lo que él había planteado previamente. Esta poca utilidad se refleja en la encuesta, donde la posiciona en el quinto lugar de seis en términos de utilidad. • Las preguntas de seguimiento las ve como un reto y le hacen pensar en cómo les enseñan en la universidad, ya que él había planteado un sort, pero el modelo confirma la lógica del profesor. • En la encuesta, P1 responde que el asistente le parece muy útil y que es capaz de completar con éxito todas las tareas. En cuanto a funcionalidades, considera que la más útil son los ejemplos en pseudocódigo y diagramas, mientras que la menos útil son las preguntas de seguimiento, que dice que están bien, pero son un extra. En cuanto a lo que podría mejorarse de la interfaz, destaca que el asistente podría ofrecer una estructura inicial. • Sugiere introducir una funcionalidad que proporcione una sugerencia al subrayar un error en el código.

Fuente: Elaboración propia

Tabla 8. *Formulario de resultados del segundo usuario*

Participante nº 2
Jordi Toscano
Lenguaje
C++

Problemas / Éxito
1- Funciones / SÍ **2-** Búsqueda de números en una secuencia / SÍ
Comentarios del participante
• "La opción de pseudocódigo está bien para entender un poco la estructura, lo que te puede ofrecer el programa o cómo debería enfocarlo. Me ha funcionado muchísimo. Cuando hacía Introducción a los Computadores en la universidad, siempre me ha parecido algo súper útil, especialmente cuando no tengo ni idea de cómo empezar el problema o si estoy muy atascado. Esto es... súper útil". • "Está bien que se apilen las respuestas. Esto de que aparezcan una encima de la otra, que vayas haciendo un seguimiento y, cuando haces una pregunta, la tengas abajo, hagas otra y se quede arriba. Que se forme una lista hacia arriba, me gusta mucho. Está muy bien hecho". • "Iría bien que, cuando pidas pistas, se muestren una por una en lugar de mostrar las tres a la vez. Y si quieres más, que haya un botón para generar una nueva pista o algo similar". • "La interfaz está bastante bonita. No la había visto hecha así en ningún sitio". • "La funcionalidad de conceptos es una pasada. Hay un montón de información. Para ser una versión prácticamente alfa, está bastante completa". • "Le pediría que me haga un roadmap". • "Cuando hago programación en la universidad, nos limitan mucho. Es decir, hay ejercicios en los que nos dicen que no podemos convertir un string o que no podemos acceder a la string como un array…". • "Muchas veces hay problemas que requieren conocimientos matemáticos. Por ejemplo, en este, tienes que saber cómo funciona la división de enteros en C++, el resto, el módulo…". • "Esta forma recursiva... Uy, a mí esto sí que no me gusta". • "El ícono se ve chafadito". • "No sé cómo se podría hacer para que se enfocara más en los conceptos que yo quiero. O sea, no que me explique cómo funciona una función y qué es una función, sino cómo aplicarla en este caso específico". • "Te da la solución y, de momento, todas las soluciones que he visto funcionaban. Cosa que, cuando le preguntas a la ChatGPT, no funciona tan bien". • "Las pistas iniciales están bastante bien. Te dan un punto de partida y una pequeña ayuda cuando estás muy atascado. Pero preferiría que no salgan todas a la vez, porque de alguna manera la pista inicial prácticamente resuelve el problema". • "Algo que me gusta en las pistas es que, cuando copias el código, todo se mantiene bien indentado. A veces copias algo de un chat o de una página y solo la primera línea queda bien, mientras que las demás quedan desajustadas. Pero aquí, cuando lo pegué, me sorprendió lo bien que se mantuvo la indentación".
Observaciones del entrevistador

- Las pistas le hacen cambiar de opinión, pero prefiere que sean progresivas y no que salgan todas a la vez. Las considera útiles cuando está atascado, pero no es la funcionalidad que más valora, al puntuarla 4 de 6 en la escala de utilidad, siendo 6 la menos útil.
- Le gustó que se apilen las respuestas en la interfaz.
- Sugiere que sería interesante no solo contemplar el enfoque de programación en conceptos, sino también el matemático. Dice que el problema del palíndromo es matemático y que, cuando haces un problema de ese tipo y no puedes usar vectores ni funciones, es importante tener una base matemática y saber cómo manipular los datos.
- Advierte que el asistente ofrece algunas pistas que pueden resultar demasiado complejas de entender para principiantes o que el profesor no permita utilizarlas por buenas prácticas. Aun así, le parecen pistas que tienen código “bonito”.
- Sugiere la mejora del icono del asistente porque “se ve chafado”.
- Sugiere que se puedan subrayar los conceptos de las respuestas que no se entiendan. En cuanto a los conceptos, considera que hay algunos que son muy básicos, como el de función.
- Sugiere que la funcionalidad de "próximo paso" tenga una mayor ventana de contexto para entender mejor el siguiente paso más óptimo en el programa.
- Sugiere una nueva funcionalidad que permita generar tests.
- En el segundo problema, una vez descubiertas las funcionalidades, empieza solicitando pseudocódigo.
- Una de las soluciones es en forma recursiva y al usuario le parece que es complicarse demasiado para el nivel principiante.
- En la encuesta, P2 responde que el asistente le parece muy útil (4 sobre 5, siendo 5 "muy útil") y que es capaz de completar con éxito todas las tareas. En cuanto a funcionalidades, considera que la más útil son los ejemplos en pseudocódigo y diagramas, mientras que la menos útil son los conceptos, pues cree que algunos son demasiado básicos y que, comparados con otras funcionalidades, son los menos valiosos. En cuanto a lo que podría mejorarse de la interfaz, destaca que en las pistas podría añadirse un botón para generar otra pista en lugar de mostrarse todas a la vez, que las flechas para pasar de conceptos no se corten, que el marco del siguiente paso es demasiado vistoso y que se mejore el icono. Además, sugiere que algunas de las opciones del menú de ayuda estén más visibles.
- Valora positivamente las explicaciones del asistente en comparación con ChatGPT. Al mismo tiempo, echa de menos un chat en la interfaz, como el que tiene el producto de OpenAI.
- Considera que las preguntas de seguimiento están bien cuando has acabado el problema, porque te hacen pensar y siempre se puede remarcar algo. Aun así, las considera la segunda menos útil (5 de 6 en términos de utilidad).

Fuente: Elaboración propia

Tabla 9. *Formulario de resultados del tercer usuario*

Participante nº 3
Marco Antonio Araujo
Lenguaje

C

Problemas / Éxito

1- Transformación de cadenas y tokenización / NO

Comentarios del participante

- "Lo primero que querría hacer es que me genere la estructura básica, que importe las librerías estándar y que me cree la función make porque no recuerdo cómo era. Yo lo que haría sería decirle: 'Dame la estructura inicial para ir empezando y, si empiezo a hacer el ejercicio, no pelearme con cosas propias del lenguaje C, sino pelearme con el problema'".
- "En pistas, lo que me gustaría es que explicase los conceptos claves. Por ejemplo, strtok: no sé qué hace. Sé que te lo invoca, pero no sé qué devuelve aquí".
- "Los conceptos claves me han gustado bastante porque se explican bien. Se supone que ya sé cómo usar las cadenas, pero no lo recuerdo bien. Y aquí me acaba de decir: 'Yo lo puedo definir así, con la notación del array estático, y tiene que llevar la barra cero'. Esto me gusta bastante".
- "Es bastante útil porque, al mirar el pseudocódigo, me di cuenta de algunas cosas".
- "A mí Buddy ya me ahorró ir a varias cosas de Google y, muchas veces, cuando estás frustrado con una solución, que te la ponga directamente aquí es muchísimo mejor que andar buscándola".
- "Es un entorno bastante bien guiado".
- "La lógica que usa para lo de '¿Es nueva palabra?' me parece un poco confusa; yo lo haría con otro *approach*".
- "El próximo paso me lo dijo muy bien".
- "Me gustó bastante que los conceptos me refrescaran; me ahorró bastante trabajo".
- "Las preguntas para resolver el ejercicio no me ayudaron porque son preguntas extra, pero sí plantean una cuestión bastante importante: cuando hacemos un problema, tenemos que tener en cuenta su extensibilidad futura. Ofrecen una visión más holística del problema".
- "Aquí (un caso de próximo paso) te dice include, pero no te dice la librería, cosa que echo en falta. Sin embargo, te lo explica perfectamente diciendo que falta la biblioteca más fundamental para las operaciones, que es necesaria para funciones como printf y scanf".

Observaciones del entrevistador

- El usuario destacó la utilidad de los conceptos para entender bien el problema. Por ejemplo, una respuesta que consideró muy buena fue la del paso por referencia de arrays: "A diferencia de otros lenguajes, C no tiene un tipo de dato string nativo, por lo que se manipulan como arrays de chars".
- Sugiere resaltar los ejemplos de los conceptos para que se aprecien mejor, ya que actualmente son difíciles de leer.
- Las pistas podrían incluir la estructura inicial del programa, ya que el usuario incluso salió del entorno para buscarla porque no recordaba cómo empezar.
- Sugiere que se puedan subrayar los conceptos de las pistas para ver qué hacen.

• Después de mirar el pseudocódigo, hace una serie de cambios que lo ayudan a avanzar en el problema, pero en un momento declara que no le gusta cómo está planteado y prefiere su propia solución. • P3 usa la opción de explicar la solución antes de terminar su programa, pero no le ayuda mucho porque el modelo plantea otra solución diferente. • En la encuesta, P3 responde que el asistente le parece muy útil (5 sobre 5, siendo 5 "muy útil"). Sin embargo, no es capaz de completar con éxito una de las tareas (la otra no pudo completarse por cuestiones de tiempo) porque fallaban los punteros. En cuanto a funcionalidades, considera que la más útil son los conceptos, porque le ayudaron a refrescar, mientras que la menos útil son las pistas iniciales, ya que no tenían nada que ver con el programa, por lo que fue la funcionalidad que más penalizó. En cuanto a lo que podría mejorarse de la interfaz, propone que las respuestas generadas por las distintas acciones se muestren en una pila de desplegables para disponer de un acceso más rápido al tenerlas a la vista. También sugiere que las flechas para cambiar entre elementos de una lista se coloquen debajo para que el ancho del recuadro no interfiera con su accesibilidad y que, en la sección "Próximo paso", se resalte el código de la misma forma que en "Pistas iniciales". • Las preguntas de seguimiento no le ayudaron en la resolución del problema, pero reconoce que plantean una cuestión bastante importante: la extensibilidad futura del código.

Fuente: Elaboración propia

Tabla 10. *Formulario de resultados del cuarto usuario*

Participante nº 4
Edu Nolla
Lenguaje
Java
Problemas / Éxito
1- Recorrido de arrays / SÍ **2-** Uso de estructuras de datos dinámicas (LinkedList) / SÍ
Comentarios del participante
• "Esto no lo había tenido en cuenta". • "No conocía el método system.printf". • "Lo ha hecho con clases, lo cual creo que sería un poco más complicado para alguien que está empezando, pero tampoco es mala idea". • "Me sorprende que no use el método Enhanced For o el bucle ForEach, que es mucho más eficiente, y que sí utilice tres clases". • "Quizá poner un icono con una papelera estaría mejor". • "Quizá en pistas podría añadirse un pequeño botón de copiar al portapapeles, como lo tiene Copilot".

- "Las pistas no son todas del mismo programa, sino que parecen distintas opciones, por lo que veo".
- "Me ha parecido bastante útil porque creo que tiene mucho potencial para ir explicando las cosas".
- "Me ha gustado esto de poder ver el próximo paso e incluso una pista, aunque hay veces que no puedo diferenciar muy bien la diferencia entre ambas".
- "La opción de 'próximo paso' me ha parecido muy útil, sobre todo porque puedo seleccionar el código específico para encontrar el error".
- "Las preguntas de seguimiento me han parecido útiles para plantearse nuevas soluciones o posibles alternativas".
- "Las pistas iniciales me han parecido excesivas. No me dejaban iniciar, sino que prácticamente ya me lo decían todo. También me gustaría profundizar un poco más. Si salieran de una en una, de alguna manera te harían pensar más".
- "En la solución, sí que profundizaría más, sobre todo para principiantes, en los métodos que hay dentro de cada linked list, por ejemplo".

Observaciones del entrevistador

- Cuando recibe por primera vez la sugerencia del próximo paso, se sorprende diciendo que no había considerado esa estrategia y se detiene a pensar si es mejor hacerlo como indica el asistente (por índices).
- Ante un error en el print, decide usar la opción de "próximo paso" y logra resolverlo gracias a la sugerencia. Se sorprende de que no conocía ese método.
- El usuario siente cierta sorpresa porque el modelo le muestra una solución que, según su planteamiento, es ineficiente.
- Sugiere un botón de limpieza con forma de papelera para evitar confusiones y que no se borren todas las respuestas sin querer.
- Sugiere añadir un botón de copiar código en las pistas.
- Ante el uso del método iterador de Java, tiene que salir del asistente a Google para buscarlo. Esto sugiere implementar una función que permita subrayar conceptos de las pistas que puedan ser desconocidos.
- En la encuesta, P4 responde que el asistente le parece muy útil (4 sobre 5, siendo 5 "muy útil") y que es capaz de completar con éxito las dos tareas propuestas. En cuanto a funcionalidades, considera que la más útil son los conceptos por su capacidad explicativa, mientras que la menos útil son las pistas iniciales, ya que desvelan demasiado la solución. En cuanto a lo que podría mejorarse de la interfaz, sugiere reemplazar el icono de refrescar por una papelera, añadir una opción para copiar directamente al portapapeles en las pistas, hacer que las pistas aparezcan de una en una y mejorar la visibilidad de la flecha en la caja de preguntas.
- Sugiere añadir más contexto a las soluciones, sin dar por hecho los conceptos y profundizando más.

Fuente: Elaboración propia

Tabla 11. *Formulario de resultados del quinto usuario*

Participante nº 5
Lucas Fodor
Lenguaje
Python
Problemas / Éxito
1- Manipulación de listas / SÍ **2-** Uso de diccionarios / SÍ
Comentarios del participante
• "Usé conceptos claves para ver cómo funcionaba". • "La solución te ayuda a ver otra forma de hacerlo". • "Podría ser una buena idea tener una funcionalidad que explique un poco más el enunciado". • "No me acordaba de la sintaxis correcta para obtener la key y el value del diccionario, así que miré el siguiente paso para que me lo explicara un poco". • "Explicar la solución creo que ayuda mucho cuando, después de hacer un problema, ves cuál es la manera óptima de resolverlo. Porque así recuerdas qué cosas no hiciste tan bien y qué podrías cambiar en el futuro". • "Las pistas iniciales, si estás perdido, te pueden ayudar bastante". • "Las preguntas de seguimiento son útiles para comprobar si realmente entiendes lo que acabas de hacer". • "Los conceptos están bien cuando empiezas, pero después de una o dos semanas, cuando ya estás familiarizado, pueden no ser tan útiles". • "En general, creo que la UI está bastante bien". • "Buddy es muy útil porque te hace pensar. Ahora casi todos los asistentes, cuando les preguntas algo, te dan la solución directamente. Y después ya no quieres aprendértelo porque lo tienes ahí".
Observaciones del entrevistador
• Usa conceptos claves para comprender el problema, pero los pasa rápidamente. • Ante un error en el print dentro de la función, el usuario decide usar la opción "siguiente paso" y consigue resolver el problema con la sugerencia. • Sugiere que las pistas iniciales salgan de una en una. • La UI le parece bien, pero sugiere que las respuestas no se apilen, sino que se muestren una por una.

- Valora el hecho de que el asistente no revele la solución directamente y que fomente el pensamiento crítico, en comparación con otros asistentes como ChatGPT, que no apoyan directamente el aprendizaje porque simplemente dan la solución.
- En la encuesta, P5 responde que el asistente le parece muy útil (4 sobre 5, siendo 5 "muy útil") y que es capaz de completar con éxito las dos tareas propuestas. En cuanto a funcionalidades, considera que la más útil es el pseudocódigo, mientras que la menos útil son los conceptos. En cuanto a lo que podría mejorarse de la interfaz, sugiere dos cambios: mejorar las flechas para cambiar entre tarjetas y evitar que se acumulen las respuestas.

Fuente: Elaboración propia

Gracias a los resultados de estas pruebas de usabilidad, será posible planificar una futura versión del asistente que corrija los defectos e imperfecciones identificados por los usuarios evaluados. Es importante destacar que, a pesar de las observaciones realizadas por algunos usuarios, todos pudieron completar las tareas propuestas sin necesidad de ayuda externa. Cuando surgían dudas, la asistencia proporcionada por el propio asistente les permitía identificar y corregir sus errores, lo que demuestra que el asistente cumple con los requisitos mínimos de usabilidad y está listo para su puesta en producción.

8. Conclusiones y trabajo futuro

En esta sección se presentan las conclusiones obtenidas tras la realización del Trabajo Fin de Grado, junto con posibles líneas de trabajo futuras para la evolución del asistente.

8.1. Conclusiones del trabajo

Como conclusión del trabajo, y partiendo del objetivo principal del proyecto definido en el apartado de **objetivo general**, que es construir un asistente que guíe a los estudiantes para una comprensión profunda de los problemas, se considera que dicho objetivo se ha cumplido totalmente. Esto puede observarse en el vídeo funcional vinculado en la portada del trabajo y en la retroalimentación recogida durante las **pruebas**.

Se detallan a continuación las conclusiones obtenidas a nivel de los **objetivos específicos**.

Respecto al **análisis del contexto y estado del arte de los asistentes** se han descubierto nuevas funcionalidades que se pueden incorporar en la solución que se ha desarrollado para que los estudiantes tengan más componentes para resolver los problemas de programación y cubrir así las sugerencias propuestas durante las pruebas, siendo el resultado satisfactorio al descubrir el posible encaje en el mercado de los asistentes que tiene la propuesta del asistente detallada en este trabajo.

Respecto al **desarrollo de las especificaciones del asistente**, se realiza la especificación del asistente desde el punto de vista del estudiante, lo que ha permitido dar un enfoque al asistente a nivel funcional para poder obtener definiciones de conceptos básicos, generar ejemplos en pseudocódigo y diagramas de flujo, ofrecer al usuario pistas iniciales, obtener sugerencias para el próximo paso, proporcionar explicaciones detalladas de las soluciones de código y generar preguntas relacionadas con el problema para profundizar y ahondar en el mismo.

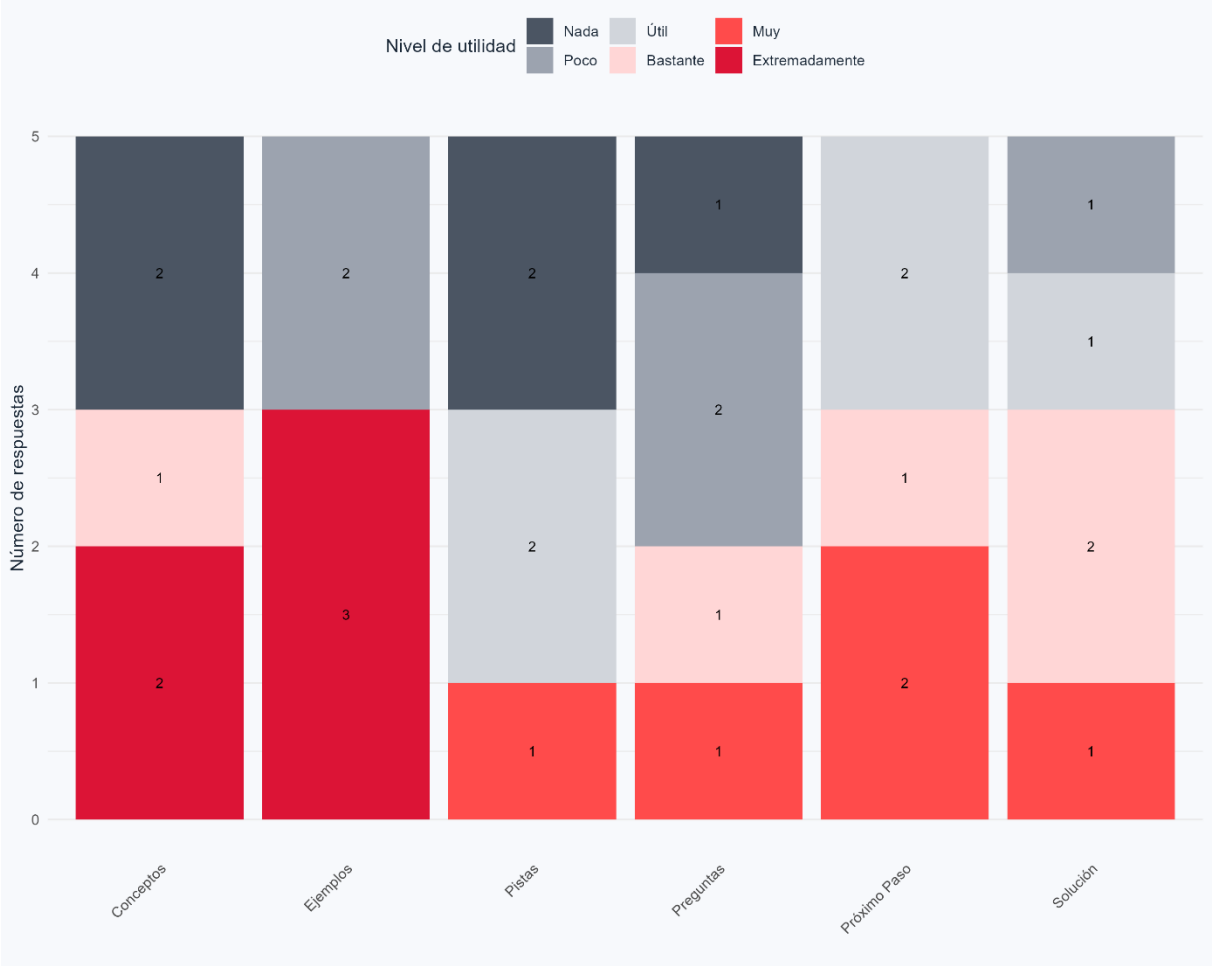
Partiendo de los **casos de uso y de la metodología**, se realiza el **diseño y la implementación del asistente** donde se define la arquitectura y los *wireframes*. Gracias a esta fase del trabajo, se han podido asentar las bases del proyecto a nivel tecnológico (tanto el *frontend* como el *backend*) en función de los casos de uso y requisitos no funcionales recogidos. Se han cubierto todas las funcionalidades definidas para la primera versión, lo que hace concluir que este objetivo se ha cubierto completamente.

Respecto a la **evaluación del asistente**, los participantes hicieron una valoración positiva del enfoque pedagógico del asistente y su capacidad para ofrecer explicaciones adaptadas a cada uno. Como ilustra P5: "Buddy es muy útil porque te hace pensar. Ahora casi todos los asistentes, cuando les preguntas algo, te dan la solución directamente. Y después ya no quieres aprendértelo porque lo tienes ahí". Se menciona en varias ocasiones el valor diferencial en comparación con ChatGPT con, por ejemplo, P1 señalando que "el programa explica mejor que ChatGPT. Te explica los conceptos, le puedes pedir pistas, te puede explicar la solución y te plantea retos". En cuanto a los patrones de uso en estas pruebas, se observaron diferentes estrategias. P2 menciona el uso estratégico que hace del pseudocódigo: "La opción de pseudocódigo está bien para entender un poco la estructura, lo que te puede ofrecer el programa o cómo debería enfocarlo. Me ha funcionado muchísimo". Al mismo tiempo, P3 también destaca la utilidad de los conceptos para entender bien el problema antes de empezar a programar. El asistente fue efectivo tanto para el aprendizaje de nuevos conceptos como para reforzar conocimientos existentes, como ilustra la experiencia de P4 al descubrir y aplicar exitosamente el método system.printf, y P3 al señalar que "A mí Buddy ya me ahorró ir a varias cosas de Google y, muchas veces, cuando estás frustrado con una solución, que te la ponga directamente aquí es muchísimo mejor que andar buscándola". Estas observaciones sugieren que el asistente logra un balance efectivo entre proporcionar ayuda inmediata y facilitar el aprendizaje activo, aunque las sugerencias de cambios, como la implementación gradual de pistas, indican margen de mejora.

Otro de los objetivos de la evaluación de este trabajo era acompañar a los estudiantes en el proceso de resolución de problemas de manera satisfactoria. Dicho proceso ha sido satisfactorio según las pruebas de usabilidad, con una media de 4.6/5 en la tasa de éxito de las tareas. El resultado es un asistente que, aunque mejorable en cuanto a interfaz de usuario, permite a los usuarios interactuar fácilmente y resolver los problemas de manera satisfactoria (3 de 5 usuarios lo valoraron como "muy útil" con un 5 sobre 5 y 2 de 5 usuarios le dieron una puntuación de 4 sobre 5).

La siguiente figura muestra la percepción de utilidad de las distintas funcionalidades según lo reportado por los participantes:

Figura 52. *Utilidad de las funcionalidades para los participantes*



Fuente: Elaboración propia

Los resultados muestran una valoración variada de las diferentes funcionalidades del asistente. Los **ejemplos (pseudocódigo/diagramas de flujo)** fueron considerados los más útiles, con 3 participantes que los calificaron como "Extremadamente" útiles. Esto confirma el enfoque de Renske & Sjaak que demuestran cómo el pseudocódigo y los diagramas de flujo son especialmente útiles para organizar las ideas antes de ponerse a programar y producen una mejora en el diseño de algoritmos y las habilidades de programación (Renske & Sjaak, 2017). Los **conceptos** presentaron una distribución polarizada, con 2 participantes que los valoraron como "Extremadamente" útiles y otros 2 como "Nada" útiles. La funcionalidad de **próximo paso** mostró una distribución más uniforme, siendo valorada principalmente como "Muy útil", “Bastante útil” y "Útil". **La explicación de la solución y las preguntas de seguimiento** recibieron valoraciones mixtas, mientras que las **Pistas iniciales** tendieron a ser

consideradas menos útiles, con 4 participantes que las calificaron como "Nada" o "Poco" útiles.

La evaluación del sistema también ha servido para ver que varios de los requisitos no funcionales se consideran cubiertos. El RNF003 (usabilidad) se demuestra en los comentarios positivos sobre la interfaz, con P2 señalando que "la interfaz está bastante bonita" y que "no la había visto hecha así en ningún sitio". Al mismo tiempo, el RNF006 (accesibilidad) tiene margen de mejora, ya que varios usuarios sugirieron modificaciones en la visibilidad de las flechas de navegación. El RNF007 (contextualización) se cumple parcialmente, como demuestra P2 cuando menciona que "cuando hago programación en la universidad, nos limitan mucho [...] hay ejercicios en los que nos dicen que no podemos convertir un string", y algunos usuarios como P4 notaron que ciertas soluciones eran demasiado complejas para principiantes. El RNF008 (portabilidad del código) se cumple exitosamente, como confirma P2: "cuando copias el código, todo se mantiene bien indentado [...] me sorprendió lo bien que se mantuvo la indentación". La evaluación también revela que los usuarios encuentran el asistente muy útil, con calificaciones de 4 o 5 sobre 5 en todos los casos, lo que sugiere un buen cumplimiento general de los requisitos de calidad establecidos.

Finalmente, se considera cubierto el último punto de la **documentación** al crear una **guía de usuario** y **desplegar públicamente** en el *marketplace* de Visual Studio Code para que el asistente esté accesible para todos los públicos.

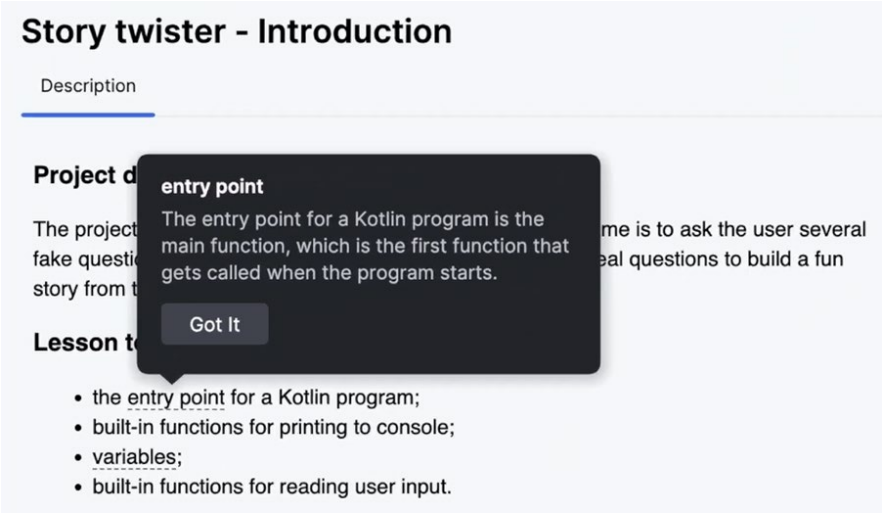
8.2. Líneas de trabajo futuro

El asistente desarrollado en el presente Trabajo Fin de Grado corresponde a una primera versión (1.0). Aunque es plenamente funcional y cumple con los objetivos planteados, es posible identificar áreas de mejora para futuras actualizaciones a partir de las pruebas de usabilidad realizadas. A continuación, se detallan los puntos a mejorar, acompañados de aspectos a considerar en próximas revisiones.

8.2.1. Implementar Theory Lookup

Una de las sugerencias más frecuentes durante las pruebas con usuarios fue la posibilidad de subrayar errores en el código o conceptos de las respuestas que no estén claros directamente en el IDE y obtener sugerencias. El proyecto de JetBrains (**Figura *53***) hace precisamente esto. Esta aproximación permitiría mejorar Buddy para que no solo genere sugerencias basadas en el problema, sino que permita explicaciones más precisas y contextuales.

Figura 53. *Theory Lookup*



Fuente: Página Web https://lp.jetbrains.com/research/education/

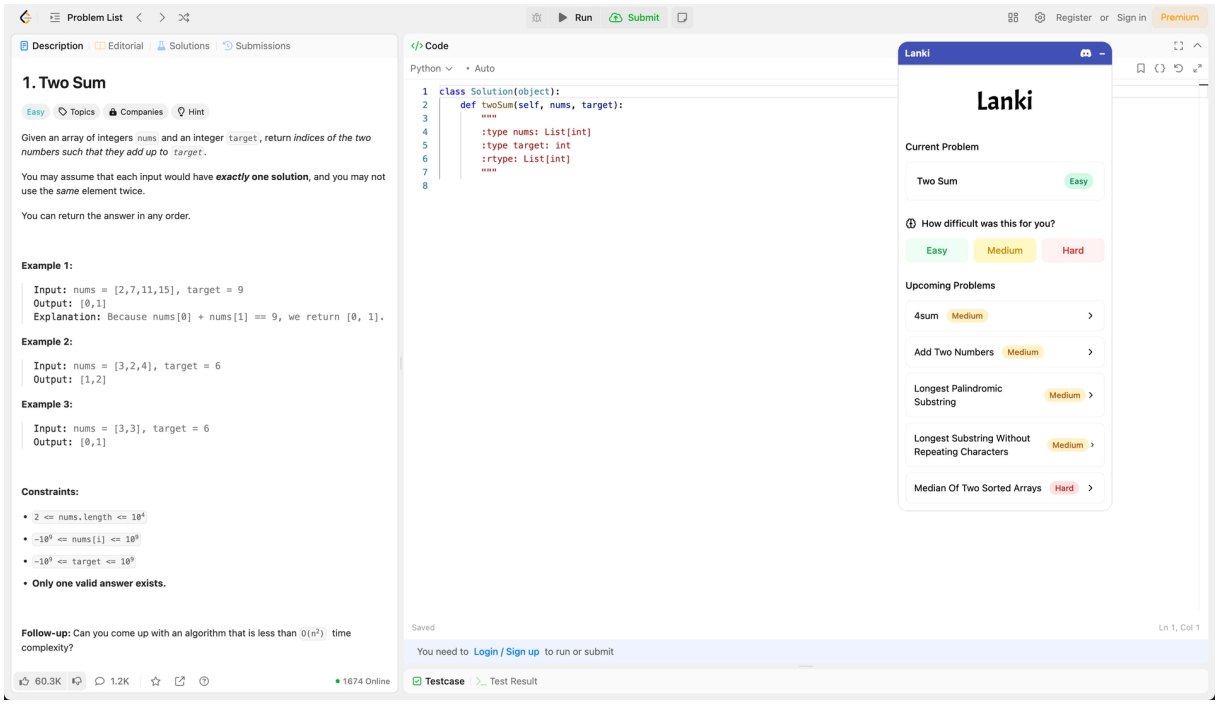
8.2.2. Mecanismo de retroalimentación

Se sabe desde hace tiempo que recibir retroalimentación detallado rápido ayuda al aprendizaje (Sadler, 1989). Ahora mismo no hay forma evaluar las respuestas del asistente. Se podría incluir una función para que el estudiante evalúe al instante la calidad de la respuesta

ya que esta puede simplemente no ayudar. Algunos comentarios de las pruebas sugieren que las respuestas del asistente pueden ser demasiado complejas para principiantes (como señaló P2 con la solución recursiva), no siempre se ajustan al nivel de conocimiento deseado (como cuando P4 notó que la solución usaba tres clases cuando podría ser más simple), o pueden plantear enfoques diferentes a los que el estudiante está aprendiendo en clase (como cuando P1 y P3 prefirieron retroceder a sus propias soluciones tras ver las sugerencias del asistente). Además, algunos participantes indicaron que ciertas explicaciones daban por sentado conceptos que podrían requerir más profundización (como mencionó P4 respecto a los métodos de *LinkedList*).

La idea es implementar una funcionalidad como tiene *Lanki*[31], una extensión de Chrome que se puede ver en la **Figura 54** y que permite marcar los problemas de Leetcode como fácil, medio o difícil al resolverlos y va recomendando problemas a resolver como siguiente paso. En el caso de Buddy se podría hacer una pregunta de ¿Encontraste lo que necesitabas? Y dos botones de Sí (*thumbs up*) y No (*thumbs down*).

Figura 54. *Feedback en Lanki*



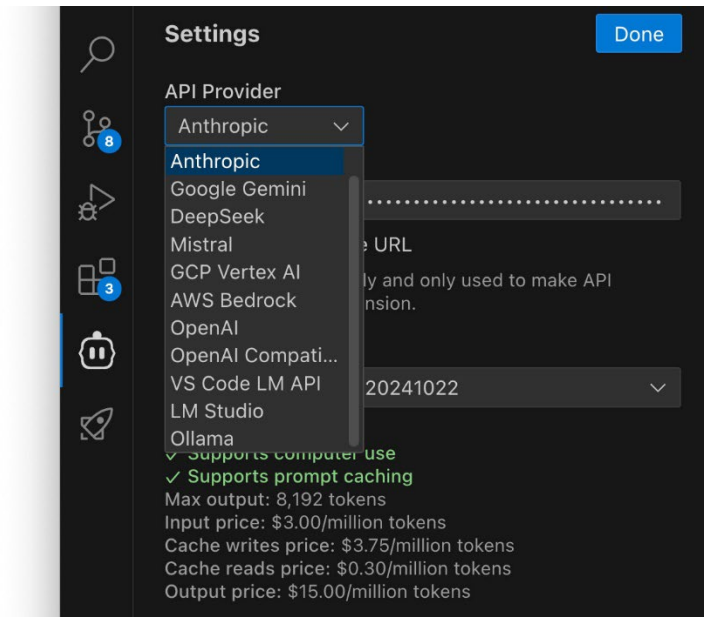
Fuente: Página Web https://www.lanki.xyz

[31] Lanki: https://www.lanki.xyz/

8.2.3. Flexibilidad del modelo

Ahora mismo Buddy está limitado a un proveedor específico (Anthropic), pero la flexibilidad de modelos como el caso de Cline[32], como se aprecia en la **Figura 55**, permitiría diferentes flujos de trabajo que aprovechen las fortalezas de diferentes modelos de IA. En un futuro se podría soportar, como se puede ver en la figura, una gama completa de modelos, incluyendo los de OpenAI, Google Gemini, DeepSeek y modelos locales a través de LM Studio/Ollama.

Figura 55. *Flexibilidad para elegir modelo*



Fuente: Página Web https://addyo.substack.com/p/why-i-use-cline-for-ai-engineering

[32] Cline: https://cline.bot/

Referencias bibliográficas

2024 Stack Overflow Developer Survey. (s. f.). Recuperado 8 de diciembre de 2024, de https://survey.stackoverflow.co/2024/

AI Copilots Are Changing How Coding Is Taught—IEEE Spectrum. (s. f.). Recuperado 28 de noviembre de 2024, de https://spectrum.ieee.org/ai-coding

Alpizar-Chacon, I. (2024). La Inteligencia Artificial Generativa y cómo podemos abordarla en nuestra enseñanza. *Investiga.TEC*, *17*(51), Article 51. https://doi.org/10.18845/itec.v17i51.7536

Birillo, A., Artser, E., Potriasaeva, A., Vlasov, I., Dzialets, K., Golubev, Y., Gerasimov, I., Keuning, H., & Bryksin, T. (2024). *One Step at a Time: Combining LLMs and Static Analysis to Generate Next-Step Hints for Programming Tasks* (No. arXiv:2410.09268). arXiv. https://doi.org/10.48550/arXiv.2410.09268

Burtch, G., Lee, D., & Chen, Z. (2024). The consequences of generative AI for online knowledge communities. *Scientific Reports*, *14*(1), 10413. https://doi.org/10.1038/s41598-024-61221-0

ChatGPT y educación universitaria | Editorial Octaedro. (s. f.). Recuperado 13 de noviembre de 2024, de https://octaedro.com/libro/chatgpt-y-educacion-universitaria/

Chatterjee, S., Liu, C. L., Rowland, G., & Hogarth, T. (2024). *The Impact of AI Tool on Engineering at ANZ Bank An Empirical Study on GitHub Copilot within Corporate Environment* (No. arXiv:2402.05636). arXiv. https://doi.org/10.48550/arXiv.2402.05636

Cody | AI coding assistant. (s. f.). Sourcegraph. Recuperado 12 de noviembre de 2024, de https://sourcegraph.com/cody

Co-Intelligence: Living and Working with AI. (s. f.). Recuperado 7 de diciembre de 2024, de https://www.goodreads.com/book/show/198678736-co-intelligence

Continue. (s. f.-a). Recuperado 25 de noviembre de 2024, de https://www.continue.dev/

Continue: The leading open-source AI code assistant. (s. f.-b). Y Combinator. Recuperado 10 de diciembre de 2024, de https://www.ycombinator.com/companies/continue

Design Thinking. (2024). *ResearchGate*. https://www.researchgate.net/publication/5248069_Design_Thinking

Dodds, K. C. (s. f.-a). *AI - «Iron Man or Ultron»*. Epic Web Dev. Recuperado 6 de diciembre de 2024, de https://www.epicweb.dev/talks/ai-iron-man-or-ultron

Dodds, K. C. (s. f.-b). *AI Assistants Tutorial*. Epic Web Dev. Recuperado 12 de noviembre de 2024, de https://www.epicweb.dev/tutorials/ai-assistants

Enhancing The Abilities Of Software Engineers With Generative AI At Tabnine. (s. f.). Data Engineering Podcast. Recuperado 25 de noviembre de 2024, de https://www.dataengineeringpodcast.com/episodepage/tabnine-generative-ai-developer-assistant-episode-400

Experiencia de Usuario: Principios y Métodos by Yusef Hassan Montero | Goodreads. (s. f.). Recuperado 9 de diciembre de 2024, de https://www.goodreads.com/book/show/24944791-experiencia-de-usuario

freeCodeCamp.org (Director). (2024, noviembre 15). *To code is to struggle! I interview Tech with Tim [Podcast #150]* [Video recording]. https://www.youtube.com/watch?v=mgWOMu2yo_U

Gates, B. (s. f.). *Sal Khan is pioneering innovation in education…again*. gatesnotes.com. Recuperado 12 de noviembre de 2024, de https://www.gatesnotes.com/Brave-New-Words

¡He probado GPT-4o! Para Programar Backend desde Cero—YouTube. (s. f.). Recuperado 17 de noviembre de 2024, de https://www.youtube.com/watch?v=N1-lsH8IXwg

How the Python Tutor visualizer can help students in your C or C++ courses—Python Tutor. (s. f.). Recuperado 6 de diciembre de 2024, de https://pythontutor.com/articles/c-cpp-visualizer.html

How the Python Tutor visualizer can help students in your Java programming courses—Python Tutor. (s. f.). Recuperado 6 de diciembre de 2024, de https://pythontutor.com/articles/java-visualizer.html

Impact Research. (2024). *AI Chatbots in Schools: Findings from a Poll of K-12 Teachers, Students, Parents, and College Undergraduates*. https://youthtoday.org/2024/06/ai-chatbots-in-schools-findings-from-a-poll/

Incorporating LLM-based Interactive Learning Environments in CS Education: Learning Data Structures and Algorithms using the Gurukul platform. (s. f.). Recuperado 12 de noviembre de 2024, de https://vtechworks.lib.vt.edu/items/3d08a8cd-effe-4e41-9830-0204637e53da

Introducing the Realtime API. (s. f.). Recuperado 8 de diciembre de 2024, de https://openai.com/index/introducing-the-realtime-api/

Jaipuria, T. (2024, septiembre 30). *Three Archetypes of AI Application Startups*. https://www.tanayj.com/p/three-archetypes-of-ai-application

Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & Grossman, T. (2023). Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 1-23. https://doi.org/10.1145/3544548.3580919

Kazemitabaar, M., Ye, R., Wang, X., Henley, A. Z., Denny, P., Craig, M., & Grossman, T. (2024). CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 1-20. https://doi.org/10.1145/3613904.3642773

Khemka, M., & Houck, B. (2024). *Towards Effective AI Support for Developers: A Survey of Desires and Concerns*. *22*(3). https://www.microsoft.com/en-us/research/publication/towards-effective-ai-support-for-developers-a-survey-of-desires-and-concerns/

Lanki. (s. f.). Recuperado 6 de diciembre de 2024, de https://www.lanki.xyz/

Lau, S., & Guo, P. (2023). From «Ban It Till We Understand It» to «Resistance is Futile»: How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, *1*, 106-121. https://doi.org/10.1145/3568813.3600138

Leading in the era of AI code intelligence with Quinn Slack, Co-founder & CEO of Sourcegraph (Changelog Interviews #580). (2024, febrero 28). Changelog. https://changelog.com/podcast/580

Learn AI-Assisted Python Programming. (s. f.). Recuperado 30 de noviembre de 2024, de https://www.manning.com/books/learn-ai-assisted-python-programming

Liu, J. (2016, febrero 4). Visualizing the 4 Essentials of Design Thinking. *Good Design*. https://medium.com/good-design/visualizing-the-4-essentials-of-design-thinking-17fe5c191c22

Mapp Digital – R. Kiran Khan. (s. f.). Recuperado 6 de enero de 2025, de https://rkirankhan.com/work/mapp/

Nam, D., Macvean, A., Hellendoorn, V., Vasilescu, B., & Myers, B. (2024). Using an LLM to Help With Code Understanding. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 1-13. https://doi.org/10.1145/3597503.3639187

Nico Orellana (Director). (2024, mayo 25). *Gen-z, la generación perdida | Freddy Vega #E56* [Video recording]. https://www.youtube.com/watch?v=MTnG_c-PYeM

Orosz, G. (2024, enero 30). *Measuring Developer Productivity: Real-World Examples*. https://newsletter.pragmaticengineer.com/p/measuring-developer-productivity-bae

Paradis, E., Grey, K., Madison, Q., Nam, D., Macvean, A., Meimand, V., Zhang, N., Ferrari-Church, B., & Chandra, S. (2024). *How much does AI impact development speed? An enterprise-based randomized controlled trial* (No. arXiv:2410.12944). arXiv. https://doi.org/10.48550/arXiv.2410.12944

Peixoto, A. R., Glória, A., Silva, J. L., Pinto-Albuquerque, M., Brandão, T., & Nunes, L. (2024). Use of Programming Aids in Undergraduate Courses. *5th International Computer Programming Education Conference (ICPEC 2024)*, 20:1-20:9. https://doi.org/10.4230/OASIcs.ICPEC.2024.20

Pelayo. (2023, marzo 2). *Los dilemas (éticos) del ChatGPT*. Ethic. https://ethic.es/2023/03/los-dilemas-eticos-del-chatgpt/

Python Tutor—Python Online Compiler with Visual AI Help. (s. f.). Recuperado 25 de noviembre de 2024, de https://pythontutor.com/

Sadler, D. R. (1989). Formative assessment and the design of instructional systems. *Instructional Science*, *18*(2), 119-144. https://doi.org/10.1007/BF00117714

SEDaily. (2024, enero 16). *JetBrains AI with Jodie Burchell*. Software Engineering Daily. https://softwareengineeringdaily.com/2024/01/16/jetbrains-ai-with-jodie-burchell/

Team, T. (2024, febrero 28). *Code refactoring principles, techniques, and automation with generative AI*. Tabnine. https://www.tabnine.com/blog/code-refactoring-with-generative-ai/

Tech With Tim (Director). (2024, marzo 10). *Can ChatGPT Actually Teach You How To Code?* [Video recording]. https://www.youtube.com/watch?v=lSyGTNttrME

The SPACE of Developer Productivity: There's more to it than you think.: Queue: Vol 19, No 1. (s. f.). Recuperado 6 de diciembre de 2024, de https://dl.acm.org/doi/10.1145/3454122.3454124

Uvapl/terra. (2024). [JavaScript]. UvA Programming Lab. https://github.com/uvapl/terra (Obra original publicada en 2024)

Vahid, F. (2024). AI in CS Education: Opportunities, Challenges, and Pitfalls to Avoid. *ACM Inroads*, *15*(3), 52-57. https://doi.org/10.1145/3679205

van den Berg, M., Heeren, B., & Rahimi, E. (2024). Parsons Problems for Equivalence Proofs in Logic. *Proceedings of the 24th Koli Calling International Conference on Computing Education Research*, 1-12. https://doi.org/10.1145/3699538.3699551

Webinar: Navigating AI tools for software engineering 092024. (s. f.). Recuperado 12 de diciembre de 2024, de https://thoughtworks.wistia.com/medias/f2ziwflq9v?wtime=0

What Is Insight? The 5 Principles of Insight Definition. (2023, enero 6). Thrive. https://thrivethinking.com/2023/01/06/what-is-insight-the-5-principles-of-insight-definition/

Problem Solving and Algorithmic Development with Flowcharts | Proceedings of the 12th Workshop on Primary and Secondary Computing Education. (s. f.). Recuperado 27 de febrero de 2025, de https://dl.acm.org/doi/10.1145/3137065.3137080

Anexo A. Tareas de código

Tabla 12. *Problemas de Python*

Problema 1
Dada la siguiente tupla. elements = ([-3, -2, -1, 0], [1, 2, 4, 5], [6, 7, 8, 9, 10]) Desarrolla un programa que nos permita añadir el número tres (3) al segundo elemento en la tupla. El número 3 deberá encontrarse en la posición 2, de tal forma que pueda generar una secuencia coherente con los demás elementos.
Problema 2
Define una función la cual nos permita conocer todas las llaves dentro de un diccionario. La función debe cumplir con los siguientes requerimientos: La función debe tener por nombre sequences_items. La función debe recibir como argumento un diccionario. La función deberá imprimir en consola cada una de la llaves y sus valores dentro del diccionario. Diccionario: "nombre": "Lucas", "edad": 19, "ciudad": "Barcelona", "universidad": "ESADE"

Fuente: Elaboración propia

Tabla 13. *Problemas de C++*

Problema 1
Escribe un programa que lea una secuencia de números naturales e imprima la posición del primer número par encontrado.
Problema 2
Escribe una función que determine si un número natural n es un número palíndromo o no.

Fuente: Elaboración propia

Tabla 14. *Problemas de C*

Problema 1
Escribir un programa con una función que tome una cadena de caracteres como parámetro, que ponga las iniciales de cada palabra en mayúsculas, y devuelva el número de palabras encontradas. Después de invocar la función desde main() e imprimir los resultados.
Problema 2
Escribir un programa en C que lea una palabra y que nos diga cuántas vocales de cada tipo tiene. Emplear un bucle while para ir recorriendo los caracteres de la palabra hasta llegar al carácter de fin de cadena, y una estrutura tipo switch para ir incrementando los contadores de cada tipo de vocal.

Fuente: Elaboración propia

Tabla 15. *Problemas de Java*

Problema 1
Un grupo de amigos de La Salle decidió correr el Maratón de Boston. Se tiene una lista con sus nombres y sus tiempos (en minutos). Escribe un programa que: 1. Encuentre al corredor más rápido. 2. Imprima el nombre del corredor y su tiempo (en minutos).
Problema 2
Crea un programa que permita almacenar títulos de canciones en una lista enlazada y las muestre en pantalla. Canciones: "Bohemian Rhapsody" "Yesterday" "Hotel California"

Fuente: Elaboración propia

Índice de acrónimos

API: Application Programming Interface

CED: Computing Education

CSS: Cascading Style Sheets

DOM: Document Object Model

ERS: Especificación de requisitos de software

HTML: HyperText Markup Language

IDE: Integrated Development Environment

LLMs: Large Language Models

NPM: Node Package Manager

NSF: National Science Foundation

RLHF: Reinforcement Learning from Human Feedback

UI: User Interface

UML: Unified Modeling Language