

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“DESARROLLO DE UN MOTOR DE GENERACIÓN
AUTOMÁTICA DE RÚBRICAS Y RETROALIMENTACIÓN
PARA LA EVALUACIÓN DE EJERCICIOS DE
PROGRAMACIÓN EN CURSOS INTRODUCTORIOS”

DIEGO IGNACIO BAHAMONDES MORRIS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Federico Meza
Profesora Correferente: Andrea Vásquez

Julio - 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Desarrollo de un motor de generación automática de rúbricas y retroalimentación para la evaluación de ejercicios de programación en cursos introductorios.

Nombre del candidato(a): Diego Bahamondes Morris

Carrera / Grado: Ingeniero civil en informática

Campus: Casa Central **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Federico Meza, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

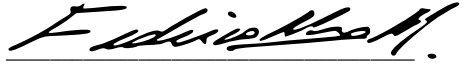
☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 10 de julio de 2026 Firma: 

Estudiante o Candidato(a):

Fecha: 10 de julio de 2026 Firma: 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

A mi pareja, familia y amigos, por apoyarme en cada etapa de este proceso y formar parte fundamental de mi vida.

AGRADECIMIENTOS

Quiero expresar mi agradecimiento hacia mi pareja, familia y amigos por el apoyo entregado a lo largo de mi carrera universitaria. Su compañía y constante motivación fueron fundamentales para terminar con éxito este proceso.

De manera muy especial, agradezco a mi pareja por estar siempre presente, escuchándome y sosteniéndome en los momentos de mayor estrés, siendo un pilar indispensable en mi día a día. Asimismo, dedico un agradecimiento profundo a mi mamá, quien hizo todo lo posible por facilitarme las cosas para que pudiera enfocarme por completo en mis estudios.

A este agradecimiento sumo a mis amigos de la universidad, con quienes compartí muy buenos momentos que hicieron mucho más grato el paso por la universidad.

Finalmente, quiero agradecer al profesor Federico, quien me ayudó y facilitó las posibilidades de evaluar esta propuesta, otorgándome el acceso a los materiales y canales necesarios para el desarrollo de este trabajo.

RESUMEN

Resumen— Esta memoria presenta un motor de software basado en LLMs orientado a optimizar los procesos evaluativos de la asignatura Introducción a la Programación de la UTFSM. La solución articula una arquitectura RAG conectada al registro histórico del curso, *prompts* especializados por tarea y un entorno de ejecución, permitiendo generar rúbricas institucionales, evaluar entregas y construir retroalimentación personalizada combinando análisis semántico con evidencia de ejecución. La validación en el semestre 2026-1 demostró alta eficiencia con un tiempo promedio de procesamiento de 22,56 segundos. El cuerpo docente destacó la coherencia de las rúbricas generadas, mientras que el estudiantado valoró positivamente la claridad y enfoque socrático de la retroalimentación, consolidando la viabilidad operativa de la herramienta para mitigar la sobrecarga administrativa sin sustituir el criterio experto del evaluador.

Palabras Clave— Generación de rúbricas; Evaluación automatizada; Retroalimentación formativa; Modelos de lenguaje; Educación en programación.

ABSTRACT

Abstract— This thesis presents an LLM-based software engine oriented to optimize the assessment processes of UTFSM's Introduction to Programming course. The solution coordinates a RAG architecture connected to historical course records, specialized prompts per task, and an execution sandbox, generating institutional rubrics, evaluating submissions, and constructing personalized feedback by combining semantic analysis with execution evidence. Validation during the 2026-1 semester demonstrated high efficiency, averaging a processing time of 22.56 seconds. The faculty highlighted the consistency of the generated rubrics, while students positively valued the clarity and Socratic approach of the feedback, confirming the tool's operational viability to mitigate administrative workload without replacing the evaluator's expert judgment.

Keywords— Rubric generation; Automated assessment; Formative feedback; Large language models; Programming education.

GLOSARIO

AlaaS (Artificial Intelligence as a Service): Inteligencia Artificial como Servicio, modelo de consumo de IA mediante proveedores externos.

API (Application Programming Interface): Interfaz de Programación de Aplicaciones, protocolo que permite la comunicación e intercambio de datos entre sistemas de software.

GPT (Generative Pre-trained Transformer): Modelo de lenguaje de gran escala utilizado como motor principal de la solución.

HNSW (Hierarchical Navigable Small World): Algoritmo de indexación vectorial para búsquedas eficientes de similitud semántica.

IA: Inteligencia Artificial.

JSON (JavaScript Object Notation): Formato estándar de intercambio de datos estructurados para la salida del sistema.

KH (Knowledge about how to proceed): Dimensión de retroalimentación que orienta al estudiante sobre cómo corregir sus errores y dar el siguiente paso, sin revelar directamente la solución.

KM (Knowledge about mistakes): Dimensión de retroalimentación que informa con precisión la naturaleza de los fallos lógicos o de estilo del estudiante.

LLMs (Large Language Models): Modelos de Lenguaje de Gran Escala.

PDF (Portable Document Format): Formato de documento utilizado para la extracción de texto de los enunciados.

RAG (Retrieval-Augmented Generation): Arquitectura que combina la recuperación de documentos históricos con la capacidad generativa de un LLM.

REST (Representational State Transfer): Estilo arquitectónico utilizado para exponer las funcionalidades del sistema mediante *endpoints*.

SMOJ (Santa María Online Judge): Plataforma institucional de evaluación automática basada en casos de prueba.

USD (United States Dollar): Dólar estadounidense, moneda de referencia para el cálculo de costos de la API de OpenAI.

UTFSM: Universidad Técnica Federico Santa María.

UVA (Unidad Virtual de Aprendizaje): División curricular que segmenta los contenidos y restricciones académicas del curso.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	IX
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Situación actual y ecosistema evaluativo	3
1.2 Actores involucrados	4
1.3 Árbol del problema	5
1.3.1 Problema central	5
1.3.2 Causas del problema	5
1.3.3 Consecuencias del problema	6
1.4 Objetivos de la memoria	6
CAPÍTULO 2: MARCO CONCEPTUAL	8
2.1 Evaluación del aprendizaje en programación inicial	8
2.1.1 Evidencias del pensamiento computacional	8
2.1.2 Comprensión estructural y modelos mentales	9
2.1.3 Calidad de Código	10
2.2 Retroalimentación formativa efectiva	10
2.2.1 Características de un buen <i>feedback</i>	11
2.3 LLMs en educación	12
2.3.1 LLMs en aprendizaje de programación	12
2.3.2 Ingeniería de <i>prompts</i> en educación de programación	13
2.3.3 Beneficios de LLMs en educación	14
2.3.4 Desafíos y limitaciones de LLMs en educación	14
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	16
3.1 Descripción del sistema	16
3.1.1 Arquitectura del sistema	16
3.1.2 Modelo e infraestructura de inteligencia artificial	18
3.2 Generación de rúbricas automatizada	18
3.2.1 Corpus histórico e índice vectorial	19
3.2.2 Pipeline de generación	19
3.2.3 Ejemplo de Rúbrica Generada	21
3.3 Evaluación y retroalimentación automatizada	21

3.3.1	Pipeline de Evaluación	22
3.3.2	Integración de restricciones académicas	23
3.3.3	Evaluación y generación de retroalimentación	24
3.3.4	Ejemplo de evaluación realizada	24
3.4	Estrategia de <i>Prompt Engineering</i>	25
3.4.1	Prompt generación de rúbricas	26
3.4.2	Prompt de evaluación	29
3.5	Metodología de implementación	33
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		34
4.1	Alcance y limitaciones de la evaluación	34
4.2	Eficiencia del motor	35
4.3	Evaluación de calidad de las rúbricas y criterios usados	38
4.3.1	Cobertura del enunciado	40
4.3.2	Precisión de los descriptores	41
4.3.3	Estructura de la rúbrica	42
4.3.4	Adopción y valoración global	44
4.4	Percepción de los estudiantes sobre la retroalimentación	46
4.4.1	Comprensión de la retroalimentación	48
4.4.2	Utilidad para mejorar	49
4.4.3	Justicia de la evaluación	53
4.5	Discusión de resultados	54
CAPÍTULO 5: CONCLUSIONES		56
5.1	Cumplimiento de los objetivos específicos	56
5.2	Principales resultados y contribuciones	57
5.3	Limitaciones del estudio	58
5.4	Trabajo futuro	59
ANEXOS		60
REFERENCIAS BIBLIOGRÁFICAS		70

ÍNDICE DE FIGURAS

1	Árbol del problema.	6
2	Arquitectura del sistema y flujo de interacción entre componentes.	17
3	Diagrama de flujo del proceso de generación de rúbrica	20
4	Rúbrica generada automáticamente a partir del enunciado de la Tarea 4 del semestre 2026-1.	21
5	Diagrama de flujo del proceso de evaluación.	23
6	Ejemplo de evaluación sobre la entrega de un estudiante para la Tarea 3 semestre 2026-1.	25
7	Vista para configuración de prompts.	26
8	Promedios de valoración docente en la dimensión de Cobertura del Enunciado (Escala Likert 1-5).	41
9	Promedios de valoración docente en la dimensión de Precisión de los Descriptores (Escala Likert 1-5).	42
10	Distribución de frecuencias docentes respecto al orden lógico de los criterios en la rúbrica.	44
11	Promedios de valoración docente sobre el impacto temporal y el desempeño técnico del motor (Escala Likert 1-5).	45
12	Distribución porcentual docente sobre la disposición para integrar la rúbrica automatizada en el aula.	46
13	Promedios de percepción estudiantil en la dimensión de Comprensión de la Retroalimentación (Escala Likert 1-5).	48
14	Distribución de frecuencias y promedio estudiantil respecto al carácter orientador de la retroalimentación.	50
15	Distribución de frecuencias y promedio estudiantil respecto a la utilidad práctica en la optimización del código.	51
16	Distribución de frecuencias y promedio estudiantil respecto al impacto en el reforzamiento conceptual.	52
17	Promedios de percepción estudiantil en la dimensión de Justicia de la Evaluación (Escala Likert 1-5).	53
18	Distribución de frecuencias y promedios de respuestas docentes respecto a la cobertura de restricciones explícitas del enunciado.	60
19	Distribución de frecuencias y promedios de respuestas docentes respecto a la inclusión de criterios orientados a fallos lógicos y casos límite.	61
20	Distribución de frecuencias y promedios de respuestas docentes respecto al alineamiento de los criterios generados con las unidades virtuales de aprendizaje.	61
21	Distribución de frecuencias y promedios de respuestas docentes respecto a la fundamentación de los descriptores basados en evidencia observable en el código.	62
22	Distribución de frecuencias y promedios de respuestas docentes respecto a la claridad en la delimitación de fronteras entre niveles de desempeño.	62

23	Distribución de frecuencias y promedios de respuestas docentes respecto a la idoneidad del nivel de exigencia técnica y el lenguaje adaptado.	63
24	Distribución de frecuencias y promedios de respuestas docentes respecto a la densidad de criterios en relación con la complejidad de la tarea.	63
25	Distribución de frecuencias y promedios de respuestas docentes respecto al nivel de independencia de los criterios y la mitigación de redundancias. . . .	64
26	Distribución de frecuencias y promedios de respuestas docentes respecto a la proporcionalidad en la asignación de puntaje entre criterios.	64
27	Distribución de frecuencias y promedios de respuestas docentes respecto al valor percibido en la generación automatizada de pautas de evaluación. . . .	65
28	Distribución de frecuencias y promedios de respuestas docentes respecto a la valoración global del desempeño técnico del motor.	65
29	Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de comprensión de los errores identificados en el código.	66
30	Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de claridad explicativa de las causas de los errores.	67
31	Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de legibilidad del lenguaje técnico empleado.	67
32	Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de correspondencia con fallos reales del programa.	68
33	Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de pertinencia con los límites del enunciado.	68
34	Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de consistencia con el avance curricular de la asignatura.	69

ÍNDICE DE TABLAS

1	Tipos de <i>chunks</i> utilizados en el sistema de recuperación del corpus histórico. Fuente: Elaboración propia.	19
2	Métricas de tiempo del motor de software por instancia evaluativa (5 ejecuciones por tarea). Fuente: Elaboración propia.	35
3	Consumo de <i>tokens</i> por instancia evaluativa (5 ejecuciones por tarea). Fuente: Elaboración propia.	36
4	Síntesis de hallazgos del cuerpo docente respecto al diseño y estructura de la rúbrica. Fuente: Elaboración propia.	43

INTRODUCCIÓN

Durante los últimos años, la Inteligencia Artificial se ha consolidado como una herramienta ampliamente utilizada en múltiples ámbitos de la sociedad, siendo la educación superior uno de ellos. En este sector, los modelos de lenguaje de gran escala (LLMs) se han convertido en un recurso clave para optimizar diversos procesos debido a su habilidad para estructurar contenido, identificar patrones complejos y generar explicaciones coherentes, sirviendo como un soporte directo tanto para docentes como para estudiantes. Particularmente, en cursos introductorios masivos de programación, la adopción de estas herramientas permite abordar de manera eficiente desafíos estructurales como el diseño de instrumentos evaluativos y la entrega de retroalimentación a gran escala.

Esta problemática se manifiesta con especial intensidad en la asignatura Introducción a la Programación dentro de la UTFSM, curso que es dictado para las distintas carreras de la institución y que llega a recibir a más de 1 000 estudiantes simultáneamente cada semestre. Esta asignatura consta de múltiples instancias evaluativas a lo largo del semestre, las cuales requieren el diseño de rúbricas específicas y una posterior revisión manual de certámenes y tareas por parte del cuerpo docente. Ante la masividad estudiantil característica de este curso, el desarrollo de este proceso actúa como un cuello de botella que afecta directamente la oportunidad y la profundidad del *feedback* que recibe el estudiantado, un elemento determinante para su aprendizaje, progresión académica y motivación.

Frente a este escenario, la presente memoria de título propone el desarrollo de un motor de generación automática de rúbricas y retroalimentación personalizada basado en LLMs. Para asegurar la calidad de las respuestas de estos modelos, la solución articula dos componentes clave: en primer lugar, una arquitectura de Recuperación Aumentada por Generación (RAG) que nutre al sistema con información adicional proveniente del registro histórico del curso, en segundo lugar, un conjunto de *prompts* especializados para cada tarea, complementados con un módulo de ejecución de código. De este modo, el sistema entrega criterios alineados con el estándar institucional y un *feedback* de enfoque socrático que apoya la labor de corrección sin sustituir el criterio del evaluador.

La metodología empleada se estructuró bajo un enfoque de desarrollo incremental compuesto por fases consecutivas. En primer lugar, se recopilieron y procesaron las evaluaciones históricas del curso para estructurar su representación textual. Posteriormente, se implementaron de forma iterativa los módulos lógicos del motor encargados de la generación de rúbricas y la evaluación de entregas, a la par con el diseño y refinamiento continuo de las plantillas de *prompts*. Estos componentes se integraron con una interfaz web. Finalmente, el sistema consolidado se validó en un escenario real con actividades del semestre 2026-1, recopilando la percepción de utilidad de profesores, ayudantes y estudiantes.

El alcance de este trabajo es de carácter exploratorio. Se centra en evaluar la viabilidad técnica y pedagógica de automatizar componentes del proceso evaluativo en un contexto masivo,

sin pretender establecer generalizaciones estadísticas definitivas sobre el rendimiento académico de la población. No obstante, los hallazgos entregan directrices y aprendizajes valiosos sobre el despliegue responsable, seguro y viable de herramientas basadas en inteligencia artificial dentro del aula universitaria.

La estructura de este documento se organiza de la siguiente manera. El Capítulo 1 detalla la definición del problema, el contexto institucional y los objetivos del trabajo. El Capítulo 2 expone el marco teórico y conceptual que sustenta la solución. El Capítulo 3 describe el diseño arquitectónico y la implementación de los módulos de rúbricas y evaluación. El Capítulo 4 presenta la metodología de validación y el análisis de los resultados obtenidos. Finalmente, el Capítulo 5 expone las conclusiones de la memoria, identificando sus principales contribuciones, limitaciones y líneas de trabajo futuro.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

El desarrollo del pensamiento computacional se ha consolidado como un pilar fundamental en la formación académica de los ingenieros modernos. Esta competencia permite a los estudiantes abordar problemas complejos mediante la abstracción, la descomposición y el diseño de algoritmos, habilidades que trascienden la mera escritura de código. Como señala la literatura fundamental, el pensamiento computacional implica resolver problemas y diseñar sistemas recurriendo a los conceptos esenciales de la informática para la toma de decisiones sistemática.

En este contexto, la Universidad Técnica Federico Santa María imparte la asignatura Introducción a la Programación. El curso busca iniciar a los estudiantes de primer año en estas capacidades utilizando el lenguaje Python, seleccionado por su sintaxis intuitiva que facilita el enfoque en la lógica de resolución de problemas por sobre las restricciones del lenguaje. Dada su relevancia, la asignatura se dicta de forma transversal en todos los campus y sedes de la institución, alcanzando 1 821 estudiantes durante el primer semestre de 2026.

La masiva cantidad de estudiantes inscritos representa un desafío crítico para el equipo docente, compuesto por aproximadamente 110 integrantes entre profesores y ayudantes, con una proporción promedio de 16 estudiantes por persona. Esta alta demanda, sumada a la naturaleza manual de la creación de enunciados, rúbricas y del proceso de retroalimentación, genera una carga significativa de trabajo que impacta directamente en la eficiencia del proceso evaluativo.

En este contexto, la ausencia de una herramienta integrada que asista al equipo académico en la automatización o agilización de estas tareas complejas constituye una limitación relevante. Esta situación motiva el desarrollo de la presente solución, orientada a apoyar y optimizar los procesos de evaluación mediante el uso de modelos de lenguaje y generación automática de rúbricas y retroalimentación.

1.1. Situación actual y ecosistema evaluativo

La asignatura cuenta con un sistema de evaluación diseñado para medir el progreso de los estudiantes a través de diversas instancias, que difieren tanto en su formato como en su nivel de complejidad. Estas evaluaciones se complementan entre sí con el objetivo de cubrir distintas dimensiones del aprendizaje, desde la comprensión conceptual hasta la resolución de problemas de programación.

En este contexto, las principales instancias evaluativas se describen a continuación:

- **Certámenes:** Constituyen las evaluaciones de mayor ponderación en la calificación final. Se realizan dos veces por semestre en formato tradicional (papel y lápiz), enfocándose en la capacidad de síntesis y resolución lógica de forma presencial.
- **Controles de Unidad:** Evaluaciones breves realizadas en la plataforma institucional AULA. Su objetivo es medir la comprensión de contenidos específicos mediante ejercicios de modificación o completación de código.
- **Tareas:** Actividades de mayor complejidad orientadas a la integración de contenidos de múltiples unidades, donde los estudiantes deben desarrollar soluciones completas a problemas de programación.
- **Evaluaciones formativas:** Actividades no ponderadas con intentos ilimitados, destinadas a validar la comprensión de contenidos audiovisuales previos a las sesiones de aprendizaje.
- **Ejercicios de corrección automática:** Implementados en la plataforma *Santa María Online Judge* (SMOJ), permiten la práctica autónoma mediante retroalimentación automática basada en casos de prueba, sin incidencia directa en la nota final.
- **Tickets de salida:** Cuestionarios breves aplicados al término de las ayudantías, cuyo propósito es monitorear el nivel de comprensión inmediata de los contenidos trabajados en el laboratorio.

Si bien estas instancias permiten evaluar de manera integral el aprendizaje del estudiante, presentan diferencias importantes en términos de carga operativa para el equipo docente. En particular, mientras que las evaluaciones automatizadas, como los ejercicios en SMOJ, permiten una corrección prácticamente instantánea basada en casos de prueba, otras instancias como los certámenes y las tareas requieren un proceso significativamente más complejo.

En estas últimas, el proceso evaluativo depende de la elaboración de rúbricas específicas para cada enunciado y de la posterior corrección cualitativa de las entregas en función de dichos criterios. Esto introduce una carga adicional de trabajo, especialmente en contextos con alta cantidad de estudiantes, convirtiéndose en uno de los principales cuellos de botella del proceso evaluativo.

1.2. Actores involucrados

La alta carga de trabajo asociada a la evaluación cualitativa de más de 1800 estudiantes impacta de manera diferenciada tanto al equipo docente como al estudiantado, generando efectos directos en la eficiencia y calidad del proceso evaluativo.

- **Equipo Docente y ayudantes:** Este grupo enfrenta una alta demanda asociada al diseño de rúbricas claras y alineadas con los objetivos de la asignatura para cada nuevo

enunciado desarrollado semestre a semestre. Asimismo, la corrección manual de un gran volumen de entregas implica una carga significativa de trabajo, lo que dificulta mantener criterios de evaluación consistentes y estandarizados entre los distintos evaluadores.

- **Estudiantes de la Asignatura:** El estudiantado se ve afectado por la sobrecarga existente en el cuerpo docente, lo que impacta la calidad y profundidad del feedback recibido. En particular, la elevada carga de corrección puede derivar en retroalimentación menos detallada y con mayores tiempos de entrega, lo que limita la capacidad de los estudiantes para identificar y corregir oportunamente sus errores conceptuales.

1.3. Árbol del problema

El árbol del problema en la Figura 1 presenta los factores que contribuyen a la alta carga, esfuerzo y tiempo requerido por el equipo docente en el proceso evaluativo en cursos introductorios de programación.

1.3.1. Problema central

El eje central del árbol corresponde a la problemática principal: “Alta carga, esfuerzo y tiempo invertido por el equipo docente en el proceso de evaluación (elaboración de rúbricas, corrección de ejercicios, entrega de retroalimentación) en cursos introductorios de programación”.

1.3.2. Causas del problema

En la parte inferior del árbol se encuentran las raíces que representan las causas de la problemática central identificada, las cuales se detallan a continuación:

- Alta cantidad de estudiantes por profesor o ayudante en cursos masivos.
- Escasez de tiempo para elaborar rúbricas detalladas y consistentes.
- Falta de herramientas tecnológicas que apoyen el proceso de evaluación más allá del chequeo funcional del código.

1.3.3. Consecuencias del problema

Finalmente, en la parte superior del árbol se visualizan las consecuencias del problema definido:

- Falta de personalización del aprendizaje de los estudiantes al recibir evaluaciones genéricas.
- Sobrecarga de trabajo y desgaste del equipo docente.
- Desmotivación del estudiante y menor desarrollo de habilidades de programación de nivel superior.

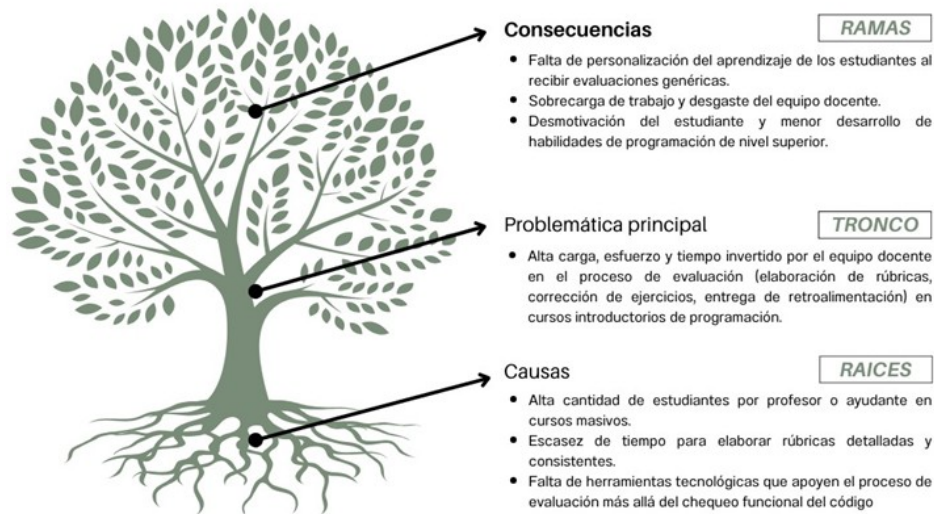


Figura 1: Árbol del problema.
Fuente: Elaboración propia.

1.4. Objetivos de la memoria

El objetivo principal de la memoria es construir un sistema de generación automática de rúbricas y retroalimentación personalizada que optimice el proceso de evaluación de ejercicios de programación en cursos introductorios. A partir de este objetivo general, se definen los siguientes objetivos específicos:

1. Identificar los criterios relevantes y dimensiones claves que deben ser consideradas en la evaluación de ejercicios de programación en cursos introductorios.

2. Determinar los principios y características de una retroalimentación formativa efectiva en cursos introductorios de programación.
3. Desarrollar un prototipo de sistema automatizado que integre la generación de rúbricas y la entrega de retroalimentación personalizada, orientado a mejorar la eficiencia y consistencia del proceso evaluativo.
4. Evaluar los efectos del sistema propuesto en la eficiencia de la evaluación docente, considerando la reducción del tiempo y esfuerzo requerido, así como la calidad de la retroalimentación entregada.

CAPÍTULO 2

MARCO CONCEPTUAL

Este capítulo expone los fundamentos teóricos y tecnológicos que sustentan el desarrollo de la solución propuesta. En él se analizan los conceptos de evaluación educativa en informática, las dimensiones de la retroalimentación formativa y el rol de la inteligencia artificial en contextos pedagógicos, estableciendo las bases necesarias para la generación automática de rúbricas de evaluación y retroalimentación personalizada en contextos educativos.

2.1. Evaluación del aprendizaje en programación inicial

La evaluación en las asignaturas introductorias de programación representa un desafío pedagógico complejo, que trasciende la exclusiva verificación de la funcionalidad técnica del código. Evaluar en este contexto requiere centrarse tanto en los elementos cognitivos del estudiante como en las características estructurales de la solución propuesta.

2.1.1. Evidencias del pensamiento computacional

El pensamiento computacional se define como el conjunto de habilidades cognitivas necesarias para formular y resolver problemas, así como para diseñar soluciones, utilizando conceptos fundamentales de la informática [Wing, 2006]. Su desarrollo está estrechamente vinculado a la comprensión estructural del estudiante, ya que las habilidades cognitivas determinan la manera en que se organiza y construye la solución en código.

Para promover y evaluar el pensamiento computacional, se espera que los estudiantes apliquen estrategias metodológicas de resolución de problemas [Dapozo *et al.*, 2022]. Según el marco de evaluación adoptado en la literatura, las dimensiones evaluables se estructuran en cinco habilidades fundamentales:

- **Abstracción:** Capacidad de simplificar un problema, omitiendo detalles innecesarios y centrando la atención en los elementos relevantes.
- **Descomposición:** Habilidad para dividir un problema complejo en partes más pequeñas y manejables.
- **Diseño Algorítmico:** Capacidad para organizar un conjunto de instrucciones de manera lógica y ordenada para resolver un problema paso a paso.
- **Generalización:** Aptitud para identificar patrones o similitudes entre problemas, permitiendo la reutilización de soluciones y lógicas en contextos diferentes.

- **Evaluación:** Habilidad para reconocer y determinar los alcances de una solución o proceso, juzgando su eficacia y eficiencia en el uso de recursos computacionales.

2.1.2. Comprensión estructural y modelos mentales

Los modelos mentales se definen como las representaciones internas que un estudiante construye sobre la estructura, organización y comportamiento dinámico de un sistema computacional durante la ejecución de un programa [Mselle, 2025]. Estas representaciones permiten comprender de manera abstracta el flujo de datos y los cambios de estado en la memoria del sistema.

El proceso de evaluación no debe limitarse únicamente a la obtención de resultados correctos o a la memorización de elementos sintácticos, ya que el estudiante puede experimentar una alta carga cognitiva que dificulta la construcción de modelos mentales adecuados sobre el comportamiento del sistema. La evidencia sugiere que el éxito funcional no implica necesariamente comprensión algorítmica; los estudiantes enfocados exclusivamente en hacer que su código supere los casos de prueba (*test cases*) pueden internalizar lógicas defectuosas o redundantes, desarrollando lo que la literatura denomina conceptos erróneos en código correcto [da Silva *et al.*, 2023].

Para estructurar la separación metodológica entre la maquinaria del código y su intención, la enseñanza de la programación se apoya en el Modelo de Bloques de Schulte [Schulte, 2008]. Este modelo establece la dualidad fundamental entre la *Estructura* del artefacto digital y su *Función*, postulando que la comprensión de un programa debe analizarse en distintos niveles de abstracción. Por lo tanto, un instrumento de evaluación integral debe ser capaz de valorar el progreso del estudiante considerando no solo la corrección técnica, sino la trazabilidad de su significado. En este contexto, la literatura organiza el análisis en torno a tres dimensiones principales:

En primer lugar, la **comprensión de la superficie del texto** corresponde al reconocimiento de la sintaxis del lenguaje y la correcta utilización de su vocabulario [Schulte, 2008, Izu *et al.*, 2024]. En este nivel, el estudiante identifica estructuras básicas del código, pero sin necesariamente comprender su comportamiento global.

A partir de esta base sintáctica, el análisis avanza hacia la **ejecución del programa**, lo que implica la capacidad de seguir mentalmente el flujo de ejecución, comprendiendo cómo varían los estados de las variables y cómo interactúan las estructuras de control dinámicamente en la máquina nocional [Izu y Mirolo, 2020].

Finalmente, el proceso culmina en la **función o propósito del programa**, que corresponde al nivel más alto de comprensión. En esta etapa, el estudiante es capaz de interpretar la intención del algoritmo, explicar su comportamiento general en lenguaje natural y relacionarlo con el problema que busca resolver [Izu y Mirolo, 2020]. Este enfoque se alinea con

propuestas de evaluación de calidad de código en cursos introductorios, donde se enfatiza la importancia de considerar dimensiones que van más allá de la corrección funcional [Izu *et al.*, 2024].

2.1.3. Calidad de Código

Una vez que se comprende cómo los estudiantes estructuran cognitivamente su conocimiento a través de modelos mentales, es posible evaluar de qué manera dicha comprensión se refleja en su propio código. La calidad del código se define como el conjunto de características estructurales de una solución que pueden ponderarse mediante la inspección estática del código fuente, trascendiendo la mera corrección funcional o el éxito ante pruebas automatizadas [Izu *et al.*, 2024]. Bajo esta perspectiva pedagógica, un código de alta calidad no se limita a la eficiencia de cara a la máquina, sino que se evalúa en función de su legibilidad, mantenibilidad y expresividad [Izu *et al.*, 2024, Kirk *et al.*, 2024].

Para sistematizar esta evaluación en el aula, la literatura reciente delimita principios de estilo esenciales orientados a resguardar la legibilidad y la mantenibilidad desde las etapas iniciales de formación [Izu *et al.*, 2024, Kirk *et al.*, 2024]. Entre estos criterios destaca, en primer lugar, el uso de un lenguaje explicativo, el cual exige tanto la selección de nombres descriptivos para variables y funciones, como la inclusión de comentarios pertinentes, garantizando que el propósito semántico del código sea evidente para cualquier lector.

En segundo lugar, se evalúan de forma complementaria la implementación apropiada y el uso de constructos simples. El primer principio implica seleccionar estructuras de control y abstracciones conceptualmente adecuadas para la naturaleza del problema a resolver. El segundo principio busca minimizar la complejidad estructural de dicha solución, reduciendo activamente las anidaciones innecesarias que derivan en una alta carga cognitiva para el lector.

Finalmente, la calidad estructural demanda un contenido no redundante y la evitación estricta de duplicación. La redundancia se combate mediante la eliminación sistemática de código muerto o inalcanzable, mientras que la evitación de duplicidad impulsa al estudiante a la modularización de bloques repetidos, promoviendo el desarrollo de soluciones algorítmicas más elegantes, compactas y mantenibles [Izu *et al.*, 2024, Kirk *et al.*, 2024].

2.2. Retroalimentación formativa efectiva

La retroalimentación o *feedback* se define como el proceso mediante el cual los estudiantes obtienen información sobre su trabajo, permitiéndoles evaluar las similitudes y diferencias entre los estándares esperados y la calidad de su propia elaboración, con el objetivo de generar mejoras continuas [Boud y Molloy, 2013]. Además, cumple un rol formativo al

constituirse como la información comunicada al estudiante para modificar su pensamiento o comportamiento con el fin de mejorar su aprendizaje [Shute, 2008].

2.2.1. Características de un buen *feedback*

Para que la retroalimentación sea verdaderamente efectiva en la enseñanza de la programación, la evidencia científica establece que debe estructurarse en torno a tres dimensiones clave:

1. **Contenido y profundidad:** El *feedback* no debe limitarse a entregar resultados simples, como una calificación numérica o un mensaje de correcto o incorrecto (componente evaluativo). Un referente clave para proporcionar un *feedback* de alta calidad es el modelo planteado por Narciss [Narciss, 2008], el cual ha sido adaptado a la programación por Keuning et al. [Keuning et al., 2018]. Bajo este enfoque, la retroalimentación se optimiza superando la mera verificación e incorporando un componente informativo basado en dos dimensiones fundamentales para el estudiante:
 - **Conocimiento sobre errores (*Knowledge about mistakes - KM*):** Se debe informar con precisión la naturaleza del error. Esta información abarca desde detalles puntuales (como indicar la línea de código del fallo o reportar un error de compilación) hasta la identificación de deficiencias de mayor profundidad. Entre estas últimas se incluyen errores lógicos (o de solución), problemas de estilo relacionados con la calidad del código (por ejemplo, nombres de variables poco descriptivos) o problemas de rendimiento y redundancia [Keuning et al., 2018].
 - **Conocimiento sobre cómo proceder (*Knowledge about how to proceed - KH*):** Junto con la identificación de los errores, se debe orientar al estudiante sobre cómo corregirlos y dar el siguiente paso, pero sin revelar directamente la solución. En su lugar, se deben entregar pistas en forma de sugerencias para la corrección (*hints*), ejemplos ilustrativos o indicaciones sobre los procesos a seguir, facilitando así la mejora autónoma de las deficiencias detectadas [Keuning et al., 2018].
2. **Estilo y tono de comunicación:** La forma en la que se redacta y entrega el mensaje es vital para su aceptación y para fomentar la autonomía en el estudiante. Para que una retroalimentación alcance un alto estándar de calidad, la literatura establece que debe cumplir con tres propiedades fundamentales [Jacobsen y Weber, 2025]:
 - **Específico:** Apunta a errores concretos utilizando la terminología técnica del dominio (ej. nombres de variables, ciclos, funciones) y explica claramente el porqué del fallo.
 - **Empático:** Está redactado en primera persona para generar un diálogo constructivo y contrarresta la crítica técnica con valoraciones positivas sobre el esfuerzo o las lógicas correctas que el estudiante sí logró implementar.

- **Activador:** Fomenta la autorregulación mediante el uso de preguntas activadoras o de reflexión (también conocidas en la pedagogía como preguntas socráticas), es decir, preguntas que invitan al alumno a analizar el comportamiento de su propio código y a deducir alternativas por sí mismo.
3. **Momento de Entrega:** El impacto psicológico y conductual del *feedback* en la resolución de problemas depende directamente de su oportunidad de entrega [Rocha, 2024]:
- El *feedback* inmediato (proactivo o reactivo) está asociado significativamente con mayores niveles de compromiso, esfuerzo y persistencia en programadores novatos.
 - El *feedback* tardío o ausente provoca estados afectivos negativos como frustración, aburrimiento y ansiedad, lo que aumenta la probabilidad de que el alumno dependa de asistencia externa no deseada o busque pistas que le ofrezcan respuestas ya hechas.

2.3. LLMs en educación

Los LLMs son sistemas de inteligencia artificial diseñados para el procesamiento avanzado del lenguaje natural. Estos modelos pueden generar respuestas de apariencia humana al identificar estructuras complejas y patrones semánticos mediante un entrenamiento con grandes volúmenes de datos. Esta capacidad posibilita la producción de secuencias de información compleja, la síntesis e integración de diversos tipos de información (tales como texto, audio e imágenes) [Hadi *et al.*, 2025].

2.3.1. LLMs en aprendizaje de programación

Los LLMs son herramientas que pueden apoyar la educación de programación bajo diversas aristas [Raihan *et al.*, 2025] como:

- **Retroalimentación formativa y depuración de código:** Generar pistas, explicaciones y mensajes de errores acordes al contexto en el que se está trabajando, ayudando a identificar sus propios errores lógicos o sintácticos.
- **Evaluación y calificación:** Se utilizan como asistentes de enseñanza automatizados para evaluar entregas y proporcionar retroalimentación a gran escala.
- **Diseño de problemas y materiales educativos:** Diseñar nuevos enunciados de problemas de programación, generar posibles soluciones e identificar conceptos erróneos comunes de los estudiantes para asociarles una retroalimentación específica.

2.3.2. Ingeniería de *prompts* en educación de programación

La ingeniería de *prompts* se define como el proceso de diseñar estímulos efectivos para que los LLMs generen respuestas precisas y contextualmente relevantes. Al respecto, Jacobsen y Weber [Jacobsen y Weber, 2025] señalan que la calidad de los resultados no depende únicamente de los algoritmos fundamentales o los datos de entrenamiento del modelo, sino de la claridad y precisión de las instrucciones proporcionadas. Esto convierte a la formulación de *prompts* en una competencia docente esencial para la integración de la IA en el aula.

Para mitigar errores y asegurar que la generación de rúbricas y retroalimentación sea pertinente, las instrucciones deben estructurarse bajo parámetros rigurosos. Basado en el manual de diseño y evaluación de *prompts* propuesto por los mismos autores [Jacobsen y Weber, 2025], un esquema efectivo para interactuar con el sistema debe articularse en torno a ocho categorías fundamentales:

- **Rol:** Define la identidad, perspectiva o personaje que deben adoptar tanto el modelo de lenguaje como el usuario (por ejemplo, actuar como un tutor experto) para encuadrar su comportamiento.
- **Audiencia objetivo:** Determina y describe de forma clara el público al que se dirige el contenido generado, adaptando el nivel de la respuesta a sus características.
- **Medio o canal:** Establece la plataforma, estilo o formato comunicativo a través del cual se presentará la información (por ejemplo, un ensayo académico o un mensaje de retroalimentación).
- **Misión:** Constituye la categoría central, describiendo de manera directa y precisa la tarea o pregunta exacta que el modelo lingüístico debe responder o ejecutar.
- **Restricciones de formato:** Define las propiedades estilísticas y las especificaciones de longitud esperadas en la salida (por ejemplo, uso de viñetas y límites en la cantidad de palabras).
- **Concisión:** Asegura que la instrucción contenga exclusivamente la información directamente relevante para la tarea, excluyendo detalles superfluos para no diluir el enfoque del modelo.
- **Especificidad de dominio:** Asegura la utilización correcta de terminología técnica y vocabulario propio del área temática analizada, permitiendo que el modelo adopte y refleje dicho léxico en su respuesta.
- **Lógica (Coherencia interna):** Asegura un flujo de lectura fluido, una secuencia coherente de la información y una conexión claramente comprensible entre el contenido y la misión solicitada.

2.3.3. Beneficios de LLMs en educación

La integración de LLMs ha transformado la pedagogía al optimizar la labor docente mediante el soporte de sistemas inteligentes. Desde una perspectiva de ingeniería educativa, estos modelos mejoran la enseñanza a través de los siguientes beneficios:

- **Escalabilidad de la evaluación:** Permiten automatizar la creación de recursos instruccionales como cuestionarios, pruebas de opción múltiple y ejercicios de código [Tang *et al.*, 2025]. Asimismo, operan como asistentes valiosos en la preparación de programas de estudio y en la generación de objetivos de aprendizaje [Hadi *et al.*, 2025].
- **Tutoría personalizada:** Los LLMs actúan como tutores virtuales capaces de ofrecer *feedback* inmediato y sensible al contexto, lo que permite adaptar las explicaciones al nivel de competencia técnica de los alumnos, un factor crítico en cursos de ciencias de la computación con alta densidad de matrícula [Pitts *et al.*, 2025, Tang *et al.*, 2025].
- **Asistencia en tiempo real:** Esta tecnología facilita el soporte para la depuración de código y la detección inteligente de errores en tiempo real, ayudando a los estudiantes a superar obstáculos sintácticos y lógicos de manera autónoma [Pitts *et al.*, 2025, Tang *et al.*, 2025].
- **Optimización del tiempo docente:** Al delegar tareas académicas intensivas en tiempo, como la preparación de material y las revisiones manuales de código, se reduce significativamente la carga de trabajo del docente [Azoulay *et al.*, 2025].

2.3.4. Desafíos y limitaciones de LLMs en educación

A pesar de sus ventajas operativas, la adopción de LLMs introduce obstáculos técnicos y éticos que impactan directamente el flujo de trabajo docente. Los principales desafíos se detallan a continuación:

- **Alucinaciones y propagación de errores:** Una limitación crítica radica en las alucinaciones, fenómeno en el que el modelo produce código, retroalimentación o explicaciones sintácticamente fluidas, pero fácticamente erróneas o inconsistentes con la lógica solicitada [Liu *et al.*, 2024, Alansari y Luqman, 2026]. Esta degradación técnica compromete el proceso de aprendizaje, pues los estudiantes corren el riesgo de asimilar de forma silenciosa lógicas defectuosas o conceptos inexactos si el contenido no es validado [Herklotz *et al.*, 2025].
- **Restricciones de la ventana de contexto efectiva:** En el análisis de tareas complejas, a medida que el volumen de *tokens* de entrada supera la ventana de contexto efectiva

máxima, se observa una disminución drástica en la exactitud del modelo y un aumento en las tasas de alucinación. Esta saturación reduce la capacidad de la IA para retener información en su memoria y mantener un razonamiento analítico consistente durante el procesamiento de secuencias largas de datos [Paulsen, 2026].

- **Riesgos éticos y operativos en la evaluación autónoma:** La delegación de procesos evaluativos a la IA introduce tres preocupaciones fundamentales en la literatura. En primer lugar, a nivel de privacidad, la información sensible introducida en los *prompts* puede ser absorbida como datos de entrenamiento [Azoulay *et al.*, 2025], sumado al riesgo de que los estudiantes confíen información personal o delicada al sistema en lugar de a un docente capacitado [Harvey *et al.*, 2025]. En segundo lugar, en el ámbito de la calificación, delegar esta tarea de forma exclusiva a la IA genera justificadas objeciones sobre la transparencia y reproducibilidad de las notas [Pitts *et al.*, 2025]. Finalmente, no se puede eliminar la supervisión humana del ciclo de retroalimentación, constituyendo un riesgo crítico, ya que el criterio del educador sigue siendo esencial para intervenir y mitigar los errores generados por el sistema automatizado [Herklotz *et al.*, 2025].

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

En el presente capítulo se describe la solución propuesta, consistente en un sistema basado en modelos de lenguaje de gran escala orientado a agilizar el proceso evaluativo en cursos introductorios de programación. La propuesta se centra en el diseño de una arquitectura modular que permite estructurar el proceso de evaluación de manera automatizada y coherente.

El sistema se organiza en torno a dos componentes principales que actúan de forma complementaria dentro del flujo evaluativo general. A lo largo del capítulo se exponen la arquitectura del sistema, el diseño de cada módulo, la estrategia de *prompt engineering* adoptada y la metodología de implementación trabajada.

3.1. Descripción del sistema

El sistema diseñado se compone de dos módulos que operan de manera complementaria para automatizar el ciclo evaluativo completo.

El primero corresponde al módulo de generación de rúbricas. Este módulo recibe como entrada un documento PDF con el enunciado de un ejercicio (certamen o tarea), junto con las unidades temáticas del curso asociadas al contenido. A partir de esto, genera una rúbrica estructurada, apoyándose en documentos históricos del curso para enriquecer el contexto de generación.

El segundo corresponde al módulo de evaluación y retroalimentación. Este módulo toma como entrada una rúbrica previamente generada dentro del sistema, junto con las soluciones entregadas por los estudiantes. A partir de esta información, produce una calificación acompañada de retroalimentación personalizada.

La relación entre ambos módulos es de dependencia secuencial, donde la salida del primer módulo constituye el instrumento evaluativo que habilita la operación del segundo, conformando así una cadena de procesamiento para la automatización del ciclo evaluativo.

3.1.1. Arquitectura del sistema

Se ha diseñado una arquitectura de software modular basada en capas con integración de servicios en la nube. El sistema consta de una interfaz web y se estructura en torno a seis componentes principales, tal como se ilustra en la Figura 2:

- **Capa de aplicación:** Implementada en Python mediante *FastAPI*, constituye el núcleo operativo del sistema. Orquesta el *pipeline* de recuperación aumentada (RAG¹), expone los *endpoints* REST y valida los esquemas de entrada y salida mediante *Pydantic*.
- **Capa de ejecución:** Implementada mediante *Docker*, ejecuta de forma aislada el código fuente de las entregas de los estudiantes y captura su salida (*stdout*, errores y archivos generados), la cual es entregada a la capa de aplicación para ser incorporada como evidencia en la evaluación.
- **Capa de extracción documental:** Procesa los archivos PDF recibidos y extrae su contenido como texto estructurado, habilitando su procesamiento posterior en el *pipeline*.
- **Capa de inteligencia artificial:** Provee los modelos de lenguaje para la generación de rúbricas y los modelos de *embeddings* para la vectorización de documentos.
- **Capa de recuperación y búsqueda:** Gestiona el índice vectorial del corpus, permitiendo búsquedas de similitud semántica para recuperar enunciados y criterios de rúbricas relacionados con el documento de entrada.
- **Capa de persistencia:** Almacena el historial de rúbricas generadas, los *prompts* editables con su historial de versiones y los datos de usuarios del sistema.

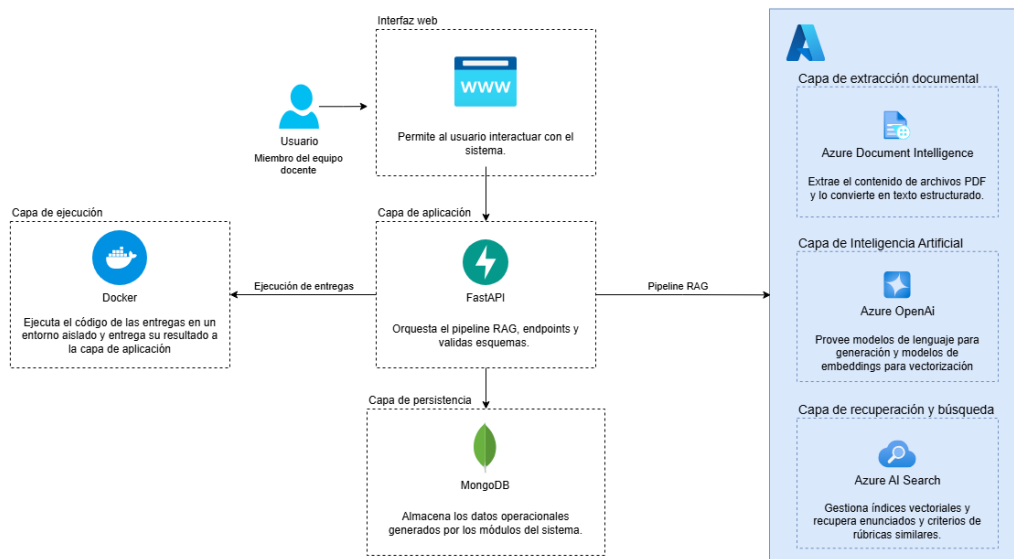


Figura 2: Arquitectura del sistema y flujo de interacción entre componentes.
Fuente: Elaboración propia.

¹*Retrieval-Augmented Generation* (RAG) es una técnica que combina la búsqueda de documentos relevantes con la generación de texto, incorporando dicha información como contexto al modelo de lenguaje.

3.1.2. Modelo e infraestructura de inteligencia artificial

La infraestructura del sistema responde a la adopción del ecosistema de *Azure* como plataforma principal de implementación, el cual proporciona un conjunto integrado de servicios para el desarrollo y despliegue de soluciones basadas en inteligencia artificial. Este entorno permite centralizar el consumo de modelos de lenguaje, así como herramientas asociadas de seguridad, trazabilidad y gestión de datos, simplificando la arquitectura general del sistema.

En este contexto, para la generación de rúbricas y la evaluación de entregas se utiliza el modelo *GPT-4o* de *Azure OpenAI*.

De manera complementaria, la representación vectorial de los documentos del corpus y de los enunciados de entrada se realiza mediante el modelo de *embeddings* de *Azure OpenAI*, generando vectores que permiten la comparación semántica entre textos.

3.2. Generación de rúbricas automatizada

Una parte fundamental del sistema es la generación automática de rúbricas a partir de enunciados de ejercicios de programación. Estas rúbricas deben ser coherentes con aquellas utilizadas históricamente en el curso de introducción a la programación y deben evaluar dimensiones relevantes en el desempeño de estudiantes principiantes.

El sistema utiliza como base un conjunto de enunciados históricos del curso, entendidos como actividades evaluativas previamente aplicadas en semestres anteriores de la asignatura. Este conjunto constituye el corpus histórico utilizado para contextualizar la generación de nuevas rúbricas.

Una aproximación inicial consistiría en incluir la totalidad de este corpus como contexto en cada consulta al modelo de lenguaje, sin embargo, este enfoque no es viable debido a las limitaciones de la ventana de contexto de los modelos actuales, las cuales restringen la cantidad de información que puede ser procesada simultáneamente. Adicionalmente, el aumento en el volumen de *tokens* por consulta impacta directamente en la eficiencia del sistema.

Para abordar esta restricción, el sistema implementa un esquema *Retrieval-Augmented Generation* (RAG), el cual selecciona únicamente los enunciados históricamente más similares al documento de entrada. De este modo, se reduce el contexto enviado al modelo, priorizando la información más relevante y favoreciendo la coherencia de las rúbricas generadas con el historial evaluativo del curso.

3.2.1. Corpus histórico e índice vectorial

El sistema de recuperación se basa en un corpus compuesto por enunciados y rúbricas correspondientes a actividades evaluativas históricas del curso. Cada enunciado se encuentra vinculado a su respectiva rúbrica mediante metadatos que registran su identificador y el semestre en el cual fue aplicado, lo que permite mantener la trazabilidad entre las actividades y sus instrumentos de evaluación.

Los documentos del corpus se organizan en unidades denominadas *chunks*, que corresponden a fragmentos de texto segmentados con el objetivo de facilitar su indexación y recuperación independiente dentro del sistema.

Se distinguen dos tipos principales de *chunks*, los cuales se resumen en la Tabla 1. El primero corresponde a *enunciado_completo*, que contiene el texto íntegro de cada actividad y se utiliza en la etapa de búsqueda semántica para identificar enunciados similares. El segundo corresponde a *rubrica_criterio*, que almacena los criterios de evaluación asociados a cada enunciado, junto con sus niveles de desempeño y ponderaciones, permitiendo la recuperación exacta de la rúbrica correspondiente.

Tabla 1: Tipos de *chunks* utilizados en el sistema de recuperación del corpus histórico. Fuente: Elaboración propia.

Tipo	Descripción	Uso en el sistema
enunciado_completo	Texto íntegro de la actividad	Búsqueda semántica de enunciados similares
rubrica_criterio	Criterio de evaluación con niveles de desempeño y ponderaciones	Recuperación exacta de la rúbrica asociada al enunciado

Cada *chunk* es representado mediante embeddings generados con el modelo de Azure OpenAI y almacenado en Azure AI Search. El índice utiliza el algoritmo HNSW para realizar búsquedas aproximadas de vecinos más cercanos en espacios vectoriales de alta dimensionalidad. Adicionalmente, los campos de metadatos como el identificador del laboratorio y el semestre permiten realizar filtrado exacto durante la fase de recuperación.

3.2.2. Pipeline de generación

El proceso de generación se estructura en tres etapas secuenciales, tal como se ilustra en la Figura 3

1. **Clasificación y validación:** Al recibir el documento PDF, el sistema realiza una llamada preliminar al modelo de lenguaje con los primeros 3.000 caracteres del documento

para verificar si el contenido corresponde a un enunciado académico de programación. Esta etapa actúa como filtro de entrada, evitando el procesamiento de documentos no pertinentes y el consumo innecesario de recursos.

2. **Recuperación semántica:** El documento es procesado mediante *Azure Document Intelligence*, que extrae el texto desde el archivo PDF. A partir de este contenido, se genera una representación vectorial utilizando *Azure OpenAI*, lo que permite comparar semánticamente el enunciado de entrada con el corpus histórico del sistema.

Luego, se realiza una búsqueda sobre los *chunks* de tipo *enunciado_completo* almacenados en *Azure AI Search*, identificando los enunciados más similares al documento de entrada.

A partir de estos resultados, se recuperan los criterios de evaluación asociados mediante filtrado exacto sobre los identificadores correspondientes, construyendo el contexto que será utilizado en la etapa de generación.

3. **Generación estructurada:** El modelo de lenguaje recibe el *prompt* enriquecido con el contexto recuperado y genera la rúbrica correspondiente. La generación se realiza con una temperatura de 0,3, lo que favorece la consistencia en la producción de criterios y niveles de desempeño. La salida se estructura en formato JSON, incluyendo dimensiones de evaluación, descriptores y puntajes asociados.

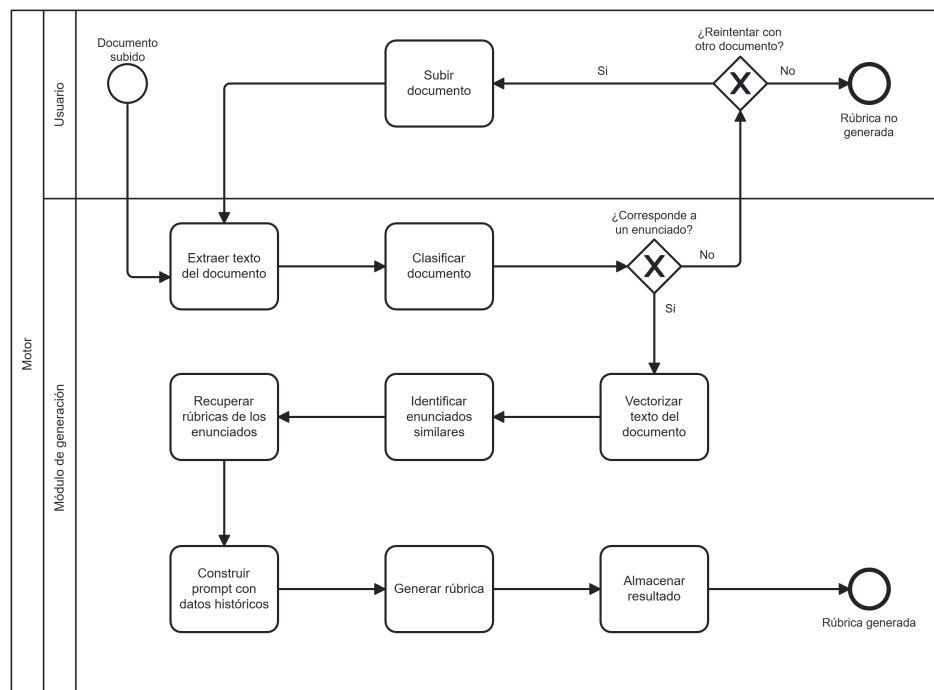


Figura 3: Diagrama de flujo del proceso de generación de rúbrica

Fuente: Elaboración propia.

3.2.3. Ejemplo de Rúbrica Generada

La Figura 4 muestra la salida obtenida al procesar un enunciado perteneciente a una tarea del curso. La rúbrica generada incluye los criterios de evaluación identificados, los niveles de desempeño con sus respectivos descriptores y los puntajes asociados a cada nivel.

Este resultado es presentado a través de la interfaz web del sistema como producto final del proceso de generación automática.

Tarea 4 (1).pdf

DESCARGAR

PARAMETROS

EJEMPLOS A INCLUIR EN EL PROMPT

5

MODELO LLM

GPT-4o

UNIDADES EVALUADAS

☒ U1 — Programas Secuenciales

☒ U2 — Condicionales

☒ U3 — Ciclos

☒ U4 — Strings

☒ U5 — Funciones

☒ U6 — Listas

☒ U7 — Diccionarios

☒ U8 — Procesamiento de Texto y Archivos

Generar rúbrica

Limpiar

ANÁLISIS DE PEDIDOS DE DELIVERY

8 criterios · 100 pts · gpt-4o · 16,370 tokens · 16,8s

Tabla · JSON · Exportar Excel · Exportar imagen · Copiar JSON

CRITERIO	NIVEL 1	NIVEL 2	NIVEL 3
P1: Procesamiento de archivo de pedidos 10 pts	No se implementa la función o no se abre el archivo de pedidos. 0 puntos	La función abre el archivo, pero no procesa correctamente las líneas (ej. no omite el encabezado, no separa los campos correctamente, o no convierte el costo a entero). 5 puntos	La función abre el archivo de pedidos, omite el encabezado, procesa correctamente las líneas, separa los campos y retorna una lista de sublists con el identificador del pedido como string y el costo como entero. 10 puntos
P2: Retorno de la lista de pedidos 10 pts	No retorna una lista, o la lista no está relacionada con los datos del archivo. 0 puntos	Retorna una lista, pero contiene datos incorrectos (ej. incluye encabezado, el identificador no es string, o el costo no es entero). 5 puntos	Retorna una lista de sublists donde cada sublist contiene el identificador del pedido como string y el costo como entero, sin incluir el encabezado. 10 puntos
P3: Clasificación de pedidos por criterios 10 pts	No clasifica los pedidos en los criterios indicados o no utiliza la función de la pregunta 1. 0 puntos	Clasifica los pedidos en algunos criterios, pero con errores (ej. no considera correctamente el retraso, el reclamo o el costo promedio). 7 puntos	Clasifica los pedidos en los criterios 'Eficiente', 'Problemático' y 'Costoso' según las condiciones del enunciado, utilizando la función de la pregunta 1. 10 puntos
P4: Ordenar pedidos dentro de cada criterio 10 pts	No ordena los pedidos dentro de cada criterio, o pierde la relación entre los datos del pedido. 0 puntos	Ordena los pedidos dentro de cada criterio, pero no por costo de menor a mayor. 7 puntos	Ordena correctamente los pedidos dentro de cada criterio por costo de menor a mayor. 10 puntos
P5: Retorno del diccionario de desempeño 20 pts	No retorna un diccionario, o el diccionario no está relacionado con los criterios solicitados. 0 puntos	Retorna un diccionario, pero contiene errores en las listas de pedidos (ej. no incluye todos los datos requeridos o no respeta el formato indicado). 10 puntos	Retorna un diccionario donde cada llave es un criterio ('Eficiente', 'Problemático', 'Costoso') y su valor es una lista de pedidos ordenada por costo, con el formato [costo, nombre_producto, tiempo_estimado, tiempo_real, categoria]. 20 puntos
P6: Exclusión de categoría en los archivos generados 10 pts	No excluye los pedidos de la categoría indicada en el parámetro. 0 puntos	Excluye parcialmente los pedidos de la categoría indicada (ej. algunos pedidos de la categoría aparecen en los archivos). 5 puntos	Excluye correctamente todos los pedidos de la categoría indicada en el parámetro de los archivos generados. 10 puntos
P7: Generación de archivos de salida 10 pts	No genera los archivos de salida, o los nombres de los archivos no corresponden a los criterios ('Eficiente', 'Problemático', 'Costoso'). 0 puntos	Genera los archivos de salida, pero con errores en el contenido (ej. no incluye los 10 pedidos menos costosos, o el formato no coincide con el ejemplo). 5 puntos	Genera un archivo por cada criterio ('Eficiente', 'Problemático', 'Costoso'), con los 10 pedidos menos costosos que no pertenecen a la categoría excluida, siguiendo estrictamente el formato indicado en el enunciado. 10 puntos
P8: Retorno de la cantidad total de pedidos 10 pts	No retorna la cantidad total de pedidos en los criterios, o el valor retornado no está relacionado con los datos procesados. 0 puntos	Retorna la cantidad total de pedidos, pero con errores en el conteo (ej. incluye pedidos de la categoría excluida). 5 puntos	Retorna correctamente la cantidad total de pedidos en los criterios, incluyendo aquellos no escritos en los archivos debido a la exclusión por categoría o límite de 10 pedidos. 10 puntos

Figura 4: Rúbrica generada automáticamente a partir del enunciado de la Tarea 4 del semestre 2026-1.

Fuente: Elaboración propia.

3.3. Evaluación y retroalimentación automatizada

El Módulo de evaluación y retroalimentación permite aplicar una rúbrica previamente generada sobre las soluciones entregadas por los estudiantes. Su ejecución depende de la existencia de una rúbrica asociada a la evaluación correspondiente, la cual estructura el proceso de análisis de las entregas.

A partir de la rúbrica y la entrega del estudiante, el módulo ejecuta el proceso, integrando múltiples fuentes de información para generar la calificación y la retroalimentación asociada.

3.3.1. Pipeline de Evaluación

El proceso de evaluación se estructura en seis etapas secuenciales que el sistema ejecuta de forma automática a partir de la recepción del archivo de entrega. Previo a estas etapas, el sistema valida que el archivo corresponda a código ejecutable con contenido no vacío, en caso contrario, se le permite al usuario reintentar la carga de la entrega. El flujo completo se ilustra en la Figura 5:

1. **Ejecución de la entrega:** Cuando el archivo corresponde a código Python, el sistema lo ejecuta de manera automática en un entorno aislado, junto con los eventuales archivos de apoyo adjuntados por el estudiante. De esta ejecución se obtiene la salida producida por el programa o en caso de existir, los errores generados, lo que permite contrastar posteriormente el comportamiento observado con lo declarado en el enunciado.
2. **Recuperación de la rúbrica:** A partir del identificador de la rúbrica seleccionada, el sistema consulta la base de datos y extrae el enunciado original de la entrega, los criterios de evaluación con sus respectivos niveles de desempeño y las unidades temáticas asociadas.
3. **Recuperación de plantillas de *prompt*:** El sistema obtiene las plantillas de *prompt* definidas para el proceso de evaluación, las cuales estructuran la interacción con el LLM.
4. **Recuperación de directrices académicas:** El sistema identifica las unidades temáticas vinculadas a la rúbrica y recupera el material normativo de la asignatura correspondiente desde la base de datos, con el fin de que el modelo cuente con el contexto necesario para distinguir entre los elementos que el estudiante puede emplear y aquellos que exceden el alcance de la unidad evaluada, tal como se describe en la Sección 3.3.2.
5. **Construcción del *prompt* e inferencia:** Una vez recopilada la información de las etapas anteriores: enunciado, rúbrica, entrega del estudiante, directrices académicas y resultado de la ejecución del código, esta es ingresada al *prompt* en un único mensaje que se envía al modelo de lenguaje.
6. **Persistencia del resultado:** El resultado de la evaluación es almacenado en la base de datos, incluyendo el archivo original del estudiante, la nota asignada, el detalle por criterio, la retroalimentación general y las plantillas utilizadas al momento de la evaluación. Este registro permite mantener una trazabilidad completa sobre cada evaluación realizada.

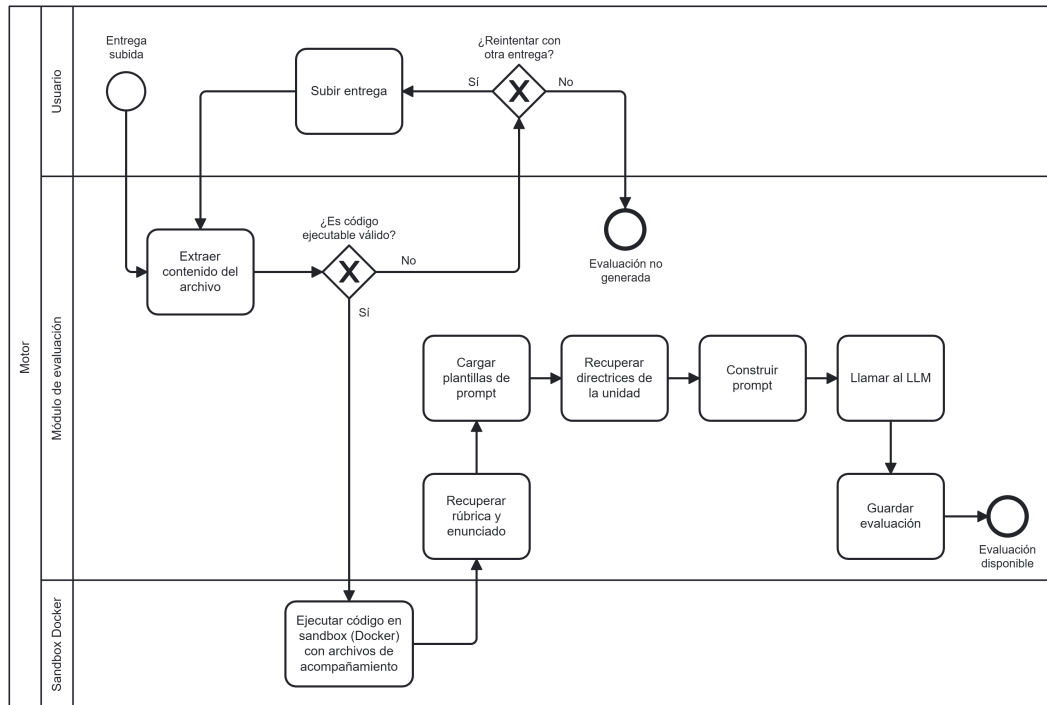


Figura 5: Diagrama de flujo del proceso de evaluación.
Fuente: Elaboración propia.

3.3.2. Integración de restricciones académicas

A diferencia de un evaluador basado exclusivamente en criterios generales, este módulo integra el documento *Objetivos y Materia de las UVAs* utilizado por el cuerpo docente dentro de la asignatura Introducción a la Programación, el cual actúa como base de conocimiento normativa que define las restricciones técnicas permitidas para cada unidad temática del curso.

- **Validación normativa por unidad:** El sistema identifica la unidad correspondiente a la evaluación y extrae las directrices específicas de lo que el estudiante puede y no puede utilizar.
- **Inyección dinámica en el *prompt*:** Estas restricciones se inyectan directamente en la plantilla de evaluación. Esto instruye al modelo de lenguaje para penalizar soluciones que, a pesar de ser funcionalmente correctas, no cumplen con las limitaciones pedagógicas impuestas para el nivel actual del estudiante.

3.3.3. Evaluación y generación de retroalimentación

El motor de evaluación integra cinco fuentes de información: el enunciado, la rúbrica seleccionada, la solución del estudiante, la salida de ejecución del código y el material normativo asociado a la unidad correspondiente. A partir de estos insumos, se realiza el proceso de evaluación descrito a continuación:

- **Mapeo de criterios y restricciones:** El modelo de lenguaje analiza la solución buscando patrones que correspondan a los descriptores de la rúbrica, contrastándolos al mismo tiempo con las restricciones académicas de la unidad.
- **Justificación de calificación basada en evidencia:** Por cada criterio, el sistema genera una explicación textual que cita partes específicas del trabajo del estudiante. Si se detecta el uso de elementos no permitidos por las directrices de la unidad, el sistema lo señala explícitamente como una desviación normativa.
- **Retroalimentación orientada a la mejora:** Se produce un *feedback* constructivo que orienta al estudiante sobre cómo alcanzar el nivel superior de desempeño, respetando las limitaciones técnicas y los objetivos de aprendizaje de la unidad actual.

3.3.4. Ejemplo de evaluación realizada

La Figura 6 ilustra el resultado de aplicar el módulo sobre una entrega real de un estudiante. Se observa la nota asignada, la justificación por criterio basada en evidencia del código entregado y la retroalimentación general generada por el sistema.

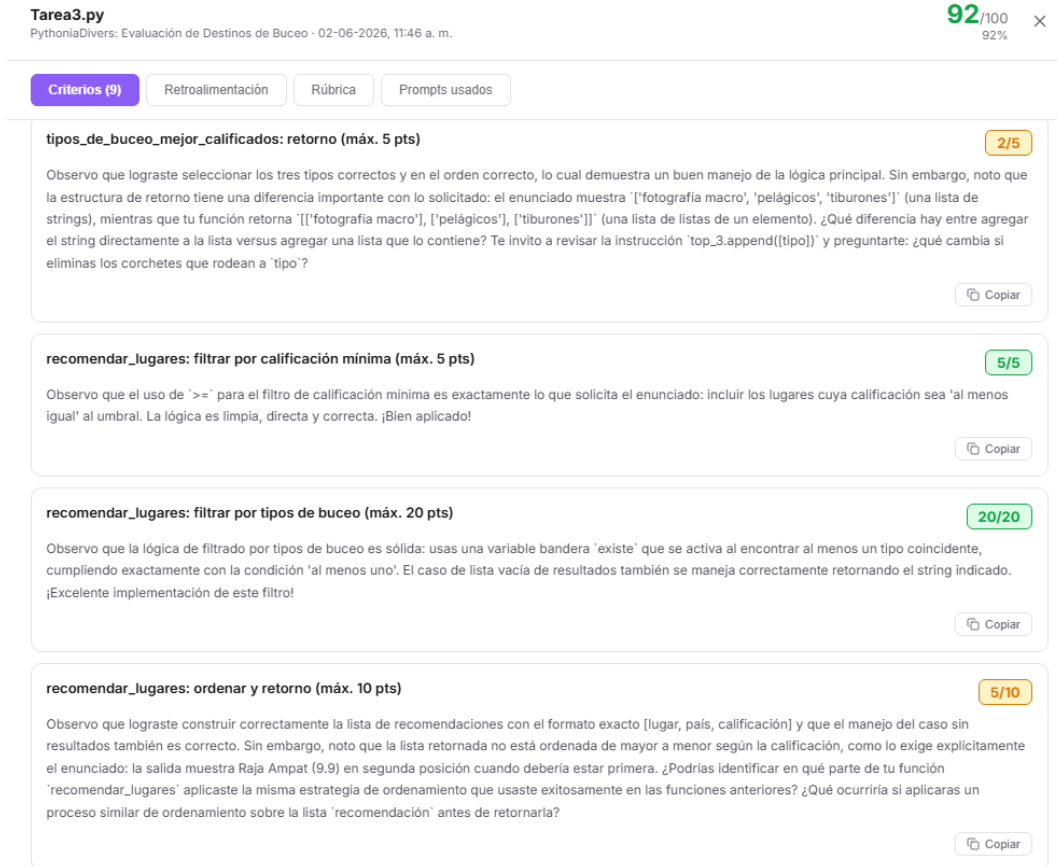


Figura 6: Ejemplo de evaluación sobre la entrega de un estudiante para la Tarea 3 semestre 2026-1.

Fuente: Elaboración propia.

3.4. Estrategia de *Prompt Engineering*

La generación de rúbricas en el sistema se basa en una estrategia de *prompt engineering* implementada mediante plantillas dinámicas almacenadas en una base de datos. Este enfoque permite modificar el comportamiento del modelo sin intervenir el código fuente, facilitando la adaptación del proceso de generación a distintos tipos de evaluaciones.

Para garantizar la validez metodológica de estas plantillas, su diseño de composición modular por capas se alinea de forma sistemática con las ocho dimensiones del manual de calidad de prompts de Jacobsen y Weber [Jacobsen y Weber, 2025] expuesto en el marco conceptual. Esta correspondencia se articula de la siguiente manera:

- **Capa de rol:** Implementa las dimensiones de rol, audiencia objetivo, medio o canal y especificidad de dominio, configurando de forma precisa la identidad del evaluador y

el lenguaje técnico.

- **Capa de instrucción de tarea y contexto:** Integra la misión y la consistencia lógica mediante el uso de datos contextuales (RAG), ejemplos estructurados de pocas pasadas (*few-shot prompting*) o plantillas normativas que restringen la simulación de ejecución.
- **Capa de formato:** Aplica de manera rigurosa las restricciones de formato y concisión, forzando esquemas de datos estructurados e interpretables exclusivamente en formato JSON sin contenido transicional redundante.

Cada plantilla puede encontrarse en estado activo o inactivo. En caso de no existir una configuración activa, el sistema utiliza versiones por defecto, asegurando la continuidad operativa del módulo. La Figura 7 muestra la interfaz de configuración de plantillas de *prompt*, la cual permite la edición y activación dinámica de los distintos componentes utilizados en los procesos de generación y evaluación.

The screenshot shows a web interface titled 'Prompts' with an 'ADMIN' button in the top right. Below the title is a '+ Nuevo prompt' button. The interface is split into two columns. The left column, 'GENERACIÓN DE RÚBRICAS', contains three prompt cards: 'Rol del Asistente' (v10), 'Tarea y Contexto RAG' (v12), and 'Formato de Salida' (v18). The right column, 'EVALUACIÓN DE ENTREGAS', contains three prompt cards: 'Estilo del Evaluador' (v13), 'Instrucciones por Criterio' (v38), and 'Estructura de Salida' (v14). Each card shows a description, a status indicator (active/inactive), and a version number. Dropdown menus are present for each card to select a different version.

Figura 7: Vista para configuración de prompts.

Fuente: Elaboración propia.

3.4.1. Prompt generación de rúbricas

El *prompt* de generación de rúbricas implementa un enfoque de aprendizaje por ejemplos, en el cual la generación se condiciona mediante contexto recuperado desde el corpus histórico del curso. El *prompt* final se construye mediante la composición de tres capas secuenciales:

1. **Rol del asistente:** Define el comportamiento global del modelo, estableciendo su rol

como generador de rúbricas, el estilo de respuesta esperado y las restricciones generales de generación.

2. **Instrucción de tarea y contexto (RAG):** Corresponde a la definición de la tarea de generación e incorpora la información variable del problema. En esta capa se realiza la sustitución de los siguientes *placeholders*:

- **{contexto}**: Conjunto de enunciados históricos similares junto con sus respectivas rúbricas, recuperados mediante el módulo de búsqueda semántica del sistema (RAG).
- **{enunciado}**: Texto del nuevo ejercicio de programación extraído desde el documento PDF de entrada.

Este contexto permite anclar la generación a decisiones evaluativas previas del curso, asegurando coherencia con los criterios docentes históricos.

3. **Formato de salida:** Define el formato de salida esperado del modelo, especificado como una estructura JSON estricta que regula la organización de criterios, niveles de desempeño y puntajes asociados.

La construcción del *prompt* final se realiza mediante la sustitución de los *placeholders* en tiempo de ejecución, integrando las tres capas anteriores en un único mensaje enviado al modelo de lenguaje. A continuación, se presenta el *prompt* utilizado:

Rol del asistente

Eres un experto en diseño de instrumentos de evaluación para cursos introductorios de programación universitaria. Construyes rúbricas analíticas en JSON que evalúan la comprensión algorítmica del estudiante, no solo la corrección funcional del código. Diseñas para el curso Introducción a la programación en la UTFSM, estudiantes principiantes de Python.

Reglas que debes aplicar en cada nivel:

- Cita el enunciado, no lo parafrasees: menciona valores, condiciones, formatos, tipos de datos y nombres específicos que aparecen en él. Prohibido usar frases vacías como “según el enunciado” o “correctamente” sin explicar qué es lo correcto.
- Transcribe los strings literales: mensajes de error, opciones de menú, prompts de entrada o salidas entre comillas se copian exactos, incluyendo errores tipográficos o caracteres anómalos del enunciado.
- Nivel 2 con error plausible: describe un error de lógica, sintaxis o caso borde concreto y plausible de un estudiante novato, no simplemente “parcialmente correcto”.

- Nivel 3 verificable: describe el comportamiento exacto, con suficiente detalle para verificarlo leyendo el código o mapeando un test unitario, sin necesitar releer el enunciado.
- Dominio cerrado: procesa el {enunciado} nuevo aislado del {contexto}. Prohibido heredar entidades, variables, tipos de datos, penalizaciones o reglas de negocio de los laboratorios de contexto. Lo no escrito explícitamente en el {enunciado} actual no existe.
- Mapeo de entidades: identifica los tipos de datos de entrada (enteros, flotantes, strings) y las entidades del dominio del problema (estructuras, variables clave, estados); mapea cada restricción explícita a una condición de evaluación cuantitativa.
- Granularidad para automatización: diseña criterios evaluables de forma independiente. Si el enunciado describe múltiples validaciones, flujos alternativos o fases, sepáralos en criterios distintos en vez de fusionarlos en uno genérico.

Instrucción de tarea y contexto (RAG)

A continuación tienes ejemplos de rúbricas usadas en el curso junto a sus enunciados: {contexto}. Genera una rúbrica para el siguiente enunciado nuevo, siguiendo el estilo, cantidad de criterios y escala de puntaje de los ejemplos anteriores: {enunciado}.

Instrucciones de formato:

- Deriva los criterios directamente del enunciado: funcionalidades, validaciones, mensajes y comportamientos explícitamente pedidos.
- Progresión de niveles: desde ausente o incorrecto (nivel 1), pasando por parcialmente correcto (nivel 2), hasta completo según el enunciado (nivel 3).

Formato de salida

Responde únicamente con un JSON válido, sin texto adicional, sin bloques de código, sin explicaciones, que contenga los siguientes elementos:

- nombre_lab: nombre descriptivo del laboratorio.
- distribucion_puntajes: un objeto que asigna a cada pregunta o sección del enunciado (identificada como P1, P2, P3, ...) un puntaje entero, de modo que la suma total sea exactamente 100.
- criterios: una lista de criterios de evaluación, donde cada uno contiene:
 - criterio: nombre del criterio, que debe comenzar con el identificador de su pregunta correspondiente (ej. "P1 ...").

- **puntaje_maximo:** puntaje entero asignado al criterio.
- **niveles:** exactamente 3 niveles de desempeño, cada uno con su número de nivel, puntaje y descripción:
 - Nivel 1: 0 puntos (comprensión mínima o ausente).
 - Nivel 2: puntaje parcial (comprensión parcial, con un error basado en el enunciado).
 - Nivel 3: puntaje completo (comprensión completa y exacta según las reglas explícitas del enunciado).
- **puntaje_total:** puntaje entero, igual a la suma de todos los **puntaje_maximo** de criterios.

Procedimiento obligatorio (en este orden):

1. Identifica las preguntas o secciones del enunciado (P1, P2, P3, ...) y asígnales un puntaje entero: si el enunciado indica pesos porcentuales, úsalos directamente ($20\% = 20$ puntos) si no, defínelos según la complejidad relativa de cada pregunta. Escribe esta asignación en **distribucion_puntajes** antes de generar criterios, su suma debe ser 100.
2. Genera los criterios de cada pregunta, de modo que la suma de sus **puntaje_maximo** sea exactamente igual al valor asignado a esa pregunta en **distribucion_puntajes**. No omitas ninguna pregunta del enunciado ni generes criterios para preguntas inexistentes.

3.4.2. Prompt de evaluación

El *prompt* de evaluación de entregas reutiliza la estructura de composición por capas utilizada en la generación de rúbricas. Sin embargo, en este caso se reemplaza el componente de recuperación semántica (RAG) por evidencia empírica proveniente de la ejecución del código del estudiante y por el material normativo asociado a la unidad correspondiente. El *prompt* final se construye mediante la composición de tres capas secuenciales:

1. **Rol del evaluador:** Define el comportamiento global del modelo, estableciendo su rol como evaluador académico experto en programación introductoria, el tono de evaluación esperado (objetivo, constructivo y académico) y la necesidad de justificar cada puntaje con evidencia observable de la entrega.
2. **Instrucción de tarea y material normativo:** Corresponde a la definición de la tarea de evaluación e incorpora información contextual asociada a la unidad del curso. En esta capa se realiza la sustitución del siguiente *placeholder*:
 - **{material}**: Restricciones y lineamientos técnicos de la unidad, indicando los elementos permitidos y no permitidos según los objetivos formativos.

Esta capa instruye al modelo a evaluar cada criterio de forma independiente, justificar los puntajes con evidencia específica y penalizar el uso de elementos no permitidos según la unidad.

3. **Contexto de evaluación y formato de salida:** Define los elementos necesarios para la evaluación y el formato de salida esperado. En esta capa se realiza la sustitución de los siguientes *placeholders*:

- **{enunciado}**: Enunciado de la tarea asociado a la rúbrica seleccionada.
- **{rubrica}**: Criterios de evaluación con niveles de desempeño y puntajes.
- **{entrega}**: Código fuente o solución del estudiante.
- **{bloque_ejecucion}**: Resultado de la ejecución del código en un entorno aislado, incluyendo salida estándar, errores y archivos generados.

El formato de salida se define como una estructura JSON estricta que incluye la nota final, el desglose por criterio con puntaje obtenido, puntaje máximo y justificación, además de una retroalimentación general.

La construcción del *prompt* final se realiza mediante la sustitución de los *placeholders* en tiempo de ejecución, integrando las tres capas anteriores en un único mensaje enviado al modelo de lenguaje. En este caso, la generación no depende de recuperación semántica, sino de evidencia directa derivada de la ejecución del código y del material normativo de la unidad. Al igual que en el *prompt* de generación de rúbricas, se eliminaron redundancias instructivas entre capas (por ejemplo, la prohibición de revelar código corregido o nombrar métodos nativos, y la validación de puntajes según los niveles de la rúbrica), consolidando cada regla en la capa donde corresponde. A continuación, se presenta el *system prompt* utilizado para configurar al evaluador:

Rol del evaluador

Eres un evaluador académico de un curso introductorio de programación en Python. Tu única función es aplicar la rúbrica proporcionada de forma estricta, literal y matemática.

Reglas que debes aplicar:

- Interpreta los criterios de la rúbrica en conjunto con el enunciado, no de forma aislada: descriptores como “correctamente”, “según lo solicitado” o “errores en el procedimiento” se definen por las condiciones, restricciones y valores que el enunciado establece explícitamente.
- Evalúa solo lo que la rúbrica y el enunciado piden. No asumas, inventes ni penalices requerimientos o comportamientos que no estén escritos explícitamente en ellos.

- El puntaje asignado debe ser exactamente uno de los valores numéricos definidos en los niveles del criterio; no generes puntajes intermedios.

La retroalimentación se escribe en primera persona, con tono constructivo (ej: “Observo que...”, “Te sugiero que...”), siguiendo este orden:

1. Reconoce el esfuerzo lógico y los aciertos visibles en el código del estudiante.
2. Explica de forma específica en qué falló la lógica o el formato, contrastando el código contra el requerimiento del enunciado.
3. Orienta la mejora con pistas conceptuales o preguntas socráticas, sin revelar la solución, nombrar funciones o métodos correctores, ni escribir código corregido.

Instrucción de tarea y material normativo

Restricciones de la unidad: {material}.

Para cada criterio de la rúbrica, sigue estos pasos en orden estricto antes de asignar un puntaje:

Paso 1 — Identificar niveles y puntajes válidos. Anota los puntajes exactos de los niveles del criterio (ej: 0, 7, 15). Solo puedes asignar uno de esos valores; no se permiten aproximaciones.

Paso 2 — Triangular y simular. Localiza la instrucción exacta del enunciado que regula este criterio, y contrástala con la lógica interna del código del estudiante (no con sus comentarios ni con la estructura visual del retorno). Luego simula mentalmente la ejecución con datos de prueba que tú mismo formules, desordenados o complejos según el enunciado, y sigue el flujo paso a paso:

- Si el fragmento de código no muestra textualmente la lógica solicitada, el criterio se considera no logrado; no asumas intenciones ni “buen camino” si el algoritmo no está escrito.
- Si el código produce el resultado correcto solo porque el caso de prueba es simple o venía ordenado de fábrica, pero la lógica interna es incorrecta o inexistente para el caso general, no otorgues el nivel máximo: la lógica manda sobre el resultado cosmético.
- Si la lógica no coincide con lo pedido, el algoritmo falla en su objetivo, o el formato de salida no es idéntico al solicitado, asigna obligatoriamente el nivel inferior correspondiente.

Paso 3 — Aplicar restricciones de la unidad. Verifica si el estudiante usó algún elemento prohibido del bloque {material}. Aplica la penalización solo si el elemento aparece textualmente en el código entregado y regístralo en la retroalimentación general.

Paso 4 — Construir el comentario del criterio. El comentario debe ser específico (usa terminología concreta del código del estudiante variables, funciones y explica qué requerimiento del enunciado falló en la lógica) y empático (redactado en primera persona, valorando el esfuerzo lógico visible antes de señalar la falencia). No nombres funciones nativas, métodos, palabras clave o soluciones que resuelvan el error directamente, ni escribas código corregido, usa preguntas socráticas para que el estudiante note la discrepancia por sí mismo.

Contexto de evaluación y formato de salida

Usa las siguientes variables para el análisis:

Enunciado del ejercicio: {enunciado}. Rúbrica de evaluación: {rubrica}. Entrega del estudiante: {entrega}. Salidas del código del estudiante: {bloque_ejecucion}.

Antes de responder, verifica de forma obligatoria:

1. El número de bloques de criterios en el JSON es exactamente igual al número de criterios de la rúbrica.
2. El valor del campo nota es la suma aritmética exacta de todos los puntaje_obtenido.

Genera únicamente un JSON válido, que contenga los siguientes elementos:

- nota: suma exacta de todos los puntaje_obtenido.
- criterios: una lista con un bloque por cada criterio de la rúbrica, donde cada uno contiene:
 - criterio: nombre exacto del criterio según la rúbrica.
 - analisis_de_ejecucion: un bloque con el requerimiento exacto del enunciado para ese criterio, la porción de código del estudiante que lo implementa (o “ausente” si no existe), la simulación paso a paso con datos de prueba desordenados, y el veredicto de la simulación (“lógica correcta” o “lógica ausente/incorrecta”, según el Paso 2).
 - puntaje_obtenido: número entero o decimal exacto que representa el nivel asignado según la rúbrica; si el veredicto de la simulación fue “lógica ausente/incorrecta”, el puntaje debe ser obligatoriamente el de nivel inferior o no logrado.
 - puntaje_maximo: puntaje máximo de ese criterio.
 - comentario: comentario en primera persona según los criterios definidos en el Paso 4.

- `retroalimentacion_general`: párrafo constructivo en primera persona dirigido al estudiante, resumiendo sus fortalezas lógicas y áreas de crecimiento conceptual, sin dar soluciones explícitamente.

3.5. Metodología de implementación

La construcción del sistema se realizó mediante un enfoque incremental, en el cual cada etapa se apoya sobre los resultados de la anterior, permitiendo avanzar desde la preparación del corpus histórico hasta la implementación de la interfaz web utilizada por el equipo docente.

1. **Recopilación y preparación del corpus:** Se realizó la selección y organización de los enunciados y rúbricas históricas del curso. En esta etapa se definió el esquema de metadatos que vincula cada enunciado con su respectiva rúbrica y semestre. Adicionalmente, los documentos en formato PDF fueron procesados mediante *Azure Document Intelligence*, obteniendo su representación textual estructurada.
2. **Construcción del índice vectorial:** Sobre el corpus preparado se ejecutó el proceso de segmentación e indexación. Se configuró un índice basado en HNSW en *Azure AI Search*, incorporando campos filtrables que permiten la recuperación en dos etapas definida por el sistema.
3. **Implementación del módulo de generación:** Se desarrolló el *pipeline* de generación de rúbricas basado en RAG, integrando la extracción de documentos y la recuperación semántica de enunciados similares. Este módulo fue expuesto mediante un *endpoint* REST para su consumo desde la interfaz.
4. **Implementación del módulo de evaluación:** Se implementó el *pipeline* de evaluación de entregas, incorporando la inyección dinámica de directrices académicas asociadas a cada unidad. Su funcionalidad fue expuesta mediante *endpoints* REST para evaluación y consulta de resultados.
5. **Desarrollo de la interfaz web:** Se construyó una aplicación en *React* que permite la gestión de rúbricas, evaluaciones, corpus histórico, plantillas de *prompts* y unidades temáticas, proporcionando una capa de interacción para el equipo docente.
6. **Ajuste iterativo de *prompts*:** Se diseñaron y refinaron las plantillas del sistema mediante pruebas iterativas con enunciados y entregas reales del curso, ajustando tanto la estructura como los parámetros de generación para mejorar la calidad de las respuestas.

El proceso descrito permitió integrar progresivamente los distintos componentes del sistema, asegurando consistencia entre la infraestructura de datos, los modelos de lenguaje y la interfaz de usuario final.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se presenta la validación de la solución propuesta, evaluando su desempeño en actividades reales de la asignatura Introducción a la Programación durante el semestre 2026-1. La validación se orienta a verificar el cumplimiento de los objetivos específicos definidos en el Capítulo 1 mediante un análisis cuantitativo y cualitativo del sistema.

La evaluación se estructura en tres dimensiones complementarias, correspondientes a la eficiencia del sistema, la calidad de las rúbricas generadas y la percepción de los estudiantes sobre la retroalimentación generada. Estas dimensiones permiten analizar la solución desde una perspectiva técnica y pedagógica.

La validación se llevó a cabo utilizando la totalidad de las actividades evaluativas programadas para la asignatura durante el semestre (Tareas 1, 2, 3 y 4), aplicando distintos procedimientos según la dimensión analizada.

En primer lugar, la eficiencia del sistema se evaluó de manera transversal utilizando los enunciados de las cuatro tareas, obteniéndose métricas globales de rendimiento arquitectónico.

En segundo lugar, la calidad de las rúbricas y la pertinencia del proceso de evaluación fueron analizadas a partir del material generado automáticamente para la Tarea 2, el cual fue sometido a revisión por parte del equipo docente mediante una encuesta de expertos.

En tercer lugar, la percepción de los estudiantes fue evaluada en las Tareas 3 y 4, donde estudiantes de los paralelos P212L y P215L participaron voluntariamente enviando sus soluciones al sistema y respondiendo posteriormente un cuestionario de percepción sobre los comentarios generados por el motor.

4.1. Alcance y limitaciones de la evaluación

Considerando los procedimientos descritos, es necesario señalar que la validación ejecutada posee un carácter preliminar y exploratorio. Esta condición responde a las restricciones contextuales propias del periodo académico, donde tanto el cuerpo docente como el estudiantado enfrentan una alta carga de responsabilidades institucionales dentro de los plazos acotados del semestre 2026-1. Esta disponibilidad restringida de tiempo influyó directamente en la naturaleza voluntaria de la participación en las pruebas y cuestionarios.

Consecuentemente, el tamaño de la muestra alcanzado, centrado en la participación voluntaria de un grupo acotado de ocho docentes y ocho estudiantes, restringe la aplicación de pruebas estadísticas inferenciales que pretendan generalizar los resultados a toda la pobla-

ción estudiantil o establecer relaciones de causalidad aisladas. Por esta razón, el análisis de los datos recolectados se aborda desde una perspectiva principalmente descriptiva para las métricas cuantitativas como los tiempos de procesamiento y los *tokens* consumidos y de análisis de tendencias para los datos de percepción cualitativa.

Los hallazgos presentados deben interpretarse, por lo tanto, como indicadores tempranos del comportamiento del sistema y como un insumo valioso para identificar vectores de mejora en la arquitectura del motor de evaluación. Lejos de buscar conclusiones definitivas sobre el impacto directo en el rendimiento académico global, esta sección se constituye como una primera aproximación empírica que valida la viabilidad técnica y pedagógica de la solución, sentando bases metodológicas para futuras iteraciones con grupos de estudiantes más numerosos.

4.2. Eficiencia del motor

Con el propósito de evaluar la estabilidad operativa y el rendimiento técnico del sistema, se diseñó un protocolo de pruebas controlado utilizando los enunciados y soluciones de las cuatro tareas programadas durante el semestre 2026-1. El experimento contempló la ejecución de cinco iteraciones consecutivas para cada actividad evaluativa. Esta replicación responde a la necesidad de mitigar sesgos por latencia de red y capturar la variabilidad inherente al comportamiento estocástico de los modelos de lenguaje externos. A partir de esta muestra, se calculó la media aritmética y la desviación estándar tanto para los tiempos de procesamiento como para el consumo de recursos informáticos.

Los resultados consolidados de latencia temporal para los procesos de generación de rúbricas y evaluación cualitativa de entregas se presentan en la Tabla 2.

Tabla 2: Métricas de tiempo del motor de software por instancia evaluativa (5 ejecuciones por tarea). Fuente: Elaboración propia.

Evaluación	Generación de Rúbricas		Evaluación y Feedback	
	Tiempo prom. (s)	Desviación (s)	Tiempo prom. (s)	Desviación (s)
Tarea 1	16,27	0,71	11,77	0,58
Tarea 2	25,41	4,91	12,36	0,61
Tarea 3	22,41	1,94	13,86	0,41
Tarea 4	26,16	5,02	14,61	1,26
Promedio	22,56	—	13,15	—

Los resultados evidencian que el sistema opera bajo márgenes temporales consistentes y acotados. El tiempo promedio general de 22,56 segundos para la construcción de rúbricas asegura un flujo de trabajo eficiente desde la perspectiva del usuario final. Este rendimiento técnico representa una optimización significativa respecto al proceso tradicional de cons-

trucción manual de instrumentos evaluativos, el cual suele requerir entre 3 y 4 horas de dedicación estimada por parte del equipo docente, según la complejidad y objetivos pedagógicos del enunciado².

En cuanto a la variabilidad temporal expuesta en la Tabla 2 para la generación, la Tarea 1 presenta la desviación estándar más reducida (0,71 s), reflejando un comportamiento altamente estable entre ejecuciones. Por otro lado, las Tareas 2 y 4 exhiben las mayores dispersiones (4,91 s y 5,02 s, respectivamente). Este fenómeno es atribuible principalmente a la naturaleza del consumo de servicios externos de Inteligencia Artificial como servicio (AlaaS), donde la concurrencia global en los servidores remotos introduce fluctuaciones de latencia fuera del control de la arquitectura local. No obstante, incluso en los escenarios de mayor dispersión registrados, los tiempos máximos se mantienen dentro de rangos plenamente tolerables para una interacción síncrona.

El módulo de evaluación y retroalimentación exhibe, en cambio, un comportamiento temporal considerablemente más predecible. El promedio general de 13,15 segundos se sitúa muy por debajo del observado en la generación de rúbricas, lo que resulta coherente con la naturaleza acotada de esta tarea, donde el modelo no debe diseñar un instrumento desde cero, sino contrastar evidencia de ejecución ya estructurada contra criterios previamente definidos. Asimismo, las desviaciones estándar del proceso de evaluación se mantienen sistemáticamente bajas (entre 0,41 s y 1,26 s), sin los aumentos de dispersión observados en la generación. Un hallazgo relevante es que el tiempo promedio de evaluación crece de manera monótona a través de las cuatro tareas (11,77 s, 12,36 s, 13,86 s y 14,61 s), a diferencia del comportamiento no monótono registrado en la generación de rúbricas. Esto sugiere que, al no depender de la creatividad estructural exigida por el diseño de instrumentos, el tiempo de evaluación responde de forma más directa y proporcional al aumento de complejidad del enunciado y del código a analizar.

Paralelamente, se recopilieron las métricas de volumen de información procesada para las mismas ejecuciones controladas, expresadas en *tokens* (unidades mínimas de texto, equivalentes a sílabas o fragmentos de palabras que el modelo procesa de forma numérica). El desglose cuantitativo de esta carga útil se detalla en la Tabla 3.

Tabla 3: Consumo de *tokens* por instancia evaluativa (5 ejecuciones por tarea). Fuente: Elaboración propia.

	Generación de Rúbricas		Evaluación y Feedback	
Evaluación	Tokens prom.	Desviación tokens	Tokens prom.	Desviación tokens
Tarea 1	15 078,6	76,0	8 017,6	105,4
Tarea 2	15 042,2	67,6	8 538,8	101,9
Tarea 3	16 708,2	36,0	9 663,8	213,2
Tarea 4	17 702,0	38,1	11 237,2	132,8
Promedio	16 132,8	—	9 364,4	—

²Estimación temporal proporcionada por Federico Meza, profesor de la asignatura.

Un hallazgo analítico clave dentro de la generación de rúbricas se deriva de los valores de desviación estándar expuestos en la Tabla 3, los cuales se presentan extremadamente acotados (oscilando entre 36,0 y 76,0 *tokens*), lo que representa menos del 0,5 % de la carga útil total. Esta mínima variabilidad matemática valida empíricamente la estabilidad y consistencia de los módulos intermedios de clasificación y recuperación semántica. El *pipeline* inyecta bloques de información idénticos en cada iteración controlada, restringiendo la dispersión de *tokens* únicamente a las sutiles diferencias sintácticas generadas por la naturaleza probabilística del modelo de lenguaje, al estructurar el texto de salida.

Respecto al consumo de *tokens* promedio, los datos demuestran que el volumen de información procesado se encuentra fuertemente influenciado por la extensión del enunciado base provisto por el docente y la densidad del contexto recuperado e indexado por el componente RAG. Esto explica por qué las Tareas 1 y 2 exhiben consumos prácticamente equivalentes (15 078,6 y 15 042,2 *tokens*, respectivamente), ya que la longitud del texto de sus especificaciones técnicas y los documentos de apoyo asociados en el *prompt* mantuvieron una volumetría similar.

Por otra parte, la introducción de problemas avanzados de programación en las Tareas 3 y 4 generó un incremento en la demanda de contexto, registrando promedios de 16 708,2 y 17 702,0 *tokens*, respectivamente. Este aumento se correlaciona con enunciados inherentemente más extensos, con mayor cantidad de restricciones explícitas y estructuras de datos más complejas que el sistema debió procesar para estructurar la rúbrica de manera pertinente.

El módulo de evaluación y retroalimentación reproduce esta misma tendencia, aunque con una dinámica particular. La desviación estándar de *tokens* en este proceso es proporcionalmente mayor que en la generación (entre 101,9 y 213,2 *tokens*). Dado que cada tarea utilizó una única entrega de referencia repetida en las cinco iteraciones, esta mayor dispersión no responde a variaciones en el código de entrada, sino a que el modelo cuenta con mayor libertad para redactar la retroalimentación de formas distintas en cada ejecución, a diferencia del *prompt* de generación, cuya salida está más acotada por la estructura fija de la rúbrica. Con lo anterior, el hallazgo más significativo es que el consumo de *tokens* en la evaluación también crece de forma monótona a través de las cuatro tareas (8 017,6, 8 538,8, 9 663,8 y 11 237,2 *tokens*), replicando el mismo patrón de escalamiento observado en la generación de rúbricas. Este comportamiento convergente entre ambos módulos permite evidenciar que la complejidad del enunciado y la complejidad del código de referencia utilizado constituyen el principal factor determinante del volumen de *tokens* procesado por el sistema, independientemente de la naturaleza específica de la tarea (generativa o evaluativa) que se esté ejecutando.

Con todo, el promedio general de 16 132,8 *tokens* consumidos en la generación y de 9 364,4 *tokens* en la evaluación representan, respectivamente, el 12,6 % y el 7,3 % de la ventana de contexto máxima admitida por el modelo GPT-4o (128 000 *tokens*). Esta holgura técnica en ambos módulos es un indicador positivo de escalabilidad, garantizando que el sistema cuenta con la capacidad de procesar especificaciones de proyectos institucionales de ingeniería

considerablemente más extensos, o entregas de código de mayor complejidad, sin riesgo de saturación o desborde de memoria.

En última instancia, considerando la tarifa oficial de OpenAI a julio de 2026 para el modelo GPT-4o y estableciendo como cota máxima el valor de 10,00 USD por millón de *tokens* de salida (*output*) aplicado a la totalidad de los *tokens*, el costo operativo asociado a la generación automática de una rúbrica se estima en aproximadamente 0,16 USD, mientras que el proceso individual de evaluación cualitativa y emisión de *feedback* para una entrega alcanza los 0,09 USD. De este modo, el costo total por ciclo completo de una entrega se sitúa en un máximo de 0,25 USD. Este bajo costo reduce significativamente las barreras operativas y viabiliza una futura adopción institucional de la plataforma en la rutina docente, constituyendo un antecedente favorable para la sustentabilidad financiera de la solución ante un eventual despliegue masivo.

4.3. Evaluación de calidad de las rúbricas y criterios usados

En el marco de la validación orientada a la identificación de criterios relevantes y dimensiones clave que deben ser consideradas en la evaluación de ejercicios de programación en cursos introductorios, se evaluó la calidad de las rúbricas generadas automáticamente por el sistema propuesto. Para ello, se utilizó la Tarea 2 del semestre 2026-1. A partir de su enunciado, el motor generó una rúbrica, la cual fue posteriormente entregada al equipo docente de la asignatura.

Con este material, se solicitó a los participantes la revisión de la rúbrica mediante la aplicación de un instrumento estructurado, el cual combinó ítems basados en una escala Likert de cinco niveles (donde 1 corresponde a *Totalmente en desacuerdo* y 5 a *Totalmente de acuerdo*) con preguntas de selección múltiple orientadas a la caracterización cualitativa de las dimensiones evaluadas. Específicamente, el instrumento abordó las siguientes dimensiones con las siguientes preguntas:

Dimensión 1: Cobertura del enunciado

Evalúa el grado en que la rúbrica automatizada abarca las directrices del problema pedagógico (Escala Likert 1–5):

- **Requisitos obligatorios:** “La rúbrica incluye todos los requisitos obligatorios y restricciones que se pedían en el enunciado.”
- **Casos de borde:** “La rúbrica incluye criterios para evaluar fallos lógicos o casos bordes propios del enunciado de la tarea.”

- **Coherencia pedagógica:** “Los criterios de evaluación son coherentes con las UVAs asociadas a la actividad.”

Dimensión 2: Precisión de los criterios

Analiza la objetividad y claridad con la que se redactaron los niveles de logro (Escala Likert 1–5):

- **Evidencia objetiva:** “Los descriptores se basan en evidencia comprobable en el código en lugar de conceptos subjetivos o ambiguos.”
- **Límites de desempeño:** “Es fácil distinguir el límite entre un nivel de desempeño y otro sin que quede a libre interpretación del corrector.”
- **Nivel de exigencia:** “El nivel de exigencia y los términos usados son apropiados para el semestre y conocimientos esperados en el estudiante.”

Dimensión 3: Diseño y estructura de criterios

Preguntas de selección múltiple orientadas a caracterizar la composición estructural de la rúbrica:

- **Cantidad de criterios:** “Sobre la cantidad de criterios generados (filas de la rúbrica): ¿Cómo califica el volumen de ítems evaluados en relación con la complejidad de la tarea?” (Opciones: *Tiene demasiados criterios (está sobrecargada)* / *Tiene más de lo necesario, pero es manejable* / *La cantidad de criterios es la adecuada (ideal)* / *Faltan algunos criterios importantes* / *Faltan muchos criterios (es muy básica)*).
- **Orden lógico:** “¿Considera que los criterios en la rúbrica siguen un orden lógico y coherente con el enunciado de la tarea?” (Opciones: *Sí, todos los criterios están organizados de manera lógica y fácil de seguir* / *Mayormente sí, pero algunos criterios podrían reorganizarse para mayor claridad* / *Parcialmente, la organización es confusa en varias secciones* / *No, los criterios están desordenados y no siguen un flujo lógico*).
- **Solapamiento:** “¿Detectó criterios que evalúan lo mismo o que penalizan dos veces un mismo error?” (Opciones: *los criterios están claramente separados y miden aspectos independientes* / *un solapamiento menor, pero que no altera significativamente la nota* / *Hay solapamiento evidente en varios puntos* / *Hay mucho solapamiento (los criterios son redundantes)*).
- **Distribución de puntaje:** “¿Considera que el puntaje asignado a cada criterio es acorde a su importancia?” (Opciones: *Sí, los puntajes son proporcionales* / *No, hay criterios*

con puntaje excesivo / No, hay criterios con puntaje muy bajo / No, la distribución es irregular).

Dimensión 4: Valoración y adopción

Reactivos enfocados en la utilidad percibida y la intención de uso de la herramienta tecnológica dentro del aula:

- **Integración en el curso:** “¿Estaría dispuesto(a) a integrar esta rúbrica dentro del proceso de evaluación de la asignatura 'Introducción a la programación'?” (*Opciones: Sí, la usaría tal cual para calificar a los estudiantes / Sí, aunque requeriría ajustes menores o intervención manual antes de su aplicación / La utilizaría únicamente como una referencia o base inicial para elaborar una rúbrica propia / No la usaría bajo ninguna circunstancia*).
- **Valor aportado:** “¿Cuánto valor aporta a su labor docente el poder generar rúbricas automatizadas en 1 minuto para optimizar sus tiempos de diseño evaluativo?” (*Escala Likert 1-5, desde 1: Sin valor hasta 5: Alto valor*).
- **Desempeño técnico:** “¿Cómo califica el desempeño técnico del motor como herramienta de apoyo docente para la asignatura?” (*Escala Likert 1-5, desde 1: Muy deficiente hasta 5: Excelente*).

El cuestionario fue respondido por ocho integrantes del equipo docente, incluyendo profesores y ayudantes. A continuación, se presentan los resultados organizados según las cuatro dimensiones del instrumento, priorizando una lectura por dimensión. El detalle individual de cada ítem, con su distribución completa de respuestas, se presenta en el Anexo.

Para facilitar la interpretación de los promedios reportados a continuación, se recuerda que la escala utilizada en los ítems de Cobertura y Precisión va de 1 (*Totalmente en desacuerdo*) a 5 (*Totalmente de acuerdo*), por lo que valores cercanos a 5 indican un alto nivel de acuerdo docente con la afirmación evaluada, valores cercanos a 3 reflejan una posición neutra, y valores cercanos a 1 indicarían desacuerdo. En los ítems numéricos de Adopción (valor de la automatización y desempeño técnico), la escala mantiene el mismo rango 1-5, pero con descriptores propios para cada pregunta (de *Sin valor* a *Alto valor*, y de *Muy deficiente* a *Excelente*, respectivamente), conservando la misma lógica. De este modo, a mayor puntaje, mayor valoración positiva.

4.3.1. Cobertura del enunciado

Con el fin de evaluar si el sistema identifica correctamente los criterios relevantes para calificar ejercicios de programación, se consultó al cuerpo docente sobre la cobertura de los

requisitos obligatorios, la inclusión de fallos lógicos y casos borde, y la coherencia de los criterios con las Unidades Virtuales de Aprendizaje (UVAs) asociadas a la actividad.

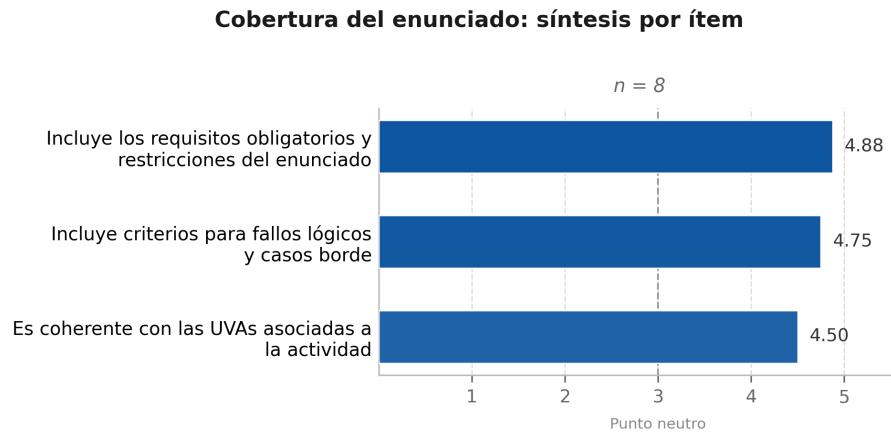


Figura 8: Promedios de valoración docente en la dimensión de Cobertura del Enunciado (Escala Likert 1-5).

Fuente: Elaboración propia

La Figura 8 resume los resultados obtenidos en esta dimensión. Los promedios calculados se mantuvieron sólidamente sobre los 4,50 puntos, alcanzando el valor más alto en la inclusión de requisitos obligatorios (4,88) y el más bajo en la coherencia con las UVAs de la asignatura (4,50). Estos puntajes sugieren que el sistema logra una alineación adecuada con el material original del curso, respaldando la percepción de una alta consistencia para traducir las restricciones técnicas y los casos borde del enunciado. Asimismo, el menor promedio obtenido en las UVAs sugiere que la ponderación automatizada de criterios podría presentar leves discrepancias con la intencionalidad pedagógica estimada por el equipo docente para esa unidad en particular.

4.3.2. Precisión de los descriptores

Para evaluar si los criterios generados son suficientemente precisos para ser aplicados de forma consistente por distintos correctores, se consultó al cuerpo docente sobre la fundamentación en evidencia comprobable en el código, la claridad de los límites entre niveles de desempeño y la pertinencia del nivel de exigencia técnica respecto al perfil del estudiante.

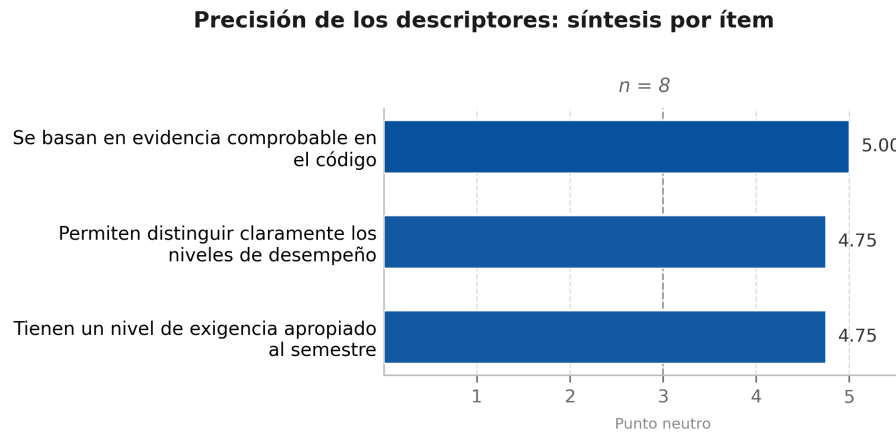


Figura 9: Promedios de valoración docente en la dimensión de Precisión de los Descriptores (Escala Likert 1-5).

Fuente: Elaboración propia.

La Figura 9 resume la evaluación de la Precisión de los descriptores. Destaca la puntuación máxima (5,00) en el ítem que evalúa si los descriptores se basan en evidencia comprobable, lo que es consistente con el diseño de la ingeniería de *prompts*, orientada a que el modelo evalúe únicamente elementos lógicos y sintácticos visibles en el código fuente. Este resultado respalda la percepción de objetividad del instrumento por parte del cuerpo docente, condición favorable para su aplicación consistente entre distintos correctores. Por su parte, la claridad en los límites entre niveles de logro en los descriptores y la pertinencia de la exigencia registraron un promedio de 4,75 puntos, lo que sugiere que el prototipo entrega rúbricas con límites percibidos como explícitos y adecuados para estudiantes de primer año.

4.3.3. Estructura de la rúbrica

Con el objetivo de evaluar el diseño y la organización de la rúbrica generada automáticamente, se consultó al cuerpo docente sobre la cantidad de criterios, la independencia entre ellos (ausencia de solapamiento) y la proporcionalidad en la distribución del puntaje. Al tratarse de ítems cualitativos con alternativas de texto exclusivas, los resultados iniciales se presentan mediante una tabla de síntesis de frecuencias (Tabla 4).

Tabla 4: Síntesis de hallazgos del cuerpo docente respecto al diseño y estructura de la rúbrica.
Fuente: Elaboración propia.

Ítem	Hallazgo principal	Respaldo
Cantidad de criterios	La cantidad de criterios generados es la adecuada para la complejidad de la tarea	7 de 8 (87,5 %)
Independencia	Los criterios están claramente separados y no penalizan dos veces un mismo error	5 de 8 (62,5 %)
Distribución de puntaje	El puntaje asignado a cada criterio es proporcional a su importancia	5 de 8 (62,5 %)

La Tabla 4 sugiere que el módulo de generación equilibra de forma adecuada la granularidad de los criterios y pondera de manera pertinente su importancia relativa. Un 87,5 % del equipo docente respalda la densidad de la rúbrica, lo que evita la fragmentación excesiva del instrumento.

En cuanto a la independencia entre criterios, un 62,5 % consideró que estos están claramente separados y no penalizan dos veces un mismo error, el 37,5 % restante percibió algún grado de solapamiento menor, lo que se identifica como una oportunidad de mejora en las restricciones del módulo de generación más que una deficiencia estructural grave, dado que mitigar la doble penalización es un desafío complejo en la evaluación automatizada de código.

De forma independiente, la distribución de puntaje también obtuvo un 62,5 % de aprobación directa respecto a su proporcionalidad con la importancia de cada criterio, mientras que el 37,5 % restante consideró que la ponderación resultaba irregular o desproporcionada en algún criterio puntual. Este resultado es consistente con la tendencia observada en el ítem de independencia, si bien la mayoría del cuerpo docente respalda el diseño estructural de la rúbrica, ambos hallazgos apuntan a que la asignación de pesos relativos entre criterios constituye un área de refinamiento adicional para futuras iteraciones del módulo de generación.

A continuación, se evaluó de forma específica la secuencia en la organización de los componentes de la rúbrica.



Figura 10: Distribución de frecuencias docentes respecto al orden lógico de los criterios en la rúbrica.

Fuente: Elaboración propia.

El ítem de orden lógico se presenta de forma aislada en la Figura 10 por concentrar la principal oportunidad de mejora identificada en esta dimensión. Mientras que 2 de los 8 participantes (25,0 %) indicaron que todos los criterios están organizados de manera lógica, los 6 restantes (75,0 %) señalaron que algunos componentes podrían reorganizarse para mayor claridad. Los comentarios cualitativos asociados apuntaron unánimemente al criterio de “Manejo de instrucciones no válidas”, sugiriendo que este elemento debería posicionarse en las filas superiores de la rúbrica para emular el flujo secuencial de ejecución del código fuente.

Al contrastar este resultado con el *prompt* del módulo generativo, se constata que la plantilla de instrucciones no incorpora una regla de ordenamiento basada en la lógica de ejecución del código para las filas de los criterios. Si bien no es posible establecer con certeza absoluta que esta omisión sea la causa exclusiva del fenómeno observado, la ausencia de dicha directriz constituye la explicación más consistente con los datos recopilados. Este hallazgo se traduce en una oportunidad de mejora concreta para futuras iteraciones del sistema, ya sea mediante la incorporación de una restricción explícita de ordenamiento dentro de la ingeniería de *prompts*.

4.3.4. Adopción y valoración global

Con el propósito de determinar la viabilidad de integración del sistema en el proceso evaluativo real de la asignatura, se consultó al cuerpo docente sobre su disposición a utilizar la

rúbrica generada automáticamente, el valor percibido de la generación en un minuto, y el desempeño técnico global del motor.

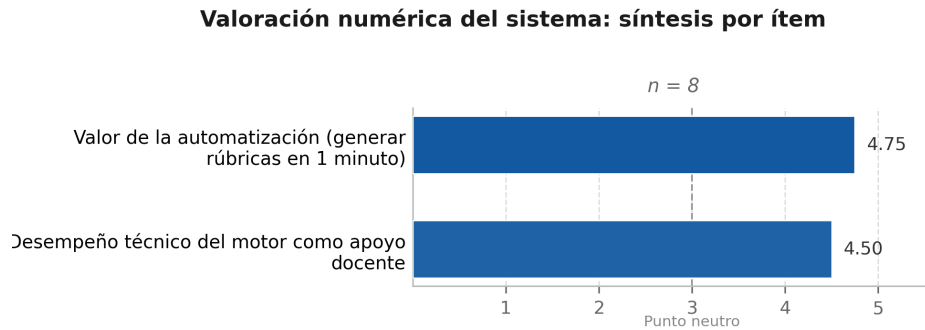


Figura 11: Promedios de valoración docente sobre el impacto temporal y el desempeño técnico del motor (Escala Likert 1-5).

Fuente: Elaboración propia.

La Figura 11 sintetiza la valoración numérica del sistema en dos dimensiones complementarias. El valor percibido de reducir el tiempo de diseño evaluativo de horas a aproximadamente un minuto registró una alta concentración en el nivel máximo de la escala, con un promedio de 4,75 puntos (donde 7 de los 8 evaluadores asignaron la calificación más alta). Por su parte, la apreciación técnica global del motor como herramienta de apoyo docente obtuvo un promedio de 4,50 puntos, con una marcada tendencia hacia valoraciones sobresalientes (5 de 8 respuestas en el nivel máximo). Estos resultados sugieren que el equipo docente reconoce el impacto práctico inmediato de la solución en la reducción de su carga laboral, y respaldan la percepción positiva sobre el desempeño técnico general de la arquitectura propuesta.

**¿Estaría dispuesto(a) a integrar esta rúbrica dentro del
proceso de evaluación de la asignatura 'Introducción a la
programación'?**

$n = 8$

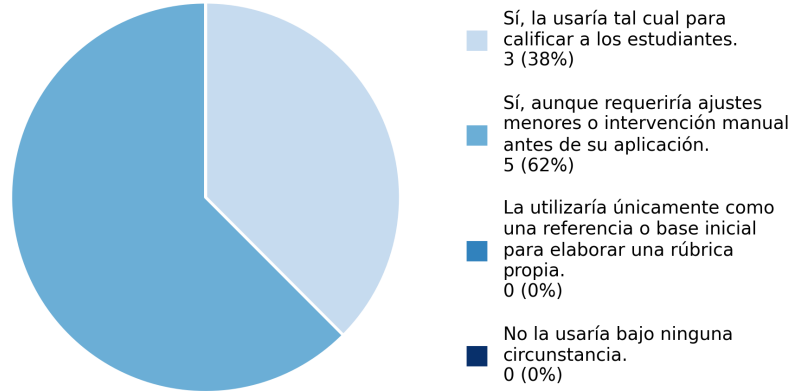


Figura 12: Distribución porcentual docente sobre la disposición para integrar la rúbrica automatizada en el aula.

Fuente: Elaboración propia.

La Figura 12 constituye el resultado más contundente de toda la validación con el cuerpo docente. En este sentido, el 100 % de los participantes manifestó una postura positiva para integrar la rúbrica en su práctica evaluativa, 3 de ellos de manera directa y los 5 restantes mediante la aplicación de ajustes menores o intervención manual previa. Ningún evaluador optó por descartar el instrumento o limitarlo a una referencia secundaria, lo que sugiere que el sistema produce un artefacto percibido como operable dentro del contexto pedagógico real, reduciendo la brecha entre generación automatizada y adopción docente. La preferencia mayoritaria por aplicar ajustes menores no implica una limitación del sistema, sino que refleja la práctica habitual de revisión que cualquier instrumento evaluativo requiere antes de su uso formal en el aula.

4.4. Percepción de los estudiantes sobre la retroalimentación

Con el objetivo de evaluar si el módulo de retroalimentación entrega un acompañamiento formativo efectivo y determinar su impacto en la calidad del proceso desde la perspectiva del usuario final, se analizó la percepción del estudiantado tras interactuar con el sistema. La metodología consistió en habilitar un portal de pre-entregas durante el periodo de desarrollo de las Tareas 3 y 4. Esta plataforma permitió a los alumnos de los paralelos P212L y P215L

subir voluntariamente sus soluciones de forma anticipada para recibir retroalimentación automatizada, permitiendo corregir o mejorar sus programas antes del envío definitivo.

Una vez concluido este ciclo, se aplicó un cuestionario de percepción estructurado en una escala Likert de cinco niveles (donde 1 corresponde a *Totalmente en desacuerdo* y 5 a *Totalmente de acuerdo*). El instrumento incorporó de forma transversal la opción “No aplica”, disponible para aquellos escenarios donde el estudiante no presentó errores en un criterio específico y, por lo tanto, no requería comentarios correctivos. Los promedios reportados a continuación excluyen dicha opción, por lo que el número de respuestas válidas varía de forma analítica entre los distintos reactivos según la experiencia particular de cada participante.

Las dimensiones y afirmaciones exactas evaluadas en el instrumento se detallan a continuación:

Dimensión 1: Comprensión del feedback

Evalúa la claridad, asimilación y el nivel del lenguaje de los comentarios entregados por el sistema (Escala Likert 1-5):

- **Identificación de errores:** “Entendí qué errores tenía mi código según el feedback recibido.”
- **Claridad de la explicación:** “El feedback explicaba claramente por qué algo estaba mal.”
- **Adecuación del lenguaje:** “El lenguaje usado en el feedback era comprensible para mi nivel en el curso.”

Dimensión 2: Utilidad para mejorar

Analiza el impacto formativo y la contribución de las sugerencias a la resolución del problema (Escala Likert 1-5):

- **Orientación estratégica:** “El feedback me orientó sobre qué revisar sin darme directamente la respuesta.”
- **Capacidad de corrección:** “Pude corregir o mejorar mi solución gracias al feedback recibido.”
- **Refuerzo conceptual:** “El feedback me ayudó a entender mejor los contenidos de la unidad.”

Dimensión 3: Justicia de la evaluación

Mide la validez de los criterios y la pertinencia de las penalizaciones aplicadas por el motor (Escala Likert 1-5):

- **Pertinencia del error:** “Los errores señalados correspondían a problemas reales de mi código.”
- **Alineación con restricciones:** “El feedback no me penalizó por aspectos que el enunciado no pedía.”
- **Consistencia curricular:** “Las sugerencias del feedback estaban basadas en lo que hemos aprendido hasta ahora, sin pedirme usar cosas que aún no vemos en el curso.”

Para la interpretación de los datos, los valores cercanos a 5 indican un alto nivel de acuerdo o valoración positiva, mientras que los valores cercanos a 3 reflejan una posición neutra, y valores cercanos a 1 indicarían desacuerdo. El detalle individual de cada ítem, con su distribución completa de respuestas, se presenta en el Anexo.

4.4.1. Comprensión de la retroalimentación

Con el fin de evaluar si el sistema entrega comentarios claros y asimilables para el nivel de un estudiante de primer año, se consultó sobre la identificación de errores, la claridad de la explicación y la adecuación del lenguaje técnico utilizado.

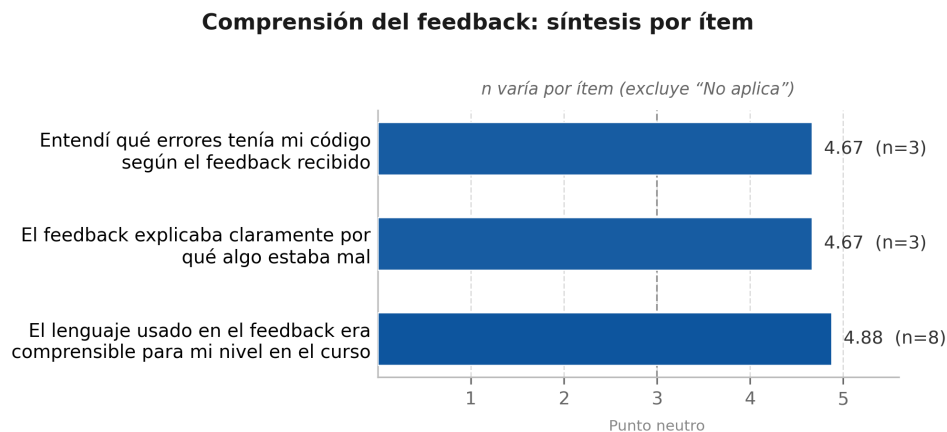


Figura 13: Promedios de percepción estudiantil en la dimensión de Comprensión de la Retroalimentación (Escala Likert 1-5).

Fuente: Elaboración propia.

La Figura 13 resume los resultados obtenidos en esta dimensión. El ítem mejor evaluado corresponde a la adecuación del lenguaje técnico, con un promedio de 4,88 puntos sobre la totalidad de los ocho estudiantes participantes, lo que indica que el vocabulario generado por el modelo de lenguaje se ajusta de forma consistente al nivel inicial del curso sin representar una barrera de entrada para el usuario.

Por su parte, la identificación de errores y la claridad de la explicación promediaron 4,67 puntos cada uno, considerando únicamente las respuestas válidas. En este caso, los tres participantes incluidos corresponden a los estudiantes cuyo código presentó errores y, por ende, recibieron retroalimentación correctiva por parte del sistema, mientras que los demás participantes respondieron *No aplica*. En consecuencia, estos últimos registros no fueron incorporados en este cálculo al no ser pertinentes para la medición de estas dimensiones. De este modo, los tres casos analizados constituyen la totalidad de observaciones efectivamente relevantes para estos reactivos.

Por lo tanto, este resultado debe interpretarse como un indicador preliminar dado el tamaño acotado de este grupo de tres observaciones, un promedio favorable posee un menor poder estadístico que uno respaldado por el total de los participantes, por lo que su valor principal es orientativo antes que representativo de la población general. Aun con esta salvedad, los resultados sugieren que, cuando el sistema interviene ante un error, el estudiantado es capaz de asimilar con precisión qué falló y por qué, proporcionando un soporte cognitivo clave para guiar el proceso autónomo de corrección de programas.

4.4.2. Utilidad para mejorar

Con el objetivo de determinar si las sugerencias entregadas por el sistema contribuyen efectivamente a la mejora del estudiante sin sustituir su proceso de razonamiento, se analizó el impacto formativo de la retroalimentación a través de tres pilares independientes, correspondientes a la orientación estratégica, la capacidad de corrección autónoma y el refuerzo conceptual. Esta dimensión constituye el respaldo para caracterizar y precisar las condiciones necesarias de un acompañamiento formativo efectivo dentro de la asignatura. Por este motivo, se presenta y analiza cada reactivo de forma individual a continuación.

En primer lugar, se evaluó si el dispositivo proporcionó orientación al estudiante sobre qué revisar sin entregarle directamente la solución.

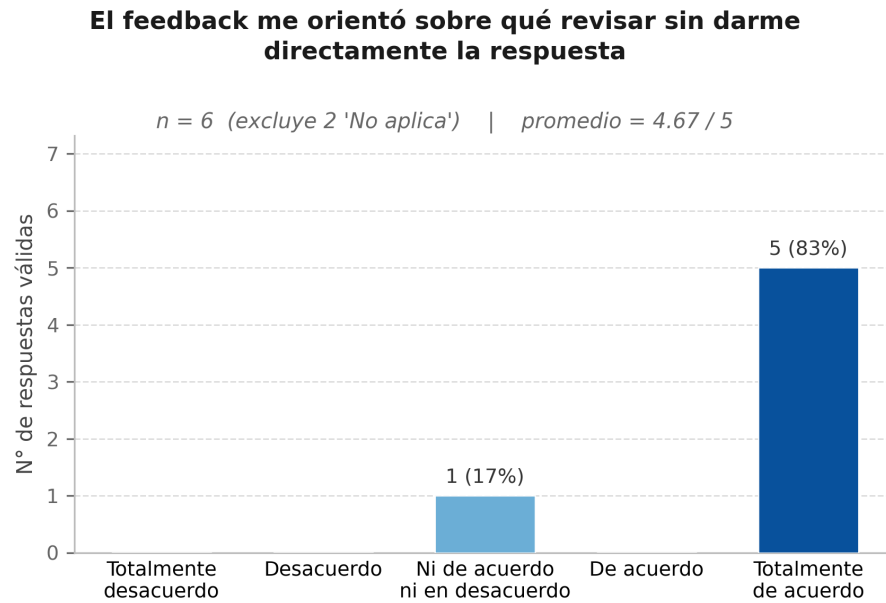


Figura 14: Distribución de frecuencias y promedio estudiantil respecto al carácter orientador de la retroalimentación.

Fuente: Elaboración propia.

La Figura 14 muestra la distribución de frecuencias de las respuestas estudiantiles respecto al rol orientador del software. El gráfico expone una marcada concentración en las valoraciones positivas, donde la gran mayoría de la muestra válida (5 de 6), se sitúa en la categoría “Totalmente de acuerdo”, resultando en un promedio de 4,67 puntos. Esta tendencia indica que los alumnos percibieron con claridad el carácter orientador de los comentarios. Este hallazgo respalda la idea de que las restricciones introducidas en las plantillas de instrucciones especializadas cumplen su propósito, en línea con la directriz pedagógica de guiar el proceso cognitivo del usuario en lugar de proveer bloques de código resueltos que fomenten una dependencia tecnológica.

A continuación, se consultó si el estudiantado logró corregir o mejorar su programa gracias a la intervención del sistema.

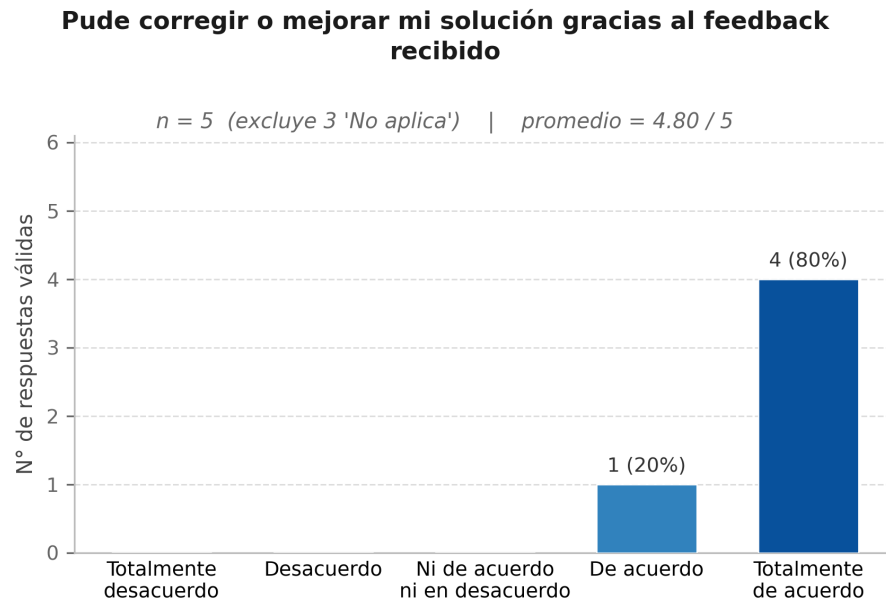


Figura 15: Distribución de frecuencias y promedio estudiantil respecto a la utilidad práctica en la optimización del código.

Fuente: Elaboración propia.

La Figura 15 ilustra el nivel de acuerdo de los participantes en relación con la utilidad práctica de las observaciones recibidas. La distribución gráfica revela un consenso sobresaliente en el extremo superior de la escala Likert, acumulando las respuestas exclusivamente entre las opciones “De acuerdo” y “Totalmente de acuerdo” para alcanzar un promedio de 4,80 puntos sobre una muestra válida de cinco estudiantes. Este comportamiento de los datos indica que la retroalimentación entregada posee un alto valor operativo percibido por el usuario. Este resultado constituye uno de los indicadores más favorables de la validación estudiantil, y permite inferir que las sugerencias de la plataforma no solo se asimilan en el plano teórico, sino que los estudiantes las asocian con acciones concretas de depuración sobre el código fuente.

Finalmente, se evaluó el impacto de la retroalimentación en el reforzamiento conceptual de la unidad académica correspondiente.

El feedback me ayudó a entender mejor los contenidos de la unidad

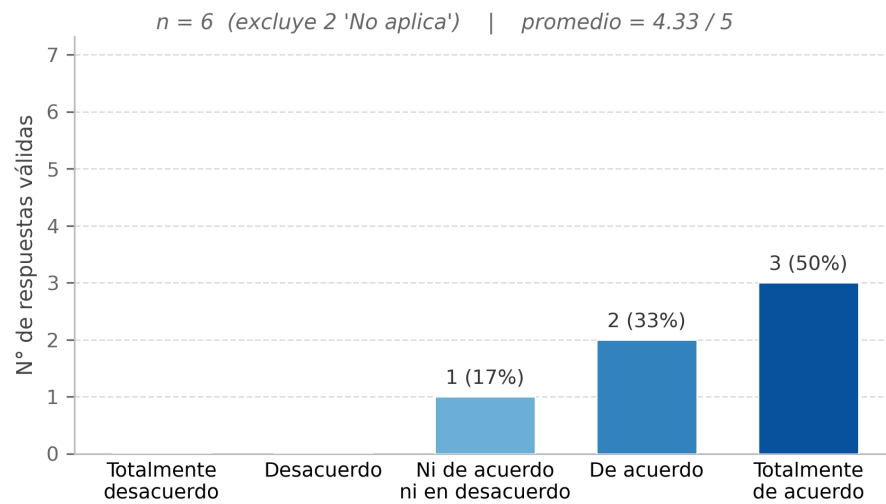


Figura 16: Distribución de frecuencias y promedio estudiantil respecto al impacto en el reforzamiento conceptual.

Fuente: Elaboración propia.

La Figura 16 detalla las frecuencias asociadas a la percepción de aprendizaje teórico derivado del uso del motor. A diferencia de los reactivos anteriores, el gráfico muestra una dispersión mayor en las respuestas, distribuyéndose entre las categorías central, alta y máxima de la escala, lo que arroja una media de 4,33 puntos sobre una muestra válida de seis participantes. Esta variabilidad indica que el impacto en el plano conceptual difiere según la experiencia y conocimientos previos de cada estudiante, sin embargo, la concentración general sobre los niveles superiores es coherente con un alcance percibido del software que supera la resolución de errores sintácticos puntuales. Este indicador apunta a que la revisión asistida podría actuar como un refuerzo que estimula el pensamiento crítico y ayuda a conectar los fallos del programa con los contenidos de la unidad.

En balance, el análisis integrado de estos tres componentes permite concluir que un modelo de retroalimentación formativa automatizada requiere equilibrar la orientación estratégica, la utilidad operativa en la corrección de errores y la transferencia hacia el aprendizaje conceptual para ser verdaderamente efectivo en el aula.

4.4.3. Justicia de la evaluación

Con el propósito de validar la pertinencia de los criterios aplicados por el motor y evitar penalizaciones injustificadas, se consultó sobre la correspondencia de los errores reportados con problemas reales del código, la alineación de la retroalimentación con las restricciones del enunciado, y la consistencia curricular de las sugerencias entregadas.

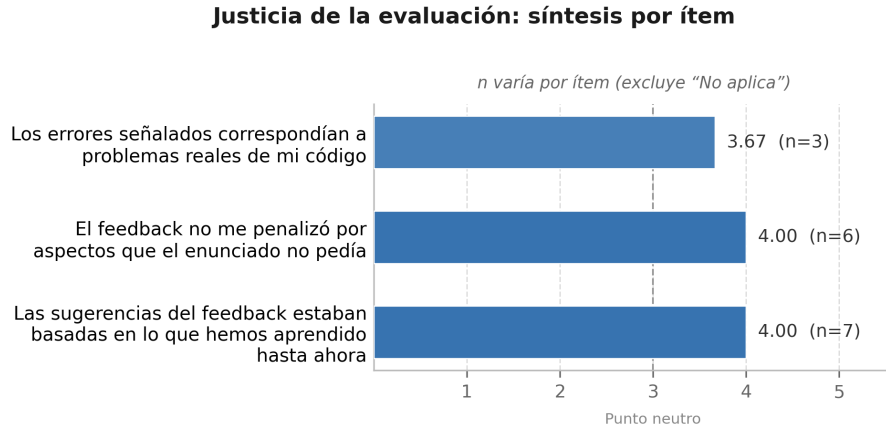


Figura 17: Promedios de percepción estudiantil en la dimensión de Justicia de la Evaluación (Escala Likert 1-5).

Fuente: Elaboración propia.

La Figura 17 resume las respuestas obtenidas para los tres ítems que componen esta dimensión. Los resultados muestran un comportamiento favorable en la alineación con las restricciones del enunciado y en la consistencia curricular, registrando promedios idénticos de 4,00 puntos sobre muestras válidas de seis y siete participantes, respectivamente. La concentración de respuestas en los niveles altos de la escala señala que el motor se ajusta de manera efectiva al marco del problema y al nivel formativo de la asignatura, lo cual es consistente con un principio clave del diseño: la retroalimentación automática debe respetar estrictamente el avance curricular para cumplir un rol verdaderamente formativo.

Por otro lado, el ítem relativo a la pertinencia del error registró la mayor dispersión y el promedio más bajo de todo el instrumento, situándose en 3,67 puntos sobre una muestra acotada a los tres participantes que presentaron fallas en sus entregas y recibieron comentarios correctivos. Este comportamiento de los datos expone un desafío técnico ligado a la naturaleza probabilística de los modelos de lenguaje, donde un fallo inexacto reportado por el sistema puede afectar la confianza del usuario final.

En balance, mientras el control del alcance conceptual sugiere un desempeño robusto dentro del aula, la precisión fina en la detección de errores específicos constituye la principal área de mejora detectada por el estudiantado. Esto sugiere la necesidad de incorporar mecanismos complementarios de verificación en el desarrollo del sistema, tales como rutinas de análisis

dinámico o pruebas de ejecución automatizadas en el *backend*, para robustecer el prototipo en futuras iteraciones.

4.5. Discusión de resultados

El análisis conjunto de las tres dimensiones abordadas en este capítulo, correspondientes a la eficiencia del sistema, la calidad de las rúbricas generadas y la percepción de los estudiantes, permite discutir de forma integrada el grado de cumplimiento de los objetivos planteados para el motor de evaluación automatizada, así como los alcances y las limitaciones observadas durante el proceso de validación.

Desde el punto de vista técnico, el comportamiento del módulo de generación automática confirma que la arquitectura basada en recuperación aumentada opera dentro de márgenes de tiempo consistentes y compatibles con un flujo de trabajo docente real. La estabilidad observada en los tiempos de respuesta respalda la solidez del *pipeline* implementado, mientras que el hecho de que el consumo de *tokens* se mantenga muy por debajo de la ventana de contexto máxima admitida por el modelo reduce el riesgo de truncamiento o pérdida de información relevante durante el procesamiento, lo cual contribuye a mitigar la probabilidad de alucinaciones y refuerza la fiabilidad técnica del sistema en escenarios de mayor complejidad.

Este respaldo cuantitativo del componente técnico se alinea de manera directa con el juicio experto sobre la calidad de las rúbricas generadas. Los altos niveles de acuerdo docente respecto a la fundamentación de los criterios en evidencia comprobable en el código permiten sostener que el sistema traduce de forma confiable las especificaciones pedagógicas del curso en instrumentos evaluativos operables. No obstante, la principal observación crítica emitida por el cuerpo docente, relativa al orden secuencial de los criterios, revela una limitación compartida entre la naturaleza probabilística del modelo de lenguaje y la ausencia de restricciones de ordenamiento cronológico explícitas dentro del *system prompt* implementado. Al no contar con una directriz de ordenamiento en sus instrucciones, el motor organiza los ítems según un criterio interno, omitiendo el flujo de ejecución secuencial que un evaluador humano aplicaría de manera intuitiva. Si bien esta explicación no puede establecerse con certeza absoluta a partir de la evidencia disponible, resulta la más consistente con los datos recopilados. En este sentido, la calidad semántica de los descriptores no basta por sí sola, ya que se requieren además reglas de ordenamiento explícitas a nivel de arquitectura o en las plantillas de instrucciones para acercar el resultado generado a los estándares de organización esperados por el equipo docente.

En la misma línea, la experiencia directa del estudiantado con el módulo de retroalimentación respalda la percepción de que el sistema cumple su propósito formativo sin comprometer el rol activo del estudiante en la resolución de problemas. La consistencia entre la percepción de orientación estratégica y la capacidad efectiva de corrección reportada por los propios estudiantes sugiere que el diseño del *prompt* logra sostener un equilibrio ade-

cuado entre guiar y no resolver, condición central para que la retroalimentación automatizada sea considerada pedagógicamente legítima. Sin embargo, la evaluación de la dimensión de justicia introduce un matiz relevante, ya que la confianza del estudiantado en el sistema depende no solo de la claridad del lenguaje empleado, sino de la exactitud técnica de los errores señalados, aspecto en el que el procesamiento basado exclusivamente en lenguaje natural muestra su límite más evidente frente a fallos que exigen una verificación lógica o de ejecución real del código fuente.

En conjunto, los resultados discutidos permiten afirmar que el prototipo cumple con el propósito definido en esta memoria, optimizando el proceso de diseño evaluativo y entregando retroalimentación formativa pedagógicamente pertinente sin sustituir el criterio experto del docente ni el razonamiento autónomo del estudiante. No obstante, el carácter exploratorio de la validación, determinado por el tamaño acotado de la muestra y la participación voluntaria del estudiantado, impide generalizar estos hallazgos a la totalidad de la población del curso. En particular, el trabajo futuro no se orienta a la aplicación de nuevos instrumentos de evaluación de la percepción, sino a la evolución técnica de la plataforma mediante la incorporación de restricciones en las plantillas de instrucciones para que el orden de los criterios siga el flujo lógico de ejecución del programa, junto con el desarrollo de mecanismos híbridos de verificación orientados a robustecer la precisión del motor frente a fallos específicos de lógica. Estas mejoras de software, en paralelo con estudios experimentales de mayor alcance, permitirán consolidar estadísticamente el impacto del sistema en el aprendizaje y en el desempeño académico de los estudiantes.

CAPÍTULO 5

CONCLUSIONES

El presente trabajo tuvo como objetivo principal construir un sistema de generación automática de rúbricas y retroalimentación que optimice el proceso de evaluación de ejercicios de programación en el curso Introducción a la Programación de la UTFSM, respondiendo directamente a la alta carga de trabajo que enfrenta el equipo de la asignatura y el consecuente impacto en la calidad de la retroalimentación y el desgaste docente. A lo largo de la memoria se abordó esta problemática de manera multidimensional, combinando una arquitectura de recuperación aumentada sobre modelos de lenguaje de gran escala con principios de retroalimentación formativa y métricas de calidad de software. A continuación, se presentan las principales conclusiones de la investigación a partir de los hallazgos obtenidos durante el proceso de desarrollo y validación.

5.1. Cumplimiento de los objetivos específicos

En cuanto a la identificación de los criterios relevantes y las dimensiones clave para la evaluación de código en etapas introductorias (Objetivo 1), la investigación determinó que una evaluación pertinente debe ir más allá de la corrección funcional, incorporando elementos de pensamiento computacional, comprensión estructural bajo el modelo de bloques de Schulte y atributos de calidad de código. Estos constructos teóricos se tradujeron en una rúbrica operativa compuesta por indicadores objetivos basados en la literatura y verificables de forma empírica en el código fuente. Este proceso se vio significativamente facilitado por el propio enunciado de la tarea, dado que la asignatura ya enseña bajo estos principios y entrega una base sólida para que el *prompt* trabaje bajo este estilo, sumado a que el diseño instruccional y los pasos secuenciales dados al LLM guían de manera óptima al modelo para cumplir este propósito. Esta alineación facilitó la integración del sistema y fue respaldada por el juicio experto del cuerpo docente, donde los ítems de cobertura del enunciado y precisión de los descriptores obtuvieron valoraciones sobresalientes sobre los 4,50 puntos, alcanzando el puntaje máximo absoluto en la fundamentación de los criterios.

Respecto a las características fundamentales de una retroalimentación formativa efectiva (Objetivo 2), se adoptó con éxito el modelo pedagógico enfocado en discriminar entre el conocimiento sobre el error y el conocimiento sobre cómo proceder, bajo atributos de especificidad y orientación socrática. Este enfoque encuentra validación en los resultados de percepción estudiantil, donde los reactivos de comprensión de los comentarios y la orientación estratégica sin entrega directa de la solución obtuvieron promedios sobre los 4,67 puntos. Esto sugiere que el diseño de las instrucciones especializadas logró trasladar de manera consistente los principios teóricos a un comportamiento del software percibido como útil por los alumnos.

Por otra parte, el desarrollo e integración de la plataforma automatizada (Objetivo 3) se cumplió formalmente mediante la implementación de una arquitectura modular basada en servicios de computación en la nube, orquestada dinámicamente bajo un esquema de ingeniería de instrucciones en capas. La existencia de una interfaz web operativa, capaz de procesar enunciados, estructurar pautas de evaluación y emitir retroalimentación individual justificada por criterio, constituye la evidencia directa de la viabilidad de la solución de software construida.

Finalmente, la evaluación del impacto del sistema en la eficiencia del proceso evidenció que la plataforma genera una rúbrica en un promedio de 22,56 segundos y evalúa una entrega individual en 13,15 segundos, consumiendo menos del 13 % de la ventana de contexto del modelo de lenguaje utilizado. A este óptimo rendimiento técnico se suma un antecedente favorable en materia económica, siendo el bajo costo operativo asociado al ciclo completo otorga a la solución un amplio margen de escalabilidad y sustentabilidad financiera.

Esta eficiencia tecnológica y económica se traduce en un alto valor operativo para el equipo docente, cuya totalidad manifestó una sólida disposición a integrar la herramienta en su práctica cotidiana. Los profesores validaron el sistema con una calificación media de 4,75 puntos, destacando especialmente su capacidad para reducir el tiempo de diseño de evaluaciones a menos de un minuto. En complemento, la calidad de la retroalimentación entregada fue validada positivamente por el estudiantado, quien reportó un promedio de 4,80 puntos en la utilidad práctica percibida para corregir sus programas, cerrando así el ciclo de eficiencia contemplado en este objetivo tanto desde la perspectiva docente como estudiantil.

Esta favorable recepción respalda la viabilidad de la solución propuesta y sugiere que esta representa un aporte concreto para abordar la problemática identificada en el curso. En consecuencia, el conjunto de metas específicas y el objetivo general de la memoria se consideran cumplidos satisfactoriamente dentro de los límites de una validación exploratoria.

5.2. Principales resultados y contribuciones

Los hallazgos de esta investigación permiten identificar cuatro contribuciones concretas. En el plano técnico, se demostró que una arquitectura de recuperación aumentada orientada al dominio educativo puede operar con latencias mínimas y una variabilidad marginal en el consumo de recursos informáticos. Una ventaja crítica de este diseño arquitectónico es que faculta al sistema para incrementar el volumen de documentos históricos de evaluación almacenados con el paso de los años, debido al filtrado selectivo de la recuperación por similitud. El LLM podría mantener su estabilidad y fiabilidad operativa en el tiempo, mitigando la degradación del contexto incluso ante repositorios de alta densidad informativa, aunque esta proyección requeriría validación empírica en estudios futuros con corpus de mayor escala.

En el ámbito pedagógico, la incorporación de restricciones académicas vinculadas a las uni-

dades de aprendizaje del curso permitió que el motor de evaluación respetara el avance curricular de los alumnos, aspecto respaldado por una valoración positiva en consistencia curricular por parte del estudiantado. En la dimensión metodológica, el proceso generó como resultado derivado un instrumento estructurado de encuesta para docentes y estudiantes que queda disponible para futuras líneas de investigación.

Adicionalmente, esta memoria documenta y evalúa un conjunto de plantillas de prompts estructuradas en capas (rol, tarea/contexto y formato), refinadas iterativamente durante el desarrollo, que constituyen un punto de partida reutilizable tanto para futuras implementaciones de sistemas educativos basados en modelos de lenguaje, como para docentes que de forma independiente quieran apoyarse en la IA para sus propios procesos evaluativos.

Este desarrollo no estuvo exento de desafíos técnicos. La etapa de mayor complejidad correspondió al afinamiento y calibración de las instrucciones del sistema, donde lograr resultados estables requirió múltiples iteraciones de diseño. Esta dificultad resalta la ingeniería de instrucciones como una competencia central para la integración responsable de software basado en inteligencia artificial. Asimismo, la validación iterativa junto a la supervisión del profesor guía resultó clave para corregir desviaciones tempranas en el criterio de la herramienta, reforzando la conclusión de que un instrumento generado automáticamente requiere supervisión experta antes de su uso formal en el aula.

Para el equipo docente, el aporte principal se traduce en una reducción sustancial del tiempo dedicado al diseño evaluativo sin perder el control experto sobre el resultado final. Para el estudiantado, el beneficio se refleja en el acceso a retroalimentación oportuna y orientadora, tal como se detalló en el cumplimiento del Objetivo 4, lo cual se complementa con valoraciones cualitativas que destacan la pertinencia de las recomendaciones entregadas.

5.3. Limitaciones del estudio

Los resultados obtenidos deben interpretarse considerando el carácter exploratorio de la validación. El tamaño reducido y la conformación voluntaria de las muestras de estudiantes y profesores impiden generalizar estos hallazgos a la totalidad de la población de la asignatura o establecer relaciones causales con el rendimiento académico. A esto se suma una participación estudiantil menor a la proyectada, lo que limitó el análisis en ítems específicos como la pertinencia del error. Como hipótesis metodológica, se plantea que el cierre temprano de la ventana de recepción de pre-entregas pudo desincentivar la participación de los alumnos que habitualmente entregan sus trabajos cerca del plazo límite. Finalmente, se identifica una limitación técnica en la dependencia exclusiva de los modelos de lenguaje. Debido a su naturaleza probabilística, el procesamiento de lenguaje natural no siempre garantiza una exactitud lógica absoluta, introduciendo una variabilidad que puede dar como resultado calificaciones más altas o bajas de lo esperado, esto va en línea con lo comentado en las encuestas estudiantiles respecto a la pertinencia del error, donde se obtuvo una de las calificaciones más bajas.

5.4. Trabajo futuro

A partir de las restricciones observadas, se proponen líneas prioritarias para dar continuidad al proyecto. En primer lugar, se plantea la oportunidad de contrastar el desempeño de distintos modelos de lenguaje (LLM) utilizando exactamente el mismo diseño metodológico e instrumentos de validación de esta investigación, con el fin de evaluar comparativamente la consistencia y el comportamiento de las métricas bajo diferentes arquitecturas de IA. Por otra parte, es necesario incorporar restricciones en las plantillas de instrucciones para asegurar que la organización de los criterios evaluativos siga el flujo lógico de ejecución del programa, atendiendo de este modo la principal observación de mejora manifestada por el equipo docente. Finalmente, se plantea la necesidad de replicar esta validación en estudios experimentales de mayor escala temporal y con muestras de mayor representatividad, ajustando los plazos de disponibilidad del portal e implementando estrategias que fomenten una mayor participación estudiantil, con el fin de consolidar estadísticamente el impacto real del sistema en el desempeño académico y el aprendizaje autónomo del estudiantado.

En conclusión, la evidencia empírica recopilada demuestra que la inteligencia artificial es capaz de optimizar significativamente el tiempo del equipo docente, y sugiere que provee un soporte cognitivo inmediato para los alumnos, actuando como una herramienta de asistencia y apoyo formativo que bajo ningún escenario debe sustituir la supervisión humana. Este trabajo aporta evidencia concreta de que es posible integrar estas tecnologías de forma responsable y sostenible en el ámbito evaluativo universitario, siempre que su arquitectura de software esté gobernada por principios pedagógicos claros y por el criterio experto de los propios docentes.

ANEXOS

Anexo A: Resultados de la evaluación del cuerpo docente

A continuación se detalla gráficamente la distribución de respuestas del equipo docente para cada afirmación del instrumento, cuyo análisis consolidado ya fue abordado en el cuerpo del documento.

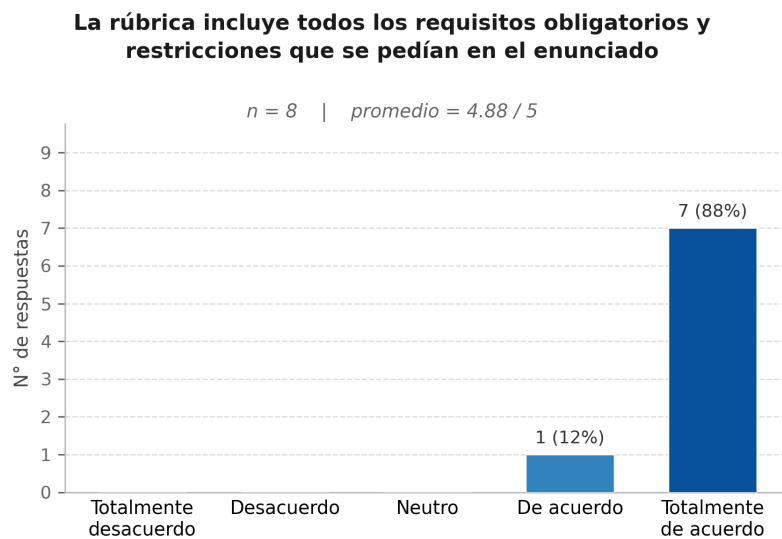


Figura 18: Distribución de frecuencias y promedios de respuestas docentes respecto a la cobertura de restricciones explícitas del enunciado.

Fuente: Elaboración propia

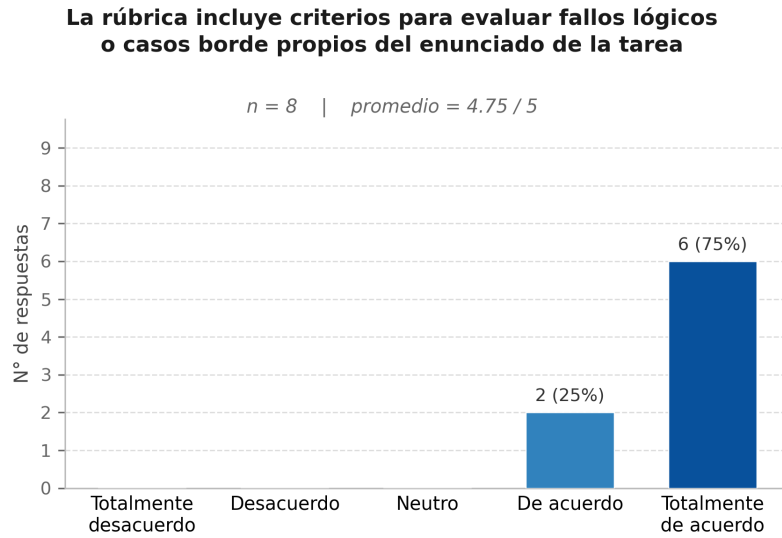


Figura 19: Distribución de frecuencias y promedios de respuestas docentes respecto a la inclusión de criterios orientados a fallos lógicos y casos límite.

Fuente: Elaboración propia

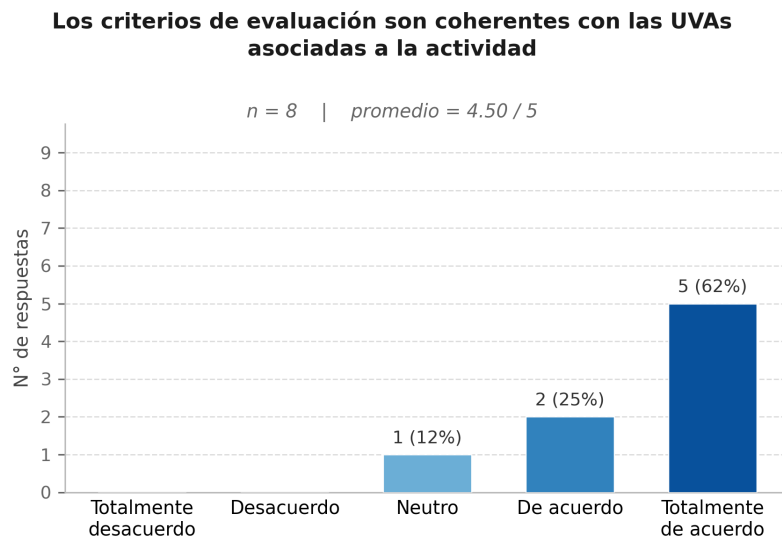


Figura 20: Distribución de frecuencias y promedios de respuestas docentes respecto al alineamiento de los criterios generados con las unidades virtuales de aprendizaje.

Fuente: Elaboración propia

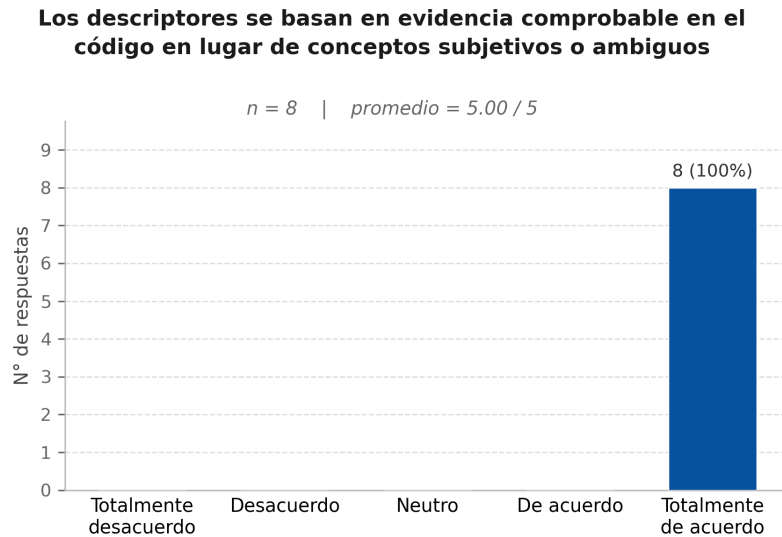


Figura 21: Distribución de frecuencias y promedios de respuestas docentes respecto a la fundamentación de los descriptores basados en evidencia observable en el código.

Fuente: Elaboración propia

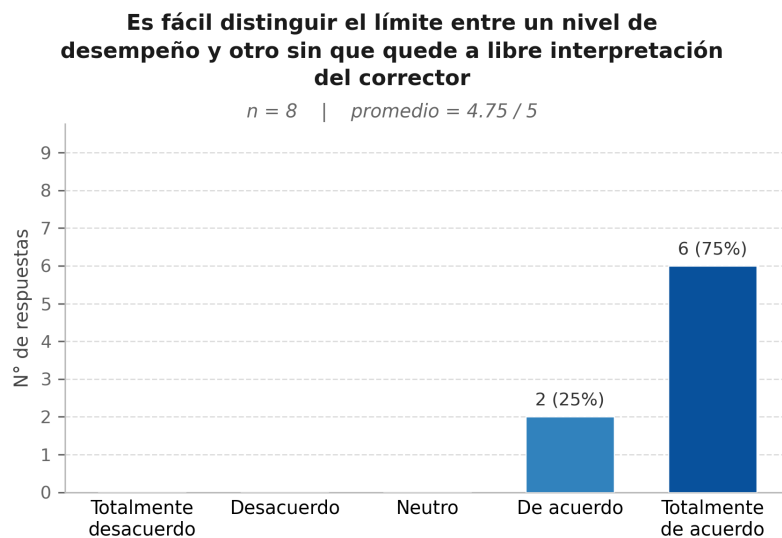


Figura 22: Distribución de frecuencias y promedios de respuestas docentes respecto a la claridad en la delimitación de fronteras entre niveles de desempeño.

Fuente: Elaboración propia

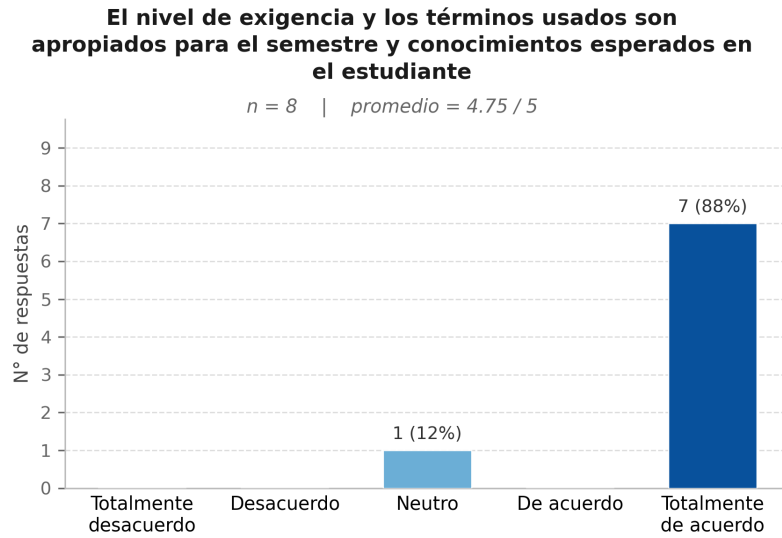


Figura 23: Distribución de frecuencias y promedios de respuestas docentes respecto a la idoneidad del nivel de exigencia técnica y el lenguaje adaptado.

Fuente: Elaboración propia

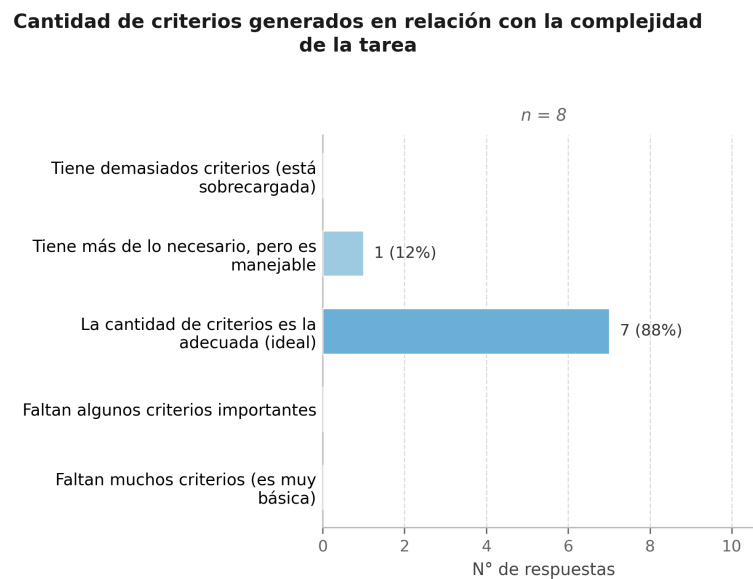


Figura 24: Distribución de frecuencias y promedios de respuestas docentes respecto a la densidad de criterios en relación con la complejidad de la tarea.

Fuente: Elaboración propia

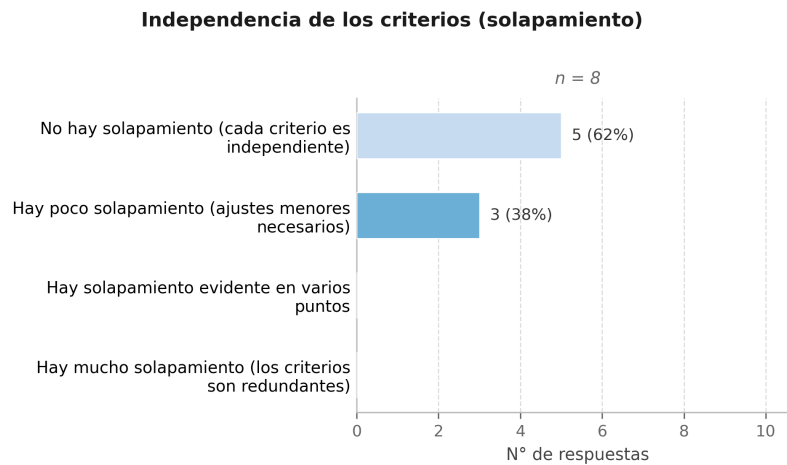


Figura 25: Distribución de frecuencias y promedios de respuestas docentes respecto al nivel de independencia de los criterios y la mitigación de redundancias.

Fuente: Elaboración propia

Distribución de puntaje según la importancia de cada criterio

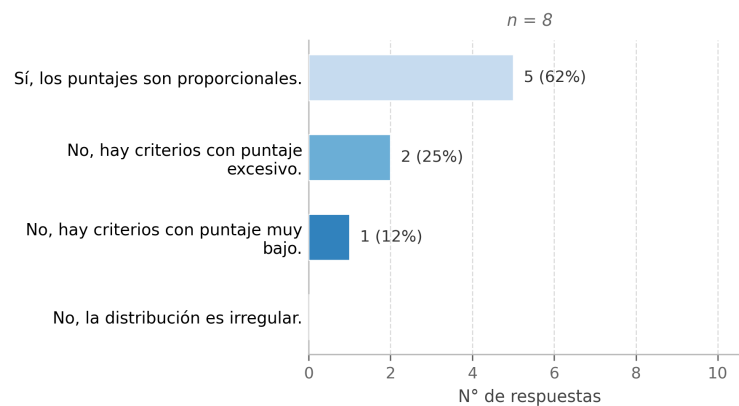


Figura 26: Distribución de frecuencias y promedios de respuestas docentes respecto a la proporcionalidad en la asignación de puntaje entre criterios.

Fuente: Elaboración propia

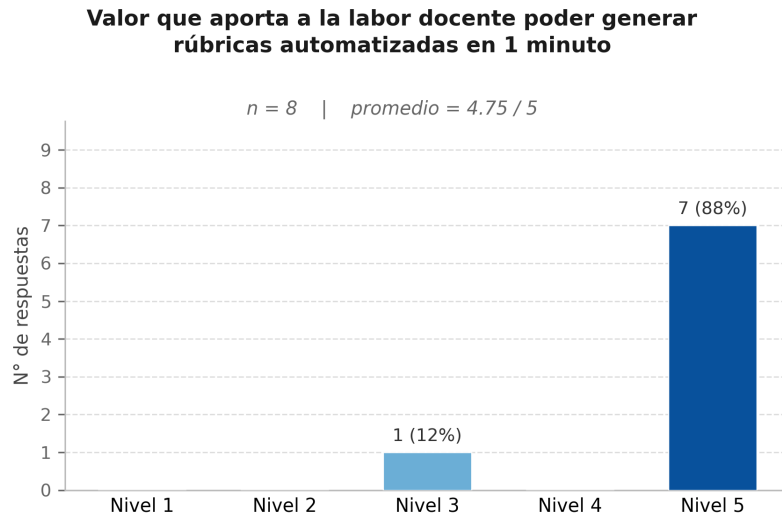


Figura 27: Distribución de frecuencias y promedios de respuestas docentes respecto al valor percibido en la generación automatizada de pautas de evaluación.

Fuente: Elaboración propia

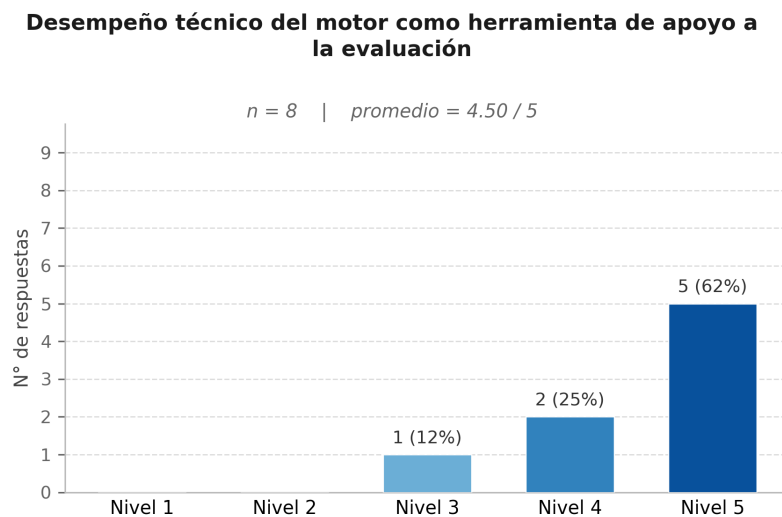


Figura 28: Distribución de frecuencias y promedios de respuestas docentes respecto a la valoración global del desempeño técnico del motor.

Fuente: Elaboración propia

Anexo B: Resultados de la percepción estudiantil

A continuación se detalla gráficamente la distribución de respuestas del estudiantado para cada afirmación del instrumento, cuyo análisis consolidado ya fue abordado en el cuerpo del documento.

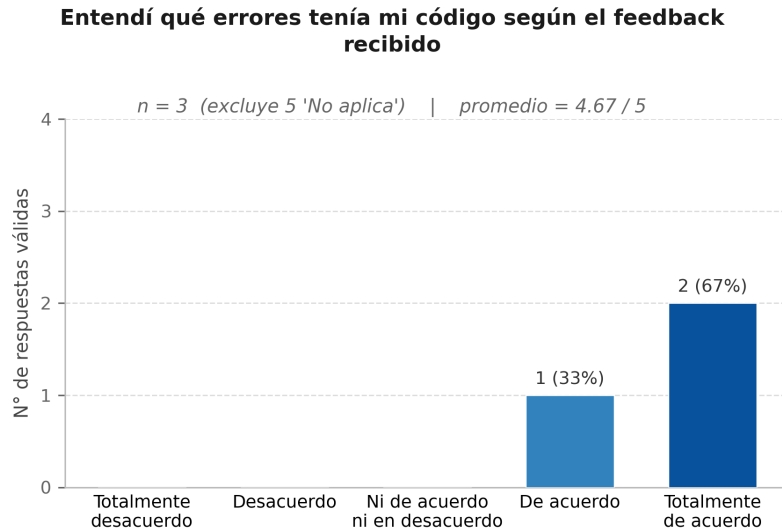


Figura 29: Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de comprensión de los errores identificados en el código.

Fuente: Elaboración propia

El feedback explicaba claramente por qué algo estaba mal

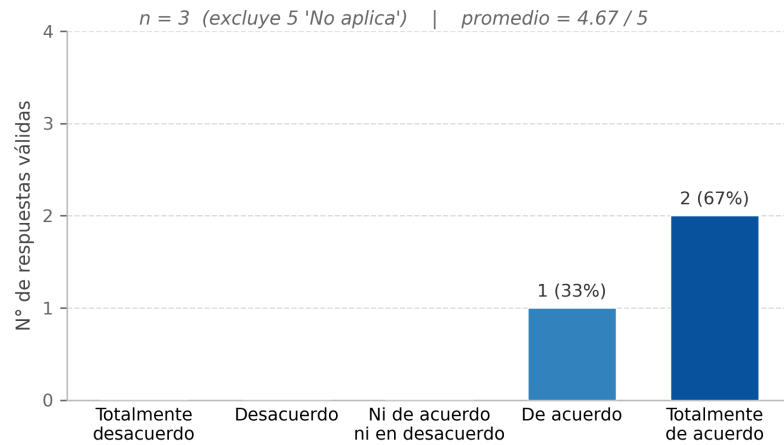


Figura 30: Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de claridad explicativa de las causas de los errores.

Fuente: Elaboración propia

El lenguaje usado en el feedback era comprensible para mi nivel en el curso

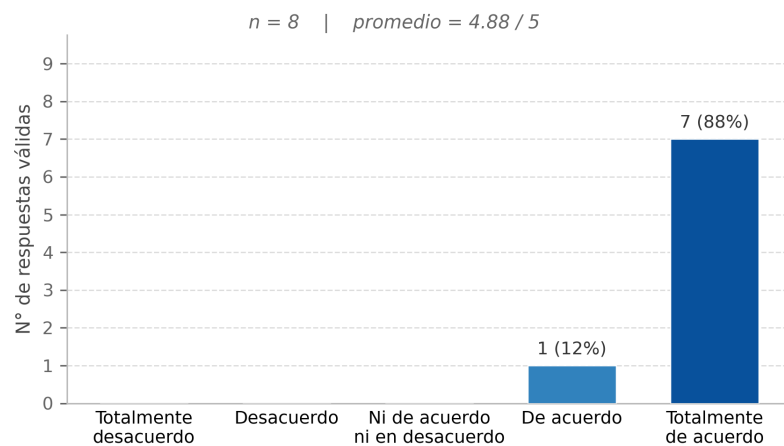


Figura 31: Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de legibilidad del lenguaje técnico empleado.

Fuente: Elaboración propia

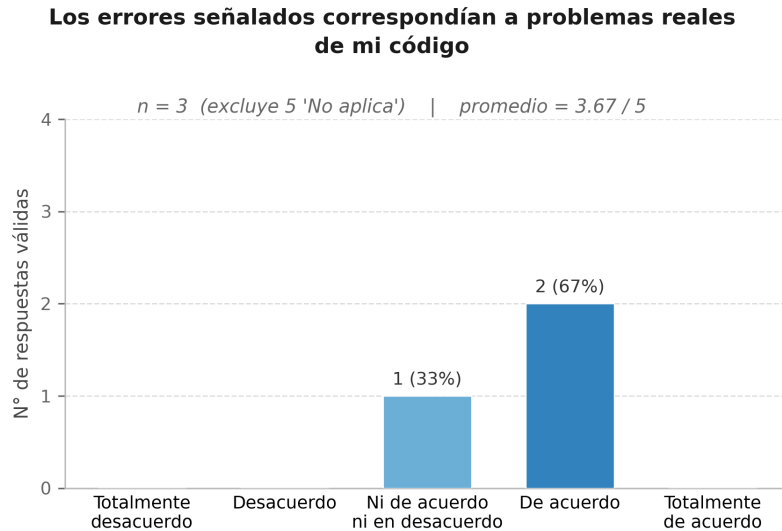


Figura 32: Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de correspondencia con fallos reales del programa.

Fuente: Elaboración propia

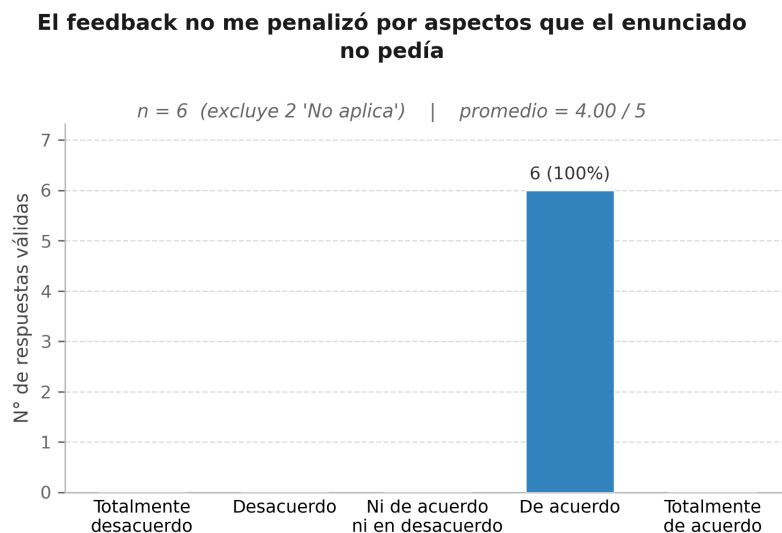


Figura 33: Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de pertinencia con los límites del enunciado.

Fuente: Elaboración propia

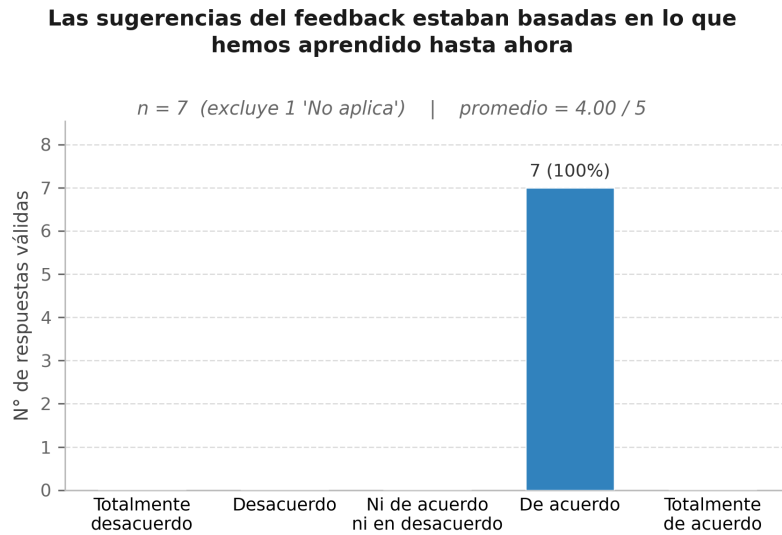


Figura 34: Distribución de frecuencias y promedios de percepción estudiantil en la dimensión de consistencia con el avance curricular de la asignatura.

Fuente: Elaboración propia

REFERENCIAS BIBLIOGRÁFICAS

- [Alansari y Luqman, 2026] Alansari, A. y Luqman, H. (2026). Large language models hallucination: A comprehensive survey. *arXiv preprint arXiv:2510.06265*.
- [Azoulay et al., 2025] Azoulay, R., Hirst, T., y Reches, S. (2025). Large language models in computer science classrooms: Ethical challenges and strategic solutions. *Applied Sciences*, 15(4):1793.
- [Boud y Molloy, 2013] Boud, D. y Molloy, E. (2013). *Feedback in higher and professional education: understanding it and doing it well*. Routledge.
- [da Silva et al., 2023] da Silva, E. P., Caceffo, R., y Azevedo, R. (2023). When test cases are not enough: Identification, assessment, and rationale of misconceptions in correct code (mc3). *Revista Brasileira de Informática na Educação (RBIE)*, 31:1165–1199.
- [Dapozo et al., 2022] Dapozo, G., Greiner, C., Petris, R., Irrazabal, E., Company, A. M., Espíndola, M. C., y Medina, Y. (2022). Evaluación de habilidades de pensamiento computacional al inicio de una asignatura de programación en una carrera de informática. *Universidad Nacional del Nordeste (UNNE), Corrientes, Argentina*.
- [Hadi et al., 2025] Hadi, M. U., Al-Tashi, Q., Qureshi, R., Shah, A., Muneer, A., Irfan, M., Zafar, A., Shaikh, M. B., Akhtar, N., Al-Garadi, M. A., Hassan, S. Z., Shoman, M., Wu, J., Mirjalili, S., y Shah, M. (2025). A survey on large language models: Applications, challenges, limitations, and practical usage. *TechRxiv*.
- [Harvey et al., 2025] Harvey, E., Koenecke, A., y Kizilcec, R. F. (2025). "don't forget the teachers": Towards an educator-centered understanding of harms from large language models in education. *arXiv preprint arXiv:2502.14592*. To appear in the 2025 ACM CHI Conference on Human Factors in Computing Systems (CHI '25).
- [Herklotz et al., 2025] Herklotz, M., Ippisch, N., y Haensch, A.-C. (2025). Can we trust llms as a tutor for our students? evaluating the quality of llm-generated feedback in statistics exams. *arXiv preprint*.
- [Izu y Mirolo, 2020] Izu, C. y Mirolo, C. (2020). Comparing small programs for equivalence: A code comprehension task for novice programmers. En *Innovation and Technology in Computer Science Education (ITiCSE '20)*, New York, NY, USA. ACM.
- [Izu et al., 2024] Izu, C., Mirolo, C., Börstler, J., Connamacher, H., Crosby, R., Glassey, R., Hal-deman, G., Kiljunen, O., Kumar, A. N., Liu, D., Luxton-Reilly, A., Matsumoto, S., de Oliveira, E. C., Russell, S., y Shah, A. (2024). Introducing code quality at cs1 level: Examples and activities. En *2024 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2024)*, Milan, Italy. ACM.

- [Jacobsen y Weber, 2025] Jacobsen, L. J. y Weber, K. E. (2025). The promises and pitfalls of large language models as feedback providers: A study of prompt engineering and the quality of ai-driven feedback. *AI*, 6(1):35.
- [Keuning et al., 2018] Keuning, H., Jeuring, J., y Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)*, 19(1):1–43.
- [Kirk et al., 2024] Kirk, D., Luxton-Reilly, A., y Tempero, E. (2024). A literature-informed model for code style principles to support teachers of text-based programming. En *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*, pp. 134–143. ACM.
- [Liu et al., 2024] Liu, F., Liu, Y., Shi, L., Yang, Z., Zhang, L., Lian, X., Li, Z., y Ma, Y. (2024). Beyond functional correctness: Exploring hallucinations in llm-generated code. *arXiv preprint arXiv:2404.00971*.
- [Mselle, 2025] Mselle, L. J. (2025). Teaching mental models in introductory programming. *Journal of ICT Systems (JICTS)*, 3(2):1–23.
- [Narciss, 2008] Narciss, S. (2008). Feedback strategies for interactive learning tasks. En Spector, J. M., Merrill, M. D., Van Merriënboer, J. J. G., y Driscoll, M. P., editores, *Handbook of research on educational communications and technology*, pp. 125–143. Lawrence Erlbaum Associates, 3rd edición.
- [Paulsen, 2026] Paulsen, N. (2026). Context is what you need: The maximum effective context window for real world limits of llms. *Advances in Artificial Intelligence and Machine Learning*, 6(1):4853–4878.
- [Pitts et al., 2025] Pitts, G., Hridi, A. P., y Lekshmi-Narayanan, A.-B. (2025). A survey of llm-based applications in programming education: Balancing automation and human oversight. *arXiv preprint arXiv:2510.03719*.
- [Raihan et al., 2025] Raihan, N., Siddiq, M. L., Santos, J. C. S., y Zampieri, M. (2025). Large language models in computer science education: A systematic literature review. En *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025)*, pp. 938–944, New York, NY, USA. ACM.
- [Rocha, 2024] Rocha, H. J. B. (2024). Exploring the effects of feedback on the problem-solving process of novice programmers. En *Anais do XXXV Simpósio Brasileiro de Informática na Educação (SBIE 2024)*, pp. 474–485. SBC.
- [Schulte, 2008] Schulte, C. (2008). Block model: An educational model of program comprehension as a tool for a scholarly approach to teaching. En *Proceedings of the 4th International Workshop on Computing Education Research, ICER '08*, pp. 149–160. ACM.
- [Shute, 2008] Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, 78(1):153–189.

- [Tang *et al.*, 2025] Tang, B., Liang, J., Hu, W., y Luo, H. (2025). Enhancing programming performance, learning interest, and self-efficacy: The role of large language models in middle school education. *Systems*, 13(7):555.
- [Wing, 2006] Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.