Herramienta de evaluación automatizada de respuesta abierta basada en inteligencia artificial para códigos de computadora.

AI-based open-response automated evaluation tool for computer code.

José Manuel Damián Aguilar (1).
Estudiante Tecnológico Nacional de México/I. T. de Tuxtla Gutiérrez.
L21270630@tuxtla.tecnm.mx.

Cesar Gustavo Farrera Molina (2). Estudiante Tecnológico Nacional de México/I. T. de Tuxtla Gutiérrez,
L21270591@tuxtla.tecnm.mx.

Luis Tomás García Andrade* (3). Tecnológico Nacional de México/I. T. de Tuxtla Gutiérrez,
luis.ga1@tuxtla.tecnm.mx.

Héctor Guerra Crespo (4). Tecnológico Nacional de México/I. T. de Tuxtla Gutiérrez. hector.gc@tuxtla.tecnm.mx.

Galdino Belisario Nango Solís (5). Tecnológico Nacional de México/I. T. de Tuxtla Gutiérrez,
galdino.ns@tuxtla.tecnn.mx.

Néstor Antonio Morales Navarro (6), Tecnológico Nacional de México/I. T. de Tuxtla Gutiérrez,
nestor.mn@tuxtla.tecnm.mx.

*corresponding author.

Artículo recibido en mayo 07, 2026; aceptado en mayo 28, 2026.

Resumen.

El presente trabajo desarrolla una herramienta de evaluación automatizada de respuesta abierta orientada a la calificación de códigos fuente en lenguaje C mediante técnicas de inteligencia artificial y reglas heurísticas. La investigación surge ante las limitaciones de la evaluación manual en entornos educativos con alta demanda, donde la revisión de ejercicios de programación requiere analizar simultáneamente sintaxis, lógica y estructura algorítmica, generando procesos lentos y susceptibles a inconsistencias. La propuesta integra una arquitectura híbrida compuesta por un motor de reglas deterministas y un modelo de aprendizaje supervisado basado en Random Forest, con el propósito de generar calificaciones coherentes con el criterio docente y ofrecer retroalimentación inmediata al estudiante. El sistema fue entrenado con 209 exámenes digitalizados de la asignatura Fundamentos de Programación y validado mediante un conjunto independiente de códigos inéditos. Para el procesamiento se implementó un mecanismo de extracción de características sintácticas y estructurales del código, considerando elementos como ciclos, condiciones y operadores. Asimismo, se diseñó una plataforma web para la interacción con los usuarios. Los resultados evidencian que la combinación entre evaluación heurística y aprendizaje automático permite mejorar la consistencia de la calificación, reducir el tiempo de revisión y proporcionar diagnósticos automáticos útiles para el aprendizaje autónomo, constituyendo una alternativa viable para la automatización de evaluaciones de respuesta abierta.

Palabras claves: Evaluación automatizada, *machine learning*, *random forest,* respuesta abierta.

Abstract.

This paper develops an automated, open-response assessment tool for grading C source code using artificial intelligence techniques and heuristic rules. The research stems from the limitations of manual assessment in demanding educational environments, where reviewing programming exercises requires simultaneously analyzing syntax, logic, and algorithmic structure, resulting in slow processes prone to inconsistencies. The proposed hybrid architecture comprises a deterministic rules engine and a supervised learning model based on Random Forest, with the aim of generating grades consistent with the instructor's criteria and providing immediate feedback to students. The system was trained using 209 digitized exams from the Fundamentals of Programming course and validated with an independent set of original code. A mechanism for extracting syntactic and structural features from the code was implemented for processing, considering elements such as loops, conditions, and operators. A web platform was also designed for user interaction. The results show that the combination of heuristic evaluation and machine learning improves grading consistency, reduces review time, and provides automatic diagnoses useful for autonomous learning, constituting a viable alternative for the automation of open-response assessments.

Keywords: Automated evaluation, machine learning, open response, random forest.

1. Introducción.

Históricamente, la automatización de la evaluación académica ya sea escolarizada o en programas de capacitación se ha limitado a sistemas de respuesta cerrada, dejando la producción textual como una tarea exclusivamente manual debido a la complejidad semántica del lenguaje.

Los procesos de revisión de respuesta abierta como ensayos o programas de computadora dependen de la disponibilidad y el criterio subjetivo del docente, lo que genera una brecha en la estabilidad de criterios y prolonga el tiempo y la calidad de entrega de resultados. Al respecto, se ha observado que la evaluación manual en entornos masivos es ineficiente, pues "la evaluación de respuestas abiertas requiere un nivel de análisis cognitivo que los sistemas tradicionales de opción múltiple no pueden replicar, limitando la retroalimentación inmediata" (Shermis & Burstein, 2013, p. 14),

El proceso de evaluación en respuesta abierta constituye uno de los retos pedagógicos más persistentes en la educación. A diferencia de otras disciplinas, la revisión, por ejemplo, de código fuente exige del docente un análisis simultáneo de corrección sintáctica, coherencia lógica y adherencia a criterios de diseño, proceso que, ante grupos numerosos, se convierte en una tarea prolongada, propensa a la inconsistencia y difícil de escalar. En las instituciones de educación superior los docentes deben gestionar la calificación de múltiples grupos de forma manual, lo que compromete tanto la objetividad de la retroalimentación como la autonomía de aprendizaje del estudiante.

Las soluciones adoptadas históricamente oscilan entre dos extremos insatisfactorios. Por un lado, la evaluación manual ofrece precisión conceptual, pero resulta ineficiente ante el volumen creciente de la matrícula y vulnerable a la fatiga del evaluador. Por otro, los exámenes de opción múltiple reducen la evaluación a la verificación de una salida esperada, ignorando la lógica interna, la estructura del algoritmo y la calidad del razonamiento del alumno.

El auge de la Inteligencia Artificial y, en particular, del Machine Learning (ML) que es una rama de la inteligencia artificial que posibilita que los sistemas aprendan de datos y realicen predicciones o clasificaciones sin ser programados de manera explícita para cada tarea. Celik et al. (2022) Señala que, en el ámbito de la evaluación automática, el ML se utiliza para entrenar modelos capaces de identificar patrones en las respuestas de los estudiantes, detectar errores comunes y generar calificaciones de forma objetiva. Mediante el uso de algoritmos supervisados y no supervisados, se pueden analizar grandes volúmenes de datos, lo que facilita la personalización del proceso educativo.

En la práctica, los sistemas basados en ML se alimentan de bases de datos con respuestas evaluadas manualmente, lo que permite que el modelo aprenda y reproduzca criterios de calificación de manera automatizada. Esta metodología no solo agiliza el proceso evaluativo, sino que también mejora la consistencia y reduce la posibilidad de sesgos

humanos. Además, el artículo de Aguado, J. M. (2025) destaca que el ML se utiliza para predecir áreas de mejora en el desempeño estudiantil, ofreciendo retroalimentación adaptativa que orienta el proceso de aprendizaje. La evidencia empírica, respaldada por estudios como los presentados en (EDUTEC. Revista Electrónica de Tecnología Educativa, [EduTec,2022]), demuestra que la aplicación del ML en la educación puede incrementar la eficiencia en la corrección y ofrecer diagnósticos más precisos sobre el rendimiento de los alumnos. Así, la integración de técnicas de Machine Learning representa un avance significativo en la modernización de los sistemas evaluativos.

Frente a esta brecha, el presente trabajo propone y valida un sistema de evaluación automática basado en una arquitectura híbrida que combina un motor de reglas heurísticas con un modelo de aprendizaje supervisado de tipo Random Forest, aplicado específicamente a la evaluación y retroalimentación de exámenes de programación en lenguaje C. La hipótesis central del sistema es que la integración de ambos componentes, la rigidez determinista del etiquetado para la detección de errores sintácticos y la capacidad de generalización estadística para la valoración de la lógica global, permite superar las limitaciones individuales de cada enfoque, produciendo calificaciones coherentes con el criterio docente incluso en escenarios de entrenamiento con datos reducidos.

El sistema fue entrenado con 209 exámenes digitalizados de la materia "Fundamentos de Programación" y validado sobre un conjunto independiente de 10 códigos no vistos durante el entrenamiento. La calificación final se obtiene mediante una función de ponderación equitativa entre ambos módulos, y es expuesta al estudiante a través de una API RESTful que devuelve retroalimentación textual diagnóstica sin revelar la solución correcta, con el propósito de preservar la autonomía de aprendizaje.

2. Métodos.

Para la construcción del prototipo se seleccionan tres algoritmos sobre cuatro propiedades numéricas. La escala de calificación es de 1 a 5 puntos estimando un examen de dos ejercicios para una hora de desarrollo en aula.

Algoritmos.

- Contar.
- Posiciones primero, segundo, último, penúltimo.
- Mayor.

Propiedades numéricas.

- Positivo.
- Negativo.
- Par.
- Impar.

Los algoritmos se pueden implementar en arreglos o lectura directa.

Se etiqueta o configura el resultado esperado, por ejemplo.

Algoritmo contar negativos, lectura directa.

Dada una lista de números.

8, -3, 4, 9, -6, 7, 4, -6

1. Contar negativos. Salida. Negativos = 3

```
#include<stdio.h>

int main()
{
    int n,c,x,i;

    scanf("%d", &n);

    c=0;
    for (i=0; i<n; i++)
    {
        scanf("%d", &x);
        if (x<0)
            c=c+1;
    }
    printf("Negativos = %d", c);
}
```

Criterios.

`#include<stdio.h>`
 	
`int main()`
`{`
`  int n,c,x,i;`
 	
`  scanf("%d", &n);`
 	
`  c=0;`
`  for (i=0; i<n; i++)`
 	
`  {`
`  scanf("%d", &x);`
`  if (x<0)`
`  c=c+1;`
`  }`
`  printf("Negativos = %d", c);`
`}`
	Biblioteca
	
	funcion_main
	Bloque_i
	Declaración_variables, indentada (con relación a la línea anterior)
	
	Leer_n
	
	Iniciar_contador, =0 // puede ser en diferente lugar
	Ciclo_control, Criterios, inicio_cero, condición <n,
	incremento_unitario
	Bloque_i
	Leer_x, indentada
	condicion_propiedad, <0
	contar, indentada
	bloque_f, desindentada, desindentada
	salida
	bloque_f, desindentada

Peso de los criterios

`#include<stdio.h>`
 	
`int main()`
`{`
`  int n,c,x,i;`
 	
`  scanf("%d", &n);`
 	
`  c=0;`
`  for (i=0; i<n; i++)`
	-0.1
	
	-0.1
	-0.1
	-0.3
	
	-0.3
	
	1
	1

`    {`
`        scanf("%d", &x);`
`        if (x<0)`
`            c=c+1;`
`    }`
`    printf("Negativos = %d", c);`
`}`
	-0.1
	-0.3
	2
	1
	-0.1
	-0.3
	-0.1

Nota. Hay puntos positivos en la evaluación del algoritmo y puntos negativos sobre elementos de apoyo al algoritmo.

De la misma forma se configura cada algoritmo.

Tecnología involucrada.

Python. Por disponer de bibliotecas especializadas a machine learning (scikit-learn, PyTorch, TensorFlow, entre otras).

Google Colab. Por el acceso inmediato a GPU y CPU sin configuración adicional y su integración con bibliotecas modernas de ML.

C y PHP. Lenguaje C es utilizado como el objeto de estudio, dado que corresponde al lenguaje en el que los estudiantes desarrollan los exámenes que serán evaluados. PHP, por su parte, se utiliza para construir una interfaz web sencilla que permite gestionar los exámenes almacenados y exportarlos hacia el sistema de análisis.

Gestión y Almacenamiento de Datos. MySQL. Se emplea para almacenar los exámenes escritos por los estudiantes y para estructurar el dataset que será posteriormente analizado. Este sistema de almacenamiento permite organizar los códigos fuente, exportarlos en formatos como JSON y alimentar el pipeline de procesamiento de manera eficiente.

Frameworks y bibliotecas de Machine Learning.

Scikit-learn (Random Forest).

Dentro del entorno de ML, se consideran varios modelos para la predicción de calificaciones y el análisis de características del código. Finalmente, scikit-learn destaca como el framework principal por las siguientes razones:

- Ofrece modelos robustos que funcionan adecuadamente incluso con datasets pequeños o medianos.
- Permite entrenar modelos como Random Forest Regressor, que manejan relaciones no lineales y reducen el riesgo de sobreajuste mediante el uso de múltiples árboles de decisión.
- Proporciona herramientas interpretables y de bajo costo computacional, ideales para contextos educativos.
- La elección de Random Forest se sustenta en su capacidad de generalizar correctamente aun con conjuntos limitados de ejemplos reales, un factor crítico en este proyecto.

Herramientas de Análisis Sintáctico y Procesamiento de Código.

El proyecto incorpora conceptos de análisis estático de código mediante expresiones regulares y técnicas tradicionales de parsing. Estas herramientas permiten:

- Identificar patrones estructurales en los exámenes en C.
- Detectar errores comunes como desbalance de llaves, uso de estructuras básicas y presencia de funciones esenciales.
- Generar métricas cuantitativas que alimentan el modelo de ML.

Aunque este análisis se implementa en capítulos posteriores, su base conceptual se presenta aquí como parte del marco tecnológico.

Herramientas Avanzadas Consideradas.

Dada la naturaleza del proyecto la evaluación automática e inteligente de código en C resultaba necesario analizar tecnologías de punta en el área de Machine Learning aplicado a código fuente. Por ello, se consideraron inicialmente distintos modelos de Deep Learning especializados en representación de código, particularmente aquellos basados en arquitecturas Transformer. Entre estos modelos destacan:

CodeBERT (Microsoft Research).

CodeBERT es un modelo basado en Transformers especializado en tareas como clasificación de código, generación automática, traducción entre lenguajes y búsqueda semántica. Se consideró por:

- Su capacidad para generar embeddings contextualizados del código.
- Su arquitectura preentrenada en grandes corpus de software público.

CodeT5.

CodeT5 es otro modelo Transformer especializado en comprensión semántica de código. Se evaluó por su capacidad para:

- Identificar estructuras profundas en el código, más allá de patrones superficiales.
- Resolver tareas avanzadas como autocompletado, reparación de código o análisis semántico.

Si bien el desempeño de ambos modelos es impresionante, su entrenamiento o fine-tuning en datasets reducidos tiende a provocar sobreajuste, limitando su utilidad en entornos educativos con pocos ejemplos.

Técnicas de recolección de datos.

Recopilación y digitalización de exámenes previamente aplicados por el docente Héctor Guerra a sus alumnos de la materia "Fundamentos de Programación".

Población: El universo total de documentos disponibles para el estudio asciende a 321 exámenes de programación en lenguaje C, correspondientes a evaluaciones previamente aplicadas para la materia de Fundamentos de programación.

Muestra: Se utilizó un muestreo no probabilístico por conveniencia para el entrenamiento y validación.

Subconjunto de Entrenamiento (Prototipo 1): Se emplearon 209 códigos correspondientes a los temas de estructuras de control y ciclos, los cuales fueron procesados y etiquetados para la fase inicial.

Subconjunto de Reserva: Los 112 códigos restantes se reservaron para fases posteriores de validación o para temas no cubiertos en la primera iteración.

Procedimiento.

El desarrollo se estructuró en cinco fases iterativas, alineadas con el modelo de prototipos:

Fase 1: Análisis de Requisitos y Criterios. Se definieron las reglas de negocio basadas en la rúbrica docente, identificando penalizaciones por sintaxis (ej. -0.1 por falta de librerías) y puntos por lógica (ej. +1.0 por uso correcto de ciclos). Dichos criterios se muestran más a detalle en la Tabla 1, algunos criterios llegan a cambiar de valor por la naturaleza el ejercicio, pero criterios como la inclusión de biblioteca, función *main*, correcto uso de llaves, uso de *scanf* y *printf* son criterios globales ya que sin importar el ejercicio se seguirá aplicando la misma penalización.

Tabla 1. Criterios de evaluación y ponderación docente para ejercicios.

Criterio	Descripción del elemento	Ponderación (Puntos)
Bibliotecas	Inclusión de #include <stdio.h>	-0.1 (Penalización)
Función Main	Estructura int main()	-0.1
Bloques	Correcta apertura y cierre de llaves { }	-0.1 por error
Indentación	Código escalonado correctamente	-0.3
Ciclos	Estructura for, inicio en 0, incremento unitario	1.0 (Acierto)
Lógica	Condición if (x%2!=0) (Paridad/Imparidad)	1.0
Asignación	Variables p=u y u=x (Lógica de reemplazo)	1.0
Salida	printf con formato correcto	-0.3

Fase 2: Modelado y Digitalización. Se diseñó la estructura de datos sencilla para facilitar el almacenamiento y futura exportación de los exámenes en formato Json para el entrenamiento, posteriormente se procedió a la transcripción limpia de los 321 exámenes, segmentándolos por tarea ("contar negativo", "Contar impares", etc.) para facilitar el aprendizaje supervisado. En la Tabla 2 se muestra Dicha estructura es ejemplificada.

Tabla 2. Diccionario de la entidad "Exámenes".

Campo	Tipo de Dato	Descripción
Id	Entero	Identificador único del examen o registro.
Tarea	Texto	Enunciado del problema o ejercicio de programación.
Código	Texto	Código fuente escrito por el alumno (digitalizado).
Calificación	Flotante	Puntuación obtenida según los criterios de evaluación.

Fase 3: Construcción del Prototipo Experimental. Se implementó una primera versión utilizando el modelo CodeT5 (Modelo Deep Learning). En esta fase se extrajeron las primeras métricas de los 209 códigos de entrenamiento.

Fase 4: Evaluación y Pivote Tecnológico. Se realizaron pruebas de desempeño que evidenciaron problemas de sobreajuste en CodeT5 debido a que por la naturaleza de este tipo de modelos se requieren cientos de exámenes para el correcto entrenamiento y evitar el sobreajuste. Se tomó la decisión metodológica de migrar hacia Random Forest, implementando un módulo de extracción de características (Feature Engineering) basado en reglas (conteo de if, else, printf, presencia de arreglos, etc.).

Fase 5: Desarrollo del Producto Final. Se integró el modelo Random Forest validado en la plataforma web por medio de una API y se desplegó para su uso en entorno controlado.

Análisis de datos.

Se realiza mediante técnicas de Aprendizaje Supervisado. Los códigos fuente se transforman en vectores numéricos mediante un algoritmo de extracción que contabiliza 14 características clave (longitud, estructuras de control, uso de operadores, etc.). Posteriormente, se emplean métricas estadísticas como el Error Cuadrático Medio y el Coeficiente de Determinación para comparar la calificación predicha por el sistema contra la calificación real asignada por el docente.

Validación.

Validación con conjunto independiente: A diferencia de la partición estadística tradicional, se optó por una validación mediante datos externos. Se generó un conjunto de pruebas exclusivo con códigos inéditos diseñados para contener casos de borde y errores específicos. Estos ejemplos no formaron parte del corpus de entrenamiento, garantizando que la evaluación mida la capacidad real del modelo para generalizar ante entradas nuevas y no su capacidad de memorización de los datos históricos.

Validación de Criterio: Comparación de los resultados automáticos contra la calificación del profesor, asegurando que la discrepancia sea estadísticamente aceptable.

3. Desarrollo.

1. Fase de Preparación de Datos:

La recolección de datos inició con la recepción de tres paquetes de exámenes históricos manuscritos, suministrados por el profesor Héctor Guerra. Dado que los documentos originales se encontraban en papel, se procedió a una etapa de digitalización y preprocesamiento manual.

```
//Examen - Jesus Jimenez, Calificacion: 5/5
/*Dada una lista de números
8,-2,9,3,-8,-6
contar negativos | salida. negativos=3
 */
 #include <stdio.h>
int main()
{
    int n,i,x;
    int c;

    printf("N: ");
    scanf("%d", &n);
    c=0;
    for (i=0; i<n; i++)
    {
        printf("X: ");
        scanf("%d", &x);
        if (x<0)
            c=c+1;
    }
    printf("La cantidad de números negativos es: %d", c);
}
```

Figura 1. Conjunto local de respuestas.

Para facilitar la inserción en la base de datos, se agruparon las respuestas en archivos de texto plano (formato .c), organizados por tipo de pregunta. En dichos archivos se documentó el código del estudiante junto con su calificación original en escala de 0 a 5, tal como se aprecia en la Figura 2.

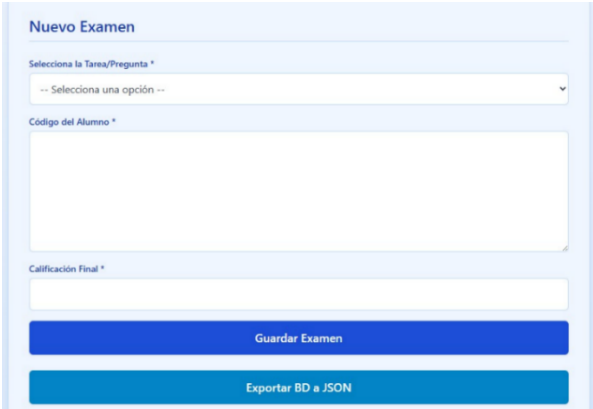

Figura 2. Formulario para ingresar exámenes a la BDD.

Posteriormente, se realizó la ingesta de estos datos en el Gestor de data desarrollado en PHP. Como se observa en la Figura 3, la interfaz permite la captura estructurada seleccionando el tipo de ejercicio (ej. "contar negativos"), ingresando el código fuente y asignando la calificación docente.

Figura 3. Estructura del conjunto de datos.

Esta plataforma centraliza la información y permite la exportación del dataset completo o segmentado por temas mediante vistas específicas, generando archivos JSON listos para el entrenamiento.

2. Fase de Entrenamiento:

El archivo JSON generado fue importado al entorno de Google Colab para el procesamiento de datos. Utilizando la librería Pandas, se estructuró el dataframe principal y se aplicó el algoritmo de extracción de características (extraer_features). Como se muestra en la Figura 4, el vector de entrenamiento resultante consolida las métricas de conteo sintáctico, la calificación heurística (reglas) y la calificación real, confirmando un volumen inicial de 209 muestras válidas.

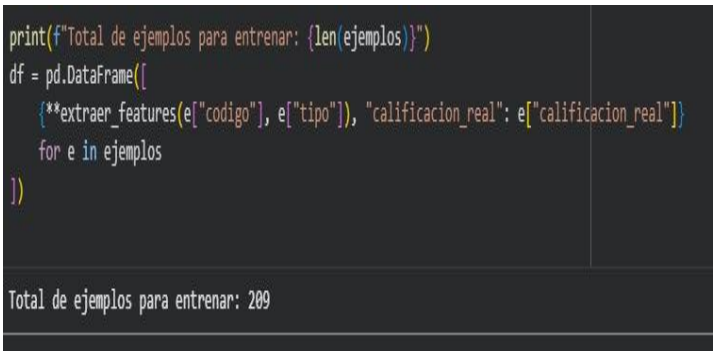

Figura 4. Conformación del vector de entrenamiento.

Finalmente, se procedió al entrenamiento del regresor Random Forest utilizando la librería Scikit-learn. La Figura 5 ilustra la configuración de los hiperparámetros base (n_estimators=100) y la ejecución del ajuste (.fit()), tras lo cual el modelo fue serializado para su exportación.

```python
from sklearn.ensemble import RandomForestRegressor
import pandas as pd

df = pd.DataFrame([
    {
        **extraer_features(e["codigo"], tipo=e["tipo"]),
        "calificacion_real": e["calificacion_real"]
    }
    for e in ejemplos
])


X = df.drop(columns=["calificacion_real"])
y = df["calificacion_real"]

modelo = RandomForestRegressor(n_estimators=100, random_state=42)
modelo.fit(X, y)

print("Modelo entrenado exitosamente con familias")
print(f"Features utilizadas: {list(X.columns)}")
```

Figura 5. Instanciación y entrenamiento del modelo predictivo.

3. Fase de Implementación:

Para la puesta en producción, el código de entrenamiento fue refactorizado en una API Restful utilizando el lenguaje Python. Esta API, alojada en la plataforma PaaS Render, orquesta el análisis híbrido cargando el modelo predictivo mediante la librería Joblib (Figura 6).

```python
import re
import pandas as pd
import joblib
import os

class EvaluadorInteligente:
    # Patrones lógicos estáticos
    PATRONES_LOGICOS = {
        # Familia Conteo
        "contar_negativos": r"if\s*\(\s*\w+\s*<\s*0\s*\)",
        "contar_pares":     r"if\s*\(\s*\w+\s*%\s*2\s*==\s*0\s*\)",
        "contar_impares":   r"if\s*\(\s*\w+\s*%\s*2\s*!=\s*0\s*\)",
        # Familia Mayor
        "mayor_negativos":  r"if\s*\(\s*\w+\s*<\s*0\s*\)",
        "mayor_pares":      r"if\s*\(\s*\w+\s*%\s*2\s*==\s*0\s*\)",
        "mayor_impares":    r"if\s*\(\s*\w+\s*%\s*2\s*!=\s*0\s*\)",
        # Familia Posición (Penúltimo y Último)
        "posicion_negativos": r"if\s*\(\s*\w+\s*<\s*0\s*\)",
        "posicion_pares":     r"if\s*\(\s*\w+\s*%\s*2\s*==\s*0\s*\)",
        "posicion_impares":   r"if\s*\(\s*\w+\s*%\s*2\s*!=\s*0\s*\)"
    }

    def __init__(self, ruta_modelo='modelo_evaluador_rf.joblib'):
        if os.path.exists(ruta_modelo):
            self.modelo = joblib.load(ruta_modelo)
            print(f"Modelo V4 cargado desde {ruta_modelo}")
        else:
            raise FileNotFoundError(f"No se encontró el archivo: {ruta_modelo}")
```

Figura 6. Implementación del motor de evaluación en la API.

Paralelamente, se desplegó un cliente web de simulación desarrollado en Node.js, diseñado para emular la experiencia del estudiante en la plataforma educativa real. Esta interfaz (Figura 7) integra controles de tiempo y validación de envíos.

Figura 7. Vista plataforma de evaluación.

Algoritmos.

El núcleo del sistema se basa en una arquitectura híbrida que combina reglas deterministas con inferencia estadística.
Algoritmo de Filtrado Heurístico (Reglas)

Antes de la inferencia del modelo, el código pasa por un filtro de expresiones regulares diseñado para validar la estructura lógica esperada. Como se detalla en la Figura 8, este módulo aplica penalizaciones directas sobre una calificación base de 5.0. Por ejemplo, la ausencia de la librería <stdio.h> o errores en la declaración de ciclos (for) descuentan puntos específicos. Este algoritmo garantiza que códigos con errores sintácticos graves sean detectados inmediatamente.

```
REGLAS = {
    "include_stdio": {
        "patron": r"#include\s*<\s*stdio\.h\s*>",
        "penalizacion": 0.1,
        "mensaje": "Falta la librería estándar <stdio.h>."
    },
    "main": {
        "patron": r"int\s+main\s*\(\s*\)",
        "penalizacion": 0.1,
        "mensaje": "La función main() no está declarada correctamente."
    },
    "declaracion_vars": {
        "patron": r"int\s+[\w,\s]+;",
        "penalizacion": 0.3,
        "mensaje": "No se detectó la declaración de variables (int ...)."
    },
    "scanf_n": {
        "patron": r"scanf\s*\(\s*\"%d\".*?&\w+\s*\)\s*;.*?for",
        "penalizacion": 0.3,
        "mensaje": "Falta la lectura de la cantidad de números (scanf inicial)."
    },
```

Figura 8. Reglas y sanción.

Algoritmo de Fusión (Ponderación Híbrida).

Debido a la limitación inicial del volumen de datos (209 ejemplos), se implementó una estrategia de ponderación para mitigar el margen de error de la Inteligencia Artificial. La calificación final (CF) se calcula mediante el promedio ponderado del Evaluador de Reglas (ER) y la Predicción de Machine Learning (ML):

$$Calificacion_{Final} = ER(0.5) + ML(0.5)$$

Como se evidencia en la Figura 9, este mecanismo permite que el sistema ofrezca una calificación equilibrada. Se proyecta que, conforme el *dataset* aumente y el error del modelo ML disminuya, esta ponderación se ajuste gradualmente hasta depender mayoritariamente de la inferencia estadística, reservando las reglas únicamente para la generación de retroalimentación textual.

```python
def calificar(self, codigo_alumno: str, tipo: str) -> dict:
    # 1. Extraer features
    features = self._extraer_features(codigo_alumno, tipo)

    # 2. Obtener nota de reglas y comentarios
    nota_reglas, lista_feedback = self._evaluar_reglas(codigo_alumno, tipo)

    # 3. Predicción IA
    try:
        X_nuevo = pd.DataFrame([features])
        prediccion_ml = self.modelo.predict(X_nuevo)[0]
    except Exception as e:
        print(f"Error ML: {e}")
        prediccion_ml = 0.0

    # 4. Cálculo final (Híbrido)
    nota_final = max(0, min(5, nota_reglas * 0.5 + prediccion_ml * 0.5))

    return {
        "nota_final": round(nota_final, 1),
        "feedback": lista_feedback, # Lista de strings con errores
        "detalles": {
            "score_reglas": nota_reglas,
            "score_ml": round(prediccion_ml, 1)
        }
    }
```

Figura 9. Lógica de promedio 50/50

Aplicación final.

El resultado tangible del proyecto es el "Evaluador Inteligente para exámenes de programación", una aplicación web funcional que integra los componentes de backend y frontend descritos.

Interfaz de Usuario y Control de Flujo: La aplicación final presenta al estudiante un entorno controlado para la resolución de exámenes. Incluye un temporizador en tiempo real que garantiza la igualdad de condiciones; si el tiempo expira antes del envío, el sistema bloquea la entrada y asigna automáticamente una calificación de "N/A", tal como se muestra en la Figura 10.

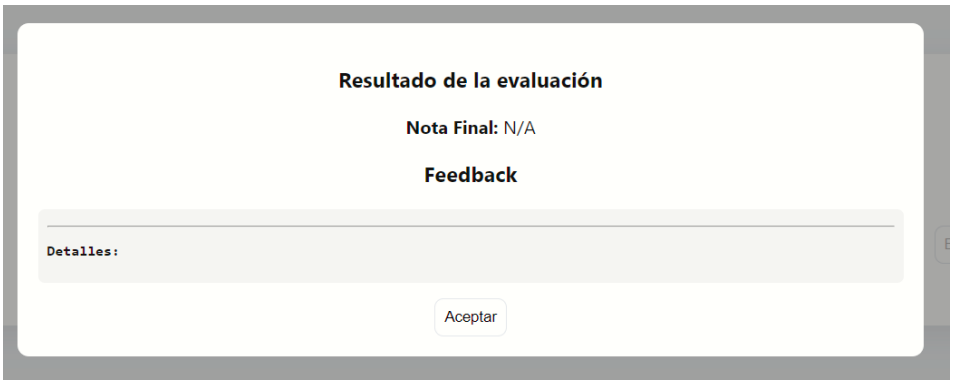

Figura 10. Pantalla de tiempo agotado

Proceso de Evaluación y Retroalimentación: Al realizar un envío exitoso dentro del tiempo límite, la información viaja a la API en Render, es procesada y retornada en formato JSON. La interfaz renderiza estos datos presentando al alumno dos elementos clave (Figura 11):

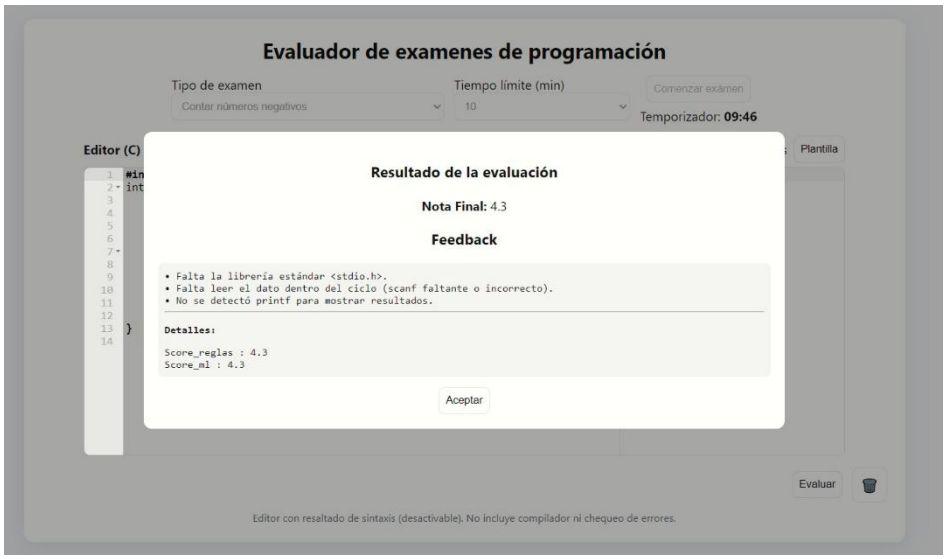

Figura 11. Pantalla de resultados y feedback

Desglose de Calificación: Muestra visualmente la contribución del evaluador de reglas y del modelo ML a la nota final.

Feedback Diagnóstico: Lista las áreas de mejora detectadas por el motor de reglas (ej. "Falta inicializar variable"), proporcionando una guía inmediata para el aprendizaje.

Conclusiones.

La investigación demuestra que la integración de técnicas de inteligencia artificial con mecanismos heurísticos constituye una estrategia efectiva para la evaluación automatizada de respuestas abiertas en ejercicios de programación. El modelo híbrido propuesto permitió combinar la precisión estructural de las reglas sintácticas con la capacidad predictiva del algoritmo Random Forest, logrando resultados consistentes aun con conjuntos de entrenamiento relativamente reducidos. La implementación del sistema evidenció que es posible disminuir significativamente la carga operativa del docente sin comprometer la calidad de la evaluación, además de proporcionar retroalimentación inmediata y objetiva al estudiante. Uno de los principales aportes del proyecto radica en la capacidad de identificar errores lógicos y sintácticos de manera simultánea, superando parcialmente las limitaciones tradicionales de los exámenes de opción múltiple y de los sistemas puramente manuales.

Créditos.

Los autores agradecen al Tecnológico Nacional de México por el financiamiento del proyecto 25452.26-P “Herramienta de evaluación automatizada basado en inteligencia artificial y método de adaptación: caso de estudio programas de capacitación en la industria autoelectromótriz”, de la convocatoria de apoyo a proyectos de desarrollo tecnológico e innovación 2026 y las facilidades del Instituto Tecnológico de Tuxtla Gutiérrez para la realización de este trabajo.

Referencias bibliográficas.

Aguado, J. M. (2024, 12 febrero). El 73% de los profesores afirma haber utilizado IA en alguna ocasión, para preparar las clases (64%) y para complementar los contenidos (50%). Actualidad Docente. https://actualidaddocente.cece.es/destacados-a-fondo/el-69-de-los-padres-y-madres-en-espana-ha-utilizado-alguna-vez-la-inteligencia-artificial-ia-y-el-78-quiere-aprender-mas-sobre-ella/

Celik, I., Dindar, M., Muukkonen, H., & Järvelä, S. (2022). The Promises and Challenges of Artificial Intelligence for Teachers: a Systematic Review of Research. TechTrends, 66(4), 616-630. https://doi.org/10.1007/s11528-022-00715-y

EduTec. (2022). Un análisis experimental de la relación entre las evaluaciones proporcionadas por la inteligencia artificial y las proporcionadas por los docentes en formación. EDUTEC. Revista Electrónica de Tecnología Educativa, 2022. https://edutec.es/revista/index.php/edutec-e/article/view/3509/1229

Shermis, M.D., & Burstein, J. (Eds) (2013). Handbook of Automated Essay Evaluation: Current applications and new directions. Journal of Writing Research, 5(2), 239-243. https://doi.org/10.17239/jowr-2013.05.02.4

Información de los autores.

José Manuel Damián Aguilar, Ingeniero en Sistemas Computacionales (ITTG) realizó su tesis en modelos de aprendizaje automático para evaluación académica. Actualmente labora como desarrollador de sistemas ERP en Node.js.

César Gustavo Farrera Molina es Ingeniero en Sistemas Computacionales (ITTG). Realizó su tesis en modelos de aprendizaje automático para evaluación académica. Actualmente labora como desarrollador *fullstack* para soluciones de software, *web apps* y automatizaciones.

Luis Tomás García Andrade, Ingeniero en Electrónica por el Instituto Tecnológico de Tuxtla Gutiérrez, Maestro en Administración de Redes y Telecomunicaciones por el Instituto de Estudios Superiores de Chiapas, Docente del Departamento de Eléctrica y Electrónica del Instituto Tecnológico de Tuxtla Gutiérrez.

Héctor Guerra Crespo es egresado del I. T. de Mérida (Yucatán, México) de la carrera de Ingeniería en Sistemas Computacionales en 1994, es Doctor en Sistemas Computacionales por la Universidad del Sur (Chiapas, México) en 2011. Es profesor en el área de Ingeniería en Sistemas Computacionales del Instituto Tecnológico de Tuxtla Gutiérrez y en el área de Licenciatura en Sistemas Computacionales de la Universidad Autónoma de Chiapas, en ambas desde 1995.

Galdino Belizario Nango Solís es Ingeniero en Sistemas Computacionales por el Instituto Tecnológico de Tuxtla Gutiérrez, Chiapas, en 1996. Es Maestro en Ciencias de la Computación por el Centro de Investigación en Computación del Instituto Politécnico Nacional, en 2001. Es Doctor en Desarrollo Tecnológico por la Universidad de Ciencia y Tecnología Descartes, en 2016. Es profesor en el área de Ingeniería en Sistemas Computacionales del Instituto Tecnológico de Tuxtla Gutiérrez, donde imparte clases en las materias de Programación Lógica y Funcional; y Lenguajes y Autómatas. Es profesor con perfil deseable PRODEP y miembro del Cuerpo Académico Tecnologías de la Información para el Desarrollo de Ventajas Competitivas en la LGAC Sistemas de Información desde 2022.

Néstor Antonio Morales Navarro es Doctor en Desarrollo Tecnológico. Docente del departamento de la carrera de Ingeniería en Sistemas Computacionales y Jefe del departamento del Centro de Cómputo del Tecnológico Nacional de México campus Tuxtla Gutiérrez. Miembro del Sistema Nacional de Investigadores, nivel Candidato. Se especializa en el área de Visión e Inteligencia Artificial.