CATEGORIZACIÓN AUTOMÁTICA APLICANDO TÉCNICAS DE MACHINE LEARNING DE ERRORES LÓGICOS EN TAREAS DE PROGRAMACIÓN CON CICLOS EN JAVASCRIPT

LÓPEZ BENAVIDES SANTIAGO DAVID

PROGRAMA DE INGENIERÍA DE SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD CESMAG
2025

CATEGORIZACIÓN AUTOMÁTICA APLICANDO TÉCNICAS DE MACHINE LEARNING DE ERRORES EN TAREAS DE PROGRAMACIÓN CON CICLOS EN JAVASCRIPT

Autores:
LÓPEZ BENAVIDES SANTIAGO DAVID

Informe final de trabajo de grado presentado como requisito para optar al título de Ingeniero de Sistemas, en modalidad investigación.

Asesor
MG. MORA PAZ HÉCTOR ANDRÉS

PROGRAMA DE INGENIERÍA DE SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD CESMAG
2025

Nota de aceptación

“Va la aceptación definida desde la universidad”.

Nota de exclusión

TABLA DE CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

LISTA DE ANEXOS

RESUMEN

El aprendizaje de la programación representa uno de los mayores desafíos en la formación de ingenieros de sistemas, especialmente en lo referente al uso correcto de los ciclos, estructuras fundamentales para la repetición de procesos en el código. Los errores lógicos que se presentan en los ciclos, como los relacionados con las condiciones de inicio, transformación o finalización, suelen ser difíciles de detectar y corregir, afectando tanto la comprensión del estudiante como la eficiencia del proceso de enseñanza. En este contexto, la presente investigación titulada "Categorización automática aplicando técnicas de Machine Learning de errores en tareas de programación con ciclos en JavaScript" propone una solución innovadora que combina la educación en programación con herramientas de inteligencia artificial para automatizar la detección y clasificación de errores en código.

El estudio se enmarca dentro de un enfoque cuantitativo y sigue la metodología CRISP-DM, ampliamente empleada en proyectos de minería de datos. Se construyó un conjunto de datos (dataset) que recopila ejemplos de código en JavaScript, centrados en el uso del ciclo while, abarcando tanto soluciones correctas como fragmentos con errores en diferentes etapas del ciclo. A partir de un proceso de generación y verificación, se obtuvieron más de 2.096.053 registros brutos, de los cuales se consolidó un dataset balanceado de 49.496 ejemplos, distribuidos en ocho categorías de clasificación: código correcto, error en estado inicial, error en transformación de estado, error en estado final y sus posibles combinaciones. Cada registro fue validado mediante pruebas automatizadas para asegurar la coherencia entre la etiqueta asignada y el comportamiento del código.

En la fase experimental, se realizaron múltiples pruebas con diferentes configuraciones de modelos de Deep Learning, aplicando técnicas de representación de texto y código mediante embeddings. El modelo basado en el embedding GraphCodeBERT mostró el mejor desempeño general, alcanzó un F1-Score de 0.9589, una accuracy del 95.9% y un loss de 0.17, demostrando una excelente capacidad para distinguir entre código correcto y los diferentes tipos de errores lógicos presentes en los ciclos. Estos resultados validan la eficacia de los modelos basados en representaciones semánticas profundas del código fuente.

Los hallazgos de esta investigación demuestran la viabilidad técnica y educativa de aplicar técnicas de Machine Learning en entornos de enseñanza de la programación. El modelo propuesto no solo permite clasificar errores de manera automática, sino que también facilita ofrecer una retroalimentación inmediata y personalizada a los estudiantes, reduciendo el tiempo de corrección por parte de los docentes y promoviendo una comprensión más profunda del funcionamiento de los ciclos. A futuro, esta metodología puede extenderse a otros lenguajes de programación y estructuras de control, contribuyendo al desarrollo de herramientas inteligentes de apoyo al aprendizaje y mejorando la calidad de la educación en ingeniería de software.

Palabras clave: aprendizaje automático, clasificación de errores, ciclos while, JavaScript.

ABSTRACT

Learning programming represents one of the greatest challenges in the training of systems engineers, especially regarding the correct use of loops, fundamental structures for repeating processes in code. Logical errors that occur in loops, such as those related to start, transformation or termination conditions, are often difficult to detect and correct, affecting both the student's understanding and the efficiency of the teaching process. In this context, the present research entitled “Automatic categorization applying Machine Learning techniques of errors in programming tasks with loops in JavaScript” proposes an innovative solution that combines programming education with artificial intelligence tools to automate the detection and classification of errors in code.

The study is framed within a quantitative approach and follows the CRISP-DM methodology, widely used in data mining projects. A dataset was built that collects examples of JavaScript code, focused on the use of the while loop, covering both correct solutions and fragments with errors at different stages of the loop. From a process of generation and verification, more than 2,096,053 raw records were obtained, of which a balanced dataset of 49,496 examples was consolidated, distributed into eight classification categories: correct code, error in initial state, error in state transformation, error in final state and their possible combinations. Each record was validated using automated tests to ensure consistency between the assigned label and the behavior of the code.

In the experimental phase, multiple tests were carried out with different Deep Learning model configurations, applying text and code representation techniques through embeddings. The model based on the GraphCodeBERT embedding showed the best overall performance, reaching an F1-Score of 0.9589, an accuracy of 95.9% and a loss of 0.17, demonstrating an excellent ability to distinguish between correct code and the different types of logical errors present in loops. These results validate the effectiveness of models based on deep semantic representations of source code.

The findings of this research demonstrate the technical and educational feasibility of applying Machine Learning techniques in programming teaching environments. The proposed model not only allows errors to be classified automatically, but also facilitates offering immediate and personalized feedback to students, reducing the correction time by teachers and promoting a deeper understanding of how loops work. In the future, this methodology can be extended to other programming languages and control structures, contributing to the development of intelligent learning support tools and improving the quality of education in software engineering.

Keywords: machine learning, error classification, while loops, JavaScript.

INTRODUCCIÓN

En la actualidad, el aprendizaje de programación se ha convertido en una habilidad fundamental para los profesionales en el campo de la ingeniería de sistemas. Sin embargo, los estudiantes suelen enfrentar desafíos importantes al momento de comprender y aplicar conceptos básicos, como el uso adecuado de ciclos en el código. Estos errores, principalmente de lógica, pueden generar dificultades en el proceso de aprendizaje y llevar a una percepción negativa sobre la complejidad de la programación [1] [2]. Este proyecto de investigación explora cómo las técnicas de Machine Learning pueden contribuir a la categorización y retroalimentación automática de errores en ciclos de programación en JavaScript, permitiendo identificar y corregir problemas comunes que enfrentan los programadores novatos.

La detección y corrección de errores en ciclos de programación es una tarea desafiante tanto para estudiantes como para instructores, y su correcta comprensión es fundamental para el éxito en cursos de programación. Sin embargo, los métodos actuales de enseñanza a menudo carecen de herramientas efectivas para proporcionar retroalimentación inmediata y específica sobre errores lógicos en ciclos, lo cual afecta la motivación y confianza de los estudiantes [2]. El desarrollo de un modelo que clasifique automáticamente estos errores y brinde retroalimentación constructiva ofrece una solución innovadora que podría optimizar el aprendizaje y reducir las tasas de abandono en materias de programación.

Este estudio tiene como objetivo general desarrollar un modelo de Machine Learning que permita la clasificación automática de errores en tareas de programación con ciclos en JavaScript, brindando una retroalimentación estática a los estudiantes. Los objetivos específicos incluyen: (1) crear un conjunto de datos con ejemplos de errores en ciclos de JavaScript, (2) aplicar técnicas de Machine Learning para entrenar un modelo que identifique estos errores y (3) evaluar la efectividad del modelo en términos de métricas de clasificación.

Para alcanzar los objetivos planteados, esta investigación utiliza un enfoque cuantitativo, basado en la metodología CRISP-DM, que es ampliamente utilizada en proyectos de minería de datos y Machine Learning [3]. El modelo de Machine Learning será entrenado y evaluado utilizando un conjunto de datos previamente estructurado que contiene diversos tipos de errores en ciclos de programación. El análisis se llevará a cabo mediante métricas de evaluación como F1-Score, precision, recall y accuracy para validar la efectividad del modelo [4].

El alcance de esta investigación se limita a la detección de errores en el uso de ciclos “while” en el lenguaje JavaScript. No se abarcarán otros lenguajes de programación ni otros tipos de estructuras de control. El modelo propuesto se enfocará en clasificar errores específicos y comunes en el ámbito de los ciclos, con el fin de ofrecer una herramienta especializada para el aprendizaje de programación en JavaScript.

I. PROBLEMA DE INVESTIGACIÓN

A. OBJETO O TEMA DE ESTUDIO

La efectividad de los modelos de clasificación en la detección de errores en ciclos en JavaScript.

B. LÍNEA DE INVESTIGACIÓN

Inteligencia artificial

Sub línea de investigación

Machine learning

C. PLANTEAMIENTO DEL PROBLEMA

Cuando los estudiantes intentan por primera vez aprender programación, se enfrentan a un nuevo mundo, desconocido y desafiante. Lamentablemente los cursos de introducción a la programación no logran enseñar con éxito los conceptos fundamentales [1], y el problema no se limita a una sola institución ni es causado por el uso de un lenguaje de programación en particular [5].

Saber diseñar un programa y dominar la sintaxis del lenguaje de programación es solo el comienzo de una tarea desafiante. Los programadores novatos a menudo tienen dificultades para comprender el funcionamiento de un ciclo debido a que no dominan los conceptos y la teoría sobre cómo programar iteraciones [6].

Estas dificultades pueden deberse a una comprensión insuficiente de los conceptos básicos de programación y a una limitada capacidad para aplicar las habilidades de programación en la práctica [1]. Al igual que falta de experiencia, lo cual puede conllevar a una desorientación respecto a la ubicación del código.

Los errores y fallos se encuentran comúnmente en las declaraciones condicionales y en los bucles debido a que estos errores son lógicos y no sintácticos [2]. Incluso cualquier programador, por más experiencia que tenga, puede llegar a tener un descuido que conlleve a un error lógico, lo que podría llevarlo a gastar mucho tiempo en identificar su origen.

La comprensión errónea sobre bucles disminuye la motivación, interés y confianza de los estudiantes [1] [2]. Esta impresión negativa y una perspectiva equivocada puede llevarlos a concluir

que la programación es extremadamente difícil y que requiere conceptos y habilidades complejas [1].

Los estudiantes ya de por si tienden a asociar connotaciones negativas con las materias de programación, esto se debe a las opiniones, comentarios y sugerencias negativas de compañeros que han tomado las materias anteriormente [2]. Estableciendo la creencia de que la programación requiere conceptos y habilidades complejas [1].

El aprendizaje en aspectos más avanzados de la programación se vuelve imposible, limitará la capacidad de los programadores novatos para desarrollar proyectos complejos, aumentará el número de estudiantes que abandonan sus estudios de programación, por lo cual inevitablemente reducirá la cantidad de futuros programadores y consolidará más aun la creencia que la programación es extremadamente difícil y solo accesible para unos pocos, generando un círculo vicioso.

D. FORMULACIÓN DEL PROBLEMA

¿Cómo ayudar a los programadores a comprender, detectar y corregir errores lógicos en ciclos?

E. OBJETIVOS

1) Objetivo general

Desarrollar un modelo de clasificación de errores en tareas comunes de programación con ciclos en JavaScript mediante Machine Learning.

2) Objetivos específicos

- Consolidar un conjunto de datos de tareas comunes de programación con ciclos en JavaScript para el entretenimiento del modelo.
- Aplicar el conjunto de datos para entrenar el modelo usando técnicas de Machine Learning
- Evaluar el modelo aplicando métricas del Machine Learning para evidenciar la efectividad del modelo.

F. JUSTIFICACIÓN

Los errores lógicos en ciclos de programación, representan un desafío constante tanto para programadores novatos como experimentados, estos errores no solo dificultan el aprendizaje inicial, sino que también pueden generar costos elevados en proyectos debido al tiempo invertido en la depuración y resolución de problemas. La implementación de un modelo basado en Machine

Learning que detecte y clasifique estos errores de manera automática tiene el potencial de abordar eficazmente este problema, optimizando tanto el aprendizaje como la productividad en el desarrollo de software.

Debido a la dificultad que se encuentra en la detección de errores lógicos, este proyecto ofrece una solución práctica para la identificación y corrección estos, beneficiando a estudiantes, desarrolladores profesionales e instituciones educativas. Para los programadores experimentados, el modelo acelerará la depuración y les permitirá enfocarse en tareas más complejas. En entornos educativos y empresariales, esta herramienta puede mejorar la formación en programación (mejorar la comprensión, detección y resolución de errores) y reducir los tiempos de desarrollo en un proyecto, incrementando así la eficacia y la calidad del software producido.

El proyecto contribuye al campo de la ingeniería de sistemas al ampliar el conocimiento existente sobre la eficacia de los modelos de Machine Learning en la detección y clasificación de tipos de errores en ciclos. Ofreciendo una base la cual puede ser replicada o adaptada en futuras investigaciones, como la implementación de otros lenguajes de programación o estructuras de control.

A largo plazo, un modelo capaz de clasificar correctamente, puede reducir significativamente los costos asociados a la depuración de código y acelerar los tiempos de desarrollo. En el ámbito educativo, contribuirá a una enseñanza más efectiva y al desarrollo de habilidades para mejorar la formación de programadores competentes. En entornos de producción, aumentará la eficiencia y la calidad del software, ayudando a programadores a minimizar errores en proyectos.

Los resultados de este proyecto pueden influir directamente en las decisiones estratégicas relacionadas con la enseñanza de la programación y el desarrollo de software. Instituciones académicas podrán integrar el modelo para una revisión automática de código, mientras que empresas tecnológicas podrán usarla para mejorar sus procesos de control de calidad.

En la actualidad, la programación se encuentra en el núcleo del desarrollo tecnológico, y JavaScript con un 62.3% destaca como el lenguaje más popular según encuestas recientes realizadas por Stack Overflow [7]. Esta investigación promueve a los diferentes sectores a adaptarse a las crecientes demandas de la transformación digital, asegurando su relevancia en un entorno tecnológico y en constante evolución. Además, el auge de la inteligencia artificial está transformando todos los sectores, desde el uso personal hasta en un ambiente laboral, exigiendo soluciones innovadoras que automaticen procesos.

Una investigación anterior es la plataforma Café [8], diseñada para proporcionar retroalimentación automática a cinco ejercicios de programación en un curso de Ciencias de la Computación, mientras que esta investigación busca una generalización mediante modelos de clasificación

automática (se profundiza más adelante), y una investigación doctoral que es tomada como base en esta propuesta de investigación [9], clasifica código en Python entre "correcto" o "incorrecto", en el último caso proporciona una clasificación más detallada, basándose en la teoría de iteraciones de Gries [10].

En resumen, este proyecto aborda una problemática clave en la programación mediante el desarrollo de un modelo capaz de detectar y clasificar errores en ciclos de JavaScript. Su implementación beneficiará tanto a estudiantes, como a programadores experimentados, al optimizar procesos de aprendizaje, depuración y desarrollo de software. Además, representa una contribución significativa al campo de la ingeniería de sistemas, alineándose con las tendencias actuales y futuras en el uso de tecnologías inteligentes para resolver problemas complejos.

G. DELIMITACIÓN

El proyecto se desarrollará en un equipo personal o en la plataforma Google Colab [11] en el lenguaje Python con bibliotecas de Machine Learning como scikit-learn [12] para experimentar y comparar con los diferentes algoritmos de clasificación, tales como K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Trees y Random Forest; Además de TensorFlow [13] para definir la arquitectura de la Artificial Neural Network (ANN) incluyendo Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Siamese Neural Network y Transformers, al igual de embeddings tanto para Natural Language Processing (NLP) BERT [14], como para el código fuente CodeBERT [15] o InferCode [16].

En la consolidación del conjunto de datos se espera abarcar entre 6.000 y 6.500 registros de tareas o retos comunes de programación que involucran concretamente el ciclo "while", cada registro tendrá su respectivo enunciado de la tarea, la solución en JavaScript y su etiqueta de clasificación.

Se pretender realizar una clasificación multiclase de 8 etiquetas, las cuales serán "correcto", error de "estado inicial", error de "transformación de estado", error de "estado final" y las diferentes combinaciones de los errores.

II. MARCO TEÓRICO

A. ANTECEDENTES

1) Internacionales

El artículo «Café: Automatic correction and feedback of programming challenges for a CS1 course» presenta una plataforma en línea que ofrece evaluación y retroalimentación automática de cinco ejercicios de programación a los estudiantes de Ciencias de Computación 1, permitiéndoles corregir sus errores y mejorar sus soluciones mediante múltiples intentos, no solo evalúa el código resultante, sino también el proceso cognitivo en su construcción, basándose en la metodología de Loop Invariant propuesto por Dijkstra [8]. Mientras que Café se centra en la evaluación y retroalimentación automática de cinco ejercicios a estudiantes, esta investigación se centra en la categorización de errores en tareas de programación mediante algoritmos de aprendizaje automático.

Los artículos «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding» y «CodeBERT: A Pre-Trained Model for Programming and Natural Languages» presentan un modelo pre-entrenado que aporta significativamente en la comprensión del lenguaje natural y la programación, su principal contribución es permitir la generación de representaciones semánticas comunes tanto para el lenguaje natural como para los lenguajes de programación [17] [18]. Lo que permite aprender mejor las conexiones semánticas entre el código y la tarea de programación deseada.

El articulo «code2vec: Learning Distributed Representations of Code» realiza una importante contribución al aprendizaje de representaciones distribuidas de fragmentos de código, convirtiendo dichos fragmentos en vectores de longitud fija que capturan las propiedades semánticas del código y su capacidad para descomponer el código en rutas dentro de su árbol de sintaxis abstracta (AST) [19]. El modelo aprende la representación atómica de cada ruta y cómo representar un conjunto de ellas en un solo vector, esto permite modelar la correspondencia entre fragmentos de código y etiquetas semánticas.

El articulo «SCC-GPT: Source Code Classification Based on Generative Pre-Trained Transformers» presenta un modelo de clasificación automática de fragmentos de código basado en transformadores generativos pre-entrenados (GPT), aprovechado su poder para generar representaciones profundas del código lo que permite identificar con precisión el etiquetado correspondiente a su lenguaje de programación de plataformas como Stack Overflow. SCC-GPT brinda una base de cómo los modelos de lenguaje pre-entrenados pueden aplicarse a la clasificación automatizada de código, no depende de etiquetas manuales preexistentes ni requiere un entrenamiento adicional extenso, lo que lo hace altamente eficiente y práctico para clasificar grandes volúmenes de código en múltiples lenguajes de manera automática y precisa [20]. Mientras SCC-GPT se centra en la clasificación de lenguajes de programación a partir de fragmentos de

código, el proyecto de se enfoca en identificar y clasificar errores específicos dentro de ciclos en JavaScript.

El articulo «Speeding Up Automated Assessment of Programming Exercises» propone un método para evaluación automática de ejercicios de programación, con una serie de fases para reducir el tiempo y los recursos necesarios en la evaluación de código presentado por los estudiantes, incluye un sistema de caché para evitar pruebas redundantes en envíos similares, análisis estático para detectar errores como bucles infinitos, el uso de un modelo de aprendizaje automático para predecir la corrección del código sin ejecutarlo y, finalmente la última fase, la ejecución de un conjunto tradicional de pruebas unitarias solo cuando sea necesario [21]. La introducción de un modelo de aprendizaje automático para evaluar la corrección del código sin ejecutarlo es especialmente relevante para la clasificación automática de errores, ya que permite identificar rápidamente patrones comunes de errores en el código. A diferencia de la investigación este no identifica que tipo error lógico se presenta ni se especializa en ciclos, sino que retroalimenta en qué fase no cumplió con el comportamiento esperado.

El articulo «Automatic Classification of Error Types in Solutions to Programming Assignments at Online Learning Platform» presenta un método para mejorar la retroalimentación en los cursos de programación en línea mediante la clasificación automática de errores en las soluciones a los ejercicios de programación, se basa en la generación de "edit scripts" que representan las modificaciones necesarias para transformar una solución incorrecta en una correcta, se agrupan estos "edit scripts" en clusters, identificando patrones comunes de errores, expertos etiquetan manualmente estos clusters según el tipo de error, y esta información se utiliza para entrenar un clasificador que puede identificar el tipo de error en nuevas soluciones incorrectas [22]. La evaluación y entrenamiento del método se realiza con un conjunto de datos de dos tareas de programación que incluye soluciones incorrectas y sus respectivas correcciones en Java.

2) Nacionales

Como principal referente de esta investigación está el articulo «IMProB-It : automatic feedback model for iterative programming tasks in introductory programming courses» [9]. El cual presenta un modelo capaz de identificar y clasificar errores lógicos en retos de programación que involucren ciclos en Python con una métrica de F1-Score de 89%. Fue entrenada con 6.000 datos que fueron categorizados entre "correcto", error en "estado inicial", "transformación de estado", "estado final" y las diferentes combinaciones entre los errores. Aporta una base sólida de referencia para fortalecer el modelo y aumentar el soporte de lenguaje de programación.

El artículo «Sistema de reconocimiento facial y de emociones aplicado a la educación básica y media de una institución educativa en Colombia con herramientas de la 4RI» describe el desarrollo de una solución tecnológica basada en el reconocimiento facial y de emociones, aplicada en una institución educativa en Colombia [23]. Aporta en el uso de Machine Learning y Deep Learning para resolver problemas específicos a través de la clasificación y el análisis de datos complejos, en

el que destacan la importancia del preprocesamiento de datos y experimentación de diferentes modelos para obtener mejores resultados. En el caso del reconocimiento facial, se utilizan algoritmos para clasificar emociones, mientras que, en esta investigación, los modelos se emplean para clasificar errores en código.

El articulo «Modelo de machine learning para la clasificación de municipios por cultivos ilícitos en Colombia de 2010 a 2020» presenta un estudio en el que se utiliza el algoritmo de clasificación no supervisado K-means para categorizar los municipios con presencia de cultivos de coca en Colombia. La investigación emplea la metodología CRISP-DM para minería de datos y el Análisis de Componentes Principales (PCA) para correlacionar variables. Se analizaron datos sobre hectáreas de coca, erradicación manual, destrucción de laboratorios y otras variables socioeconómicas entre 2010 y 2020 [24]. Brinda un aporte a la aplicación de algoritmos de clasificación y minería de datos, aunque en contextos muy diferentes, emplea la metodología CRISP-DM y técnicas de procesamiento y análisis de grandes volúmenes de datos.

El articulo «Computers' Interpretations of Knowledge Representation Using Pre-Conceptual Schemas: An Approach Based on the BERT and Llama 2-Chat Models» aborda la interpretación automática de esquemas pre-conceptuales por parte de computadoras, utilizando modelos de lenguaje avanzados como BERT y Llama 2-Chat. Los esquemas pre-conceptuales son representaciones gráficas que encapsulan el conocimiento mediante un lenguaje controlado, facilitando la comunicación de ideas complejas [25]. Se resalta el uso de modelos de aprendizaje automático para mejorar la comprensión y representación del conocimiento, enfatiza la importancia del procesamiento del lenguaje natural (NLP) y el entrenamiento específico de modelos para adaptarse a contextos particulares.

El articulo «Analysis of Satellite Images Using Deep Learning Techniques and Remotely Piloted Aircraft for a Detailed Description of Tertiary Roads» presenta un enfoque innovador para la identificación y clasificación de la infraestructura vial terciaria en la región de Taminango, Colombia, utilizando imágenes satelitales y técnicas de aprendizaje profundo [26]. Emplea redes neuronales convolucionales totalmente conectadas (FCN) para etiquetar imágenes y destaca la importancia del preprocesamiento y la estructuración adecuada de los datos.

3) Regionales

El articulo «Nature: A Tool Resulting from the Union of Artificial Intelligence and Natural Language Processing for Searching Research Projects in Colombia» se centra en el desarrollo de una herramienta llamada que combina inteligencia artificial y procesamiento de lenguaje natural para facilitar la búsqueda semántica de proyectos de investigación en la Universidad de Nariño. La metodología empleada incluye la creación de una ontología que organiza y estructura los datos, el uso de algoritmos de aprendizaje automático para entrenar modelos que interpretan y procesan consultas [27]. Establece el uso de técnicas de aprendizaje automático y procesamiento de lenguaje natural, empleando modelos entrenados para interpretar datos complejos y mejorar la eficiencia en

la clasificación y búsqueda de información, utilizando un modelo semántico para organizar proyectos de investigación.

Trabajo de master «Comparativo de kernels sobre predicción de oferta de fuentes alternativas de energía» se centra en la evaluación de diferentes funciones kernel en modelos de aprendizaje automático para predecir la oferta de energías limpias, presenta un análisis detallado que incluye la formulación del problema, el contexto actual de las energías alternativas y una revisión del estado del arte relacionado con el uso de máquinas de soporte vectorial (SVM) y redes neuronales artificiales (ANN). La investigación busca identificar cuál de las funciones kernel disponibles proporciona mejores resultados en términos de precisión y eficiencia en la predicción, contribuyendo así a optimizar la gestión y planificación de recursos energéticos sostenibles [28]. Realiza uso de técnicas avanzadas de aprendizaje automático para resolver problemas específicos, se centra en el desarrollo y evaluación de modelos que requieren un entrenamiento cuidadoso y una selección adecuada de algoritmos para optimizar el rendimiento y el uso de funciones kernel para mejorar las predicciones.

El artículo «Use of Trained Convolutional Neural Networks for Analysis of Symptoms Caused by Botrytis fabae Sard» se centra en la aplicación de redes neuronales convolucionales (CNN) para la identificación y clasificación de síntomas de la enfermedad conocida como “mancha de chocolate”, causada por el hongo Botrytis fabae. En este estudio, se utilizaron imágenes de hojas sanas y afectadas de cultivos experimentales de haba para entrenar modelos de CNN con el objetivo de automatizar el reconocimiento de síntomas y facilitar la toma de decisiones en la gestión agrícola. Al implementar técnicas de inteligencia artificial, el estudio busca mejorar la precisión en la identificación de enfermedades, lo que podría llevar a estrategias más efectivas de control y prevención en el cultivo [29].

El trabajo de grado «Gestión de información académica de solicitudes estudiantiles y docentes en la Jefatura de Software de la UNICESMAG mediante Chatbot aplicando PLN» se enfoca en el desarrollo de un chatbot que utiliza técnicas de procesamiento del lenguaje natural (PLN) para optimizar la gestión de información académica en la Jefatura de Software de la UNICESMAG. La investigación implica el uso de PLN para permitir al chatbot comprender y procesar las solicitudes de los usuarios formuladas en lenguaje humano [30].

El trabajo de grado «Implementación de una red neuronal para la clasificación de imágenes histológicas de Cáncer de Mama» se centra en el desarrollo y aplicación de una red neuronal convolucional (CNN) para mejorar la clasificación de imágenes histológicas relacionadas con el cáncer de mama. El proyecto aborda la importancia del cáncer de mama como una de las principales causas de mortalidad a nivel mundial y la necesidad de diagnósticos tempranos y precisos. La investigación se enfoca en superar los desafíos asociados con el diagnóstico del cáncer de mama, incluyendo la variabilidad en la interpretación de imágenes histológicas por parte de los patólogos. Los resultados muestran que la utilización de redes neuronales puede mejorar significativamente la precisión en el diagnóstico del cáncer de mama. Este trabajo es relevante por su enfoque

innovador en el uso de inteligencia artificial para mejorar diagnósticos médicos y abordar un problema crítico en salud pública [31].

B. SUPUESTOS TEÓRICOS

1) Teoría de programación con Iteraciones

En el ámbito de la programación y la verificación formal de software, uno de los mayores desafíos es garantizar que los algoritmos, especialmente aquellos que incluyen ciclos, se comporten de manera correcta bajo todas las condiciones posibles. Para lograr esto, es fundamental desarrollar un entendimiento profundo de cómo funcionan los ciclos. En este sentido, un concepto clave es el invariante de ciclo (loop invariant), una condición lógica que permanece verdadera antes, durante y después de cada iteración de un ciclo [10] [32].

Este concepto es esencial para la verificación formal de la corrección de los ciclos, ya que asegura que un ciclo es correcto si, durante la ejecución del programa, el invariante permanece verdadero [10]. Existen varios invariantes en un ciclo, y encontrar el adecuado es un paso crucial en la verificación. El invariante adecuado proporciona información fundamental sobre el ciclo, mostrando lo que está tratando de lograr y cómo lo consigue; de hecho, muchas personas consideran que es imposible entender completamente un ciclo sin conocer su invariante [32].

Comúnmente, encontrar el invariante adecuado para un ciclo es responsabilidad de un ser humano, ya sea la persona que realiza la verificación o el programador que escribe el ciclo en primer lugar [32].

2) Metodologías

La metodología es un conjunto estructurado de principios y procedimientos que especifica cómo llevar a cabo un proceso, definiendo no solo las fases principales, sino también las tareas que deben realizarse y cómo ejecutarlas, su función es guiar la planificación y ejecución de manera sistemática y no trivial, para alcanzar los objetivos [33]. Su importancia radica en la capacidad de proporcionar una secuencia lógica y ordenada que facilita la obtención de resultados coherentes y replicables, ayudando a transformar datos en conocimiento.

Aunque muchas metodologías fueron desarrolladas inicialmente para minería de datos, su uso es crucial para proyectos de machine learning. Estas metodologías ayudan a estructurar el ciclo de vida de los proyectos, desde la comprensión del problema hasta la implementación de modelos predictivos. Algunas de las más conocidas son CRISP-DM, SEMMA, KDD y Catalyst, que ofrecen distintos enfoques para llevar a cabo un análisis de datos eficiente y que se aborden todas las fases necesarias, desde la preparación de datos hasta la evaluación de los resultados [33].

a) CRISP-DM

Cross-Industry Standard Process for Data Mining es un modelo de metodología y procesos estándar para llevar a cabo proyectos de minería de datos. Fue desarrollado a finales de 1996 por un consorcio de líderes de la industria, incluyendo Daimler-Benz, ISL, NCR y OHRA, con el objetivo de crear un modelo estándar que fuera neutral en cuanto a la industria, herramientas y aplicaciones [3]. CRISP-DM se ha convertido en la metodología más utilizada en el desarrollo de proyectos de Data Mining [33].

Como se muestra en la Fig. 1, la metodología CRISP-DM se compone de seis fases principales [34]:

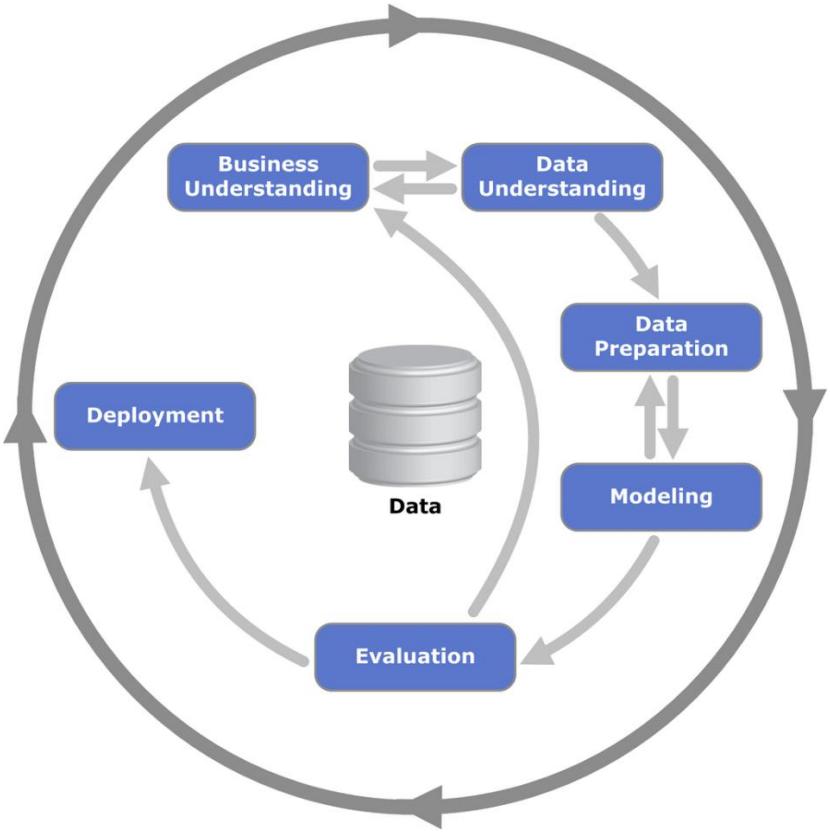


Fig. 1. Diagrama del proceso de las fases de CRISP-DM

Nota: fuente https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining

1. Comprensión del negocio: Definir los objetivos del proyecto y entender el contexto del negocio.
2. Comprensión de los datos: Recolectar y explorar los datos disponibles para familiarizarse con ellos.

3. Preparación de los datos: Limpiar y transformar los datos para que estén listos para el modelado.
4. Modelado: Aplicar técnicas de modelado a los datos preparados para crear modelos predictivos.
5. Evaluación: Evaluar el modelo para asegurarse de que cumple con los objetivos del negocio y es adecuado para su implementación.
6. Despliegue: Implementar el modelo en un entorno de producción y monitorear su rendimiento.

3) Machine Learning

Machine Learning es el proceso mediante el cual las computadoras adquieren la capacidad de resolver problemas al analizar patrones en los datos, en lugar de recibir instrucciones directas sobre los pasos a seguir. Desarrollan sus propios métodos para completar tareas basadas en la información que se les proporciona [35].

a) Algoritmos supervisados

El aprendizaje supervisado consiste en utilizar una muestra finita de datos etiquetados para entrenar y lograr aprender patrones o relaciones, con el propósito de hacer predicciones sobre nuevas muestras no vistas previamente, su objetivo es lograr una generalización. Los escenarios más comunes de aplicación son los problemas de regresión, jerarquización y clasificación (del cual se centra este proyecto) [36].

La elección del algoritmo o modelo juega un papel crucial en la generalización, un modelo rico o complejo puede llevar a un sobreajuste, en el que elige una función que se ajusta perfectamente a los datos de entrenamiento, pero generaliza mal a los datos no vistos. Por otro lado, un modelo demasiado simple puede llevar a un infrajuste, lo que implica que no logra una precisión suficiente ni en los datos de entrenamiento ni en los de prueba [36].

Algunos algoritmos supervisados para clasificación son:

Support Vector Machines (SVM): son un método de aprendizaje automático supervisado para la clasificación de datos en dos grupos. SVM se basan en la idea de mapear los datos de entrada en un espacio de características de alta dimensionalidad y luego construir un hiperplano optimo, que separe de la mejor forma posible, es decir, maximizando el margen que existe entre las clases [37]. El algoritmo solo puede encontrar un hiperplano optimo en problemas que permiten la separación lineal, cuando el problema es de clasificación no lineal, las funciones kernel permiten mapear los datos que no son linealmente separables a un espacio de características de mayor dimensión donde la separación se vuelve posible [28].

Para construir esos hiperplanos óptimos sólo hay que tener en cuenta una pequeña cantidad de los datos de entrenamiento, los llamados vectores de soporte, que determinan ese margen, como se observa en Fig. 2, los vectores de soporte, marcados con cuadros grises, definen el margen de mayor separación entre las dos clases [38].

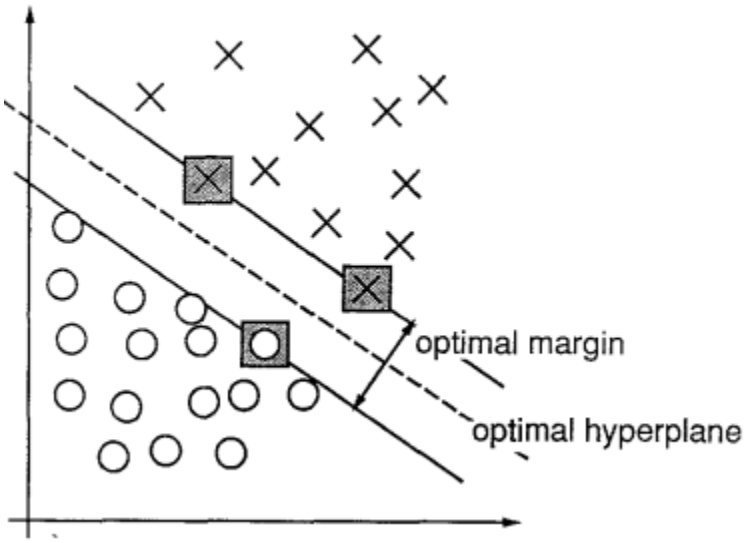


Fig. 2. SVM - Clasificación en un espacio bidimensional

Nota: fuente [38]

Aunque las SVM se diseñaron originalmente para problemas de clasificación binaria, la forma más común de realizar la clasificación multiclase es combinar varios clasificadores binarios [39]:

- Uno contra todos, la clasificación de las instancias se realiza mediante una estrategia de "el ganador se lo lleva todo", se entrena un clasificador binario por cada clase, clasificando una clase en concreto contra todas las demás combinadas, cada clasificador compite por la predicción del conjunto de características.
- Uno contra uno, la clasificación se realiza mediante una estrategia de votación de máximo ganador, se entrena un clasificador para cada combinación de par de clases, cada clasificador vota por una de las dos clases que maneja, la clase con más votos determina la clasificación del conjunto de características.

Decision Trees: son una técnica de aprendizaje supervisado que se utiliza tanto para clasificación como para regresión. Estos modelos funcionan mediante la creación de una estructura de datos de árbol, donde cada nodo interno representa una característica o atributo del conjunto de datos, cada rama representa una decisión o regla basada en el valor del atributo, y cada nodo hoja indica una

predicción o resultado. Los árboles de decisión tienen varias ventajas, entre ellas su facilidad de interpretación y visualización, así como la capacidad de capturar relaciones no lineales en los datos. Sin embargo, presentan limitaciones como la tendencia al sobreajuste cuando el árbol es demasiado profundo, lo que puede llevar a una generalización deficiente del modelo, este problema suele abordarse mediante la poda, un proceso que simplifica el árbol eliminando nodos poco relevantes o agrupando varios nodos en uno solo [40].

Random Forest: es un método de aprendizaje supervisado que combina múltiples árboles de decisión (Decision Trees) para realizar tareas de clasificación y regresión. Se basa en la idea de que, al combinar los resultados de múltiples arboles de decisión, se puede obtener una precisión mayor que la de un solo árbol y reducir la probabilidad de sobreajuste (una limitación común en los Decision Trees) [41] [42].

Cada árbol se entrena con un subconjunto de datos diferente, lo que permite que cada árbol desarrolle un modelo ligeramente diferente del mismo problema. Al combinar el resultado de múltiples árboles, el algoritmo utiliza el voto mayoritario (para clasificación) o el promedio (para regresión) de las predicciones individuales. Sin embargo, Random Forest puede tener desventajas como la complejidad computacional debido a la construcción de muchos árboles, así como la pérdida de interpretabilidad, ya que la combinación de múltiples árboles reduce la claridad de cada decisión individual, especialmente en bosques grandes [41] [42].

K-Nearest Neighbors (KNN): es un método de clasificación de aprendizaje supervisado no paramétrico, lo que significa que no hace suposiciones específicas sobre la distribución de los datos. Usado por su simplicidad y efectividad en problemas donde la relación entre las características y la salida es compleja o difícil de modelar de forma explícita [43].

Calcular la distancia entre el punto de prueba (el dato que se desea clasificar o predecir) y todos los puntos en el conjunto de datos de entrenamiento. El algoritmo selecciona los K vecinos más cercanos (de allí su nombre) y realiza la predicción con base en esos vecinos, en clasificación, la predicción se basa en el voto mayoritario de las clases de los vecinos más cercanos, es decir, el punto se clasifica en la clase que prevalece entre sus vecinos. El valor de K (el número de vecinos a considerar) es un hiperparámetro clave en el rendimiento del algoritmo [43].

b) Métricas

Las métricas de evaluación son herramientas fundamentales en el aprendizaje supervisado para medir de un modo objetivo la efectividad de los modelos y garantizar que sus predicciones sean precisas y útiles. Estas métricas varían según el tipo de problema (clasificación o regresión) y permiten una evaluación objetiva de la capacidad predictiva y la generalización del modelo [44].

Este proceso de evaluación generalmente implica la construcción de una matriz de confusión, que registra las predicciones del clasificador frente a las etiquetas reales, y la posterior aplicación de una métrica para resumir esta matriz en un solo número [45].

Matriz de Confusión: es una herramienta gráfica que facilita el análisis de la calidad del modelo en problemas de clasificación, muestra el conteo de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos como se ve en la Fig. 3, tomando como referencia una clasificación multiclase y centrándonos en la clase B, la definición de TP, TN, FP y FN respectivamente es la siguiente [4]:

- Verdaderos positivos: Pertenece la clase B y el modelo lo clasifica en la clase B.
- Verdadero negativo: No pertenece a la clase B y el modelo no lo clasifica en la clase B.
- Falso positivo: No pertenece a la clase B y el modelo lo clasifica en la clase B.
- Falso negativo: Pertenece a la clase B y el modelo no lo clasifica en la clase B.

		PREDICTED classification				
	Classes	a	b	c	d	Total
ACTUAL classification	a	50	37	24	39	150
	b	10	480	5	3	498
	c	14	10	765	1	790
	d	0	2	9	101	112
	Total	74	529	803	144	**1550**

		PREDICTED classification			
	Classes	a	b	c	d
ACTUAL classification	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN

Fig. 3. Matriz de confusión

Nota: fuente [4]

Accuracy: mide el porcentaje de predicciones correctas en relación al total de predicciones, conocido también como exactitud [45]. La ecuación (1) se utiliza para obtener la exactitud de un modelo de clasificación binaria, es decir $K = 2$, para un modelo multiclase ($K = n$) vea (2).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \qquad K = 2 \tag{1}$$

Nota: fuente [4]

En clasificación multiclase el accuracy también es conocido como Micro F1-Score [4].

$$Micro\ F1\ Score = \frac{\sum_{k=1}^{K} TP_k}{Total} \quad (2)$$

Nota: fuente [4]

Precision: calcula la proporción de predicciones positivas correctas en relación al total de predicciones positivas realizadas de una clase k, como se ve en (3), para obtener la precision promedio vea (4). Una interpretación de precision es que dado un conjunto de n predicciones diferentes ¿Cuál es la probabilidad de sacar aleatoriamente una predicción correcta? [45].

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \quad (3)$$

Nota: fuente [4]

$$AveragePrecision = \frac{\sum_{k=1}^{K} Precision_k}{K} \quad (4)$$

Nota: fuente [4]

Recall: mide la proporción de verdaderos positivos en relación al total de verdaderos positivos y falsos negativos de una clase k, como se ve en (5), para obtener la precision promedio vea (6). Su interpretación es que si seleccionamos una clase al azar y de ella extraemos una predicción al azar ¿Cuál es la probabilidad de que sea una predicción correcta? [45].

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \quad (5)$$

Nota: fuente [4]

$$AverageRecall = \frac{\sum_{k=1}^{K} Recall_k}{K} \quad (6)$$

Nota: fuente [4]

F1-Score: combina precisión y recall en una sola métrica mediante su media armónica como se ve en (7), lo que es útil para balancear ambas métricas en problemas donde tanto los falsos positivos como los falsos negativos son costosos [45].

$$F1\ Score = 2 \cdot \frac{AveragePrecision \cdot AverageRecall}{AveragePrecision + AverageRecall} \quad (7)$$

Nota: fuente [4]

Entropía cruzada: es una función de perdida (loss) utilizada en problemas de clasificación, especialmente en modelos de redes neuronales y problemas de clasificación multiclase. Esta función mide la diferencia entre dos distribuciones de probabilidad, una que representa la distribución real de los datos (y) y otra que representa la distribución predicha por el modelo ($\hat{y}$) como se ve en (8). Una entropía cruzada baja indica que el modelo está generando predicciones de alta confianza para las clases correctas, mientras que valores altos reflejan una baja precisión en las predicciones, el modelo pretende minimizar el valor de esta función [4].

$$CrossEntropy = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(\hat{y}_{ik}) \quad (8)$$

Nota: fuente [4]

4) Deep Learning

Deep Learning, también conocido como aprendizaje profundo, es un subcampo del aprendizaje automático que se centra en el uso de redes neuronales con múltiples capas (profundas) para aprender representaciones de datos con múltiples niveles de abstracción. Estas representaciones jerárquicas permiten a los modelos de Deep Learning abordar tareas complejas de procesamiento de información [46].

Una neurona artificial imita de forma simplificada el comportamiento de una neurona biológica, cada neurona recibe entradas, las procesa y genera una salida, mediante una suma ponderada como se evidencia en (9) donde x es el conjunto de características o entradas, w y b los parámetros internos de la neurona que debe ajustar (aprender) [47].

$$\hat{y} = \sum_{i=1}^{N} x_i w_i + b \quad (9)$$

Nota: fuente [47]

El concepto de Deep learning se fundamenta en la arquitectura de redes neuronales profundas, que consiste en una estructura de nodos (neuronas) organizadas en capas como se ven en la Fig. 4: una capa de entrada (Input), múltiples capas ocultas (Hidden) y una capa de salida (Output). Cada capa transforma la información de la capa anterior mediante funciones de activación. Aprenden a través

de un proceso de entrenamiento basado en el ajuste de parámetros (pesos y bias) mediante un algoritmo de optimización de la función de coste (minimizar el loss) [44].

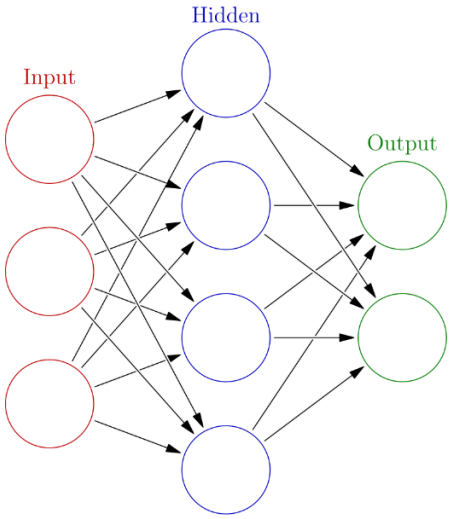


Fig. 4. Artificial Neural Network (ANN)

Nota: fuente https://en.wikipedia.org/wiki/Neural_network_(machine_learning)

Las funciones de activación son funciones matemáticas utilizadas en las capas de una red neuronal para transformar la salida en un valor, este determina si una neurona "se activa" o no, permitiendo que se capture relaciones en los datos. Las funciones de activación introducen no linealidad, que es fundamental para que puedan modelar relaciones complejas entre entradas y salidas. Sin una función de activación, independientemente de su profundidad, una red neuronal solo sería capaz de aprender relaciones lineales, lo cual limitaría considerablemente su capacidad de resolver problemas complejos [44].

a) Convolutional Neural Network (CNN)

Las Redes Neuronales Convolucionales son un tipo de red neuronal profunda que ha demostrado un inmenso poder predictivo en una variedad de tareas de aprendizaje, particularmente en visión artificial, gracias a su capacidad para aprender y extraer características espaciales y jerárquicas directamente de los datos [48]. Se diseñan específicamente para procesar datos que vienen en forma de múltiples matrices, imágenes, señales y secuencias, incluyendo el lenguaje [46].

Capas Convolucionales: son las capas centrales de una CNN, estas utilizan filtros que se deslizan sobre los datos de entrada, aplicando convoluciones (operaciones matemáticas) en pequeñas secciones. Cada filtro genera un mapa de características que representa la activación de las neuronas al detectar patrones específicos (patrones más complejos en capas profundas) en ciertas áreas de los datos. Los filtros aprenden sus propios pesos durante el entrenamiento, optimizando los patrones que detectan, mejorando el rendimiento [46].

Capas de Pooling: es la capa encargada de reducir la dimensionalidad de los mapas de características, conservando las características más relevantes, esto reduce la complejidad computacional y previene el sobreajuste al simplificar la representación sin perder la información más importante. Existen varios tipos de pooling, siendo el más común el Max Pooling, que toma el valor máximo en una región determinada del mapa de características [49].

b) Recurrent Neural Networks (RNN)

Las Redes Neuronales son un tipo de arquitectura de red neuronal profunda diseñada para procesar secuencias de datos, lo cual las hace especialmente adecuadas para tareas en las que el orden y la relación temporal entre los elementos son importantes, como en el procesamiento de lenguaje natural, el análisis de series temporales y el reconocimiento de voz. Se denominan recurrentes porque la salida calculada para la entrada actual depende tanto de la entrada como de los resultados de los cálculos anteriores, de hecho, la salida actual retroalimenta a la red y se utiliza (junto con la entrada actual) para producir la siguiente salida, dándoles una especie de "memoria" que retiene información de estados anteriores para mejorar las predicciones [50].

Long Short-Term Memory (LSTM): son una arquitectura de redes neuronales recurrentes diseñada para abordar el problema de capturar dependencias a largo plazo en datos secuenciales, una limitación que presentan las RNN tradicionales debido al desvanecimiento del gradiente. Las LSTM se destacan por su célula de memoria y sus tres puertas (de olvido, entrada y salida), que regulan el flujo de información y controlan qué información se recuerda o se olvida a medida que se avanza en la secuencia, en cada paso de la secuencia, la puerta de olvido decide cuánta información previa debe desecharse, la puerta de entrada determina qué información nueva debe almacenarse, y la puerta de salida decide qué información de la célula de memoria será utilizada en el paso actual. Esta estructura de puertas permite a las LSTM "recordar" información relevante durante períodos largos [51] [52].

Gated Recurrent Units (GRU): son una variante de las LSTM que simplifican su estructura al utilizar solo dos puertas: actualización y reinicio, la puerta de actualización regula la cantidad de información que se mantiene de un estado a otro, mientras que la puerta de reinicio decide qué parte de la información previa se ignorará al procesar una nueva entrada en la secuencia. Al reducir el número de puertas y simplificar las operaciones, ofrecen un balance entre precisión y eficiencia computacional, logrando un rendimiento comparable a LSTM, pero con menor complejidad de cálculo. Esta eficiencia permite ser entrenadas más rápidamente y con menos recursos [53].

Se ha demostrado que tanto las LSTM como las GRU son eficaces para una gran variedad de tareas, como el reconocimiento de la actividad humana, el reconocimiento de voz y el procesamiento del lenguaje natural [54].

c) Transformers

Los Transformers son una arquitectura de red neuronal profunda diseñada para procesar datos secuenciales, revolucionando el campo del procesamiento de lenguaje natural (NLP). A diferencia de las redes recurrentes, que procesan datos de manera secuencial, los Transformers utilizan un mecanismo de atención que permite que cada elemento en una secuencia preste atención a todos los demás elementos simultáneamente, esta característica es fundamental para capturar contextos y dependencias entre palabras o tokens en una oración, mejorando la comprensión del significado y la relación entre diferentes partes del texto [55].

La arquitectura de los Transformers se basa en un modelo de codificador-decodificador, el codificador transforma la entrada en una representación interna, mientras que el decodificador utiliza esta representación para generar la salida. El mecanismo de autoatención calcula la importancia de cada palabra en relación con las demás, y la técnica de multi-head attention permite que múltiples relaciones se analicen simultáneamente, esta estructura paralelizada no solo acelera el entrenamiento en comparación con modelos recurrentes, sino que también hace que los Transformers sean escalables y eficaces para aprender patrones complejos en grandes volúmenes de datos, convirtiéndose en la base de muchos modelos avanzados en NLP [55].

5) Natural Language Processing (NLP)

El procesamiento de lenguaje natural, es un campo de la inteligencia artificial (IA) enfocado en la interacción entre las computadoras y el lenguaje humano. El objetivo principal del NLP es desarrollar modelos y algoritmos que permitan a las máquinas entender, interpretar, generar y responder a texto y habla de manera similar a como lo haría una persona. Este campo abarca un conjunto de técnicas y métodos para tareas como el análisis de sentimientos, la traducción automática, el resumen de texto, la generación de texto y el reconocimiento de voz [56] [57].

a) Preprocesamiento

El preprocesamiento es una etapa fundamental que consiste en preparar los datos para su análisis y entrenamiento de los modelos de aprendizaje automático [58]. Dado que el lenguaje natural es complejo y desestructurado, el preprocesamiento permite reducir esta complejidad, eliminar ruido y estructurar el texto de una forma que los algoritmos puedan entender. Este proceso incluye varias etapas, como la tokenización, que separa el texto en palabras o tokens individuales [59] [47]; la eliminación de stopwords, que quita palabras comunes sin carga semántica significativa (como “y” o “el”); la normalización, que convierte palabras a una forma estándar, mediante técnicas como la lematización o el stemming, para reducir las palabras a sus formas base o raíz y la eliminación de signos de puntuación y la conversión a minúsculas, que ayudan a uniformar el texto [57].

b) Embeddings

Los embeddings son representaciones vectoriales densas de datos que permiten transformar palabras, frases o incluso fragmentos de código en vectores de números que capturan relaciones

semánticas. En el contexto de NLP, los embeddings han revolucionado el procesamiento de lenguaje, ya que permiten que las máquinas comprendan la similitud y el contexto entre palabras, frases o sentencias, en lugar de tratar cada palabra como una entidad completamente aislada, estas representaciones se obtienen a través del entrenamiento de modelos que, al capturar patrones y relaciones entre los datos de entrada, proyectan el significado semántico en un espacio de menor dimensión donde palabras similares se representan con vectores cercanos [57] [60].

En el procesamiento del lenguaje natural, un embedding es una representación matemática de una palabra o frase en forma de vector. Estos vectores, también llamados embeddings, capturan aspectos del significado de las palabras y permiten a las computadoras "entender" el lenguaje de una manera más profunda

Lenguaje Natural: Los embeddings de lenguaje natural representan palabras en un espacio vectorial donde la cercanía entre vectores indica similitud semántica. Modelos como Word2Vec, GloVe y, más recientemente, BERT y GPT, han permitido mejorar significativamente la capacidad de los sistemas de NLP para entender y procesar texto, en estos modelos, las palabras con significados y contextos similares ocupan posiciones cercanas en el espacio vectorial, lo que facilita que los algoritmos detecten relaciones semánticas, sinónimos y contexto de uso en textos extensos. Estas representaciones son esenciales para tareas como el análisis de sentimientos, la clasificación de texto, la traducción automática y la generación de texto, permitiendo que los modelos "interpreten" el significado en lugar de solo procesar secuencias de caracteres [61] [17].

Código Fuente: Los embeddings de código fuente son representaciones vectoriales de fragmentos de código que buscan capturar la lógica y funcionalidad de las instrucciones, más allá de su estructura sintáctica, al igual que en el lenguaje natural, los embeddings de código permiten agrupar y comparar fragmentos de código con similar funcionalidad o propósito, independientemente de detalles específicos de sintaxis o estilo de codificación. Modelos de embeddings como Code2Vec, CodeBERT y GraphCodeBERT son ejemplos de arquitecturas entrenadas específicamente en grandes volúmenes de código fuente y diseñadas para capturar las relaciones entre funciones, variables y estructuras de control, útiles en tareas como la búsqueda de fragmentos de código, la generación automática de código, la detección de similitudes o duplicados y la traducción entre lenguajes de programación, ofreciendo una base para desarrollar herramientas avanzadas de asistencia al desarrollo y análisis de software [19] [18].

C. VARIABLES DEL ESTUDIO

1) Variables independientes

- Modelo de aprendizaje automático
- Configuración del modelo
- Representación de los datos

- Preprocesamiento de los datos

2) Variables dependientes

- Métricas de evaluación
- Sobreajuste e infrajuste

3) Definición nominal de variables

- ***Modelo de aprendizaje automático:*** es un algoritmo que aprende o se entrena con un conjunto de datos para mejorar su desempeño en una tarea en especifica [62], en este caso clasificación de errores en tareas comunes con ciclos while en JavaScript.
- ***Configuración del modelo:*** La mayoría de los algoritmos de aprendizaje automático cuentan con hiperparámetros, que son varias configuraciones para controlar su comportamiento [44]. Al igual que la arquitectura en Deep Learning, que se refiere a la estructura general de la red neuronal, como sus capas y como deben conectarse entre sí [44].
- ***Representación de los datos:*** De qué forma y que atributos se presentan en el conjunto de datos.
- ***Preprocesamiento de los datos:*** el conjunto de datos se suele preprocesar para transformarlo en un conjunto en el que se espera que el problema de aprendizaje sea más fácil de resolver [58]. En el ámbito del Natural Language Processing (NLP) se encuentran la tokenización (de texto a una lista de tokens, es decir palabras o caracteres, según su diseño [59] [47]) y los embeddings (de token a vector, representando una noción de similitud vectorial con cada token según su semántica [60]).
- ***Métricas de evaluación:*** son funciones que mide la calidad de las clasificaciones del modelo utilizando el conjunto de validación (distinto al conjunto de entrenamiento) [59].
- ***Sobreajuste e infrajuste:*** el sobreajuste es cuando el modelo se ajusta demasiado a los datos de entrenamiento y no se realiza una generalización para nuevos datos, se suele caracterizar por tener una pérdida promedio muy pequeña en el conjunto de entrenamiento, pero una gran pérdida promedio en el conjunto de prueba, por otro lado, el infrajuste es cuando el modelo no es lo suficientemente complejo para resolver el problema [63].

4) Definición operativa de variables

- ***Modelo de aprendizaje automático:*** se probará con los algoritmos de aprendizaje automático, específicamente k-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Trees, Random Forests and Artificial Neural Network (ANN) [62].
- ***Configuración del modelo:*** se manipulará los hiperparámetros dependiendo del algoritmo de aprendizaje automático y en el caso de la red neuronal artificial su arquitectura.
- ***Representación de los datos:*** se establece la descripción la tarea común, código fuente dividido en el apartado de estado inicial, transformación de estado y estado final; junto con las diferentes etiquetas de clasificación del error.
- ***Preprocesamiento de los datos:*** la tokenización puede tener varios enfoques basados en reglas y estadísticas o en simples palabras, cada una de estos tratan diferente a los signos de puntuación, espacios y otras estructuras gramaticales [59] [47].
- ***Métricas de evaluación:*** la medición de la calidad de la clasificación se realiza con diferentes funciones, como lo son Accuracy, Recall, Precision y F1-Score [44].
- ***Sobreajuste e infrajuste:*** se evidenciará con la relación de la métrica Accuracy y con la función de perdida Cross-entropy [64].

D. FORMULACIÓN DE HIPÓTESIS

1) Hipótesis de investigación

Los modelos de clasificación cuentan con una relación F1-Score tiempo superior que un modelo aleatorio, en todas las etiquetas de clasificación de errores en JavaScript que involucran ciclos while.

2) Hipótesis nula

Los modelos de clasificación no cuentan con una relación F1-Score tiempo superior que un modelo aleatorio, en ninguna de las etiquetas de clasificación de errores en JavaScript que involucran ciclos while.

3) Hipótesis alterna

Los modelos de clasificación cuentan con una relación F1-Score tiempo superior que un modelo aleatorio, en algunas etiquetas de clasificación de errores en JavaScript que involucran ciclos while.

III. METODOLOGÍA

A. PARADIGMA

El paradigma de este proyecto de investigación es de tipo positivista, característico de una investigación de conocimiento científico y ciencias exactas para la verificación de conocimiento a través de predicciones [65] [66]. El proyecto pretende estudiar el desempeño de los diferentes modelos de clasificación en la detención de errores de programación en JavaScript que involucran el ciclo "while", realizando diferentes variantes de configuración de hiperparámetros, de arquitectura y de preprocesamiento de los datos.

B. ENFOQUE

De acuerdo con el paradigma positivista, la investigación se enmarcará en el enfoque cuantitativo [65] [66], debido a que se pretende realizar una evaluación de la eficacia de los diferentes modelos de clasificación, esta evaluación es medible y definidas por las métricas establecidas en el MARCO TEÓRICO.

C. MÉTODO

La investigación abordara el método empírico-analítico también conocido como método científico, caracterizado por observar y analizar las cusas y efectos [65]. Con el fin de determinar que modelos tienen un rendimiento superior y que preprocesamiento o ajuste de hiperparámetros es mejor para cada uno.

D. TIPO DE INVESTIGACIÓN

El tipo de investigación establecido es la investigación aplicada, se ajusta al presente proyecto debido a que pretende la generación de conocimiento con aplicación directa a mediano o largo plazo en la sociedad y en el sector productivo [67]. Esta investigación pretende contribuir a la disminución de búsqueda de modelos, sintonización de hiperparámetros y técnicas que preprocesamiento para la clasificación de errores de programación en JavaScript que involucran el ciclo "while".

E. DISEÑO DE LA INVESTIGACIÓN

El diseño de la investigación establecido es cuasi-experimental, debido a que se manipulan deliberadamente las variables independientes para ver su efecto sobre las variables dependientes establecidas en la sección VARIABLES DEL ESTUDIO [65].

F. POBLACIÓN

Al igual que las investigaciones tituladas «Evaluación De Métodos De Reducción De Dimensión Para La Preservación Topológica De Los Datos Mediante Métricas Rnx» y «Comparativo de funciones kernel en la predicción de enfermedades cardiovasculares en Redes Neuronales Artificiales (ANN) y Máquinas de Soporte Vectorial (SVM)», esta investigación tiene como fin el estudio de la parte epistemológica y matemática del objeto de estudio, que en este caso es la efectividad de los modelos de clasificación en la detección de errores en ciclos en JavaScript, por lo tanto, no se requiere población ni muestra [68] [69]. La naturaleza de esta investigación establece un enfoque basado en datos o data-driven approach en el cual las relaciones entre variables se extraen directamente de los datos experimentales mediante el uso de algoritmos de aprendizaje automático [70].

G. TÉCNICAS DE RECOLECCIÓN DE INFORMACIÓN

Una de las técnicas de recolección de información es la extracción mediante fuentes o datos secundarios, información previamente recopilada por otras investigaciones, instituciones o bases de dato. Será utilizada para recopilar la descripción de las diferentes tareas o retos comunes que existen en la programación que involucran el ciclo “while”.

Al igual que utilización de prompts de IA para obtener diferentes variantes o formas de establecer la descripción del planteamiento del reto de programación y su posible solución en JavaScript. Como se presentó en los resultados de la tesis llamada «Gestión de información académica de solicitudes estudiantiles y docentes en la Jefatura de Software de la UNICESMAG mediante Chatbot aplicando PLN» el uso de GPT-3 fue el más efectivo para ampliar el conjunto de datos [30].

Otra técnica de recolección de información es la de experimentos controlados, debido a que, en la presente investigación, se manipulara las variables independientes, para observar su efecto en las variables dependientes mencionadas en la sección VARIABLES DEL ESTUDIO.

1) Validez de las técnicas de recolección

El uso de datos secundarios hace posible la obtención de retos-tareas de programación comunes confiables, lo cual reduce el tiempo de recolección de datos, por lo que directamente reduce costos. La estructura del dataset ya está probada al igual que los diferentes experimentos que se han realizado en la tesis doctoral que brinda la base-viabilidad de este proyecto de investigación [9].

Los modelos de inteligencia artificial como GPT ya han alcanzado una buena madurez tanto en Natural Language Process (NLP) como en la generación de código fuente [71], se probará que la solución generada sea correcta, en caso contrario se implementará una de forma manual al igual

que la inserción de errores en el código fuente y la etiquetación, garantizado que los datos estén etiquetados correctamente.

2) Confiabilidad de las técnicas de recolección

El control de la manipulación de las variables independientes permite resultados consistentes y repetibles (con el mismo contexto, representación de datos, preprocesamiento, etc.), contar con métricas hace posible determinar si el modelo generaliza y clasifica correctamente, afirmando o refutando la hipótesis planteada en la sección FORMULACIÓN DE .

H. INSTRUMENTO DE RECOLECCIÓN DE DATOS

La herramienta establecida para medir la generalización o la correcta clasificación, se hará atreves de graficas épocas vs loss en entrenamiento y de validación como también para la exactitud, y métricas de evaluación obtenidas a partir de los datos de prueba donde se obtendrá la matriz de confusión que evidencia los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos, vistos en el MARCO TEÓRICO, como también reportes de clasificación donde se relacione la exactitud, precisión, recall, F1-Score.

IV. RESULTADOS DE LA INVESTIGACIÓN

En este capítulo se presentan los resultados obtenidos siguiendo la metodología CRISP-DM [3]. Se describen las etapas de construcción del dataset, la aplicación de los algoritmos de aprendizaje automático y la evaluación final del modelo entrenado. Cada fase del proceso se documenta en detalle, junto con las decisiones tomadas y las métricas clave obtenidas.

A. CONSTRUCCIÓN DEL DATASET

En esta sección se detallan los resultados de las primeras tres etapas de la metodología CRISP-DM, comprensión del negocio, comprensión de los datos y elaboración del dataset, para poder dar por cumplido el primer objetivo específico.

1) Comprensión del negocio

En esta fase se debe de identificar el problema a resolver, definir los objetivos y contexto del proyecto, para dar una posible solución. Como problema a resolver, se identificó que existe una dificultad de los estudiantes al comprender el funcionamiento de un ciclo (como se definió en la sección PLANTEAMIENTO DEL PROBLEMA), para ayudar a los programadores a comprender, detectar y corregir errores lógicos en ciclos, la solución planteada (objetivo) es desarrollar un modelo de Machine Learning que clasifique automáticamente errores comunes en ciclos while de JavaScript (Los objetivos específicos se definieron en la sección OBJETIVOS).

Para ello se estableció crear un dataset etiquetado con ejemplos de código que contengan o no errores lógicos en ciclos. Se identificaron las categorías de errores relevantes basándose en la literatura previa (como se ve en la sección ANTECEDENTES), un ciclo contiene un error si cualquiera de sus etapas no cumple con su función especificada en la Tabla I. De modo que el modelo pueda distinguir entre código correcto y los distintos errores lógicos en la inicialización, transformación y condición de salida del bucle.

Tabla I
Etapas de un bucle y sus funciones

Etapa	Función
Estado inicial	Son las precondiciones necesarias para que un ciclo funcione correctamente, define las condiciones que deben cumplirse antes de que el bucle comience. La función del estado inicial es que el propósito (invariante) del ciclo se cumpla antes del ciclo y al menos en la primera iteración.
Transformación de estado	Son las modificaciones necesarias para que el invariante del ciclo se cumpla en cada iteración.
Estado final	Define en qué momento el ciclo termina, y que transformaciones debe realizar para cumplir con el invariante.

Nota: Elaboración propia

Las diferentes categorías de errores de un ciclo, dependen de en qué etapa se produjo el error. Puede haber más de una etapa involucrada, por lo que hay categorías que pueden combinar cualquiera de las tres categorías. De este modo, hay siete categorías de error, en total contando con la categoría correcta, hay ocho.

2) Comprensión de los datos

En esta fase se debe de recolectar y explorar los datos disponibles para familiarizarse con ellos, como no se encontró un dataset disponible para cumplir con el objetivo establecido es necesario elaborarlo, para ello si se encontró con un dataset base [9] que se tomará como guía para la elaboración del dataset, cuenta con aproximadamente 6.000 datos con la siguiente estructura:

Tabla II
Estructura dataset inicial

Campo	Descripción
No	Auto incremental
Problema	Descripción del problema a resolver
Solución	Solución en Python para la tarea de programación
Estado 1	Etiqueta parcial del 0 al 1. (0 = Error, 1 = Correcto)
Estado 2	Estado de la tarea del 0 al 7, considerando los tipos de errores: estado inicial, transformación de estado, estado final y los diferentes tipos de combinaciones

Nota: Elaboración propia

Hay otra variante de este dataset, agrega los campos de Estado inicial, Estado final y Transformación de estado, además de algunos cambios en la descripción de Estado 1 y Estado 2:

Tabla III
Variante dataset inicial

Campo	Descripción
No	Auto incremental
Problema	Descripción del problema a resolver
Solución	Solución en Python para la tarea de programación
Estado inicial	Fragmento de código de la solución (antes del while)
Estado final	Fragmento de código de la solución (condición del while)
Transformación de estado	Fragmento de código de la solución (dentro del while)
Estado 1	Etiqueta parcial del 0 al 1. (0 = Error, 1 = Correcto)
Estado 2	Estado de la tarea del 0 al 7, considerando los tipos de errores: estado inicial, transformación de estado, estado final y los diferentes tipos de combinaciones

Nota: Elaboración propia

Este dataset sirvió como base sólida a la presente investigación, del cual se recolecto 89 preguntas (descripción del problema a resolver) las cuales algunas compartían el mismo objetivo, cambiando solo los argumentos, por ejemplo: “Sumar los primeros diez números” y “Sumar los primeros veinte números”, el objetivo es sumar n números, donde cada pregunta solo varia en los primeros n números que hay que sumar.

Los demás datos como la solución (que está en Python, por lo que no aplica en esta investigación) y el etiquetado, solo sirvieron como guía para la elaboración del dataset explicado en la siguiente sección.

3) Elaboración del dataset

Si bien esta fase no la comprende como tal la metodología CRISP-DM, es necesaria debido a que no se encontró un dataset listo para su uso, por lo que comparte objetivo con la fase de comprensión de los datos. Esta fase consistió en generar un conjunto de datos listo para la aplicación de algoritmos de Machine Learning.

Con las preguntas recopiladas del dataset base [9], las preguntas que compartían el mismo objetivo se unificaron en 29 preguntas, escogidas manualmente y reescritas en el siguiente formato: "Write a JavaScript function that returns … using a 'while' loop.".

Los problemas fueron estructurados en carpetas, donde cada carpeta contiene cuatro archivos markdown (.md) que corresponden a las cuatro etiquetas principales, tres de error y uno correcto como se ve en la Tabla IV. El uso de un archivo markdown (.md) en lugar de un texto plano (.txt), fue por la comodidad de escribir secciones de código y tener el resaltado de sintaxis en un editor de código.

Tabla IV
Archivos markdown por problema

Nombre	Descripción del contenido
correct.md	Posibles soluciones correctas
initial.md	Posibles soluciones que contienen errores de estado inicial
transformation.md	Posibles soluciones que contienen errores de transformación de estado
final.md	Posibles soluciones que contienen errores de estado final

Nota: Elaboración propia

Inicialmente se comenzó con el archivo "correct.md" de cada problema, recopilando las posibles soluciones correctas del problema a resolver, aplicando diferentes variaciones en cada respuesta. Las variaciones aplicadas fueron:

- Cambiar orden de declaración de variables
- Cambiar la ejecución del bucle de creciente a decreciente, es decir en lugar de ir de 0 hasta 10, se va de 10 hasta 0.
- Cambiar operadores de comparación
- Cambiar valores iniciales de las variables
- Agregar variable de transformaciones

Por cada variación se aplicó su ajuste correspondiente, para que el código siga resolviendo el problema correctamente. Como codificar estas variaciones manualmente a las demás etiquetas requería bastante tiempo, se optó por una estrategia más eficiente. En lugar de codificar las respuestas completas, se dividió por secciones del código, como se establecen en la siguiente tabla.

Tabla V
Tipos de secciones

keyword	content
js	declaración de la función y declaración del ciclo while con su condicional.
initial	sección del código que va dentro de la función, pero antes del ciclo while.
transformation	sección del código que va dentro del ciclo while.
final	sección del código que va dentro de la función, pero después del ciclo while.

Nota: Elaboración propia

Así, se agregaban las diferentes variaciones solo a la sección que le correspondía (eliminando la duplicación de código). Como las variaciones generaban incompatibilidad de algunas secciones con otras, debido a que algunas variaciones repercutían en más de una sección, se optó por dividir por grupos, donde cada grupo garantizaba que todas las secciones al combinarse eran compatibles entre sí. La Fig. 5 muestra un ejemplo usando el primer enfoque y la Fig. 6 lo muestra aplicando el nuevo enfoque.

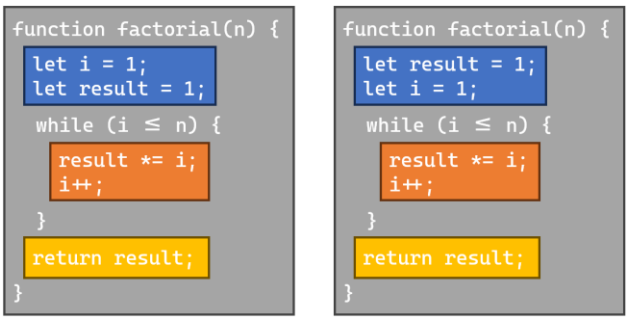


Fig. 5. Ejemplo respuestas usando primer enfoque

Nota: Elaboración propia

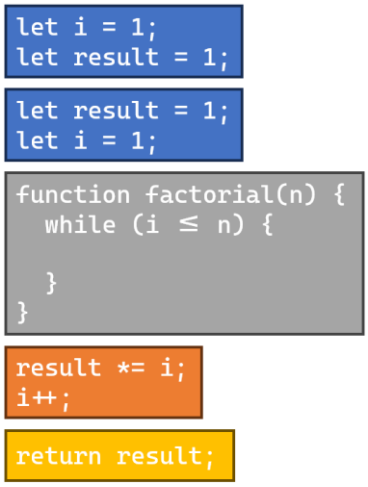


Fig. 6. Ejemplo respuestas usando nuevo enfoque

Nota: Elaboración propia

Para definir las secciones, divisores de grupos y bloques en el archivo markdown, se hace de la siguiente manera.

Tabla VI
Definición de bloque, grupo y sección en markdown

Definición	Caracteres	Explicación
Divisor de bloque	======	seis iguales seguidos
Divisor de grupo	===	tres iguales seguidos
Sección	```keyword content ```	keyword: palabra clave que define que tipo de sección es. content: el contenido de la sección.

Nota: Elaboración propia

El uso del nuevo enfoque acelero la elaboración del dataset, de esta forma cada archivo markdown solo modifica sus secciones asignadas. La siguiente tabla define que secciones es responsable cada archivo markdown.

Tabla VII
Archivos markdown con sus secciones asignadas

Nombre	Secciones asignadas
correct.md	initial, js, transformation, final
initial.md	initial
transformation.md	transformation
final.md	js, final

Nota: Elaboración propia

Si bien este enfoque permitió reducir el tiempo en terminar los cuatro archivos markdown de cada problema, fue necesario la creación de un script para transpilar estos grupos de secciones en un conjunto de respuestas, es decir pasar de la Fig. 6 a la Fig. 5. Para hacer esta transpilación es necesario que cada grupo contenga las cuatro secciones, de caso contrario no se genera ninguna respuesta. Como cada archivo solo se hace responsable por su sección, las secciones faltantes son tomadas de “correct.md”.

Los datos etiquetados con las cuatro etiquetas principales, fueron redactados-codificados manualmente, este enfoque solo fue un truco para compactar y evitar la repetición. Sin embargo, también esto permitió la generación automática de datos para las etiquetas combinadas. Al unir el archivo “initial.md” con “transformation.md” y las secciones faltantes obtenidas de “correct.md”, se obtienen respuestas con errores de estado inicial y errores de transformación de estado, así mismo para las demás etiquetas combinadas.

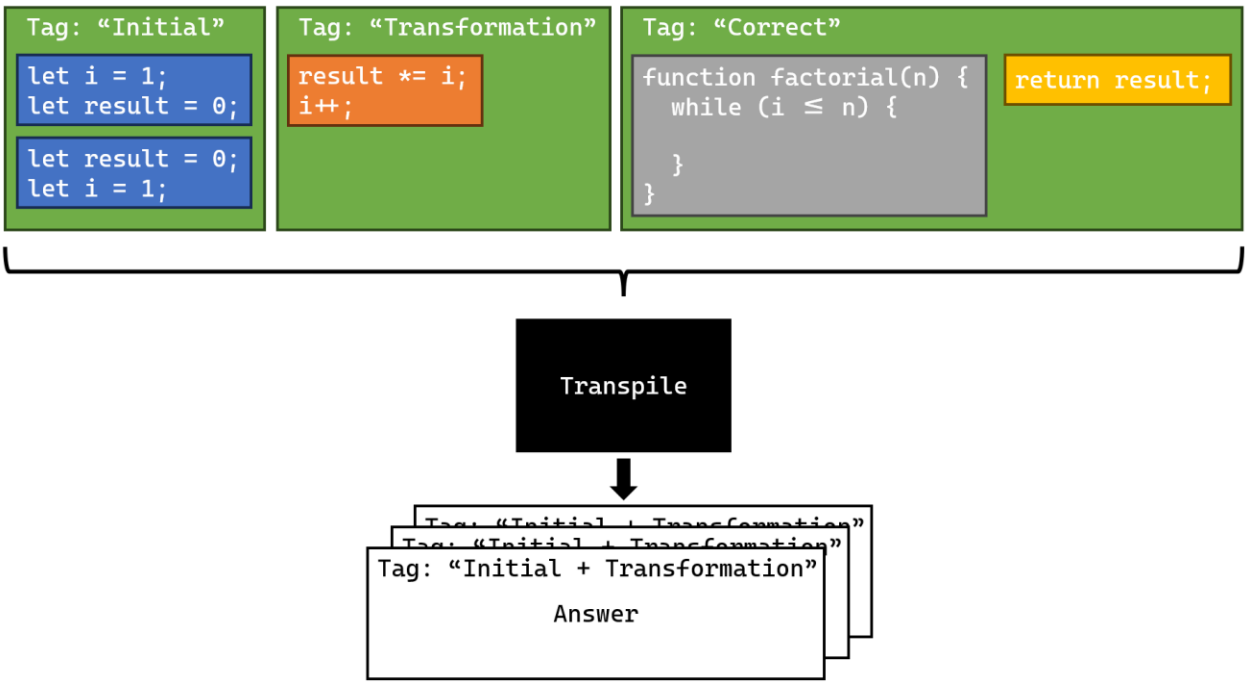


Fig. 7. Ejemplo generación automática para etiquetas combinadas

Nota: Elaboración propia

Como se miró anteriormente los divisores de grupo sirven para agrupar secciones compatibles entre sí, de otras que no (agrupadas en otro grupo), como tal no agrupan, sino que marcan la frontera de un grupo con otro. Los divisores de bloque son un caso especial, fueron pensados para los archivos “transformation.md” y “final.md”, estos archivos son diferentes porque las secciones que tienen asignados tienen la capacidad de generar un bucle infinito. Estos divisores de bloque marcan la frontera del conjunto de grupos que no generan un bucle infinito de uno que sí. Además, como el archivo “final.md” tiene asignado dos secciones, este divisor de bloque también se usa para marcar

la frontera entre el conjunto de grupos que tienen las variaciones de la sección “js” y el conjunto de grupos que tienen las variaciones de la sección “final”.

Es importante que cada archivo entre si cuenten con la misma cantidad de grupos, para permitir la transpilación al unir archivos. En el caso de que el archivo cuente con divisores de bloque, entonces cada bloque tiene que contar con la misma cantidad de grupos de los demás archivos. Así es posible unir el bloque dos del archivo “final.md” con los grupos de “correct.md”.

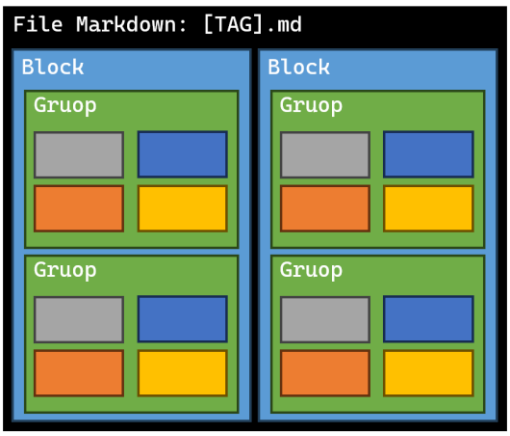


Fig. 8. Estructura grafica del archivo markdown

Nota: Elaboración propia

Para generar el dataset, un script lee las carpetas y transforma los archivos markdown en un dataset en formato CSV, con la siguiente estructura.

Tabla VIII
Estructura del dataset

Campo	Descripción
No	Auto incremental
Problem ID	Corresponde al ID de cada carpeta (1 al 29).
Question	Pregunta del problema a resolver.
Answer	Respuesta o posible solución en JavaScript al problema.
Etiqueta	Etiqueta numérica del 0 al 7

Nota: Elaboración propia.

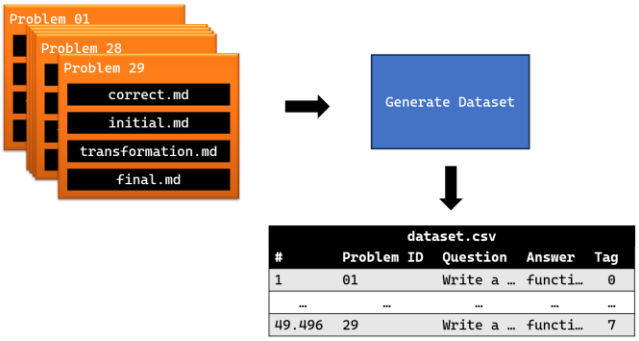


Fig. 9. Diagrama general de la generación del dataset

Nota: Elaboración propia

Para asegurar que las respuestas generadas están correctamente etiquetadas, se hizo un script que al transpilar los archivos markdown toma las respuestas y ejecuta algunos tests con estas. Lo primero que verifica es si esta respuesta contiene en cualquier lugar "#(ignore-test)", esta es una marca especial, su propósito es omitir la ejecución de los tests (para esa respuesta) y marca la respuesta como correctamente etiquetada.

Para las respuestas que producen un bucle infinito, es fundamental esta marca, debido a que, sin esta, no se obtendrá una respuesta de si esta correctamente etiquetada o no. Esta marca es muy peligrosa, ya que puede hacer pasar datos incorrectamente etiquetados como correctamente etiquetados en el dataset.

Los tests tienen una limitación, solo verifican si una respuesta es correcta o contiene algún error, sin importar en que etapa está el error. Por lo que, para una respuesta etiquetada con error de estado inicial, los tests aseguran que la respuesta contiene un error, no aseguran que este error este en la etapa de estado inicial.

Para cada problema se definió unos casos de prueba, para las respuestas etiquetadas como correctas, tiene que pasar todos los casos de prueba, caso contrario se marca la respuesta como incorrectamente etiquetada. Las respuestas con las demás etiquetas, tienen como mínimo NO pasar un caso de prueba para ser marcadas como correctamente etiquetadas.

Para las etiquetas combinadas no se realizaron tests, debido a que ya se hicieron a cada una de las etiquetas principales, esto puede afectar la integridad de los datos, por lo que se recomienda para futuras investigaciones implementar validaciones a todas las etiquetas combinadas y en caso de que alguna respuesta no pase los tests, que no se incluya en el dataset.

Antes de generar el dataset, se eliminan las posibles respuestas duplicadas, se limpian las respuestas quitando "#(ignore-test)" (si lo hay) y finalmente balancear el dataset, para que cada etiqueta contenga la misma cantidad de datos que todas las demás. Como resultado se generaron en total 2.096.053 datos y 49.496 datos con el dataset balanceado. Superando con creces la generación de datos esperados, el dataset balanceado es el que se utilizara en la preparación de los datos.

Todos los scripts fueron hechos en Python, debido a la comodidad de su sintaxis y a bibliotecas existentes que ayudan a leer, transformar y limpiar los datos de dataset. El código fuente para generar el dataset y realizar los tests (como también los demás scripts o modelados que se mencionaran a continuación) se encuentran en un repositorio de GitHub[1]. Al igual que el dataset[2] que cuenta con 49.496 datos, donde cada etiqueta contiene exactamente 6.187 datos, por lo que está perfectamente balanceado, su estructura ya se definió en la Tabla VIII.

Así se da por cumplido el primero objetivo específico "Consolidar un conjunto de datos de tareas comunes de programación con ciclos en JavaScript para el entretenimiento del modelo."

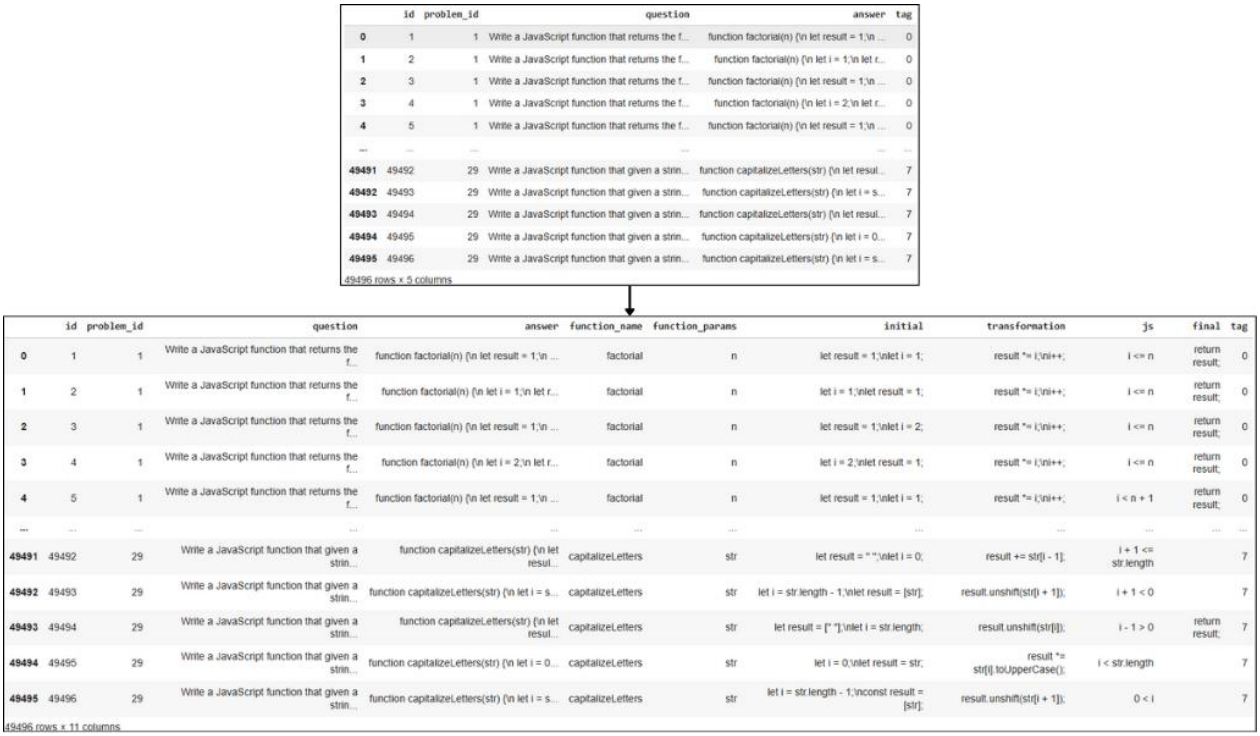
B. APLICACIÓN DE ALGORITMOS DE MACHINE LEARNING

En esta sección se detallan los resultados de las siguientes dos etapas de la metodología CRISP-DM, preparación de los datos y modelado, para poder dar por cumplido el segundo objetivo específico.

1) Preparación de los datos

Para la fase de preparación de los datos se decidió ampliar las columnas del dataset original incorporando cuatro nuevas, que toman secciones del código de respuesta: initial (código anterior al while), transformation (cuerpo del while), js (condición del while) y final (código posterior al while). La extracción de estas secciones se automatizó mediante un script en Python que analiza cada respuesta de JavaScript y separa las secciones correspondientes, incorporándolas como columnas independientes en el dataset final. La decisión de expandir las columnas se basó en la heurística de la investigación previa [9], que observó que esta representación explícita del código, aumentó la capacidad del modelo para discriminar errores según la etapa del ciclo.

[1] https://github.com/sanpezlo/research-project
[2] https://raw.githubusercontent.com/sanpezlo/research-project/refs/heads/main/data/xlsx/dataframe_balanced.csv



	id	problem_id	question	answer	tag
0	1	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let result = 1;\n ...	0
1	2	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let i = 1;\n let r...	0
2	3	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let result = 1;\n ...	0
3	4	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let i = 2;\n let r...	0
4	5	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let result = 1;\n ...	0
...	...	...	...	...	...
49491	49492	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let resul...	7
49492	49493	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let i = s...	7
49493	49494	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let resul...	7
49494	49496	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let i = 0...	7
49495	49496	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let i = s...	7

49496 rows × 5 columns

	id	problem_id	question	answer	function_name	function_params	initial	transformation	js	final	tag
0	1	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let result = 1;\n ...	factorial	n	let result = 1;\nlet i = 1;	result *= i;\ni++;	i <= n	return result;	0
1	2	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let i = 1;\n let r...	factorial	n	let i = 1;\nlet result = 1;	result *= i;\ni++;	i <= n	return result;	0
2	3	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let result = 1;\n ...	factorial	n	let result = 1;\nlet i = 2;	result *= i;\ni++;	i <= n	return result;	0
3	4	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let i = 2;\n let r...	factorial	n	let i = 2;\nlet result = 1;	result *= i;\ni++;	i <= n	return result;	0
4	5	1	Write a JavaScript function that returns the f...	function factorial(n) {\n let result = 1;\n ...	factorial	n	let result = 1;\nlet i = 1;	result *= i;\ni++;	i < n + 1	return result;	0
...	...	...	...	...	...	...	...	...	...	...	...
49491	49492	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let resul...	capitalizeLetters	str	let result = "";\nlet i = 0;	result += str[i - 1];	i + 1 <= str.length		7
49492	49493	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let i = s...	capitalizeLetters	str	let i = str.length - 1;\nlet result = [str];	result.unshift(str[i + 1]);	i + 1 < 0		7
49493	49494	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let resul...	capitalizeLetters	str	let result = [" "];\nlet i = str.length;	result.unshift(str[i]);	i - 1 > 0	return result;	7
49494	49495	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let i = 0...	capitalizeLetters	str	let i = 0;\nlet result = str;	result *= str[i].toUpperCase();	i < str.length		7
49495	49496	29	Write a JavaScript function that given a strin...	function capitalizeLetters(str) {\n let i = s...	capitalizeLetters	str	let i = str.length - 1;\nconst result = [str];	result.unshift(str[i + 1]);	0 < i		7

49496 rows × 11 columns

Fig. 10. Dataset con aumento de columnas

Nota: Elaboración propia.

Antes de entrenar cualquier modelo de aprendizaje automático es importante preparar los datos. Esto mejora la calidad del entrenamiento y facilita que el modelo aprenda lo que realmente importa, en este proyecto esa preparación tiene una importancia especial porque los ejemplos combinan dos tipos de información muy diferentes: por un lado, el enunciado en lenguaje natural (la pregunta del problema) y por otro, código en JavaScript (la respuesta y las cuatro nuevas columnas). Cada uno de estos tipos de dato tiene su propia estructura y semántica, por lo que tratarlos correctamente es la base para obtener buenos resultados.

Tanto la pregunta del problema como la respuesta son datos de longitud variable, para transformarlos a datos con longitud fija son necesario los embeddings. Un embedding es una transformación que convierte una entrada textual en un vector numérico de dimensión fija. Ese vector actúa como una “firma” compacta que captura propiedades importantes del contenido. Capturan relaciones semánticas/contextuales, que hacen posible combinar, información de texto y de código en un mismo modelo.

Se experimentó comparativamente con los siguientes modelos especializados para generar embeddings:

- BERT para las preguntas (lenguaje natural), produce embeddings contextualizados que representan el significado del enunciado en su contexto.
- CodeBERT, GraphCodeBERT y CodeT5 para las respuestas (código en JavaScript), que está afinado para código y conserva información sintáctica y semántica del programa.

Para realizar el proceso de embeddings, lo primero que hay que hacer es la tokenización. La tokenización transforma una secuencia de caracteres de longitud variable en una secuencia de tokens, estos tokens pueden ser palabras, subpalabras o símbolos que luego se convierten en números (cada token tiene un numero asignado) y finalmente en vectores (embeddings).

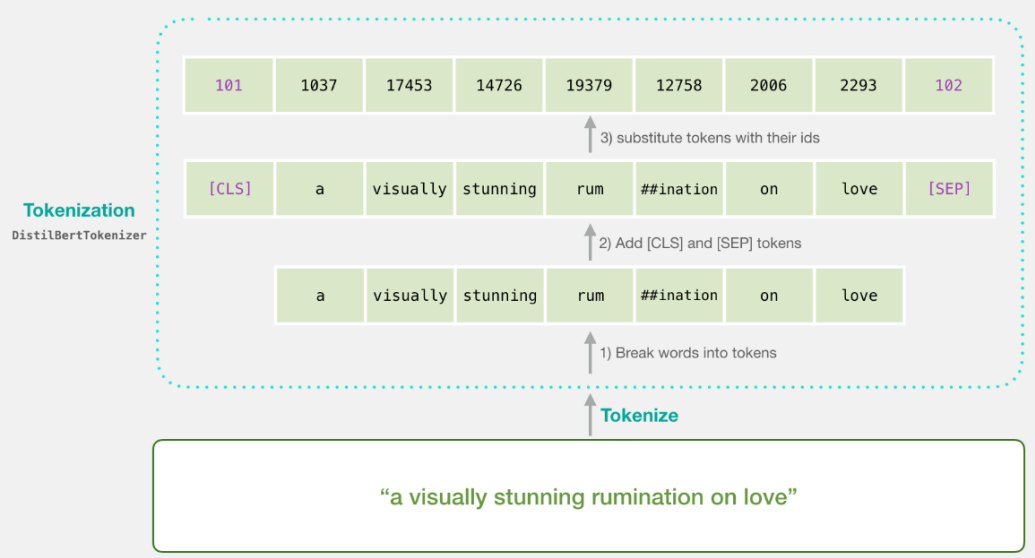


Fig. 11. Tokenización

Nota: fuente https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Tanto BERT como CodeBERT, GraphCodeBERT y CodeT5 devuelven un vector de 768 de longitud por token. Para tener un solo vector la práctica más común y con mejor resulta es calcular el promedio (mean pooling) de todos los vectores de token. Así se obtiene un solo vector con 768 de longitud.

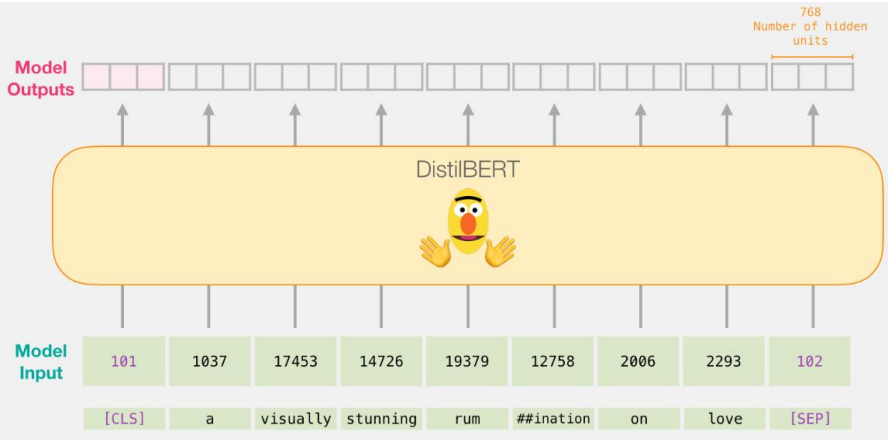


Fig. 12. Embedding

Nota: fuente https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

El dataset se dividió en un 80% para entrenamiento y un 20% para tests, a ambos conjuntos se les aplicó los diferentes embeddings mencionados. Los scripts para la preparación de los datos y transformación a embeddings también se encuentran en Google Drive[3] junto con los diferentes scripts de entrenamiento del modelo.

2) Modelado

El modelado es la fase en la que se define la arquitectura del modelo y se ajustan sus parámetros para que aprenda y generalice el conocimiento. En esta etapa se optó por arquitecturas de redes neuronales totalmente conectadas (Fully Connected), dado que en la investigación previa [9] este tipo de arquitectura obtuvo las mejores métricas y comportamiento para la tarea objetivo. Por este motivo no se llevaron a cabo experimentos con clasificadores tradicionales (como RandomForest o K-Nearest Neighbors) en esta fase de experimentación, esto permitió centrar el análisis en cómo los diferentes embeddings afectan el rendimiento de las redes.

Como las entradas provienen de embeddings diferentes (lenguaje natural y las diferentes secciones del código), se diseñó una arquitectura con ramas paralelas, donde cada rama procesa su embedding correspondiente para luego fusionarlas en una etapa final de decisión. A partir de esta idea se definieron dos arquitecturas base. La estructura del Modelo 1 se presenta en la Fig. 13, y la estructura del Modelo 2 corresponde exactamente a la que se utilizó en la investigación de referencia [9] como se muestra en la Fig. 14. Para garantizar comparabilidad entre configuraciones, cada instancia del modelo se entrenó con la misma configuración, los mismos criterios de optimización y regularización; la única variable de la comparación fue el tipo de embedding aplicado al código (el embedding con BERT para NL se mantuvo fijo).

[3] https://drive.google.com/drive/folders/1r77sbRKggjegqbgpObQgshnoEjUiMxWR

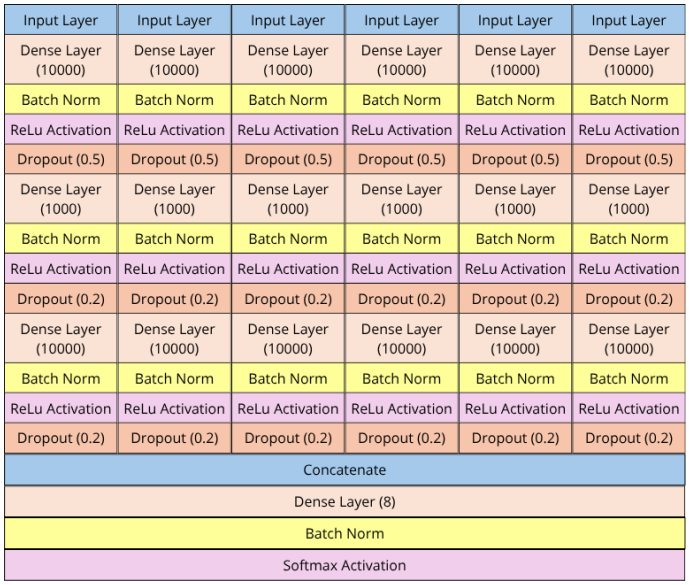


Fig. 13. Estructura del Modelo 1

Nota: Elaboración propia

Fig. 14. Estructura del Modelo 2

Nota: Elaboración propia

Para evitar un sobreajuste se aplicaron técnicas de regularización como el Dropout, Batch Normalization y Early stopping. Para el Modelo 1 el Dropout NO es constante, la primera capa es de 50% y las siguientes capas de 20%, para el Modelo 2 el Dropout es de 20% constante en todas

las capas. El Batch Normalization se aplicó a cada etapa para estabilizar el entrenamiento a los dos modelos por igual.

Se entreno con un optimizador Adam con un learning_rate de 0.00001, con una función de perdida Categorial Cross Entropy debido a que es una clasificación multiclase. Las métricas que se usaron son Accuracy y F1-Score.

Se dividió el dataset (que ya se había reducido previamente en un 80%) en 80% para entrenamiento y 20% para validación. Por lo que en general hay 3 divisiones, un 64% para el entrenamiento, 16% para validación y 20% para tests.

El entrenamiento del modelo realiza como máximo 150 épocas, debido a que se empleó un early stopping con paciencia de 10 para controlar la parada temprana y evitar un entrenamiento innecesario o sobreajuste. El early stopping vigila la métrica val_loss en cada época y si ésta no disminuye durante 10 épocas consecutivas, se detiene el entrenamiento. Al parar se restaura automáticamente los pesos correspondientes a la época con el menor val_loss registrado.

Tanto las configuraciones como arquitecturas se basaron en las utilizadas por la investigación de referencia [9], por lo que ya se contaba con experimentaciones previas y se ajustaron levemente. Para todas las divisiones del dataset, mezclas o inicialización de pesos se usó la misma semilla, con el valor de 2025, junto con un entrenamiento con operaciones determinista, para garantizar una replicación de los resultados.

a) CodeBERT

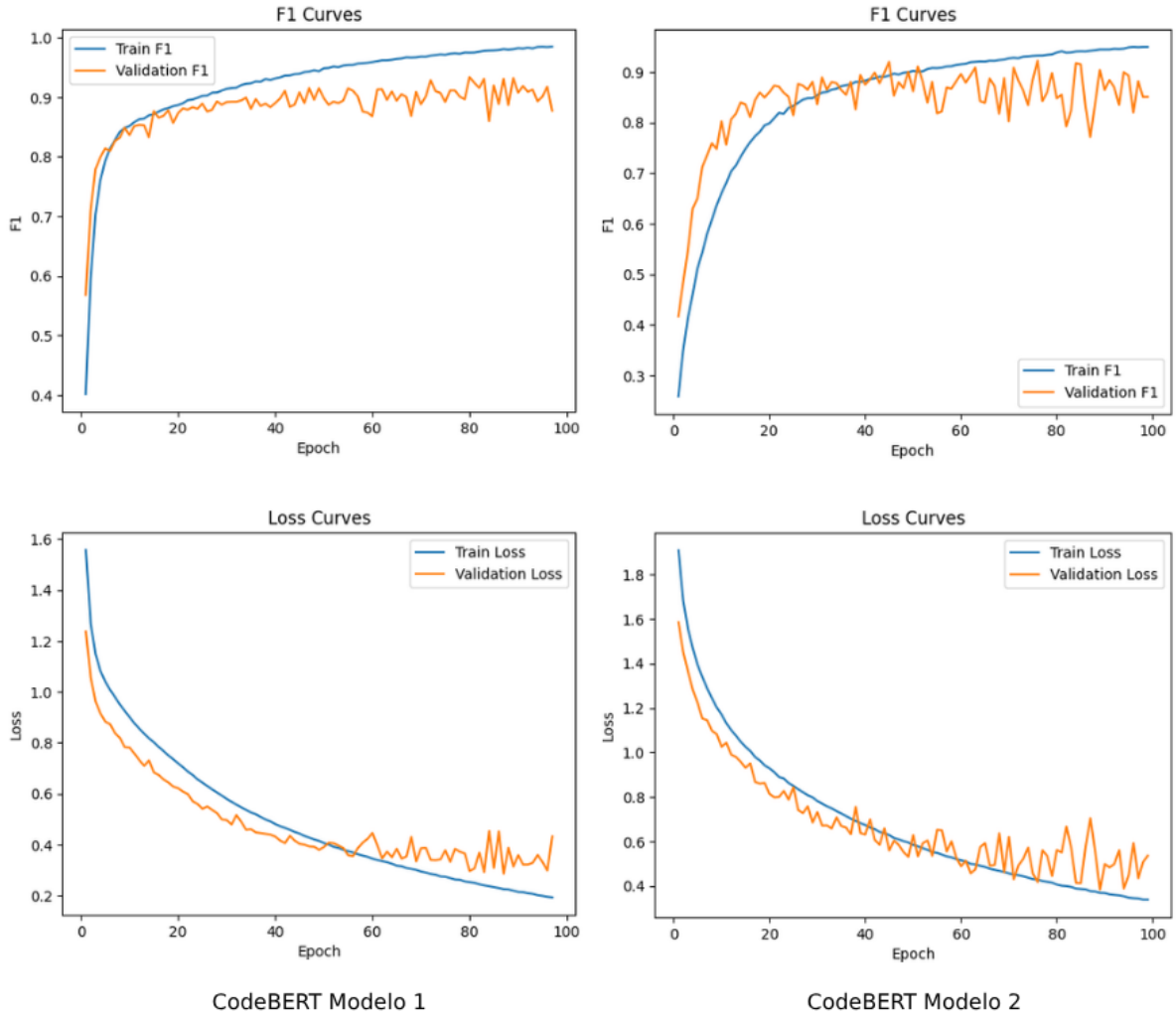


Fig. 15. CodeBERT - Modelo 1 vs Modelo 2

Nota: Elaboración propia

Los resultados obtenidos con el embedding CodeBERT muestran diferencias significativas entre ambas arquitecturas propuestas. El Modelo 1 alcanzó su mejor desempeño en la época 87 de un total de 97 épocas de entrenamiento, logrando un accuracy de 0.9806 y un F1-Score de 0.9804 en el conjunto de entrenamiento. En el conjunto de validación, este modelo registró un accuracy de 0.9310, un F1-Score de 0.9307 y una función de pérdida de 0.2869.

Por su parte, el Modelo 2 completó 99 épocas y alcanzó su mejor rendimiento en la época 89. Los resultados en entrenamiento fueron de 0.9428 en accuracy y F1-Score, con una pérdida de 0.3730. En validación, el modelo obtuvo un accuracy de 0.9046, un F1-Score de 0.9034 y una pérdida de 0.3837. Estos valores son consistentemente inferiores a los del Modelo 1.

Al comparar ambas arquitecturas, el Modelo 1 supera al Modelo 2 en todas las métricas evaluadas. La diferencia es particularmente notable en el conjunto de validación, donde el Modelo 1 obtiene aproximadamente 2.64 puntos porcentuales más de accuracy y F1-Score. Además, presenta una función de pérdida considerablemente menor (0.2869 vs 0.3837), lo que sugiere predicciones con mayor confianza y menor incertidumbre. El análisis de las curvas de entrenamiento revela que el Modelo 1 alcanza la convergencia de manera más estable, mientras que el Modelo 2 muestra mayor oscilación en las métricas de validación, lo cual podría indicar una mayor sensibilidad a las variaciones en los datos.

b) GraphCodeBERT

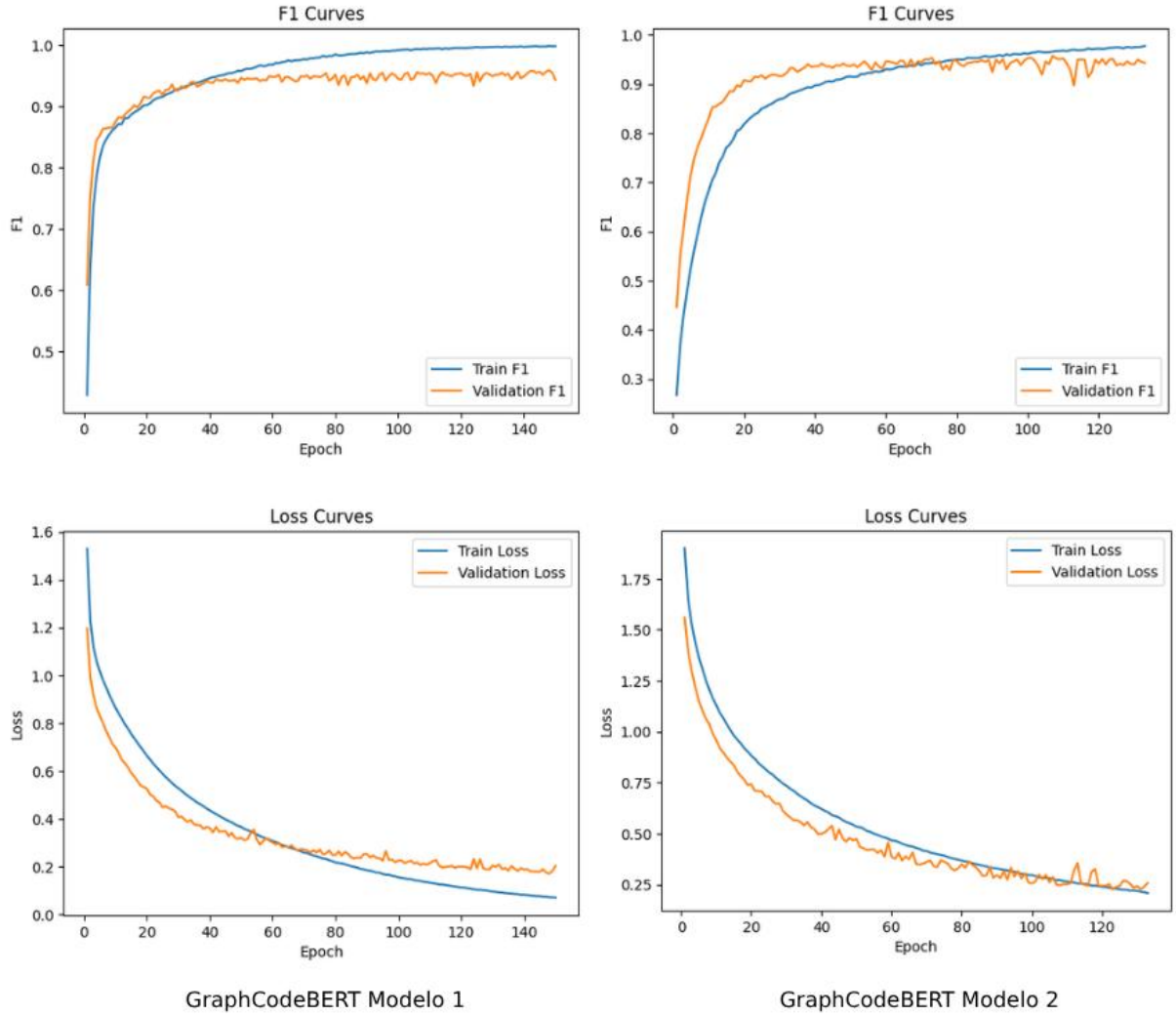


Fig. 16. GraphCodeBERT - Modelo 1 vs Modelo 2

Nota: Elaboración propia

GraphCodeBERT, diseñado específicamente para capturar la estructura del código mediante representaciones basadas en grafos, mostró un rendimiento superior en comparación con CodeBERT. El Modelo 1 requirió 150 épocas de entrenamiento y alcanzó su mejor desempeño en la época 148. En entrenamiento, logró un accuracy de 0.9982 y un F1-Score de 0.9982, con una pérdida excepcionalmente baja de 0.0714. En el conjunto de validación, obtuvo un accuracy de 0.9590, un F1-Score de 0.9589 y una pérdida de 0.1705.

El Modelo 2 completó 133 épocas, alcanzando su óptimo en la época 123. Sus resultados en entrenamiento fueron de 0.9757 en accuracy y F1-Score, con una pérdida de 0.2272. En validación, registró un accuracy de 0.9502, un F1-Score de 0.9500 y una pérdida de 0.

La comparación entre arquitecturas revela nuevamente la superioridad del Modelo 1, aunque con una brecha menor que en el caso de CodeBERT. El Modelo 1 supera al Modelo 2 por 0.88 puntos porcentuales en accuracy y F1-Score de validación. Sin embargo, lo más destacable es la notable reducción en la función de pérdida del Modelo 1 (0.1705) frente al Modelo 2 (0.2255), lo que indica predicciones con mayor certeza. Ambos modelos lograron un ajuste excelente a los datos de entrenamiento, alcanzando valores de accuracy superiores al 97%. Las curvas de entrenamiento muestran que el Modelo 1 requirió más épocas para converger, pero logró una estabilización más sólida en las métricas de validación.

c) CodeT5

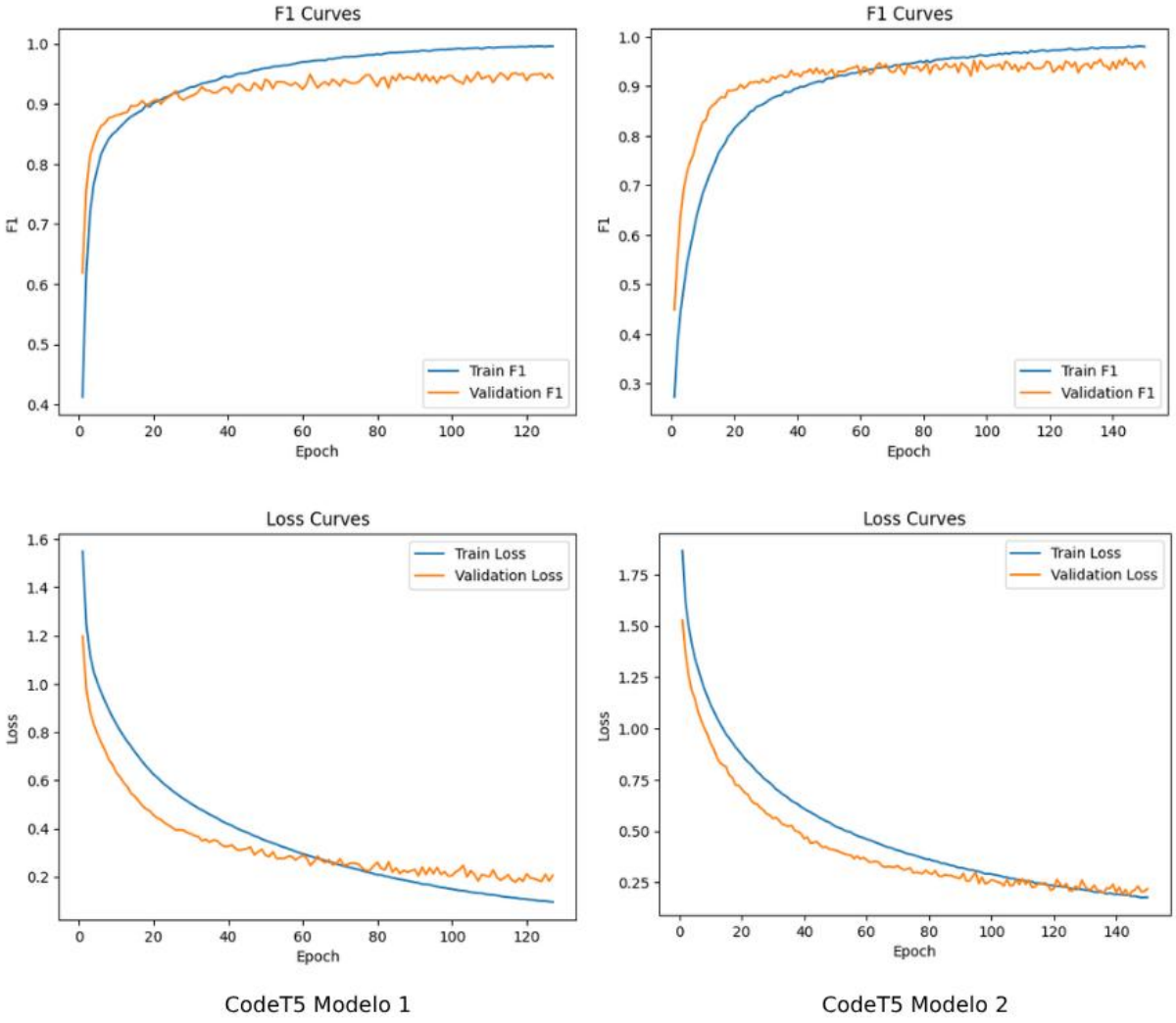


Fig. 17. CodeT5 – Modelo 1 vs Modelo 2

Nota: Elaboración propia

CodeT5, un modelo basado en arquitectura Transformer específicamente pre-entrenado para tareas de código. El Modelo 1 se entrenó durante 127 épocas y alcanzó su mejor rendimiento en la época 117. En el conjunto de entrenamiento, registró un accuracy de 0.9934, un F1-Score de 0.9934 y una pérdida de 0.1152. En validación, obtuvo un accuracy de 0.9522, un F1-Score de 0.9518 y una pérdida de 0.1786.

El Modelo 2 completó el máximo de 150 épocas permitidas, con su mejor desempeño en la época 144. Los resultados en entrenamiento fueron de 0.9786 en accuracy y F1-Score, con una pérdida

de 0.1883. En validación, logró un accuracy de 0.9571, un F1-Score de 0.9568 y una pérdida de 0.1862.

A diferencia de los resultados con CodeBERT y GraphCodeBERT, los modelos con CodeT5 presentaron un comportamiento más equilibrado entre arquitecturas. Si bien el Modelo 1 mostró un mejor ajuste en entrenamiento (accuracy de 0.9934 vs 0.9786), el Modelo 2 obtuvo métricas de validación ligeramente superiores, con una diferencia de 0.49 puntos porcentuales en accuracy y F1-Score a favor del Modelo 2. Las funciones de pérdida fueron relativamente similares (0.1786 para Modelo 1 vs 0.1862 para Modelo 2), lo que sugiere niveles de confianza comparables en las predicciones. Las curvas de entrenamiento revelan que ambos modelos convergieron de manera estable sin signos evidentes de sobreajuste, a pesar de que el Modelo 2 alcanzó el límite de épocas establecido. Esto sugiere que podría haberse beneficiado de un entrenamiento más prolongado, aunque las mejoras adicionales probablemente serían marginales dado el nivel de desempeño ya alcanzado.

Así se da por cumplido el segundo objetivo específico "Aplicar el conjunto de datos para entrenar el modelo usando técnicas de Machine Learning"

C. EVALUACIÓN DEL MODELO

En esta sección se detallan los resultados de las dos etapas restantes de la metodología CRISP-DM, evaluación y despliegue, para poder dar por cumplido el tercer objetivo específico.

1) Evaluación

Para evaluar el rendimiento de los modelos se emplearon métricas globales (accuracy, F1-score loss y MCC en las diferentes divisiones del dataset) y un análisis detallado por clase mediante reportes de clasificación y matrices de confusión. Además, se inspeccionaron previamente las curvas de entrenamiento (épocas vs. loss y vs. F1) para verificar convergencia y posibles signos de sobreajuste o inestabilidad.

Tabla IX
Reporte General de cada Modelo

Dataset	Train			Validation			Test
Modelo	Accuracy	F1	Loss	Accuracy	F1	Loss	MCC
CodeBERT M1	0.9806	0.9804	0.2283	0.9310	0.9307	0.2869	0.9177
CodeBERT M2	0.9428	0.9426	0.3730	0.9046	0.9034	0.3837	0.8966
GraphCodeBERT M1	**0.9982**	**0.9982**	**0.0714**	**0.9590**	**0.9589**	**0.1705**	**0.9512**
GraphCodeBERT M2	0.9757	0.9757	0.2272	0.9502	0.9500	0.2255	0.9387
CodeT5 M1	0.9934	0.9934	0.1152	0.9522	0.9518	0.1786	0.9463
CodeT5 M2	0.9786	0.9786	0.1883	0.9571	0.9568	0.1862	0.9477

Nota: Elaboración propia.

La métrica Accuracy indica la proporción de predicciones correctas. La métrica F1-Score indica el rendimiento del modelo de clasificación, combinando la precisión (precision) y la exhaustividad (recall). La función de perdida Categorial Cross Entropy (Loss) mide la diferencia entre dos distribuciones de probabilidad, la distribución predicha por el modelo y la distribución real de las clases. La métrica Matthews Correlation Coefficient (MCC) indica la calidad de un clasificador, que oscila entre -1 y 1. Una puntuación de 1 representa una predicción perfecta, 0 una predicción aleatoria promedio y -1 una predicción inversa.

Al comparar los tres embeddings evaluados (CodeBERT, GraphCodeBERT y CodeT5), se observan diferencias significativas en el rendimiento de los modelos, siendo GraphCodeBERT con Modelo 1 claramente superior en todas las métricas de evaluación relevantes. La superioridad de GraphCodeBERT puede atribuirse a su capacidad de capturar no solo la semántica del código sino también su estructura sintáctica mediante representaciones basadas en grafos de flujo de datos. Esta característica permite al modelo comprender las relaciones entre variables, operaciones y estructuras de control de manera más profunda que CodeBERT, que trata el código principalmente como secuencias de tokens.

GraphCodeBERT Modelo 1 demostró el mejor desempeño en el conjunto de validación, mostrando las métricas más altas entre todas las configuraciones evaluadas. Adicionalmente, registró la función de pérdida en validación más baja, lo que indica predicciones con mayor confianza y menor incertidumbre. En el conjunto de entrenamiento, GraphCodeBERT Modelo 1 también destacó notablemente, evidenciando un excelente ajuste a los datos sin signos de sobreajuste, como se puede corroborar en la consistencia entre las métricas de entrenamiento y validación.

CodeT5 mostró resultados competitivos, particularmente el Modelo 2, pero se ubicó por debajo de GraphCodeBERT en todas las métricas clave. Las diferencias son pequeñas pero consistentes: GraphCodeBERT Modelo 1 supera a CodeT5 Modelo 2 por 1.9 puntos porcentuales en accuracy de validación, y presenta una pérdida significativamente menor (0.1705 vs 0.1862), lo que se traduce en predicciones más confiables y estables. Por su parte, CodeBERT presentó el desempeño más bajo de los tres embeddings evaluados, con pérdidas en validación considerablemente más altas, reflejando mayor incertidumbre en sus clasificaciones y menor capacidad de generalización.

En cuanto a la arquitectura, el Modelo 1 demostró ser consistentemente más efectivo con CodeBERT y GraphCodeBERT, superando al Modelo 2 en todas las métricas evaluadas. Con CodeT5, aunque el Modelo 2 mostró un ligero mejor desempeño en validación, ninguno alcanzó los niveles de GraphCodeBERT Modelo 1.

Para evaluar el rendimiento del modelo seleccionado GraphCodeBERT Modelo 1 se emplearon múltiples métricas y visualizaciones que permiten un análisis exhaustivo tanto a nivel global como por clase individual. Esta evaluación se realizó utilizando el conjunto de datos de prueba (20% del

total), que no fue empleado durante el entrenamiento ni la validación, garantizando así una evaluación imparcial de la capacidad de generalización del modelo.

Como se vio en las métricas presentadas en la Tabla IX, todos los modelos superan el 0 en la métrica MCC, concretamente superan más del 0.89, gracias a esto se puede demostrar que las métricas F1-Score de TODOS los modelos son superiores a un modelo aleatorio, por lo que la hipótesis planteada en la sección FORMULACIÓN DE HIPÓTESIS se cumple con creces. Sin embargo, para comprender a profundidad el comportamiento del clasificador es necesario analizar su desempeño en cada una de las ocho categorías de errores. A continuación, se presentan diferentes visualizaciones y reportes que permiten evaluar la calidad de las predicciones desde múltiples perspectivas.

a) Análisis de Curvas ROC

Las curvas ROC (Receiver Operating Characteristic) permiten evaluar la capacidad del modelo para discriminar entre cada clase y el resto de categorías, mostrando la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes umbrales de decisión. En un problema de clasificación multiclase como el presente, se genera una curva ROC para cada categoría aplicando el enfoque "uno contra todos" (one-vs-rest).

La Fig. 18 presenta las curvas ROC para las ocho categorías de clasificación del modelo GraphCodeBERT Modelo 1. El área bajo la curva (AUC) proporciona una medida agregada del rendimiento: un AUC de 1.0 indica una clasificación perfecta, mientras que 0.5 representa un clasificador aleatorio.

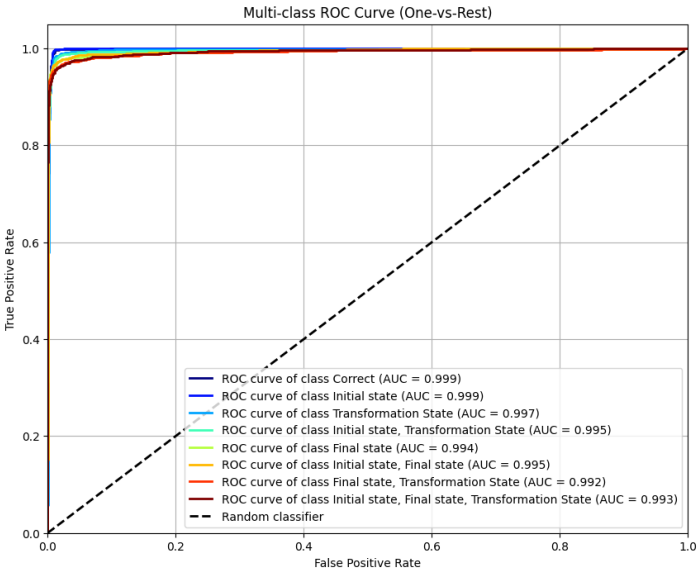


Fig. 18. AUC-ROC

Nota: Elaboración propia.

Como se observa en las curvas ROC, el modelo alcanza valores de AUC excepcionalmente altos para todas las categorías. Las clases Correct, Initial state, Transformation state y Final state presentan las curvas más cercanas a la esquina superior izquierda (punto óptimo), con AUC cercano o superior a 0.99, lo que indica una excelente capacidad de discriminación para estas categorías fundamentales. Las categorías correspondientes a errores combinados muestran curvas ROC ligeramente menos pronunciadas, aunque siguen manteniendo valores de AUC superiores a 0.99 en todos los casos. Esto es consistente con la hipótesis de que discriminar múltiples errores simultáneos representa un desafío mayor para el modelo. No obstante, incluso en estos casos más complejos, el modelo supera ampliamente el rendimiento de un clasificador aleatorio.

b) Análisis de Curvas Precisión-Recall

Las curvas Precisión-Recall ofrecen una perspectiva complementaria a las curvas ROC, siendo especialmente informativas cuando existe desbalance entre clases o cuando el costo de los falsos positivos y falsos negativos es diferente, si bien el dataset está balanceado al hacer la división aleatoria se desbalanceo mínimamente. La precisión mide la proporción de predicciones positivas que son correctas, mientras que el recall mide la proporción de casos positivos reales que fueron identificados correctamente.

La Fig. 19 muestra las curvas Precisión-Recall para cada una de las ocho categorías del modelo GraphCodeBERT Modelo 1. El área bajo la curva Precisión-Recall (AP - Average Precision) resume el desempeño del modelo, valores cercanos a 1.0 indican que el modelo mantiene alta precisión incluso cuando se incrementa el recall.

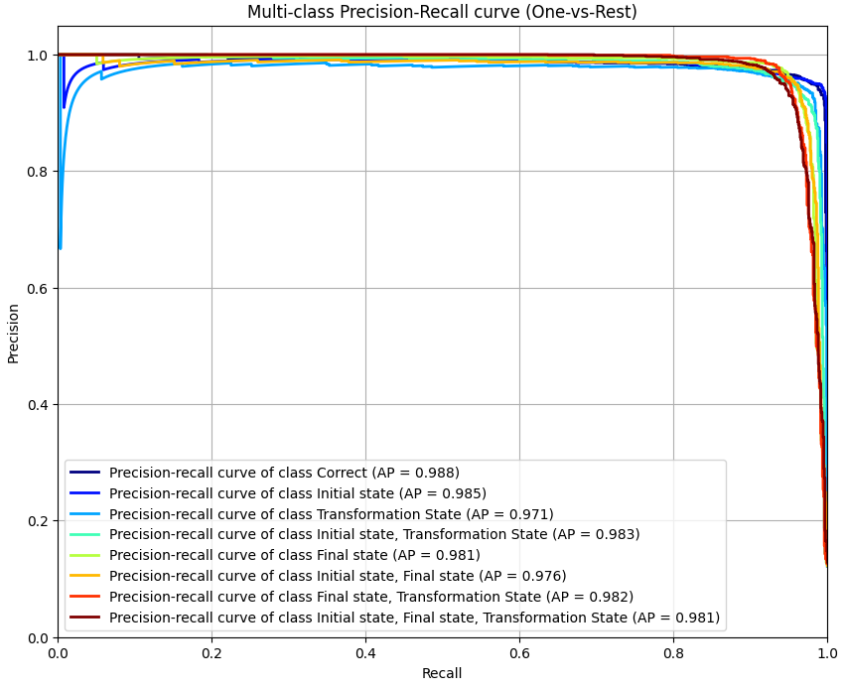


Fig. 19. Precision-Recall

Nota: Elaboración propia.

El análisis de las curvas Precisión-Recall confirma los resultados observados en las curvas ROC. Las categorías de errores combinados presentan curvas con mayor variabilidad, particularmente en valores altos de recall, donde se observa una ligera disminución en la precisión. Este comportamiento es esperado dado que estas categorías representan casos más complejos que combinan múltiples tipos de errores. Sin embargo, los valores de AP se mantienen por encima de 0.97 en todos los casos, lo que demuestra un balance adecuado entre precisión y exhaustividad.

c) Análisis de Matriz de Confusión

La matriz de confusión proporciona una visualización detallada de los patrones de error del clasificador, mostrando la relación entre las etiquetas reales y las predicciones del modelo para cada categoría. Los elementos de la diagonal principal representan las predicciones correctas, mientras que los elementos fuera de la diagonal indican confusiones entre categorías.

La Fig. 20 presenta la matriz de confusión normalizada del modelo GraphCodeBERT Modelo 1 sobre el conjunto de pruebas. La normalización por filas permite interpretar cada celda como la proporción de casos de una clase real que fueron clasificados en cada categoría predicha, facilitando la comparación entre clases independientemente de su frecuencia.

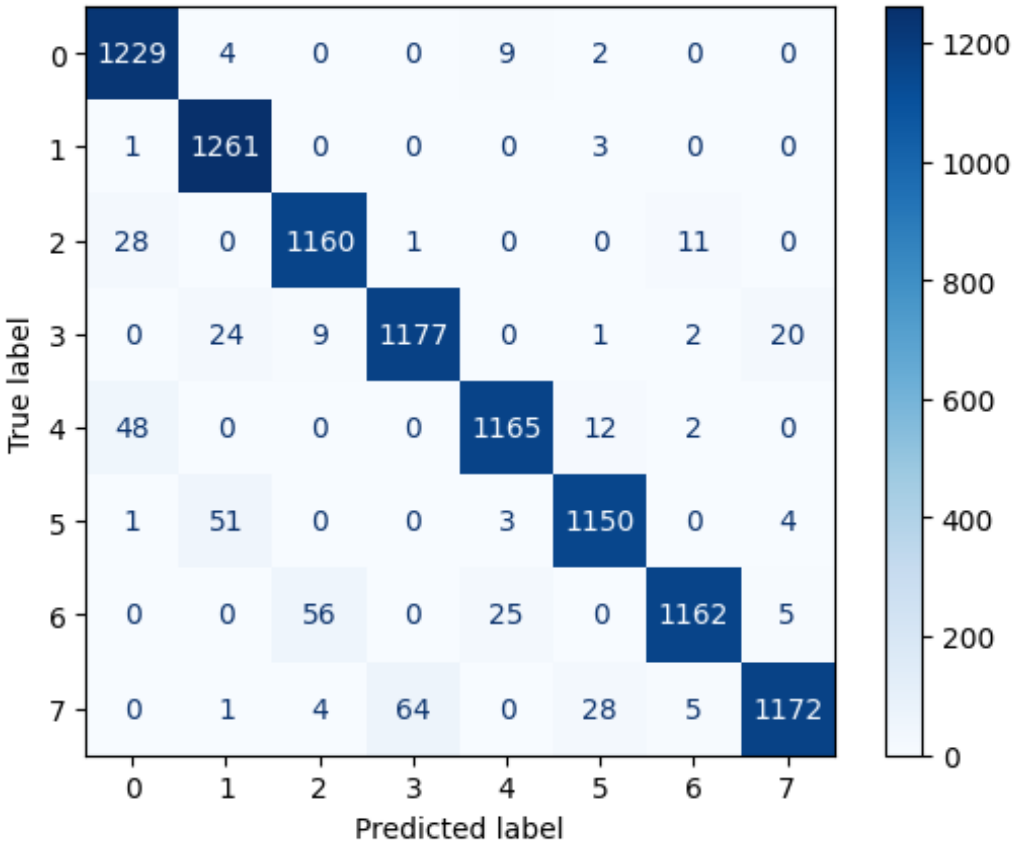


Fig. 20. Matriz de confusión

Nota: Elaboración propia.

La matriz de confusión revela el excelente desempeño del modelo a través de la fuerte intensidad de color en la diagonal principal, confirmando que la gran mayoría de las predicciones son correctas. Las categorías principales mantienen tasas de acierto superiores al 95%, siendo particularmente destacable la clase Initial state con 1,261 de 1,265 predicciones correctas, alcanzando casi el 100% de precisión.

Las categorías de errores combinados presentan tasas de acierto entre 92% y 95%, ligeramente inferiores, pero igualmente competitivas. Este comportamiento es esperado dado que representan casos más complejos donde múltiples etapas del ciclo presentan errores simultáneamente.

El análisis de las confusiones fuera de la diagonal revela patrones coherentes y predecibles. Las confusiones más significativas ocurren entre errores compuestos y sus componentes simples: por ejemplo, la clase 7 (Initial-Transformation-Final) se confunde ocasionalmente con clase 3 (Final state) en 64 casos, y la clase 6 (Transformation-Final) con clase 2 (Transformation state) en 56 casos. Esto sugiere que cuando múltiples errores están presentes, el modelo a veces identifica correctamente que existe un error, pero no siempre detecta todas las etapas problemáticas.

Un hallazgo importante desde la perspectiva educativa es el análisis de las confusiones entre código correcto e incorrecto. Se identificaron 78 casos donde código con errores fue clasificado como correcto (principalmente 48 casos de Initial-Transformation y 28 de Transformation state), mientras que, en sentido inverso, solo 15 casos de código correcto fueron clasificados como erróneos. Esta asimetría representa un aspecto crítico a considerar: aunque el modelo mantiene un buen desempeño general, existe un mayor riesgo de generar falsos negativos (no detectar errores existentes) que falsos positivos (señalar errores inexistentes). Para una aplicación educativa, los 78 casos de código incorrecto clasificado como correcto (6.3% de los errores simples) constituyen situaciones donde estudiantes podrían recibir retroalimentación engañosa, lo cual debe considerarse en el despliegue del sistema. No obstante, este porcentaje sigue siendo relativamente bajo en el contexto del volumen total de clasificaciones.

d) Análisis del Reporte de Clasificación

El reporte de clasificación proporciona un resumen cuantitativo del desempeño del modelo para cada categoría, incluyendo las métricas de precisión (precision), exhaustividad (recall), F1-Score y el número de muestras de soporte para cada clase. Esta visualización mediante mapa de calor permite identificar rápidamente las fortalezas y debilidades del modelo a través de la intensidad del color: tonos rojizos indican valores más altos (cercanos a 1.0), mientras que tonos azulados señalan valores más bajos.

La Fig. 21 presenta el reporte de clasificación detallado del modelo GraphCodeBERT Modelo 1 sobre el conjunto de prueba. Cada fila corresponde a una de las ocho categorías de clasificación, mientras que las últimas filas presentan promedios agregados del desempeño global.

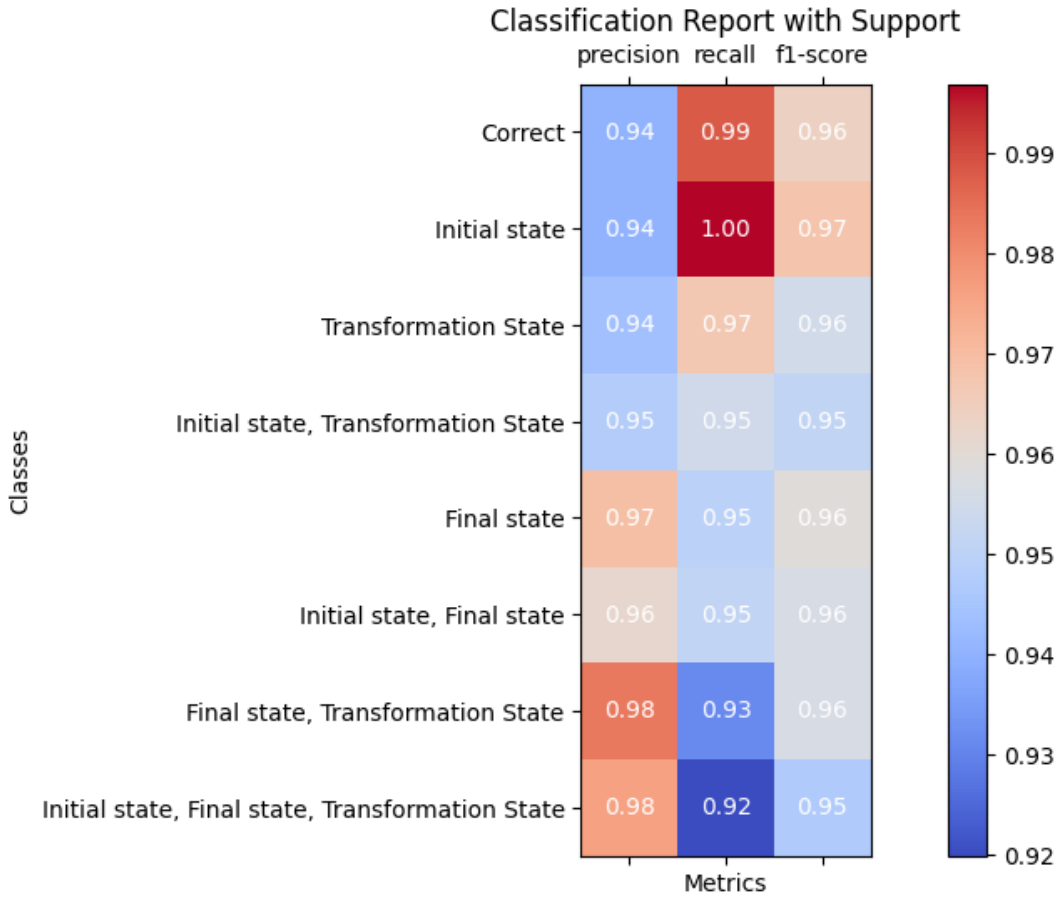


Fig. 21. Reporte de Clasificación

Nota: Elaboración propia.

Las categorías de errores simples y código correcto mantienen métricas consistentemente altas y balanceadas. Initial state destaca particularmente con un recall casi perfecto, representado por el color rojo más intenso del reporte, lo que significa que el modelo identificó correctamente casi todos los casos de esta categoría sin casi ningún falso negativo. Las demás categorías principales (Correct, Transformation State y Final state) también presentan desempeño sobresaliente, con valores muy cercanos al óptimo en las tres métricas evaluadas.

Se observa un patrón distintivo en las categorías de errores combinados: estas presentan precision más alta, pero recall ligeramente más bajo en comparación con las categorías simples. Este comportamiento es particularmente evidente en Final state, Transformation State e Initial state, Final state, Transformation State, donde la columna de precision muestra los colores más intensos del reporte, mientras que la columna de recall presenta tonalidades más claras. Este patrón indica que cuando el modelo predice estos errores compuestos, generalmente acierta (alta precision), pero tiende a no identificar todos los casos existentes (menor recall). El modelo es conservador al asignar etiquetas de error múltiple, requiriendo mayor certeza antes de clasificar un caso como error compuesto.

El F1-Score, que equilibra precision y recall, se mantiene notablemente estable y alto a través de todas las categorías, como lo evidencia la uniformidad de color en esa columna. Las categorías principales muestran una ligera ventaja en F1-Score, pero la diferencia con las categorías compuestas es mínima, demostrando que el modelo mantiene un balance adecuado incluso en los casos más complejos.

Así se da por cumplido el tercer objetivo específico “Evaluar el modelo aplicando métricas del Machine Learning para evidenciar la efectividad del modelo.”.

2) Despliegue

Una vez analizado y establecido cual es el mejor modelo, se optó por desplegarlo en una aplicación web de prototipo, la cual tendrá dos campos de entrada, el enunciado de la tarea de programación a resolver y la posible solución en código JavaScript; al enviar estas entradas el sistema mostrara el resultado de la predicción, indicando si el código es correcto, o si presenta algún error y donde se encuentra este.

Para lograr esto se realizó una API con el framework FastAPI, por su facilidad para el desarrollo y despliegue. Se cargo el modelo de predicción junto con sus dos modelos de generación de embeddings GraphCodeBERT y BERT.

El cliente realiza una petición POST al servidor, este procesa la respuesta (answer) de JavaScript extrayendo las secciones initial, transformation, js y final, estos campos serán procesados por GraphCodeBERT que generara los embeddings correspondientes. Por otro lado, el enunciado (question) será procesado por BERT que también generara su embedding correspondiente, los embeddings generados serán la entrada del modelo y este responderá al cliente la clasificación y las probabilidades de cada etiqueta.

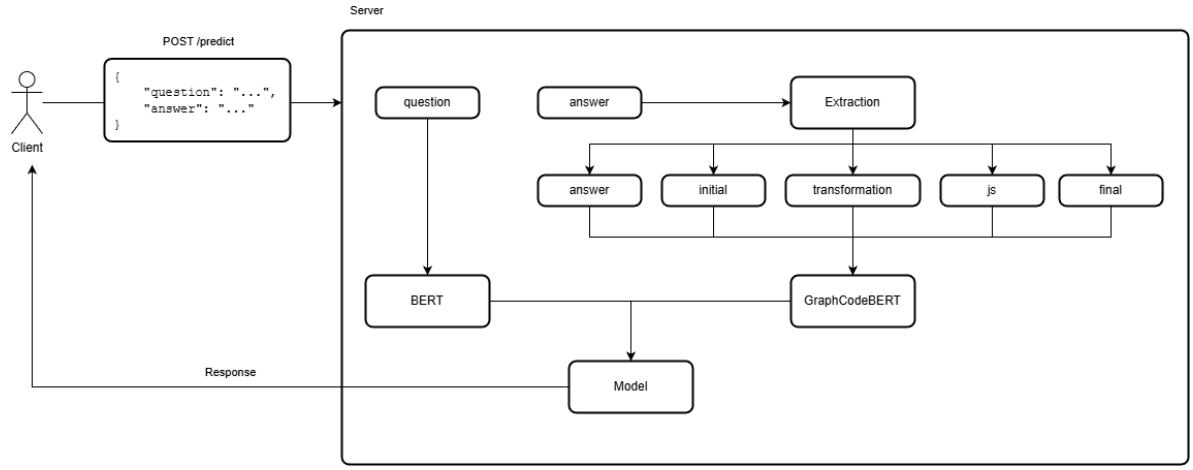


Fig. 22. Diagrama del Prototipo

Nota: Elaboración propia.

Para el cliente se optó por usar Vanilla JavaScript y obviamente HTML y CSS. Por la simplicidad del prototipo no se optó por algún framework en específico, esto solo retrasaría el desarrollo. El sistema contará con una lista desplegable para mostrar los diferentes enunciados y un editor de código (Monaco Editor) donde el usuario podrá colocar su respuesta en código JavaScript. Una vez los campos estén diligenciados se enviará a procesar los inputs y el servidor contestara con la clasificación, ahí se mostrará si bien el mensaje "El código es correcto" o cual y donde se encuentra el error.

Tanto el frontend como el backend fueron dockerizados, usando respectivamente imágenes base de Nginx y Python. Se desplegaron y ejecutaron en un VPS de Oracle Cloud y que son manejados por Nginx Proxy Manager.

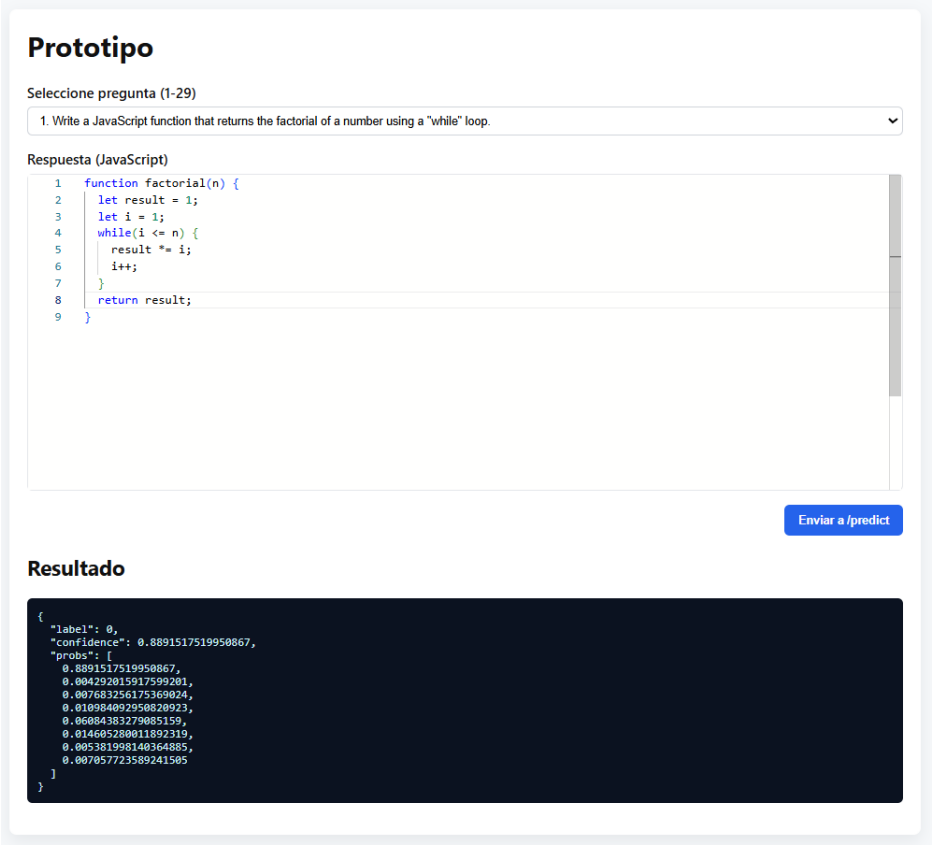


Fig. 23. Prototipo

Nota: Disponible en https://model.sanpezlo.duckdns.org/

Si bien esta aplicación es un prototipo, sirve como base para una implementación en un contexto educativo real, servirá tanto como una herramienta de aprendizaje para los estudiantes, como una herramienta de calificación automática que servirá de apoyo a los profesores.

V. ANÁLISIS Y DISCUSIÓN DE LOS RESULTADOS

Todos los scripts tanto de elaboración del dataset, preparación de los datos y modelado, fueron hechos en Python, debido a su sintaxis sencilla, su gran ecosistema de bibliotecas para el tratamiento de datos y creación de modelos de deep learning, su flexibilidad y la enorme comunidad que lo respaldan.

Para el desarrollo del prototipo fue necesario dividirlo en backend y frontend. Para el desarrollo del backend se realizó una API en Python, debido a que en este se construyó el modelo, era casi obligatorio que debía de estar en Python, sobre todo para agilizar el proceso de desarrollo. La decisión de usar el framework FastAPI sobre otros fue por su simpleza y para acelerar aún más el desarrollo.

Para el desarrollo del frontend no se quería complicarlo de más añadiendo algún framework, ya que no aportaría ninguna agilidad al ser un prototipo tan simple con solo un propósito. Por lo que se usó las tecnologías básicas para desarrollar una aplicación web HTML, CSS y JavaScript. Para brindar una mejor experiencia de usuario se utilizó Monaco Editor que es el editor de código que utiliza Visual Studio Code, por lo que los usuarios se sentirán familiarizados con su uso.

La plataforma de despliegue fue en un VPS de Oracle Cloud, debido a que ya se contaba con la posesión personal del mismo.

Como se mencionó anteriormente la presente investigación toma de referencia al trabajo "IMProB-It : automatic feedback model for iterative programming tasks in introductory programming courses" [9], de ahí se tomó como base el dataset y se utilizó las experimentaciones con arquitecturas de modelos, hiperparámetros y demás como referencia. Por lo que se hará una comparativa de los resultados logrados en esta investigación con la investigación de referencia.

En la elaboración del dataset, si bien no se aumentó el conjunto de enunciados o tareas comunes de programación, se amplió con gran diferencia las soluciones o respuestas de código JavaScript por cada enunciado. Se aumento un 825% tomando como referencia el dataset balanceado, gracias a la generación semi-automática del dataset.

También se identificó que algunas respuestas del dataset de referencia estaban incorrectamente etiquetadas, por esto se creó la necesidad de agregar unas pruebas unitarias por cada enunciado para garantizar que el dataset sea integro. Si bien el sistema actual de pruebas tiene algunas falencias mencionadas a continuación, y no es el definitivo, sirve como una base sólida para futuras investigaciones.

Las causas de que el dataset pueda tener algunos datos incorrectamente etiquetados es debido a la etiqueta de ignore-test, como se mencionó anteriormente evita que el test se ejecute para esa respuesta y confía de que esta correctamente etiquetada, fue necesaria para evitar los loops infinitos, pero en la generación del dataset puedo generar algunas respuestas etiquetadas incorrectamente sin loops infinitos que tenían la marca de ignore-test. Otra posible causa son las etiquetas combinadas, durante la fase de construcción del dataset, las etiquetas principales (Correct, Initial state, Transformation state, Final state) fueron generadas y validadas manualmente con pruebas automatizadas que verificaban el comportamiento del código. Sin embargo, las etiquetas combinadas se generaron automáticamente mediante la unión de secciones de diferentes archivos markdown.

Este método de generación automática introduce un riesgo potencial que al combinar secciones de distintos archivos que individualmente contienen errores, existe la posibilidad de que estos errores se compensen mutuamente, resultando en código que funciona correctamente a pesar de estar etiquetado como erróneo. Por ejemplo, si se toma una sección de inicialización del archivo initial.md que establece un contador en un valor incorrecto, y se combina con una sección de transformación del archivo transformation.md que modifica ese contador de forma también incorrecta, ambos errores podrían anularse entre sí, produciendo un resultado correcto. Sin embargo, el sistema automáticamente etiquetaría este código como error en Initial-Transformation. Por lo que para futuras investigaciones se recomienda agregar tests para etiquetas combinadas y si la respuesta no pasa los test excluirlo del dataset final.

El mejor modelo de la investigación de referencia obtuvo un 90% en la métrica de Accuracy y también un 90% en F1-Score, por lo que también se mejoró en este aspecto, donde el mejor modelo obtuvo un 95.9% de Accuracy y un 95.8% en F1-Score. La mejora puede deberse a diversos factores, tanto al aumento de datos, la integridad del dataset, la mejora en la arquitectura del modelo o incluso que los embeddings presenten mejores relaciones para el lenguaje JavaScript que para Python.

En esta investigación se probó si los modelos de clasificación basados en embeddings específicos para código, son capaces de identificar y categorizar errores en tareas con ciclos while en JavaScript. Los resultados agregados muestran que los modelos entrenados superan con claridad el comportamiento de un modelo aleatorio y alcanzan puntuaciones de F1 muy altas, lo cual permite ACEPTAR, la hipótesis de investigación “Los modelos de clasificación cuentan con una relación F1-Score tiempo superior que un modelo aleatorio, en todas las etiquetas de clasificación de errores en JavaScript que involucran ciclos while”.

El análisis por clases revela matices importantes. Para las etiquetas principales, es decir, Correct, Initial state, Transformation state y Final state, los modelos alcanzan altos valores de precision y recall, lo que se traduce en F1 altos y en una predicción confiable para errores aislados. Sin embargo, la mayor dificultad aparece en las etiquetas compuestas (combinaciones de fallos en más de una etapa), donde se registran las confusiones más frecuentes. Esto indica que, aunque el modelo

identifica correctamente errores simples, discriminar simultáneamente múltiples fallos requiere más señal o más ejemplos representativos.

Respecto al proceso de construcción del dataset y su impacto en el desempeño, la representación explícita del código en columnas separadas (initial, transformation, js, final) facilitó que los modelos aprendieran la correspondencia entre la etapa del ciclo y el tipo de error. El método de generación permitió crear variaciones sin repetir código. No obstante, hay fuentes de ruido que deben considerarse al interpretar los resultados: los tests automatizados empleados durante la generación validan si la solución es correcta o incorrecta, pero no siempre garantizan que un error detectado corresponda inequívocamente a la etapa señalada por la etiqueta.

Existe posibilidad de sesgo por la forma en que se generaron y balancearon los datos, el modelo puede bajar las métricas con ciertas soluciones creadas por estudiantes reales. Por último, la comparación entre modelos se hizo manteniendo fijo el embedding de NL y variando embeddings de código, por lo que se desconoce si estos cambios simultáneos en ambos embeddings afectarían el desempeño.

CONCLUSIONES

La presente investigación demostró la viabilidad técnica y educativa de aplicar técnicas de Machine Learning, específicamente modelos basados en Deep Learning con embeddings especializados en código, para la clasificación automática de errores lógicos en tareas de programación con ciclos while en JavaScript. Los resultados obtenidos validan la hipótesis planteada y confirman que los modelos de clasificación desarrollados superan ampliamente el desempeño de un clasificador aleatorio en todas las categorías de errores evaluadas.

El proceso de construcción del dataset representó uno de los aportes más significativos de esta investigación. Se logró generar un conjunto de datos balanceado de 49.496 ejemplos distribuidos equitativamente en ocho categorías de clasificación, partiendo de 29 problemas de programación y aplicando un enfoque innovador de generación mediante archivos markdown estructurados por secciones. Este método permitió crear variaciones de código de manera eficiente y escalable, reduciendo significativamente el tiempo de desarrollo. El dataset resultante constituye una contribución valiosa para futuras investigaciones en el área de análisis automático de código educativo en JavaScript.

La experimentación con diferentes embeddings especializados (CodeBERT, GraphCodeBERT y CodeT5) reveló diferencias sustanciales en el rendimiento de los modelos. GraphCodeBERT con Modelo 1 emergió como la configuración óptima, alcanzando un accuracy de 0.959 y un F1-Score de 0.958.

El desarrollo y despliegue de un prototipo funcional mediante una aplicación web demostró la factibilidad práctica de implementar el modelo en contextos educativos reales. Este prototipo sienta las bases para una herramienta de apoyo tanto para estudiantes, que pueden recibir retroalimentación inmediata sobre sus errores, como para docentes, que pueden utilizarla como sistema de calificación automática y análisis de patrones de error en sus grupos.

Finalmente, esta investigación contribuye al campo de la educación en programación al demostrar que es posible automatizar la detección y categorización de errores lógicos específicos, un tipo de error particularmente difícil de identificar mediante métodos tradicionales de análisis estático. Los resultados sientan un precedente para futuras investigaciones que extiendan este enfoque a otros lenguajes de programación, otras estructuras de control o tipos de errores más complejos, contribuyendo al desarrollo de sistemas inteligentes de apoyo al aprendizaje de la programación.

RECOMENDACIONES

Se recomienda prestar especial atención a la fase de construcción y validación del dataset, que constituye el fundamento del desempeño del modelo. La metodología de generación mediante archivos markdown estructurados por secciones demostró ser efectiva, pero es fundamental implementar un sistema robusto de validación que verifique no solo que el código presenta errores, sino que los errores se encuentran específicamente en las etapas indicadas por la etiqueta. Para las etiquetas combinadas, se sugiere sustituir la marca "#(ignore-test)" por un entorno de ejecución controlado o pruebas estáticas más sofisticadas que permitan verificar comportamientos sin riesgo de bloqueo por bucles infinitos.

Es crítico considerar la posible compensación de errores en la generación automática de etiquetas combinadas. Al combinar secciones de diferentes archivos que individualmente contienen errores, existe el riesgo de que estos errores se anulen mutuamente, resultando en código funcional pero etiquetado como erróneo. Para mitigar este problema, se recomienda implementar pruebas de validación exhaustivas para cada combinación generada y excluirla del dataset.

En cuanto a la arquitectura del modelo, aunque el Modelo 1 demostró ser superior con CodeBERT y GraphCodeBERT, se recomienda experimentar con arquitecturas jerárquicas que implementen un enfoque en dos etapas: un primer clasificador binario que distinga entre código correcto e incorrecto, seguido de un clasificador multiclase que identifique el tipo específico de error. Esta estrategia podría mejorar significativamente las métricas de las categorías compuestas, que presentaron el mayor desafío en la clasificación.

Para futuras investigaciones se sugiere:

- **Enriquecer el dataset con código real de estudiantes:** Incorporar ejemplos auténticos extraídos de entornos docentes reales, garantizando el anonimato y el consentimiento correspondiente. Esto permitirá evaluar la robustez del sistema frente a estilos de codificación diversos y ayudará a identificar posibles sesgos introducidos por la generación automática de datos.
- **Optimización automatizada de hiperparámetros:** Utilizar herramientas como Keras Tuner para explorar sistemáticamente el espacio de hiperparámetros y encontrar configuraciones óptimas que puedan mejorar aún más el desempeño del modelo.
- **Extensión a otros lenguajes y estructuras:** Replicar la metodología en lenguajes como Python, Java o C++, y extenderla a otras estructuras de control como ciclos for, do-while, condicionales anidados o recursividad. Esto contribuiría a desarrollar un sistema integral de análisis de código educativo.
- **Estudio en contextos educativos:** Implementar el sistema en cursos reales de programación y realizar estudios controlados que midan su impacto en el aprendizaje, la motivación estudiantil y la carga de trabajo docente.

REFERENCIAS BIBLIOGRÁFICAS

[1] C. Y. Tsai, "Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy," *Comput. Human Behav.*, vol. 95, pp. 224–232, Jun. 2019, doi: 10.1016/J.CHB.2018.11.038.

[2] C. S. Cheah, "Factors Contributing to the Difficulties in Teaching and Learning of Computer Programming: A Literature Review," *Contemp. Educ. Technol.*, vol. 12, no. 2, p. ep272, May 2020, doi: 10.30935/CEDTECH/8247.

[3] C. Shearer, "The CRISP-DM model: the new blueprint for data mining," *Journal of data warehousing*, vol. 5, no. 4, pp. 13–22, 2000.

[4] M. Grandini, E. Bagli, and G. Visani, "Metrics for Multi-Class Classification: an Overview," Aug. 2020, Accessed: Oct. 29, 2024. [Online]. Available: https://arxiv.org/abs/2008.05756v1

[5] T. Sirkiä and J. Sorva, "Exploring programming misconceptions: An analysis of student mistakes in visual program simulation exercises," *Proceedings - 12th Koli Calling International Conference on Computing Education Research, Koli Calling 2012*, pp. 19–28, 2012, doi: 10.1145/2401796.2401799.

[6] R. Caceffo, P. Frank-Bolton, R. Souza, and R. Azevedo, "Identifying and validating Java misconceptions toward a CS1 concept inventory," *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, pp. 23–29, Jul. 2019, doi: 10.1145/3304221.3319771.

[7] "2024 Stack Overflow Developer Survey." Accessed: Aug. 26, 2024. [Online]. Available: https://survey.stackoverflow.co/2024/

[8] S. Liénardy, L. Leduc, D. Verpoorten, and B. Donnet, "Café: Automatic correction and feedback of programming challenges for a CS1 course," *ACE 2020 - Proceedings of the 22nd Australasian Computing Education Conference, Held in conjunction with Australasian Computer Science Week*, pp. 95–104, Feb. 2020, doi: 10.1145/3373165.3373176.

[9] G. Leytón, J. F. Diaz, and A. M. Castillo Robles, "IMProB-It : automatic feedback model for iterative programming tasks in introductory programming courses," 2025, *Universidad del Valle*. Accessed: Sep. 10, 2025. [Online]. Available: https://hdl.handle.net/10893/36036

[10] D. Gries, "The Science of Programming," *The Science of Programming*, 1981, doi: 10.1007/978-1-4612-5983-1.

[11] "Precios de los servicios pagados de Colab." Accessed: Sep. 13, 2024. [Online]. Available: https://colab.research.google.com/signup

[12] "scikit-learn/scikit-learn: scikit-learn: machine learning in Python." Accessed: Oct. 06, 2024. [Online]. Available: https://github.com/scikit-learn/scikit-learn

[13] "tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone." Accessed: Oct. 06, 2024. [Online]. Available: https://github.com/tensorflow/tensorflow

[14] "google-research/bert: TensorFlow code and pre-trained models for BERT." Accessed: Oct. 06, 2024. [Online]. Available: https://github.com/google-research/bert

[15] “microsoft/CodeBERT: CodeBERT.” Accessed: Oct. 06, 2024. [Online]. Available: https://github.com/microsoft/CodeBERT

[16] “bdqnghi/infercode: [ICSE 2021] - InferCode: Self-Supervised Learning of Code Representations by Predicting Subtrees.” Accessed: Oct. 06, 2024. [Online]. Available: https://github.com/bdqnghi/infercode

[17] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, pp. 4171–4186, Oct. 2018, Accessed: Sep. 16, 2024. [Online]. Available: https://arxiv.org/abs/1810.04805v2

[18] Z. Feng *et al.*, “CodeBERT: A Pre-Trained Model for Programming and Natural Languages,” *Findings of the Association for Computational Linguistics Findings of ACL: EMNLP 2020*, pp. 1536–1547, Feb. 2020, doi: 10.18653/v1/2020.findings-emnlp.139.

[19] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, “code2vec: Learning Distributed Representations of Code,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, Mar. 2018, doi: 10.1145/3290353.

[20] M. D. Alahmadi, M. Alshangiti, and J. Alsubhi, “SCC-GPT: Source Code Classification Based on Generative Pre-Trained Transformers,” *Mathematics 2024, Vol. 12, Page 2128*, vol. 12, no. 13, p. 2128, Jul. 2024, doi: 10.3390/MATH12132128.

[21] S. Sarsa, J. Leinonen, C. Koutcheme, and A. Hellas, “Speeding Up Automated Assessment of Programming Exercises,” *ACM International Conference Proceeding Series*, Sep. 2022, doi: 10.1145/3555009.3555013.

[22] A. Lobanov, T. Bryksin, and A. Shpilman, “Automatic Classification of Error Types in Solutions to Programming Assignments at Online Learning Platform,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11626 LNAI, pp. 174–178, 2019, doi: 10.1007/978-3-030-23207-8_33.

[23] S. Enrique Campaña Bastidas, A. Méndez Porras, A. Milena Santacruz Madroñero, A. Alejandro Díaz Torro, and Á. José Cervelión Bastidas, “Sistema de reconocimiento facial y de emociones aplicado a la educación básica y media de una institución educativa en Colombia con herramientas de la 4RI,” *Encuentro Internacional de Educación en Ingeniería*, pp. 1–12, Sep. 2023, doi: 10.26507/PAPER.3342.

[24] A. E. N. Figueroa, G. C. Castellano, J. Sebastián, and G. Sanabria, “Modelo de machine learning para la clasificación de municipios por cultivos ilícitos en Colombia de 2010 a 2020,” *Inge CuC*, vol. 19, no. 1, pp. 47-60–47–60, Jan. 2023, doi: 10.17981/INGECUC.19.1.2023.05.

[25] G. Alfano, A. J. García, F. Parisi, J. Insuasti, F. Roa, and C. M. Zapata-Jaramillo, “Computers’ Interpretations of Knowledge Representation Using Pre-Conceptual Schemas: An Approach Based on the BERT and Llama 2-Chat Models,” *Big Data and Cognitive*

Computing 2023, Vol. 7, Page 182, vol. 7, no. 4, p. 182, Dec. 2023, doi: 10.3390/BDCC7040182.

[26] J. Homepage *et al.*, "Analysis of Satellite Images Using Deep Learning Techniques and Remotely Piloted Aircraft for a Detailed Description of Tertiary Roads," *Revista Facultad de Ingeniería*, vol. 30, no. 58, p. e13816, Dec. 2021, doi: 10.19053/01211129.V30.N58.2021.13816.

[27] F. Cujar-Rosero, "Nature: A Tool Resulting from the Union of Artificial Intelligence and Natural Language Processing for Searching Research Projects in Colombia," Jul. 01, 2021. Accessed: Sep. 16, 2024. [Online]. Available: https://papers.ssrn.com/abstract=3927551

[28] P. por, M. Paz, H. Andrés Directora, M. Valets, and H. Andrés, "Comparativo de kernels sobre predicción de oferta de fuentes alternativas de energía," Oct. 2019, Accessed: Sep. 17, 2024. [Online]. Available: https://reunir.unir.net/handle/123456789/10020

[29] D.-E. Álvarez Sánchez *et al.*, "Use of Trained Convolutional Neural Networks for Analysis of Symptoms Caused by Botrytis fabae Sard," *Revista de Ciencias Agrícolas*, vol. 40, no. 1, pp. e1198–e1198, Apr. 2023, doi: 10.22267/RCIA.20234001.198.

[30] B. C. Jamanoy Bacca and J. F. Montenegro Rosero, "Gestión de información académica de solicitudes estudiantiles y docentes en la Jefatura de Software de la UNICESMAG mediante Chatbot aplicando PLN," Universidad CESMAG, 2023. Accessed: Sep. 17, 2024. [Online]. Available: http://repositorio.unicesmag.edu.co:8080/xmlui/handle/123456789/998

[31] M. Jose, D. Guaquez, T. Julieth, and V. Celis, "Implementación de una red neuronal para la clasificación de imágenes histológicas de Cáncer de Mama," Sep. 2024, Accessed: Sep. 17, 2024. [Online]. Available: https://repositorio.umariana.edu.co//handle/20.500.14112/28517

[32] C. A. Furia, B. Meyer, and S. Velder, "Loop invariants: analysis, classification, and examples," *ACM Comput. Surv.*, vol. 46, no. 3, Nov. 2012, doi: 10.1145/2506375.

[33] M. G. Estayno, G. N. Dapozo, L. R. Cuenca Pletsch, C. L. Greiner, and Y. Medina, "Estudio comparativo de metodologías para minería de datos," *XIII Workshop de Investigadores en Ciencias de la Computación*, 2011, Accessed: Oct. 13, 2024. [Online]. Available: http://sedici.unlp.edu.ar/handle/10915/20034

[34] P. Chapman, "CRISP-DM 1.0: Step-by-step data mining guide," 2000.

[35] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, "Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming," *Artificial Intelligence in Design '96*, pp. 151–170, 1996, doi: 10.1007/978-94-009-0279-4_9.

[36] A. T. Mehryar Mohri, Afshin Rostamizadeh, "Foundation of Machine Learning," *The MIT Press*, vol. 4, no. 1, pp. 88–100, 2018.

[37] V. N. Vapnik, "The Support Vector method," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1327, pp. 261–271, 1997, doi: 10.1007/BFB0020166.

[38] C. Cortes, V. Vapnik, and L. Saitta, "Support-vector networks," *Machine Learning 1995 20:3*, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/BF00994018.

[39] K. B. Duan and S. S. Keerthi, “Which Is the Best Multiclass SVM Method? An Empirical Study,” *Lecture Notes in Computer Science*, vol. 3541, pp. 278–285, 2005, doi: 10.1007/11494683_28.

[40] J. R. Quinlan, “Induction of decision trees,” *Machine Learning 1986 1:1*, vol. 1, no. 1, pp. 81–106, Mar. 1986, doi: 10.1007/BF00116251.

[41] T. K. Ho, “Random decision forests,” *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, vol. 1, pp. 278–282, 1995, doi: 10.1109/ICDAR.1995.598994.

[42] T. K. Ho, “The random subspace method for constructing decision forests,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, 1998, doi: 10.1109/34.709601.

[43] T. M. Cover and P. E. Hart, “Nearest Neighbor Pattern Classification,” *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, 1967, doi: 10.1109/TIT.1967.1053964.

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[45] J. Opitz, “A Closer Look at Classification Evaluation Metrics and a Critical Reflection of Common Evaluation Practice,” *Trans. Assoc. Comput. Linguist.*, vol. 12, pp. 820–836, Oct. 2024, doi: 10.1162/TACL_A_00675/122720/A-CLOSER-LOOK-AT-CLASSIFICATION-EVALUATION-METRICS.

[46] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature 2015 521:7553*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[47] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, “Dive into Deep Learning,” *Journal of the American College of Radiology*, vol. 17, no. 5, pp. 637–638, Jun. 2021, doi: 10.1016/j.jacr.2020.02.005.

[48] A. Daigavane, B. Ravindran, and G. Aggarwal, “Understanding Convolutions on Graphs,” *Distill*, vol. 6, no. 9, p. e32, Sep. 2021, doi: 10.23915/DISTILL.00032.

[49] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 6999–7019, Dec. 2022, doi: 10.1109/TNNLS.2021.3084827.

[50] S. Abbaspour, F. Fotouhi, A. Sedaghatbaf, H. Fotouhi, M. Vahabi, and M. Linden, “A Comparative Analysis of Hybrid Deep Learning Models for Human Activity Recognition,” *Sensors 2020, Vol. 20, Page 5707*, vol. 20, no. 19, p. 5707, Oct. 2020, doi: 10.3390/S20195707.

[51] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/NECO.1997.9.8.1735.

[52] X. Zhu, P. Sobhani, and H. Guo, “Long Short-Term Memory Over Tree Structures,” Mar. 2015, Accessed: Oct. 12, 2024. [Online]. Available: https://arxiv.org/abs/1503.04881v1

[53] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with LSTM,” *IEE Conference Publication*, vol. 2, no. 470, pp. 850–855, 1999, doi: 10.1049/CP:19991218.

[54] A. Madsen, “Visualizing memorization in RNNs,” *Distill*, vol. 4, no. 3, p. e16, Mar. 2019, doi: 10.23915/DISTILL.00016.

[55] A. Vaswani *et al.*, “Attention Is All You Need,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-December, pp. 5999–6009, Jun. 2017, Accessed: Oct. 12, 2024. [Online]. Available: https://arxiv.org/abs/1706.03762v7

[56] “NLP - overview.” Accessed: Oct. 12, 2024. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/nlp/overview_history.html

[57] Dan. Jurafsky and J. H. . Martin, “Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition,” p. 934, 2000.

[58] C. M. Bishop, *Pattern recognition and machine learning*. 2006. Accessed: Sep. 30, 2024. [Online]. Available: https://link.springer.com/book/9780387310732

[59] J. Howard and S. Gugger, *Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD*. O’Reilly Media, Incorporated, 2020. [Online]. Available: https://books.google.no/books?id=xd6LxgEACAAJ

[60] J. Morris, “Do text embeddings perfectly encode text?,” *The Gradient*, 2024.

[61] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global Vectors for Word Representation,” *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1532–1543, 2014, doi: 10.3115/V1/D14-1162.

[62] A. Géron, “Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems,” p. 821, 2019.

[63] M. Peter. Deisenroth, A. Aldo. Faisal, and C. Soon. Ong, “Mathematics for machine learning,” p. 371, 2020, Accessed: Sep. 30, 2024. [Online]. Available: https://books.google.com/books/about/Mathematics_for_Machine_Learning.html?hl=es&id=pbONxAEACAAJ

[64] T. Hastie, R. Tibshirani, and J. Friedman, “The Elements of Statistical Learning,” 2009, doi: 10.1007/978-0-387-84858-7.

[65] A. J. Quijano Vodniza, *Guía de investigación cuantitativa*. Institución Universitaria CESMAG, 2009.

[66] L. Cuenya and E. Ruetti, “Epistemological and Methodological Controversies Between the Qualitative and Quantitative Paradigm in Psychology,” *Revista Colombiana de Psicología*, vol. 19, pp. 271–277, Oct. 2010.

[67] J. O. Lozada, “Investigación Aplicada: Definición, Propiedad Intelectual e Industria,” *CienciAmérica: Revista de divulgación científica de la Universidad Tecnológica Indoamérica, ISSN-e 1390-9592, Vol. 3, Nº. 1, 2014, págs. 47-50*, vol. 3, no. 1, pp. 47–50, 2014, Accessed: Oct. 20, 2024. [Online]. Available: https://dialnet.unirioja.es/servlet/articulo?codigo=6163749&info=resumen&idioma=ENG

[68] C. D. Correa Lozano, D. F. Urrea Burgos, and J. A. Lozano Thomé, “Evaluación De Métodos De Reducción De Dimensión Para La Preservación Topológica De Los Datos Mediante Métricas Rnx,” Universidad CESMAG, 2021.

[69] M. R. Rodríguez Rodríguez and C. A. Delgado Calpa, “Comparativo de funciones kernel en la predicción de enfermedades cardiovasculares en Redes Neuronales Artificiales (ANN) y Máquinas de Soporte Vectorial (SVM),” Universidad CESMAG, 2024. Accessed: Oct. 21, 2024. [Online]. Available: http://repositorio.unicesmag.edu.co:8080/xmlui/handle/123456789/1181

[70] R. O. Tinoco, E. B. Goldstein, and G. Coco, “A data-driven approach to develop physically sound predictors: Application to depth-averaged velocities on flows through submerged arrays of rigid cylinders,” *Water Resour. Res.*, vol. 51, no. 2, pp. 1247–1263, Feb. 2015, doi: 10.1002/2014WR016380.

[71] “Introducing ChatGPT | OpenAI.” Accessed: Oct. 27, 2024. [Online]. Available: https://openai.com/index/chatgpt/

ANEXOS

Anexo A Carta aval del asesor

San Juan de Pasto, 09 de noviembre de 2025

Señores:
Comité de Investigaciones Programa de
Ingeniería de Sistemas Universidad
CESMAG

Asunto: Aval Trabajo de Grado

Saludo de paz y bien,

Yo, Mg. Héctor Andrés Mora Paz identificado con cédula de ciudadanía Nro. 1085251119, doy aval que el trabajo de grado, "CATEGORIZACIÓN AUTOMÁTICA APLICANDO TÉCNICAS DE MACHINE LEARNING DE ERRORES EN TAREAS DE PROGRAMACIÓN CON CICLOS EN JAVASCRIPT", realizado por el estudiante: SANTIAGO DAVID LÓPEZ BENAVIDES se puede presentar el documento final a revisión de Jurados, de igual manera, notifico que los estudiantes asistieron a las respectivas asesorías conforme a cronograma de actividades.

Atentamente,

Mg. Héctor Andrés Mora Paz
C.C. 1085251119
Asesor del proyecto

UNIVERSIDAD CESMAG NIT: 800.109.387-7 VIGILADA MINEDUCACIÓN	**CARTA DE ENTREGA TRABAJO DE GRADO O TRABAJO DE APLICACIÓN – ASESOR(A)**	**CÓDIGO:** AAC-BL-FR-032
		VERSIÓN: 1
		FECHA: 09/JUN/2022

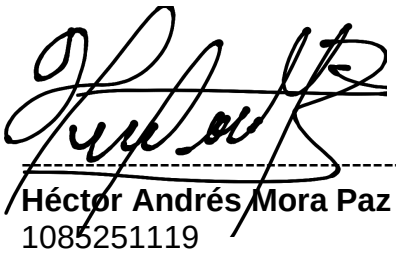
San Juan de Pasto, 28 de abril de 2026

Biblioteca
REMIGIO FIORE FORTEZZA OFM. CAP.
Universidad CESMAG
Pasto

Saludo de paz y bien.

Por medio de la presente se hace entrega del Trabajo de Grado / Trabajo de Aplicación denominado Categorización automática aplicando técnicas de machine learning de errores lógicos en tareas de programación con ciclos en javascript, presentado por el autor Santiago David Lopez Benavides del Programa Académico Ingeniería de Sistemas al correo electrónico biblioteca.trabajosdegrado@unicesmag.edu.co. Manifiesto como asesor(a), que su contenido, resumen, anexos y formato PDF cumple con las especificaciones de calidad, guía de presentación de Trabajos de Grado o de Aplicación, establecidos por la Universidad CESMAG, por lo tanto, se solicita el paz y salvo respectivo.

Atentamente,



Héctor Andrés Mora Paz
1085251119
Ingeniería de Sistemas
3172537641
hamora@unicesmag.edu.co

INFORMACIÓN DEL (LOS) AUTOR(ES)	
Nombres y apellidos del autor: Santiago David Lopez Benavides	**Documento de identidad:** 1080040001
Correo electrónico: sdlopez.0001@unicesmag.edu.co	**Número de contacto:** 3103771821
Nombres y apellidos del asesor: Héctor Andrés Mora Paz	**Documento de identidad:** 1085251119
Correo electrónico: hamora@unicesmag.edu.co	**Número de contacto:** 3172537641
Título del trabajo de grado: Categorización automática aplicando técnicas de machine learning de errores lógicos en tareas de programación con ciclos en javascript	
Facultad y Programa Académico: Ingeniería de sistemas	

En mi (nuestra) calidad de autor(es) y/o titular (es) del derecho de autor del Trabajo de Grado o de Aplicación señalado en el encabezado, confiero (conferimos) a la Universidad CESMAG una licencia no exclusiva, limitada y gratuita, para la inclusión del trabajo de grado en el repositorio institucional. Por consiguiente, el alcance de la licencia que se otorga a través del presente documento, abarca las siguientes características:

a) La autorización se otorga desde la fecha de suscripción del presente documento y durante todo el término en el que el (los) firmante(s) del presente documento conserve (mos) la titularidad de los derechos patrimoniales de autor. En el evento en el que deje (mos) de tener la titularidad de los derechos patrimoniales sobre el Trabajo de Grado o de Aplicación, me (nos) comprometo (comprometemos) a informar de manera inmediata sobre dicha situación a la Universidad CESMAG. Por consiguiente, hasta que no exista comunicación escrita de mi(nuestra) parte informando sobre dicha situación, la Universidad CESMAG se encontrará debidamente habilitada para continuar con la publicación del Trabajo de Grado o de Aplicación dentro del repositorio institucional. Conozco(conocemos) que esta autorización podrá revocarse en cualquier momento, siempre y cuando se eleve la solicitud por escrito para dicho fin ante la Universidad CESMAG. En estos eventos, la Universidad CESMAG cuenta con el plazo de un mes después de recibida la petición, para desmarcar la visualización del Trabajo de Grado o de Aplicación del repositorio institucional.
b) Se autoriza a la Universidad CESMAG para publicar el Trabajo de Grado o de Aplicación en formato digital y teniendo en cuenta que uno de los medios de publicación del repositorio institucional es el internet, acepto(amos) que el Trabajo de Grado o de Aplicación circulará con un alcance mundial.
c) Acepto (aceptamos) que la autorización que se otorga a través del presente documento se realiza a título gratuito, por lo tanto, renuncio(amos) a recibir emolumento alguno por la publicación, distribución, comunicación pública y/o cualquier otro uso que se haga en los términos de la presente autorización y de la licencia o programa a través del cual sea publicado el Trabajo de grado o de Aplicación.
d) Manifiesto (manifestamos) que el Trabajo de Grado o de Aplicación es original realizado sin violar o usurpar derechos de autor de terceros y que ostento(amos) los derechos patrimoniales de autor sobre la misma. Por consiguiente, asumo(asumimos) toda la responsabilidad sobre su contenido ante la Universidad CESMAG y frente a terceros, manteniéndose indemne de cualquier reclamación que surja en virtud de la misma. En todo caso, la Universidad CESMAG se

compromete a indicar siempre la autoría del escrito incluyendo nombre de(los) autor(es) y la fecha de publicación.

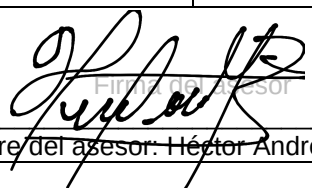
e) Autorizo(autorizamos) a la Universidad CESMAG para incluir el Trabajo de Grado o de Aplicación en los índices y buscadores que se estimen necesarios para promover su difusión. Así mismo autorizo (autorizamos) a la Universidad CESMAG para que pueda convertir el documento a cualquier medio o formato para propósitos de preservación digital.

NOTA: En los eventos en los que el trabajo de grado o de aplicación haya sido trabajado con el apoyo o patrocinio de una agencia, organización o cualquier otra entidad diferente a la Universidad CESMAG. Como autor(es) garantizo(amos) que he(hemos) cumplido con los derechos y obligaciones asumidos con dicha entidad y como consecuencia de ello dejo(dejamos) constancia que la autorización que se concede a través del presente escrito no interfiere ni transgrede derechos de terceros.

Como consecuencia de lo anterior, autorizo(autorizamos) la publicación, difusión, consulta y uso del Trabajo de Grado o de Aplicación por parte de la Universidad CESMAG y sus usuarios así:

- Permito(permitimos) que mi(nuestro) Trabajo de Grado o de Aplicación haga parte del catálogo de colección del repositorio digital de la Universidad CESMAG por lo tanto, su contenido será de acceso abierto donde podrá ser consultado, descargado y compartido con otras personas, siempre que se reconozca su autoría o reconocimiento con fines no comerciales.

En señal de conformidad, se suscribe este documento en San Juan de Pasto a los 28 días del mes de Abril del año 2026

 Firma del autor	Firma del autor
Nombre del autor: Santiago David Lopez Benavides	Nombre del autor:
 Firma del asesor Nombre del asesor: Héctor Andrés Mora Paz	