



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CULIACÁN



Desarrollo Backend Para Integrar Modelos De Ia En Una Página Web

TESIS

para

OBTENER EL GRADO DE

INGENIERO EN SISTEMAS COMPUTACIONALES

POR:

Jhonatan Alexis De La Torre Castro

DIRECTOR DE TESIS:

Arcelia Judith Bustillos Martínez

CODIRECTOR DE TESIS:

Héctor Rodríguez Rangel

CULIACÁN, SINALOA

21 de octubre del 2025



Educación
Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Culiacán

DIVISION DE ESTUDIOS PROFESIONALES
ASUNTO: **Autorización de Impresión**

Culiacán, Sinaloa, 23/09/2025 19:23

C. JHONATAN ALEXIS DE LA TORRE CASTRO
PASANTE(S) DE LA CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES
P R E S E N T E.

De acuerdo con fallo emitido por la Comisión Revisora Integrada por:

Cédula Profesional

Presidente:	Maestría en Ciencias en Ciencias Computacionales Arcelia Judith Bustillos Martínez	13118410
Secretario:	Maestría en Ciencias de la Computación José Luis Medina Jiménez	13778869
Vocal:	Maestría en Informática Aplicada Rogelio Prieto Alvarado	13976653

Y considerando que reúne los requisitos establecidos en la Opción: **TITULACIÓN INTEGRAL: TESIS PROFESIONAL "DESARROLLO BLACKEND PARA INTEGRAR MODELOS DE IA EN UNA PAGINA WEB"**, damos a usted nuestra autorización para que proceda a imprimir su trabajo profesional.

Sin otro particular, quedo de usted.

ATENTAMENTE
Excelencia en Educación Tecnológica®

CONCEPCION MENDOZA ROSALES
LIC. CONCEPCION MENDOZA ROSALES
JEFA DE LA DIVISION DE ESTUDIOS PROFESIONALES



Educación
Secretaría de Educación Pública



INSTITUTO TECNOLÓGICO DE CULIACÁN

DEPARTAMENTO DE
DIVISION DE ESTUDIOS PROFESIONALES



2025
Año de
La Mujer
Indígena

Juan de Dios Bátiz 310 Pte. Col. Guadalupe C.P. 80220
Culiacán, Sinaloa. Tel. 667-454-0100
tecnm.mx www.culiacan.tecnm.mx



Hoja de firmas

Dedicatoria

A mis amados padres, quienes han sido mi faro y mi roca. Gracias por su fe inquebrantable en mí, por las incontables horas de paciencia, por cada palabra de aliento y por enseñarme el valor del esfuerzo. Esta tesis es el reflejo de su sacrificio y de todo el apoyo incondicional que me han dado a lo largo de los años. Sin ustedes, este sueño no hubiera sido posible.

A mi querido hermano, mi primer amigo y mi confidente. Gracias por ser mi compañero de risas y mi apoyo en los momentos difíciles. Tu presencia y tu cariño han hecho este camino mucho más llevadero y alegre.

Este logro es tanto de ustedes como mío. Gracias por ser mi mayor inspiración.

Agradecimientos

Me gustaría agradecer a mis familiares en especial a Eunice Castro García y Juan De La Torre Rodríguez, por brindarme el apoyo constante y creer en mi, contribuyendo a mi formación académica, reconocer su dedicación y esfuerzo para salir adelante, reconocer a mi hermano mayor, Carlos Daniel Castañeda Castro por estar para mi en los momentos más difíciles donde pensé que no podía, por su apoyo incondicional, sin su apoyo no podría aspirar a culminar una carrera.

Reconocer al Doctor Héctor Rodríguez Rangel por aceptarme como residente junto al M.C. José Luis Medina Jiménez, ambos docentes en el Instituto Tecnológico Nacional de México campus Culiacán, ambos docentes de Inteligencia Artificial quienes anteriormente sentaron las bases para realizar este proyecto y me dieron esta oportunidad.

Finalmente reconocer el apoyo de mis compañeros y amigos cercanos, otorgando una mención especial a Derian Molina López, quien me enseñó a trabajar y resolver problemas en equipo, a Kenyi Shigeru Cervantes Creano quien al igual que mi familia me demostró que el esfuerzo y dedicación nos llevan lejos, a Cesar Macdiel Lizarraga Beltrán quien estuvo para escucharme en momentos difíciles.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Jhonatan Alexis De La Torre Castro

Culiacán, Sinaloa, México, 2024

Resumen

La calidad del agua es fundamental para la salud pública y el equilibrio de los ecosistemas acuáticos. Sin embargo, los métodos de análisis tradicionales son lentos y costosos, lo que limita su aplicación frecuente. Este proyecto busca resolver esta problemática mediante el desarrollo de un backend para una aplicación web que predice parámetros y nutrientes clave en muestras de agua de forma rápida y accesible.

El objetivo principal fue el diseño e implementación de este backend, una solución tecnológica que optimiza el proceso de análisis al reducir significativamente el tiempo de espera. La aplicación se enfoca en predecir cinco parámetros esenciales: Oxígeno Disuelto (OD), Temperatura del agua, Conductividad de campo, pH y Sólidos Suspendidos Totales (SST).

El sistema fue construido con Python y el framework Flask, lo que asegura una arquitectura eficiente y ligera. Se utilizó PostgreSQL para la base de datos, garantizando una gestión óptima de la información de las muestras. Para asegurar su disponibilidad y escalabilidad global, el backend fue desplegado en Heroku. El desarrollo se realizó con metodologías ágiles, permitiendo la entrega de resultados funcionales en cada etapa.

En conclusión, este proyecto ofrece una solución tecnológica avanzada y eficiente para el monitoreo de la calidad del agua. El backend desarrollado no solo acorta los tiempos de análisis, sino que también sirve como una herramienta accesible y escalable, especialmente útil para su uso en campo o en entornos remotos. Este sistema sienta las bases para futuras aplicaciones en el monitoreo ambiental y la gestión sostenible de los recursos hídricos, facilitando la toma de decisiones informadas para la protección del medio ambiente.

Palabras clave

Aquí se escriben las palabras que juegan un papel importante en el desarrollo de este trabajo de investigación y aquellas que logran condensar los temas más importantes que presenta el trabajo.

Inteligencia Artificial

Backend

Python

Java

Flask

ONNX

Índice general

1.	Introducción	1.1-1
1.1.	Descripción del problema	1.1-2
1.2.	Hipótesis	1.2-3
1.3.	Objetivos.....	1.3-3
1.3.1.	Objetivo general.....	1.3-3
1.3.2.	Objetivos específicos.....	1.3-4
1.4.	Justificación.....	1.4-4
2.	Marco teórico.....	1.4-6
2.1.	Aprendizaje Maquina.....	2.1-6
2.1.1.	Aplicaciones de ML	2.1-6
2.1.2.	Datos	2.1-7
2.1.3.	Tipos de algoritmos de ML.....	2.1-8
2.1.4.	Tipos de escaladores de datos	2.1-9
2.2.	API como servicios	2.2-9
2.2.1.	¿Qué es una API?	2.2-10
2.2.2.	Funcionamiento de una API	2.2-10
2.2.3.	Beneficios de las API	2.2-11
2.2.4.	Aplicaciones comunes de las API	2.2-12
2.2.5.	Tipos de API.....	2.2-14
2.2.6.	Tipos de protocolos de API	2.2-14
2.2.7.	Arquitecturas.....	2.2-14
2.2.8.	Arquitectura cliente servidor	2.2-15
2.2.9.	Arquitectura MVC	2.2-15
2.2.10.	Arquitectura MVVM.....	2.2-16
2.2.11.	API y arquitectura de nube	2.2-16
3.	Estado del arte	2.2-18
3.1.	Creación de API	3.1-18
3.2.	Identificación del modelo	3.2-20
3.3.	Procesamiento de datos	3.3-21
4.	Desarrollo del proyecto.....	3.3-24
4.1.	Análisis	4.1-24

4.1.1.	Análisis de requisitos.....	4.1-24
4.1.2.	Requisitos funcionales	4.1-25
4.1.3.	Requisitos de Calidad	4.1-26
4.1.4.	Actores	4.1-26
4.1.5.	Casos de uso.....	4.1-27
4.2.	Diseño	4.2-28
4.2.1.	Arquetipos.....	4.2-28
4.2.2.	Relación de arquetipos	4.2-29
4.2.3.	Diagrama de contexto.....	4.2-30
4.2.4.	Modelo arquitectónico	4.2-30
4.2.5.	Capas Lógicas	4.2-31
4.2.6.	Componentes.....	4.2-31
4.2.7.	Tablas de mapeo	4.2-32
4.2.8.	Herramientas	4.2-33
4.2.9.	Bibliotecas	4.2-34
4.3.	Desarrollo.....	4.3-35
4.3.1.	Selección, uso de modelos y preprocesamiento.....	4.3-35
4.3.2.	Comprobación de datos	4.3-35
4.3.3.	Algoritmo para selección de modelo y escalador	4.3-35
4.3.4.	Estructura de la API.....	4.3-35
4.3.5.	Creación de rutas	4.3-36
4.3.6.	Despliegue de modelo de aprendizaje máquina.....	4.3-37
4.3.7.	Comunicación Cliente-Servidor.....	4.3-37
4.4.	Despliegue	4.4-38
5.	Pruebas y Resultados	4.4-40
5.1.	API.....	5.1-40
5.2.	Base de datos.....	5.2-41
5.3.	Desarrollo Movil.....	5.3-42
6.	Conclusiones y trabajo futuro	5.3-44
6.1.	Conclusiones del proyecto.....	6.1-44
6.2.	Recomendaciones.....	6.2-45
6.2.1.	Mejorar la infraestructura del sistema	6.2-45
6.2.2.	Integrar la API con otros sistemas.....	6.2-46

6.2.3.	Mejorar los modelos de Inteligencia Artificial	6.2-46
7.	Bibliografía	6.2-47
8.	Anexos	Error! Bookmark not defined.
8.1.	Título del anexo A	Error! Bookmark not defined.
8.2.	Título del anexo B	Error! Bookmark not defined.

Índice de figuras

Figura 1.- Proceso de atención API (Fuente: Elaboración propia)	2.2-10
Figura 2.-Arquitectura Cliente-Servidor (Fuente: Elaboración propia).....	2.2-15
Figura 3.-Relacion de arquetipos (Fuente: Leyva López, N.).....	4.2-29
Figura 4.- Diagrama de contexto (Fuente: Leyva López, N)	4.2-30
Figura 5.-Estructura de archivos (Fuente: Elaboración propia)	4.3-36
Figura 6.-Flujo completo de consumo (Fuente: Elaboración propia).....	4.4-39
Figura 7.- Prueba de la API en linea, finalizada (Fuente: Elaboración propia)	5.1-40
Figura 8.- Grafico de entradas en la base de datos. (Fuente: Elaboración propia).....	5.2-41
Figura 9.- Tabla de predicciones en la base de datos. (Fuente: Elaboración propia)	5.2-41
Figura 10.-Panel de control de Heroku (Fuente: Elaboración propia)	5.2-42
Figura 11.- Codigo para convertir modelos de IA a formato ONNX (Fuente: Elaboración propia). 5.3-	42
Figura 12.- Prueba de resultado del modelo de IA implementado en la app móvil. (Fuente: Elaboración propia)	5.3-43
Figura 13.-Datos guardados en archivo CSV (Fuente: Elaboración propia)	5.3-43

Índice de tablas

Tabla 1.- Áreas del aprendizaje maquina (Fuente: Elaboración propia)	2.1-7
Tabla 2.-Tipos de algoritmo según clasificación (Fuente: Elaboración propia).....	2.1-8
Tabla 3.- Tipos de escaladores de datos. (Fuente: Elaboración propia)	2.1-9
Tabla 4.- Tipos de API según su uso (Fuente: Elaboración propia)	2.2-14
Tabla 5.- Protocolos usados en APIs (Fuente: Elaboración propia)	2.2-14
Tabla 6.-Análisis de requisitos (Fuente: Elaboración propia).....	4.1-25
Tabla 7.-Requisitos funcionales (Fuente: Elaboración propia).....	4.1-25
Tabla 8.-Requisitos de calidad (Fuente: Elaboración propia).....	4.1-26
Tabla 9.-Actores del sistema (Fuente: Elaboración propia)	4.1-27
Tabla 10.-Casos de uso (Fuente: Elaboración propia).....	4.1-27
Tabla 11.-Arquetipos (Fuente: Elaboración propia).....	4.2-28
Tabla 12.-Mapeo de responsabilidad (Fuente: Elaboración propia).....	4.2-33

Capítulo 1

1. Introducción

La calidad del agua es un pilar fundamental para la salud de los ecosistemas acuáticos y el bienestar humano. La presencia de nutrientes y otros parámetros, como el oxígeno disuelto, la temperatura y el pH, son indicadores clave de la salud ambiental. Comprender estos indicadores nos permite identificar a tiempo desequilibrios ecológicos, como la eutrofización, o la presencia de contaminantes. Sin embargo, los métodos de laboratorio tradicionales, aunque precisos, son a menudo lentos, lo que impide una respuesta ágil ante situaciones de riesgo.

El análisis de muestras de agua para determinar los nutrientes presentes es una tarea fundamental para comprender el estado de los ecosistemas acuáticos y su dinámica. Estos estudios son esenciales para garantizar la calidad del agua, identificar desequilibrios ecológicos y tomar decisiones informadas sobre el manejo de los recursos hídricos. Tradicionalmente, este proceso involucra la recolección de muestras en campo, su adecuado transporte y conservación, y su posterior análisis en laboratorios especializados. Aunque esta metodología es confiable, resulta compleja y requiere un manejo delicado que, en muchos casos, prolonga los tiempos de espera para obtener resultados.

El principal desafío del método tradicional radica en su lentitud, ya que los análisis de laboratorio pueden tardar días o incluso semanas en completarse, dependiendo de la disponibilidad de equipos, reactivos y personal capacitado. Este retraso no solo afecta la rapidez en la toma de decisiones, sino que también puede limitar la capacidad de respuesta ante situaciones críticas, como la detección de zonas contaminadas o la evaluación de áreas que requieren atención prioritaria.

Ante esta problemática, surge la necesidad de implementar soluciones tecnológicas que optimicen el proceso de análisis de muestras de agua. La integración de modelos de inteligencia artificial (IA) en sistemas web representa un complemento innovador que permite realizar estimaciones rápidas relativamente confiables a partir de datos obtenidos en campo mediante pruebas sencillas. Este enfoque no solo reduce significativamente los

tiempos de análisis, sino que también democratiza el acceso a herramientas avanzadas, permitiendo a los usuarios en campo tomar decisiones de manera más ágil y eficiente

1.1. Descripción del problema

La creciente popularidad de la inteligencia artificial (IA) ha impulsado el desarrollo de modelos de machine learning (ML) y aprendizaje profundo cada vez más potentes y variados. Esto permite su implementación como un complemento de alta velocidad para analizar los parámetros de calidad del agua, como el oxígeno disuelto (OD), la temperatura del agua, la conductividad de campo, el pH y los sólidos suspendidos totales (SST), con una fiabilidad aproximada del 84%.

Sin embargo, a diferencia de los análisis de laboratorio especializados que requieren materiales y reactivos específicos, además de tiempos prolongados para la obtención de resultados, nuestro principal desafío técnico radica en la necesidad de un sistema backend robusto. Este sistema no solo debe implementar los modelos de IA, sino también asegurar su disponibilidad y realizar el preprocesamiento de datos requerido para que funcionen correctamente. Esta tarea se vuelve particularmente compleja en el contexto del desarrollo de una aplicación móvil.

El problema es que se requieren técnicas de implementación y preprocesamiento de datos muy específicas para estos modelos, lo que añade una capa de complejidad que limita la reutilización de la tecnología. Esta limitación frena el avance en el área, pues la adopción y adaptación de nuevas herramientas es fundamental para la innovación, tanto para fines académicos como personales. El objetivo es ofrecer una aproximación fiable a través de la IA, utilizando parámetros de entrada que no requieren largos períodos de tiempo en su obtención, y de esta manera, proporcionar una herramienta ágil y accesible para la evaluación de la calidad del agua.

1.2. Hipótesis

Es factible desarrollar e implementar un sistema web basado en modelos de machine learning que, al ser expuesto como un servicio, permita el análisis de los siguientes parámetros presentes en muestras de agua:

- Oxígeno disuelto (OD): medido en miligramos por litro (mg/L).
- Temperatura del agua: expresada en grados Celsius (°C).
- Conductividad de campo: medida en microsiemens por centímetro ($\mu\text{S}/\text{cm}$).
- pH: en una escala de 0 a 14.
- Sólidos suspendidos totales (SST): medidos en miligramos por litro (mg/L).

Este sistema permitirá reducir significativamente los tiempos de respuesta y mejorará la accesibilidad a la información en comparación con los métodos de laboratorio tradicionales. Además, este sistema evaluará de manera fiable y accesible, en tiempo real y desde cualquier ubicación con acceso a internet, funcionando como una herramienta complementaria y eficaz para la toma de decisiones en campo.

1.3. Objetivos

Desarrollar un sistema backend para una aplicación web que, mediante el uso de modelos de aprendizaje profundo, permita estimar las concentraciones de nutrientes clave en muestras de agua de manera rápida y fiable, optimizando el proceso de monitoreo de la calidad del recurso hídrico.

1.3.1. Objetivo general

Desarrollar una aplicación web que integre los modelos de estimación de nutrientes de la calidad del agua permite la estimación de contaminantes futuros mediante el uso de modelos de aprendizaje profundo.

1.3.2. Objetivos específicos

1. Definir y documentar los requisitos funcionales y no funcionales del sistema, estableciendo las especificaciones técnicas para la arquitectura del backend y la integración de los modelos.
2. Diseñar y desarrollar una API RESTful que sirva como punto de entrada para las solicitudes de análisis y gestione la comunicación entre el cliente y el servidor.
3. Integrar y estructurar los modelos de machine learning previamente entrenados dentro de la arquitectura del backend, asegurando su correcta ejecución y gestión de datos.
4. Desplegar el sistema backend en un entorno de producción en la nube para garantizar su alta disponibilidad, escalabilidad y accesibilidad desde la web.
5. Validar el correcto funcionamiento del sistema mediante la ejecución de pruebas funcionales y de rendimiento que aseguren la fiabilidad de las predicciones y la estabilidad del servidor.

1.4. Justificación

El análisis de nutrientes en el agua es una tarea crucial para organizaciones como la Comisión Nacional del Agua (CONAGUA), ya que permite evaluar la calidad del recurso hídrico y fundamentar decisiones relacionadas con su manejo y preservación. Sin embargo, el proceso tradicional, que implica la recolección de muestras en campo y su envío a laboratorios especializados, presenta una latencia significativa. El transporte, la preparación de las muestras y los análisis químicos correspondientes pueden prolongarse durante semanas, impactando negativamente la capacidad de respuesta y dificultando tanto el control eficiente de las zonas de muestreo como la gestión oportuna de los recursos hídricos.

Para optimizar este proceso, se propone el desarrollo de un innovador sistema auxiliar basado en tecnologías web y modelos de inteligencia artificial (IA). Este sistema está diseñado para proporcionar resultados fiables en cuestión de minutos, utilizando datos obtenidos de pruebas rápidas realizadas directamente en campo. Las principales ventajas que ofrece esta solución son:

Portabilidad y Accesibilidad: El sistema puede ser consultado desde cualquier dispositivo con conexión a internet a través de una página web o una aplicación móvil, lo que facilita su uso por parte del personal técnico en campo.

Optimización de la Eficiencia: A partir de un conjunto de datos simples y de rápida obtención, el sistema genera una estimación confiable de los nutrientes presentes en el agua. Esto permite a los usuarios evitar procesos de laboratorio largos y costosos para obtener un diagnóstico preliminar.

Valor Estratégico: Al ofrecer información inmediata, la herramienta facilita la identificación de zonas críticas o de interés prioritario. Esto mejora sustancialmente la toma de decisiones y permite reaccionar con celeridad ante contingencias relacionadas con la calidad del agua.

Este enfoque no busca reemplazar los análisis de laboratorio, sino fungir como un complemento ágil y potente que permita obtener datos preliminares y tomar decisiones informadas de manera inmediata. La implementación de este sistema no solo reduce drásticamente los tiempos de análisis, sino que también promueve un manejo más proactivo y sostenible de los recursos hídricos.

Capítulo 2

2. Marco teórico

El presente capítulo establece los fundamentos teóricos y conceptuales que sustentan el desarrollo de este proyecto de investigación. Dada la naturaleza de la implementación de modelos de Inteligencia Artificial (IA) tanto en una aplicación web como en una aplicación móvil, cada una de estas plataformas será abordada de forma individual en sus respectivas secciones. Asimismo, y considerando que el objetivo principal de este trabajo es la exposición de modelos de IA como servicios disponibles, se dedicará una sección específica al concepto de APIs (Interfaces de Programación de Aplicaciones) como servicios, así como a las arquitecturas de software relevantes

2.1. Aprendizaje Maquina

El Aprendizaje Automático (Machine Learning - ML), una disciplina fundamental dentro del campo de la Inteligencia Artificial (IA), se define como el subcampo que dota a los sistemas informáticos de la capacidad de aprender a partir de datos, sin ser programados explícitamente para cada tarea. A diferencia de los algoritmos tradicionales, que requieren instrucciones detalladas y predefinidas para cada acción, los algoritmos de ML desarrollan la habilidad de identificar patrones, tomar decisiones o hacer predicciones basándose en la experiencia adquirida a través de los datos.

2.1.1. Aplicaciones de ML

Las aplicaciones del ML abarcan un amplio espectro de dominios, demostrando su versatilidad y relevancia en diversas industrias. Un ejemplo destacado se encuentra en el Procesamiento del Lenguaje Natural (PLN), donde el ML posibilita funcionalidades como el reconocimiento de voz en asistentes inteligentes, la traducción automática de idiomas y la extracción de información de textos.

En el ámbito de la visión artificial, el ML permite a los sistemas automatizados, como los robots utilizados en plantas industriales, reconocer y clasificar objetos

Área	Descripción	Ejemplo
Aprendizaje Supervisado	Se entrena el modelo con datos etiquetados, de modo que aprenda a predecir la salida correcta para nuevas entradas.	Clasificar emails como spam o no a partir de sus características.
Aprendizaje No Supervisado	Se busca descubrir patrones o estructuras ocultas en datos que no cuentan con etiquetas previas.	Agrupamiento (clustering) de clientes en segmentos mediante PCA.
Aprendizaje por Refuerzo	Un agente interactúa con el entorno y aprende a tomar decisiones mediante recompensas o penalizaciones, optimizando sus acciones a lo largo del tiempo.	Entrenamiento de robots, juegos (como ajedrez o Go) o el control de sistemas de tráfico.
Aprendizaje Semi-Supervisado	Combina un pequeño conjunto de datos etiquetados con una gran cantidad de datos sin etiquetar para mejorar la precisión del modelo.	Clasificación de imágenes cuando sólo se dispone de pocas etiquetas
Deep Learning	Utiliza redes neuronales profundas con múltiples capas para extraer características complejas y resolver tareas avanzadas.	Reconocimiento facial, procesamiento del lenguaje natural
Meta-Aprendizaje	"Aprender a aprender": el modelo se adapta rápidamente a nuevas tareas o entornos con pocos datos, optimizando el proceso de aprendizaje.	Técnicas de few-shot learning para ajustar un modelo a nuevas categorías con muy pocos ejemplos.
Aprendizaje en Línea	Permite actualizar continuamente el modelo conforme llegan nuevos datos, ideal para entornos dinámicos y en tiempo real.	Sistemas de recomendación en plataformas de streaming o análisis de datos provenientes de sensores en tiempo real.

Tabla 1.- Áreas del aprendizaje maquina (Fuente: Elaboración propia)

2.1.2. Datos

En el ámbito computacional, los datos se definen como representaciones simbólicas de atributos o variables, sean estas cuantitativas o cualitativas. Constituyen la información o los valores que son procesados por algoritmos y pueden presentarse en una vasta diversidad de formatos (Editorial Etecé, 2021). La relevancia de los datos en las ciencias computacionales radica en su capacidad para permitir el almacenamiento y el análisis organizado de información. La obtención de datos se lleva a cabo mediante diversos métodos, desde encuestas hasta el monitoreo de redes sociales.

En el área específica del ML, los datos son fundamentales, ya que se utilizan para entrenar los modelos con el fin de que estos produzcan resultados optimizados. Resulta conveniente clasificar los problemas de aprendizaje según el tipo de datos que emplean. Esta categorización es particularmente útil al enfrentar nuevos desafíos, dado que problemas similares suelen manifestarse con tipos de datos análogos y, por ende, pueden resolverse mediante la aplicación de técnicas comparables (Smola, 2018).

2.1.3. Tipos de algoritmos de ML

Tipo de Modelo	Descripción	Ejemplo
Modelos de Regresión	Se emplean para predecir valores continuos a partir de variables independientes, modelando la relación funcional entre ellas.	Regresión lineal, regresión polinómica.
Modelos de Clasificación	Asignan etiquetas o categorías a nuevos datos basándose en patrones aprendidos a partir de datos etiquetados.	Árboles de decisión, SVM, k-NN.
Modelos de Clustering	Agrupan datos sin etiquetas en clusters o grupos según similitud o distancia, descubriendo estructuras ocultas en los datos.	K-means, clustering jerárquico.
Modelos Bayesianos	Utilizan el teorema de Bayes y conceptos probabilísticos para inferir la pertenencia de datos a determinadas clases o estimar distribuciones.	Clasificador Naive Bayes, Redes Bayesianas.
Redes Neuronales	Imitan el funcionamiento del cerebro mediante nodos interconectados, aprendiendo representaciones complejas a partir de los datos.	Redes feedforward, redes neuronales convolucionales (CNN).
Modelos de Ensamble	Combinan las predicciones de varios modelos para mejorar la precisión y robustez del sistema en general.	Random Forest, métodos de Boosting (AdaBoost, Gradient Boosting).
Modelos de Aprendizaje por Refuerzo	Un agente interactúa con un entorno y aprende a tomar decisiones mediante recompensas o penalizaciones, optimizando su política a lo largo del tiempo.	Q-learning, Deep Q-Network (DQN).

Tabla 2.-Tipos de algoritmo según clasificación (Fuente: Elaboración propia)

En la Tabla 2 se muestran los tipos de algoritmos de ML que se pueden generar, en este proyecto se empleó el Random Forest, que es considerado un modelo de ensamble con ser un cumulo de árboles de decisión (Oladipupo Ayodele, 2010).

2.1.4. Tipos de escaladores de datos

Tipo de escalador	Descripción	Ejemplo de uso
Standard Scaler	Estandariza las características removiendo la media y escalando a varianza unitaria (media = 0 y desviación estándar = 1).	Algoritmos sensibles a la escala, como SVM o regresión lineal.
Min-Max Scaler	Normaliza los datos para que se ajusten a un rango específico, comúnmente [0, 1].	Redes neuronales que requieren entradas en un rango fijo.
Robust Scaler	Escala los datos utilizando la mediana y el rango intercuartílico, lo que lo hace menos sensible a valores atípicos.	Conjuntos de datos con outliers significativos.
MaxAbs Scaler	Escala cada característica dividiéndola por su valor absoluto máximo, manteniendo la forma de la distribución original.	Datos dispersos, por ejemplo, en representaciones de texto.
Normalizer	Normaliza cada muestra individualmente a una norma (por ejemplo, L2), ajustando la magnitud de cada vector de características.	Procesamiento de texto o sistemas de recomendación, donde se comparan distancias.
Power Transformer	Aplica una transformación (como Box-Cox o Yeo-Johnson) para estabilizar la varianza y aproximar la distribución a una normal.	Corrección de asimetrías en distribuciones de datos.

Tabla 3.- Tipos de escaladores de datos. (Fuente: Elaboración propia)

En la Tabla 3 se presentan los escaladores existentes, su descripción y uso, para efectos de este proyecto se implementaron min-max scaler y robust scaler. Los datos con los que se alimentan los modelos obtenidos deben estar escalados previamente por los mismos con el fin de obtener una salida con la mayor precisión posible.

2.2. API como servicios

Las Interfaces de Programación de Aplicaciones (API) actúan como intermediarias que facilitan la comunicación e integración entre diferentes sistemas y aplicaciones de software. Su función principal es exponer funcionalidades específicas —como el acceso a datos, el procesamiento de pagos o la provisión de servicios de análisis— a través de Internet. Esto permite que otros desarrolladores o sistemas puedan consumir estas funcionalidades sin necesidad de conocer la infraestructura subyacente. La implementación de APIs contribuye

a estandarizar las interacciones, escalar recursos y mantener altos niveles de seguridad, al mismo tiempo que acelera el desarrollo de nuevas aplicaciones mediante la reutilización de servicios ya existentes.

2.2.1. ¿Qué es una API?

Una API (Application Programming Interface) es un conjunto de reglas y definiciones que permite que diferentes aplicaciones se comuniquen entre sí. Actúa como un intermediario que facilita la interacción entre sistemas de software, permitiendo que una aplicación acceda a funciones o datos de otra sin necesidad de conocer su implementación interna.

2.2.2. Funcionamiento de una API



Figura 1.- Proceso de atención API (Fuente: Elaboración propia)

Como se mencionó previamente, las Interfaces de Programación de Aplicaciones (API) son un conjunto de reglas y protocolos programados que dictan cómo se debe gestionar, procesar y resolver una solicitud de comunicación entre dos sistemas de software.

En esencia, una API actúa como un intermediario que facilita la interacción entre diferentes aplicaciones. Permite que una aplicación (el cliente) solicite y acceda a funcionalidades o datos de otra aplicación (el servidor) sin necesidad de comprender su compleja implementación interna. Este proceso típicamente involucra los siguientes pasos:

- 1.-Solicitud (Request): La aplicación cliente envía una solicitud a la API, que puede incluir parámetros específicos o datos necesarios para la operación. Esta solicitud se realiza siguiendo las reglas y el formato definidos por la API (por ejemplo, a través de una petición HTTP).
- 2.-Procesamiento: La API recibe la solicitud y la dirige al sistema backend o servicio correspondiente. Este sistema procesa la solicitud, realiza la operación requerida

(como consultar una base de datos, ejecutar un cálculo o invocar otra función) y prepara una respuesta.

3.-Respuesta (Response): El sistema backend envía la respuesta de vuelta a la API. La API, a su vez, formatea esta respuesta de acuerdo con sus propias reglas (por ejemplo, en formato JSON o XML) y la envía de regreso a la aplicación cliente.

4.-Recepción: La aplicación cliente recibe la respuesta de la API y utiliza los datos o la confirmación de la operación para continuar con sus procesos internos.

Este mecanismo estandarizado permite que los desarrolladores construyan aplicaciones modulares y distribuidas, donde diferentes componentes pueden comunicarse y colaborar eficazmente, sin acoplamiento directo entre sus implementaciones internas.

2.2.3. Beneficios de las API

Las API (Interfaz de Programación de Aplicaciones) son un componente esencial en el desarrollo de software contemporáneo, actuando como puentes que posibilitan la comunicación fluida entre distintas aplicaciones y sistemas. Sus beneficios son multifacéticos, impactando positivamente tanto el proceso de desarrollo como la funcionalidad y el valor final de un proyecto.

A nivel de desarrollo, el uso de una API acelera significativamente los flujos de trabajo al permitir la reutilización de funcionalidades ya existentes, eliminando la necesidad de programar soluciones desde cero. Esto no solo ahorra tiempo, sino que también garantiza la fiabilidad y la eficiencia, ya que se integran soluciones probadas y optimizadas. Además, las API fomentan un desarrollo modular, donde cada componente del sistema puede ser gestionado de forma independiente. Esto facilita el trabajo en paralelo de los equipos, mejora la colaboración y simplifica el mantenimiento y las actualizaciones, ya que cualquier modificación en un módulo específico no compromete el resto del sistema.

Desde la perspectiva del proyecto, una API robusta y bien diseñada potencia la escalabilidad y la flexibilidad. Al desvincular la interfaz de usuario del backend, una API permite que la aplicación se adapte a diversas plataformas —ya sean web, móviles o de escritorio— sin tener que reescribir la lógica subyacente. Esta capacidad facilita la expansión del proyecto a nuevos mercados y usuarios. Por ejemplo, una aplicación con una API bien definida puede

lanzar una versión móvil o integrarse con sistemas de terceros de manera ágil. Finalmente, las API promueven la interoperabilidad y la innovación. Al exponer datos y funcionalidades de manera controlada, un proyecto puede integrarse en un ecosistema más amplio, permitiendo que otros desarrolladores construyan nuevas herramientas y servicios. Esto no solo enriquece la propuesta de valor del proyecto original, sino que también fomenta la creación de soluciones que el equipo principal tal vez no había considerado.

2.2.4. Aplicaciones comunes de las APIs

Debido a su gran flexibilidad y capacidad de integración, las API están en todos lados, en la actualidad es difícil imaginar una aplicación o servicio que no consuma dichas API, entre sus usos más comunes están:

Autenticación y Seguridad:

- Las API de autenticación, como OAuth o OpenID, permiten a los usuarios acceder a múltiples aplicaciones con una sola cuenta, por ejemplo, iniciar sesión en un sitio web con una cuenta de Google o Facebook.
- Las API de seguridad, como reCAPTCHA, ayudan a detectar bots y evitar accesos no autorizados.

Integración de Servicios en la Nube:

- Las API permiten conectar plataformas como Google Drive, Dropbox y OneDrive, facilitando la sincronización y almacenamiento de archivos en la nube.
- Aplicaciones como Zapier o IFTTT usan APIs para automatizar tareas entre servicios distintos, como enviar correos automáticos o guardar archivos en la nube.

Chatbots y Asistentes Virtuales:

- Los asistentes virtuales como Siri, Alexa y Google Assistant usan APIs para procesar lenguaje natural y obtener información de distintos servicios.
- Los chatbots en WhatsApp, Messenger o Telegram utilizan APIs para interactuar con usuarios y responder preguntas automáticamente.

Streaming de Contenido:

- Netflix, Spotify y YouTube usan APIs para gestionar catálogos de contenido, recomendaciones y transmisión de datos de video o audio.
- Las API de Twitch permiten a los desarrolladores integrar transmisiones en vivo en sitios web o aplicaciones de terceros.

E-commerce y Gestión de Inventarios:

- Las APIs de plataformas como Shopify, Amazon y eBay permiten a las tiendas administrar productos, recibir pagos y actualizar inventarios en tiempo real.
- Los sistemas de pago como PayPal, Stripe y MercadoPago usan APIs para facilitar transacciones seguras y rápidas.

Banca y Finanzas:

- Los bancos utilizan APIs para ofrecer servicios de banca en línea, permitiendo transferencias de dinero, pagos de facturas y consulta de estados de cuenta.
- Las API de trading financiero permiten acceder a información de mercados en tiempo real y realizar transacciones automáticas.

Automatización de Tareas Empresariales:

- Aplicaciones como Slack, Microsoft Teams y Trello usan APIs para integrar flujos de trabajo y automatizar la gestión de tareas dentro de una empresa.
- Las API de facturación electrónica permiten a empresas generar y validar facturas automáticamente.

IoT (Internet de las Cosas):

- Las APIs permiten conectar dispositivos inteligentes como termostatos Nest, cámaras Ring y luces Philips Hue con aplicaciones móviles y asistentes de voz.
- Los autos inteligentes utilizan APIs para recibir actualizaciones de software, monitorear estado del vehículo y enviar datos a aplicaciones móviles.

2.2.5. Tipos de API

Tipo de API	Descripción
APIs Abiertas (Públicas)	Cualquiera puede acceder y utilizarlas, generalmente requieren una clave API.
APIs Privadas	Solo son accesibles dentro de una organización o sistema cerrado.
APIs de Socios	Diseñadas para integraciones específicas con terceros mediante acuerdos comerciales.
APIs Compuestas	Combinan múltiples APIs para realizar tareas más complejas.

Tabla 4.- Tipos de API según su uso (Fuente: Elaboración propia)

En la Tabla anterior se describen los tipos de API existentes, debido a que el proyecto desarrollado en este documento es privado se desarrolló una API Privada, con acceso limitado desde ciertos clientes.

2.2.6. Tipos de protocolos de API

Protocolo	Formato de datos	Velocidad	Casos de uso
REST (HTTP/HTTPS)	JSON, XML	Media	APIs web, servicios públicos
SOAP	XML	Lenta	Servicios bancarios, transacciones seguras
GraphQL	JSON	Alta	Aplicaciones con datos complejos (redes sociales)
gRPC	Protobuf	Muy alta	Microservicios, Big Data
WebSockets	JSON, Binario	Alta	Chats, videojuegos, trading en vivo
MQTT	Binario	Muy alta	IoT, dispositivos conectados
AMQP	JSON, Binario	Alta	Sistemas de mensajería empresarial

Tabla 5.- Protocolos usados en APIs (Fuente: Elaboración propia)

Con la necesidad de cubrir las necesidades de los sistemas informáticos nacen diversos protocolos de comunicación, el más común para sistemas webs en la actualidad es el REST(HTTP/HTTPS) pues sus características cubren la mayoría de las necesidades de los sistemas existentes, Mismo protocolo que se emplea en el presente documento.

2.2.7. Arquitecturas

La arquitectura de software se refiere a la estructura fundamental de un sistema, que incluye sus componentes, sus relaciones entre sí y con el entorno, y los principios que guían su diseño y evolución. Una arquitectura bien definida es crucial para el desarrollo de sistemas robustos, escalables y mantenibles. Un enfoque prominente en la actualidad es el diseño modular, que

busca dividir las aplicaciones en componentes más pequeños e interdependientes. La implementación de una aplicación como un conjunto de servicios separados facilita las pruebas, el mantenimiento y la escalabilidad. Esta metodología ha ganado relevancia en la era de las aplicaciones en la nube, ya que permite a los desarrolladores trabajar en componentes independientes, agilizando el ciclo de desarrollo y despliegue.

2.2.8. Arquitectura cliente servidor

Es un modelo distribuido donde las tareas se dividen entre proveedores de recursos o servicios (servidores) y solicitantes de servicios (clientes). En este modelo, el cliente envía una solicitud al servidor, y este último procesa la solicitud y devuelve una respuesta. Este enfoque centraliza la gestión de datos y recursos en el servidor, lo que facilita la seguridad y el control. Es una arquitectura fundamental para muchas aplicaciones web y móviles, permitiendo que múltiples clientes accedan a funcionalidades y datos compartidos de manera eficiente.

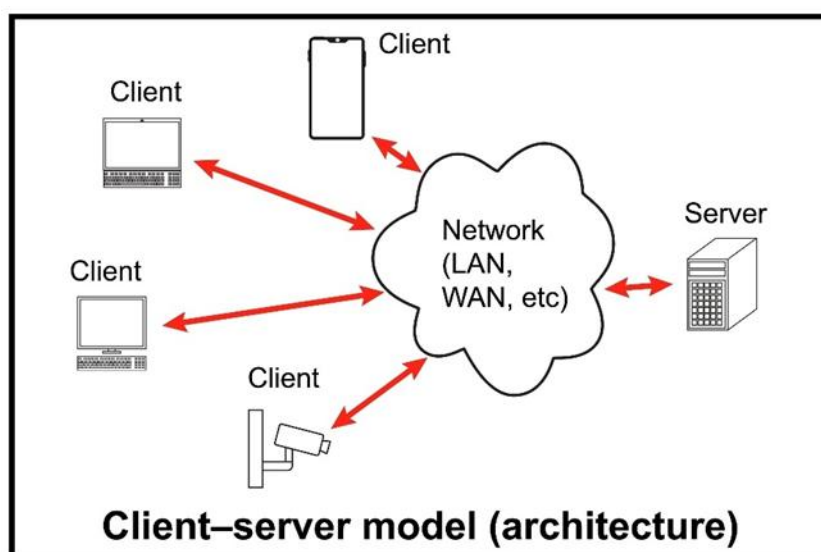


Figura 2.-Arquitectura Cliente-Servidor (Fuente: Elaboración propia)

2.2.9. Arquitectura MVC

Es un patrón de diseño arquitectónico que separa una aplicación en tres componentes interconectados:

- Modelo (Model): Representa la lógica de negocio y los datos de la aplicación. Es independiente de la interfaz de usuario.

- Vista (View): Es la interfaz de usuario. Muestra los datos del Modelo al usuario y recibe la interacción del usuario.

- Controlador (Controller): Actúa como intermediario entre el Modelo y la Vista. Procesa los eventos de entrada del usuario, actualiza el Modelo según sea necesario y selecciona la Vista apropiada para mostrar la respuesta.

El patrón MVC mejora la separación de responsabilidades, lo que facilita el desarrollo, las pruebas y el mantenimiento de aplicaciones complejas, especialmente aquellas con interfaces de usuario ricas.

2.2.10. Arquitectura MVVM

El patrón de diseño MVVM (Model-View-ViewModel) es una arquitectura utilizada en el desarrollo de software, especialmente para interfaces de usuario. Su objetivo principal es separar la lógica de la aplicación de su presentación.

Aquí está el desglose de sus componentes:

- Modelo (Model): Representa los datos y la lógica de negocio. Es completamente independiente de la interfaz de usuario. Por ejemplo, en una aplicación de lista de tareas, el Modelo sería la clase que define una tarea con sus propiedades (título, descripción, estado, etc.).
- Vista (View): Es la interfaz de usuario. Muestra lo que el usuario ve en la pantalla. Puede ser una página web, una pantalla de aplicación móvil, etc. La Vista solo sabe cómo mostrar los datos y enviar las acciones del usuario, pero no contiene lógica de negocio.
- Vista-Modelo (ViewModel): Actúa como un intermediario entre el Modelo y la Vista. El ViewModel expone los datos del Modelo de una manera que la Vista puede consumir fácilmente. No tiene una referencia directa a la Vista, pero la Vista tiene una referencia al ViewModel. Esto permite que la Vista se actualice automáticamente cuando los datos del ViewModel cambian.

En resumen, el patrón MVVM ayuda a crear aplicaciones más fáciles de mantener, probar y entender, al mantener la lógica de la aplicación y la interfaz de usuario desacopladas.

2.2.11. API y arquitectura de nube

En el contexto de las arquitecturas de nube, la creación de APIs (Interfaces de Programación de Aplicaciones) que se adapten a las necesidades actuales es fundamental. El desarrollo de aplicaciones nativas de la nube se basa intrínsecamente en la interconexión de microservicios a través de APIs que no solo facilitan la comunicación interna, sino que también exponen funcionalidades y datos a usuarios o sistemas externos.

Los servicios, incluso dentro de un ecosistema de microservicios, suelen emplear marcos de desarrollo de mensajería comunes, como los utilizados por las API RESTful. Esto promueve una comunicación abierta y estandarizada con el sistema operativo subyacente, eliminando la necesidad de integrar capas adicionales complejas o gestionar transacciones de datos de forma engorrosa. Una ventaja crucial de este enfoque modular es la independencia de los servicios: cualquier servicio o característica específica puede ser suspendida, reemplazada o mejorada sin afectar negativamente a otros servicios del sistema. Esta dinámica optimiza el uso de los recursos en línea y sienta las bases para mejorar significativamente el rendimiento, las pruebas y la escalabilidad de las APIs en entornos de nube.

Capítulo 3

3. Estado del arte

Esta sección presenta una revisión de la literatura relevante y un análisis del estado del arte sobre el tema central de este proyecto. Se llevó a cabo una exhaustiva búsqueda, lectura crítica y análisis de numerosas fuentes bibliográficas directamente relacionadas con la problemática abordada en este trabajo. Adicionalmente, se examinaron artículos científicos y publicaciones técnicas estrechamente vinculadas al enfoque de la investigación.

Los resultados de esta investigación bibliográfica permitieron definir con mayor precisión el alcance de este trabajo y establecer un flujo de trabajo realista, que incluye el conjunto de actividades necesarias para la consecución de los objetivos propuestos.

La estructura de esta sección se organiza en torno a los siguientes ejes temáticos clave: la creación de APIs, la definición de formatos de archivo para el intercambio de datos, el preprocesamiento de datos y la provisión de modelos de Machine Learning (ML) como servicios a través de un servidor.

3.1. Creación de API

Dado que este proyecto busca ofrecer una aplicación como servicio, es imperativo que cuente con una infraestructura de API robusta que facilite su publicación y disponibilidad para los usuarios. Esta API debe poseer una arquitectura específica que permita la conexión y la formación de repositorios tanto de algoritmos de Machine Learning (ML) como de Deep Learning (DL). Para sustentar esta propuesta, se ha realizado una investigación de trabajos previos que abordan desafíos y soluciones arquitectónicas similares.

Miura et al. (2017) presentan una infraestructura novedosa para el análisis de contenido autorizado (ASCA), la cual responde a una necesidad crítica en la recolección de información. La distinción clave de su enfoque reside en el diseño de la arquitectura utilizando contenedores Docker para la migración de servicios. Esta infraestructura tiene como propósito principal proveer APIs comunes para comunidades inteligentes,

desplegándolas en contenedores Docker. La elección de Docker se justifica por su capacidad para encapsular las librerías y el software necesario para la ejecución de procesos y servicios, ofreciendo un beneficio significativo en términos de rendimiento y portabilidad.

La arquitectura propuesta por Miura et al. (2017) se compone de capas de conexión entre el cliente y la API, además de una capa dedicada a la gestión de contenedores. Este diseño permite la ejecución de hasta ocho servicios distintos sin experimentar una disminución apreciable en el rendimiento. El uso de contenedores también mejora sustancialmente la portabilidad de los servicios, incluyendo la migración entre redes. En consecuencia, la plataforma demuestra la capacidad de transferir eficientemente datos compartidos a los procesos de servicios. Este estudio es relevante para nuestra propuesta por el uso de contenedores y el enfoque en rendimiento para una infraestructura de API.

Haselböck et al. (2018) proponen una metodología para la identificación y organización de opciones potenciales de diseño relacionadas con los conceptos tecnológicos que se pretenden utilizar en un sistema. La premisa central de este trabajo es la aplicación de microservicios para fragmentar el software en un conjunto de servicios desacoplados. Estos microservicios están diseñados para soportar el desarrollo, despliegue y operación de forma independiente, lo que confiere una notable flexibilidad al proceso de desarrollo. Todo lo anterior permite identificar los espacios de diseño posibles para el manejo de API en sistemas basados en microservicios y captura de decisión de modelos.

También presenta el proceso de los casos de uso básicos para este tipo de desarrollos. Éste identifica y analiza el espacio de diseño, se realiza la toma de decisiones y se validan el acercamiento con la documentación pertinente. También se demuestra que el espacio de diseño puede ser manejado paralelamente, y por último propone la guía de creación de API tomando en cuenta el anterior análisis.

Esto fue mostrado en pruebas de campo, dónde se identificaron las áreas de diseño para organizar las tareas en microservicios, estos no solo son un conjunto de decisiones, sino la retroalimentación en la utilidad del acercamiento para el análisis de la documentación.

3.2. Identificación del modelo

Considerando el enfoque de una aplicación web, se identifican principalmente dos ecosistemas robustos: Scikit-learn para el aprendizaje automático clásico y PyTorch para el aprendizaje profundo. Scikit-learn, reconocido por su API intuitiva y su eficiencia en conjuntos de datos de tamaño pequeño a mediano, se integra comúnmente en APIs web RESTful utilizando marcos como Flask o FastAPI, a menudo empaquetados en contenedores Docker para asegurar la consistencia y reproducibilidad (Muller & Guido, 2018). Un aspecto crucial en el uso de modelos Scikit-learn, como los Random Forest, es la aplicación de escaladores (ej., StandardScaler, MinMaxScaler) a través de Pipelines, lo que garantiza que el preprocesamiento de datos se aplique de manera uniforme tanto en el entrenamiento como en la inferencia, mitigando la desviación entrenamiento-servicio (Pedregosa et al., 2011). La persistencia de estos modelos se realiza preferentemente con formatos seguros como skops.io o ONNX, en lugar de pickle, para evitar vulnerabilidades de seguridad (Scikit-learn developers, 2023).

Por otro lado, PyTorch, favorecido por su flexibilidad y gráficos de cómputo dinámicos, se despliega a menudo utilizando TorchServe, una herramienta especializada que facilita la inferencia escalable, el versionado de modelos y el monitoreo (Paszke et al., 2019; TorchServe Developers, 2022). El preprocesamiento de datos para modelos PyTorch, especialmente para imágenes, se maneja con transformaciones encadenadas (ej., torchvision.transforms.Compose) que se integran directamente en la lógica de la API de inferencia (Paszke et al., 2019). Además, ONNX juega un papel vital en el despliegue multi-marco de modelos PyTorch, permitiendo que se sirvan de manera eficiente en entornos sin Python y mejorando la seguridad (Open Neural Network Exchange, 2024).

La tendencia futura en el despliegue de estos modelos se centra en la adopción de prácticas de MLOps, que buscan automatizar y monitorear continuamente todo el ciclo de vida del ML, desde el desarrollo hasta la operación (Kreuzberger et al., 2023). Esto incluye el uso de arquitecturas de servicio distribuido, como las facilitadas por Ray Serve, para manejar la creciente escala y complejidad de los modelos y datos (Moritz et al., 2018). Las plataformas MLaaS gestionadas en la nube (ej., Google Cloud Vertex AI, Kubeflow, MLflow) están simplificando este proceso al proporcionar entornos integrados y escalables (Gartner, 2024;

Microsoft Azure, 2024; Google Cloud, 2024). Finalmente, la creciente importancia de la IA Explicable (XAI) impulsará la necesidad de transparencia y responsabilidad en los modelos desplegados, integrando capacidades de interpretabilidad en las tuberías de MLOps (Adadi & Berrada, 2018).

En el contexto del desarrollo de esta tesis, la selección de herramientas para el despliegue del modelo se realizó considerando las limitaciones y características del entorno de producción. Si bien se reconocen las ventajas de PyTorch para el aprendizaje profundo y la promesa de MLOps para automatizar el ciclo de vida completo del ML (Paszke et al., 2019; Kreuzberger et al., 2023), las condiciones específicas del hosting y la configuración de Python proporcionada por el proveedor de servicios y el proveedor de modelos hicieron que la elección de Scikit-learn y la persistencia mediante Joblib fueran la solución más viable. Scikit-learn ofreció la robustez y simplicidad necesarias para los modelos de aprendizaje automático clásico empleados (Muller & Guido, 2018), mientras que Joblib, pese a sus consideraciones de seguridad, al contar con una fuente de modelo confiable, se adaptó a la configuración de entorno de Python existente y los requisitos de memoria del hosting para el manejo de modelos (Joblib developers, 2024; Scikit-learn developers, 2023). Esta decisión permitió asegurar la consistencia del preprocesamiento a través de Pipelines y el despliegue efectivo de la aplicación web, priorizando la compatibilidad operativa frente a la adopción de los marcos más recientes o complejos de aprendizaje profundo para el problema específico abordado en este trabajo.

3.3. Procesamiento de datos

Escriba texto aquí. El procesamiento de datos es una fase crítica en el ciclo de vida del aprendizaje automático, ya que prepara los datos crudos para que los modelos de IA puedan aprender de ellos de manera efectiva. Implica una serie de pasos que transforman los datos desde su formato original hasta una representación adecuada para el modelado, culminando en la estandarización o escalado para asegurar un rendimiento óptimo del algoritmo.

El camino desde los datos crudos hasta un formato listo para el modelo es integral y multifacético:

Recolección y Carga de Datos

El primer paso es recolectar los datos de diversas fuentes, que pueden incluir bases de datos, APIs, archivos de texto, imágenes, o sensores. Una vez recolectados, los datos se cargan en un entorno de trabajo donde se puedan manipular, como un DataFrame en Python.

- **Limpieza de Datos:** Los datos del mundo real a menudo están "sucios" y contienen errores, valores atípicos (outliers), o datos faltantes. La limpieza de datos aborda estas imperfecciones:
- **Manejo de valores faltantes:** Se pueden imputar (reemplazar) valores faltantes con la media, mediana, moda, o con métodos más avanzados, o simplemente eliminar las filas o columnas incompletas.
- **Detección y tratamiento de valores atípicos:** Los valores extremos pueden distorsionar el análisis y el entrenamiento del modelo. Se identifican y se manejan, ya sea eliminándolos, transformándolos o ajustando el modelo para que sea robusto a ellos.
- **Corrección de errores y duplicados:** Se eliminan entradas erróneas o duplicadas que podrían sesgar el modelo.
- **Transformación de Datos:** Esta fase ajusta los datos para que sean más adecuados para los algoritmos de ML:
- **Codificación de variables categóricas:** Los algoritmos de ML generalmente requieren entradas numéricas. Las variables categóricas (como "rojo", "verde", "azul") se convierten a un formato numérico usando técnicas como One-Hot Encoding o Label Encoding.
- **Normalización o estandarización:** Los datos numéricos con diferentes rangos o unidades pueden afectar el rendimiento de ciertos algoritmos. Aquí es donde la escalación juega un papel fundamental.

Muchos algoritmos de aprendizaje automático son sensibles a la escala de las características de entrada. Si las características tienen rangos de valores muy diferentes, aquellas con magnitudes mayores pueden dominar las funciones de costo o las métricas de distancia, lo que lleva a un entrenamiento ineficiente o a resultados sesgados. Por ejemplo, en algoritmos basados en distancia como K-Means o SVM, una característica con valores de 0 a 1000

podría tener un impacto desproporcionado en comparación con una característica con valores de 0 a 1.

Los escaladores son herramientas de preprocesamiento que transforman los datos numéricos para que sus valores se encuentren dentro de un rango o distribución específica, asegurando que todas las características contribuyan de manera equitativa al proceso de aprendizaje. En Scikit-learn, el módulo preprocessing ofrece varios escaladores, cada uno con un enfoque distinto (Pedregosa et al., 2011):

- **StandardScaler:** Este es uno de los escaladores más comunes. Implementa la estandarización, que transforma los datos para que tengan una media de 0 y una desviación estándar de 1. Esto es particularmente útil para algoritmos que asumen una distribución normal o centrada de los datos, como la regresión lineal, las máquinas de vectores de soporte (SVM) o las redes neuronales.
- **MinMaxScaler:** Este escalador realiza la normalización, ajustando las características a un rango específico, típicamente entre 0 y 1. Es efectivo cuando la distribución de los datos no es gaussiana o cuando los algoritmos requieren entradas en un rango acotado.
- **MaxAbsScaler:** Escala cada característica por su valor absoluto máximo, mapeando los datos al rango $[-1, 1]$. Es útil para datos dispersos.
- **RobustScaler:** Utiliza la mediana y el rango intercuartílico (IQR) para escalar, lo que lo hace más robusto a la presencia de valores atípicos en el conjunto de datos, a diferencia de StandardScaler o MinMaxScaler que son sensibles a ellos.

La integración de estos escaladores en Pipelines de Scikit-learn es una mejor práctica para garantizar que los pasos de preprocesamiento, incluida la escalación, se apliquen de manera consistente tanto durante el entrenamiento del modelo como en la fase de inferencia con datos nuevos. Esto es vital para mitigar la "desviación entrenamiento-servicio" (training-serving skew), un problema común que ocurre cuando hay discrepancias entre las distribuciones de datos en los entornos de entrenamiento y producción, lo que degrada el rendimiento del modelo (Sculley et al., 2015).

Capítulo 4

4. Desarrollo del proyecto

Este capítulo presenta la explicación de los mecanismos y herramientas utilizados para el análisis de la problemática de investigación. Aquí se documentan y monitorean los hechos relevantes ocurrido durante el período de desarrollo del proyecto. A continuación, se define el nombre con el que se denominó este trabajo.

4.1. Análisis

4.1.1. Análisis de requisitos

El análisis de requisitos es una tarea prioritaria en la ingeniería de software donde se analizan y definen los procesos que describen el desarrollo de la API. Esto permite especificar las características de rendimiento. Para este análisis se detalla lo que debe hacer el sistema desde un punto de vista más coloquial, teniendo en cuenta las necesidades del usuario. Sobre la base de este análisis y los requisitos específicos, se han categorizado los requisitos funcionales y de calidad, los cuales se basan en los requisitos del usuario. Debido a que es un servicio público debe cumplir con ciertas exigencias de funcionalidad en el sistema, para que el usuario pueda utilizarlo.

Requisitos	Descripción
RU-01	Se debe permitir al usuario utilizar las distintas redes disponibles en el repositorio de modelos de ML y DL sin limitaciones.
RU-02	El usuario es capaz de elegir el tipo de dato que aplicará en las redes disponibles.
RU-03	El servicio debe estar disponible para acceder desde cualquier navegador web las 24 horas del día, los 365 días del año.
RU-04	El sistema que el usuario cargue archivos desde la plataforma que se esté utilizando.

RU-05	El usuario debe recibir los datos debidamente preprocesados por la red ML o DL que se haya elegido.
RU-06	El sistema debe ser gratuito.
RU-07	El sistema debe contar con una interfaz intuitiva en la que el usuario pueda identificar la funcionalidad de este de manera rápida y sencilla.

Tabla 6.-Análisis de requisitos (Fuente: Elaboración propia)

4.1.2. Requisitos funcionales

Esta sección presenta los requisitos funcionales del sistema. Estos describen el comportamiento específico de ese sistema cuando se cumplen ciertas condiciones. La definición de estos está relacionada con las necesidades del usuario, ya que la funcionalidad del sistema debe satisfacer las necesidades del usuario establecidas previamente.

Requisitos	Descripción	Prioridad
RF-01	El sistema debe tener las redes de ML y DL debidamente referenciados en su repositorio para ser usadas cuando se realice su llamado. (RU-01, RU-03 y RU-06)	Media
RF-02	El sistema debe soportar como todo tipo de formato de archivo utilizado para aprendizaje máquina (RU-02)	Alta
RF-03	El sistema ofrece los distintos tipos de redes disponibles en el repositorio de modelos de ML y DL, y que estos puedan utilizarse sin ninguna limitante. (RU-02)	Alta
RF-04	El sistema debe entregar los datos procesados en un formato útil para el usuario. (RU-05)	Media
RF-05	El sistema permite la búsqueda y carga de archivos para utilizarlos con las redes de ML y DL. (RU-04)	Media

Tabla 7.-Requisitos funcionales (Fuente: Elaboración propia)

4.1.3. Requisitos de Calidad

Los requisitos de calidad son los encargados de determinar las características cualitativas que debe ofrecer el programa. Se dividen en atributos de calidad, es decir, disponibilidad y usabilidad. Se dividen de esta manera para su clasificación y es posible priorizarlos de manera que satisfagan las necesidades en términos de requisitos del usuario. En la Tabla 3 se expone la forma en que estos fueron especificados.

Requisitos	Atributo de calidad	Descripción	Prioridad
RC-01	Disponibilidad	El sistema debe estar disponible para acceder desde cualquier navegador web las 24 horas los 365 días del año de manera gratuita. (RU-03 y RU-06)	Media
RC-02	Usabilidad	El sistema debe mostrar una interfaz de fácil uso en que los usuarios puedan encontrar la utilidad de cada una de las opciones en un tiempo corto (no más de 10 segundos). (RU-07)	Alta
RC-03	Usabilidad	El servicio permite el uso de las redes propuestas ofrecidas dependiendo del dato ingresado al sistema (RU-01, RU-02)	Alta
RC-04	Usabilidad	El usuario recibe los datos introducidos debidamente procesados (RU-05)	Alta
RC-05	Usabilidad	Se puede realizar la búsqueda de los datos de la forma en que al usuario le parezca más conveniente. (RU-04)	Media

Tabla 8.-Requisitos de calidad (Fuente: Elaboración propia)

4.1.4. Actores

los actores se definen como "entidades externas al sistema que se relacionan con él y que requieren ciertas funciones". Esto incluye a usuarios y sistemas externos. Para nuestro sistema, los actores considerados son el usuario, una interfaz gráfica de usuario, repositorio de preprocesamiento, repositorio de modelos y repositorio de intérpretes.

Actor	Descripción
Usuario	Es quien utilizará el servicio de aprendizaje máquina introduciendo los datos que desea utilizar y recibiendo estos en un formato deseado.
Repositorio de preprocesamiento	Dirección física donde se encuentran los distintos algoritmos de preprocesamiento.
Repositorio de modelos	Dirección física donde se encuentran los distintos modelos de aprendizaje máquina disponibles en el servicio.
Repositorio de intérpretes	Dirección física donde se encuentran los algoritmos que definen las relaciones entre los tipos de preprocesamiento y los modelos.

Tabla 9.-Actores del sistema (Fuente: Elaboración propia)

4.1.5. Casos de uso

Describen las actividades que deberán realizarse para llevar a cabo algún proceso mostrando la funcionalidad principal del sistema." Se definieron con base en el análisis de requerimientos funcionales para establecer mecanismos de interacción con el sistema.

Identificador	Nombre	Descripción
CU001	Carga de datos	El sistema permite que el usuario cargue cualquier dato en el servicio de ML y DL. (RF-02, RF-05)
CU002	Elección de tipo de red a utilizar	Dependiendo del tipo de dato introducido en el sistema, se elige entre las distintas redes que pudiesen utilizarse para procesarlo. (RF-01, RF-03)
CU003	Preprocesamiento de datos	Proceso que se realiza de manera automática al elegir el tipo de red que se utilizará. (RF-04)
CU004	Entrega de datos procesados	Proceso que se realiza una vez los datos han sido debidamente preprocesados y se han procesado en una red de ML y DL. (RF-04)

Tabla 10.-Casos de uso (Fuente: Elaboración propia)

4.2. Diseño

4.2.1. Arquetipos

Los arquetipos son "entidades abstractas identificadas que representan las funcionalidades más importantes de un sistema". Se definen al responder a preguntas como: "¿Cuáles son las partes clave del sistema?" y "¿Qué representan estas entidades en él?". A continuación, se describen los arquetipos seleccionados para este sistema, junto con una breve descripción de cada uno.

Arquetipo	Descripción
Usuario	Es aquel que utilizará la aplicación web, y hará uso de las distintas funciones que el sistema ofrece.
Datos	Es un conjunto de archivos utilizados en los distintos modelos de aprendizaje máquina y aprendizaje profundo para obtener predicciones.
Modelo	Es un algoritmo que permite al usuario procesar los datos.
Interfaz	Se trata de la representación del sistema que permite que el usuario interactúe con el de manera sencilla e intuitiva.
Preprocesamiento	Es la preparación de los datos para que estén en el formato correcto que permita ser utilizados en los modelos de aprendizaje máquina y profundo
Escalador	Define las relaciones entre los modelos, los tipos de datos a los cuales estos pueden ser expuestos y el preprocesamiento necesario para que la información sea utilizable.
Servicio	Es el que permite que el sistema esté disponible para cualquier usuario que desee utilizarlo.

Tabla 11.-Arquetipos (Fuente: Elaboración propia)

4.2.2. Relación de arquetipos

En este apartado se muestra cómo es que los arquetipos previamente expuestos se relacionan en el sistema, así como la manera en que estos interactúan.

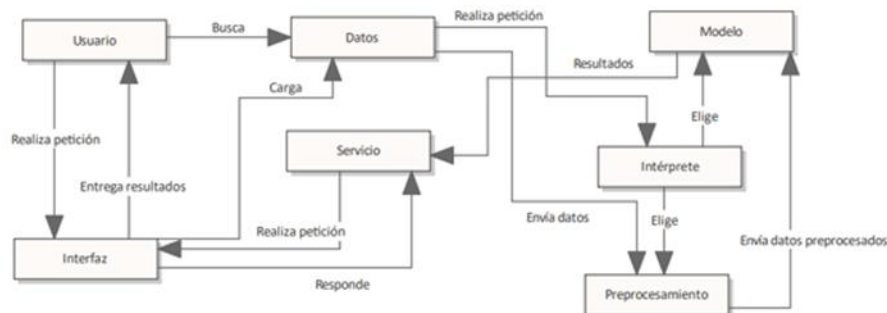


Figura 3.-Relacion de arquetipos (Fuente: Leyva López, N.)

En esta figura se muestra como el usuario se relaciona con el sistema, realizando peticiones a través de la interfaz. Esta interfaz carga los datos que el usuario proporciona y realiza la petición al intérprete, el cual permite elegir el modelo que mejor se ajusta al tipo de dato y lo relaciona con el algoritmo de preprocesamiento adecuado para el modelo elegido. El algoritmo de preprocesamiento envía los datos con el formato correcto para ser utilizados a el modelo, que a su vez envía los resultados al servicio. Dicho servicio proporciona los datos a la interfaz que hace entrega de los resultados esperados al usuario.

4.2.3. Diagrama de contexto

El diagrama de contexto se refiere a un diagrama que define el límite entre un sistema o parte de un sistema y su entorno, mostrando las entidades que interactúan con el y a menudo se dice que es un sistema de vista de nivel superior.

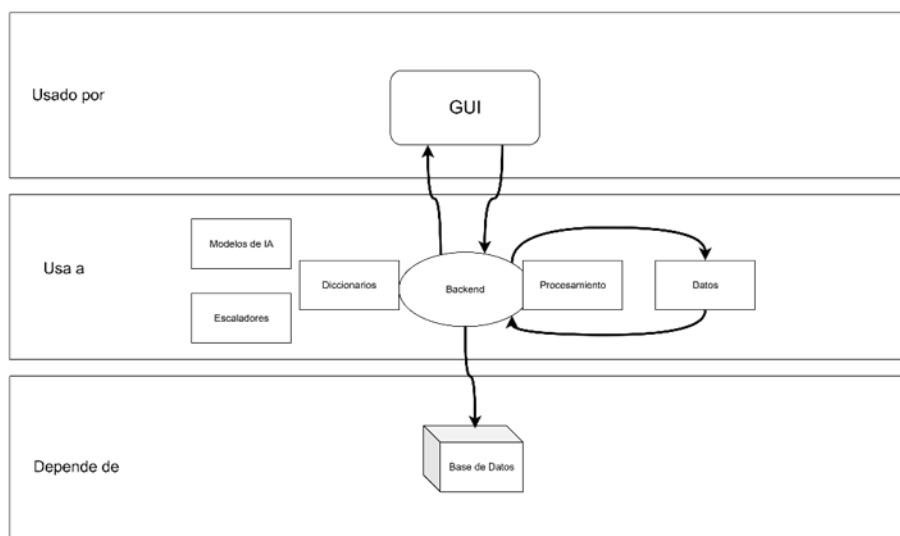


Figura 4.- Diagrama de contexto (Fuente: Leyva López, N.)

Como puede observarse en la Figura X., El sistema Backend es iniciado por el GUI (Frontend), a través de la cual un usuario final puede realizar peticiones a las APIs alojadas en el sistema, el sistema backend con el fin de responder realiza un procesamiento de datos y hace uso de un diccionario para identificar el modelo de IA y el escalador necesario en esa petición, finalmente guarda la respuesta en una DB y regresa la respuesta al usuario.

4.2.4. Modelo arquitectónico

La arquitectura de software a menudo se conoce como el diseño de nivel superior de la arquitectura de un sistema, y consiste en un conjunto de patrones y abstracciones que brindan un marco claro para interactuar con el software. Este, a su vez, define de manera abstracta la arquitectura de los componentes que realizan una tarea, sus interfaces y la comunicación entre ellos. La arquitectura de software se selecciona y diseña de acuerdo con las necesidades y limitaciones (directamente relacionadas con la tecnología disponible).

La arquitectura elegida es una mezcla entre el modelo de servicio web y el monolítico ya que sus características cubren las necesidades del sistema. Algunas de las características que resultan convenientes son que el modelo monolítico permite que todos los componentes del

sistema residan en una única base de código y se ejecuten como una sola unidad. Los componentes interactúan directamente dentro de esta única aplicación, lo que hace que este sistema sea el más adecuado para aplicaciones donde las funcionalidades están fuertemente acopladas y se desarrollan por un mismo equipo. En el caso del modelo de servicio web, este proporciona la infraestructura necesaria para que un usuario (o consumidor) se comuniquen con el sistema a través de un registro de servicio. Estas características lo vuelven una forma muy eficiente de compartir grandes cantidades de datos donde las acciones de seguridad (copias de seguridad, protección, control de acceso y recuperación de errores) se encuentran centralizadas.

Una desventaja que tiene este modelo es que el despliegue y escalado pueden volverse complejos a medida que la aplicación crece, ya que, al ser una única unidad, cualquier cambio requiere el rediseño de todo el sistema, y su escalado implica escalar todo el conjunto de funcionalidades, incluso aquellas que no lo necesitan.

4.2.5. Capas Lógicas

Las capas lógicas de un sistema muestran los componentes de software de una aplicación distribuida y representan la independencia lógica de los componentes según la naturaleza de los servicios que prestan. Dichas capas son las siguientes:

- Capa de usuario: En esta capa se presenta lo que el usuario ve en el sistema, así como la comunicación e interacción (intercambio y captura de información) entre estos.
- Capa de Negocio: Esta capa se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados, y a su vez lo hace con la capa de datos para solicitar al gestor de base de datos el almacenar o recuperar datos.
- Capa de Datos: En esta capa se almacenan los datos del sistema. Está formada por uno o más gestores de bases de datos que realizan el almacenamiento de toda la información, además, recibe solicitudes de almacenamiento, acceso y recuperación de información directo desde la capa de negocio.

4.2.6. Componentes

Los componentes de software son los bloques que proporcionan las funciones del sistema de software. Son configurables y definen cómo interactúan entre sí. Realizan tareas específicas que cumplen con la funcionalidad del sistema. En este caso particular, los componentes se

seleccionaron basándose en requisitos funcionales y de calidad, así como en plantillas predefinidas y diagramas de contexto.

Los componentes definidos para el funcionamiento del sistema son:

- **Interfaz Gráfica:** Permite a los usuarios interactuar dinámicamente a través de una interfaz web en dispositivos de computadora. Desde aquí, los usuarios realizan y solicitan las peticiones que el sistema permite.
- **Manejador de memoria:** Este componente gestiona las solicitudes para asegurar que la capacidad de la memoria del servicio no se sature.
- **Servicio:** Es el componente principal que ofrece el uso de los otros componentes del sistema.
- **Repositorio de modelos:** Contiene los diferentes modelos de aprendizaje máquina disponibles en el servicio.
- **Repositorio de intérpretes:** Almacena los algoritmos que definen las relaciones entre los tipos de preprocesamiento y los modelos.
- **Repositorio de preprocesamiento:** Guarda los distintos algoritmos de preprocesamiento.

4.2.7. Tablas de mapeo

Una tabla de mapeo es el conjunto de declaraciones que recopilan información de uno o de múltiples conjuntos en un solo esquema. A partir de este se pueden crear listas y derivar vistas. El mapeo de datos es una parte esencial del buen funcionamiento de muchas operaciones de datos, ya que es una parte esencial de cualquier proceso de integración de datos.

Caso de uso	Descripción	Interfaz Gráfica	Repositorio de modelos	Repositorio de preprocesamiento	Servicio	Base de datos
CU001	Búsqueda de datos	☒			☒	
CU002	Carga de datos	☒			☒	☒
CU003	Elección de tipo de red a utilizar	☒	☒			

CU004	Preprocesamiento de datos			☒	☒	
CU005	Obtención de datos preprocesados			☒		

Tabla 12.-Mapeo de responsabilidad (Elaboración propia)

En la Tabla 12 se expone la tabla de mapeo que relaciona los Requisitos funcionales y los casos de uso, que, al igual que la Tabla 7 permite observar de una manera más clara dichas relaciones. En este caso conocer la forma en que se enlazan dichos conceptos ayuda a delimitar mejor la forma en que el sistema actuará tomando en cuenta las diversas situaciones a las que este puede ser expuesto.

4.2.8. Herramientas

Para llevar a cabo este proyecto, fue necesario el uso de diversas herramientas case que facilitan el desarrollo de software, en este caso se usaron las siguientes:

1. Python: Es un lenguaje de programación caracterizado por su simplicidad, versatilidad y amplio soporte comunitario. Es uno de los lenguajes más utilizados en el desarrollo de aplicaciones relacionadas con inteligencia artificial (IA) y análisis de datos, gracias a su vasta colección de bibliotecas especializadas. Entre sus ventajas destacan la facilidad para escribir código claro y mantenible, y su compatibilidad con herramientas científicas y tecnológicas de alto rendimiento.
2. Visual Studio Code: Editor de código de gran potencia y versatilidad, compatible con múltiples extensiones para el lenguaje Python.
3. Git y Github: Herramienta de control de versiones, al trabajar en un desarrollo Ágil se cuenta con desarrollos incrementales y progresivos, con la intención de gestionar dichos cambios y realizar el despliegue del software desarrollado.
4. PostgreSQL: Base de datos relacional con licencia de uso gratuito, al requerir de almacenar información generada por la aplicación de opto por usar dicho SGBD.
5. Java: Es un lenguaje de programación caracterizado por su versatilidad y la capacidad de ejecutarse en cualquier equipo, Principal lenguaje para programar aplicaciones móviles en entornos Android
6. Android Studio: IDE especializado en el desarrollo de aplicaciones móviles Android con el lenguaje Java o Kotlin.

4.2.9. Bibliotecas

Flask: Un framework minimalista de Python diseñado para el desarrollo de aplicaciones web y APIs. Su enfoque ligero y modular lo convierte en una herramienta ideal para proyectos que requieren flexibilidad y personalización, especialmente en soluciones que combinan servicios web con modelos de inteligencia artificial.

Joblib: es una biblioteca de Python diseñada para facilitar el almacenamiento y la ejecución eficiente de tareas que requieren cómputo intensivo. Se usa principalmente para:

- Serialización y deserialización (persistencia de objetos): Permite guardar y cargar modelos de machine learning, matrices de datos y otros objetos grandes de Python de manera eficiente.
- Paralelización: Ofrece una forma sencilla de paralelizar bucles y funciones utilizando múltiples núcleos de la CPU.
- Optimización de memoria: Usa técnicas de almacenamiento en caché para evitar computaciones innecesarias en tareas repetitivas.

Scikit Learn: Una biblioteca de Python ampliamente utilizada para el desarrollo de soluciones de aprendizaje automático gracias a su simplicidad, flexibilidad y variedad de algoritmos implementados. Ofrece herramientas para tareas como clasificación, regresión, reducción de dimensionalidad y preprocesamiento de datos, siendo una opción ideal para construir modelos eficientes y escalables con facilidad.

psycopg2: es una biblioteca de Python que actúa como un adaptador para bases de datos PostgreSQL. Permite a las aplicaciones en Python conectarse a un servidor PostgreSQL, ejecutar consultas SQL y gestionar transacciones de manera eficiente.

Gunicorn: es un servidor WSGI (Web Server Gateway Interface) para aplicaciones web en Python. Se usa principalmente para desplegar aplicaciones web creadas con frameworks como Django o Flask en producción.

ONNX: (Open Neural Network Exchange) es un formato de código abierto para representar modelos de aprendizaje automático. Su propósito es permitir a los desarrolladores mover modelos entre diferentes frameworks, herramientas y runtimes de IA de manera más sencilla.

4.3. Desarrollo

4.3.1. Selección, uso de modelos y preprocesamiento

Para que la API construida funcione como se espera es necesario preprocesar los datos de entrada e identificar adecuadamente cuando se usara cada modelo almacenado en la aplicación para realizar la predicción de forma correcta.

4.3.2. Comprobación de datos

Con el fin de garantizar que los modelos y la API funcionen adecuadamente es necesario comprobar los tipos y los valores de los datos que se emplean, esto toma aún más relevancia en un lenguaje como Python que es de tipado dinámico y en muchos casos esto puede llevar a errores en la manipulación.

4.3.3. Algoritmo para selección de modelo y escalador

Los modelos de IA usados tienen diferentes parámetros de entrada y orden por lo que es necesario emplear estructuras y funciones que nos permitan la selección del modelo correcto de forma dinámica.

Para resolver este problema se emplearon diccionarios dado que permiten usar valores clave: valor sin importar su orden.

4.3.4. Estructura de la API

Antes de crear la aplicación, es necesario crear la estructura de la API, Esto consiste en el cómo están organizados y el cómo se almacenan (nombre y formato), Es necesario tener un método principal que liste los endpoints que serán consumidos por el cliente como punto de entrada, luego un archivo auxiliar para métodos de procesamiento de información y finalmente los modelos de IA, los escaladores y unos diccionarios para conocer qué modelo corresponde a que escalador según su entrada de datos.

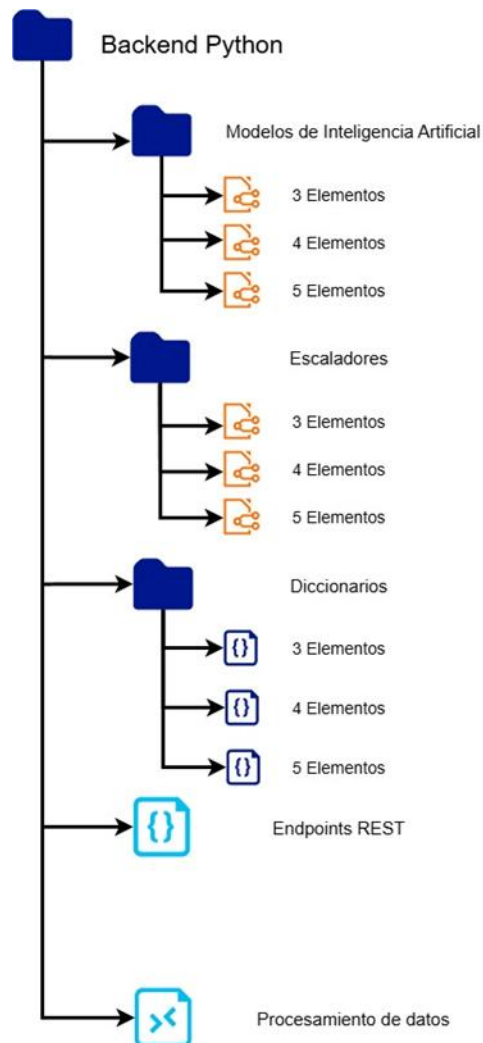


Figura 5.-Estructura de archivos (Fuente: Elaboración propia)

Se empleo la misma estructura de archivos en la aplicación móvil hecha en Java.

4.3.5. Creación de rutas

Las rutas definen qué páginas y aplicaciones redirigir (por ejemplo, ejecutar cualquier tarea de Python, mostrar páginas HTML, peticiones y respuestas). Estas rutas se crearon en la instancia del método principal. Es obligatorio el importar nuestra aplicación web, y de esta manera ligar las rutas a las tareas del programa, así cuando se realicen las peticiones, el algoritmo le redirigirá a la 69 ruta en la función que debe ser usadas. También se encargan de enviar las respuestas a el usuario (cliente).

4.3.6. Despliegue de modelo de aprendizaje máquina

Es necesario montar un modelo previamente entrenado (como se ha hecho referencia con anterioridad). Se utiliza referenciando el módulo de preprocesamiento, que realiza todo el proceso de transformación de datos para que estos sean compatibles con el tipo de modelo de ML. Es posible discernir cual es necesario debido al uso de la función de automatización de elección de modelo tomando en cuenta la extensión del archivo que se ha cargado. Así se crea todo un proceso automatizado de carga, preproceso, procesado y obtención de resultados. Para la obtención de resultados se regresa un arreglo llamado predicción que interactúa con la aplicación web, el cual después de recibirlo se lo proporciona al usuario para su uso.

4.3.7. Comunicación Cliente-Servidor

Para que la aplicación web (frontend) pueda comunicarse con la API desplegada en Heroku, el enrutamiento se gestiona directamente a través de las rutas definidas en la API de Flask.

Cuando la aplicación frontend realiza una solicitud a una URL específica de la API (por ejemplo, <https://nombre-de-tu-app.herokuapp.com/api/data>), esta solicitud viaja a través de internet directamente a los servidores de Heroku. Heroku actúa como el enrutador, dirigiendo la solicitud al contenedor (Dyno) donde se está ejecutando tu aplicación Flask.

Dentro de la aplicación Flask, el framework utiliza su sistema de enrutamiento para hacer coincidir la URL solicitada (/api/data en el ejemplo) con la función Python correspondiente que se ha definido usando los decoradores de ruta (como `@app.route('/api/data')`). Esta función procesa la solicitud y devuelve una respuesta (generalmente en formato JSON) que Heroku envía de vuelta a la aplicación frontend.

De esta manera, no es necesario configurar manualmente puertos en un router, ya que Heroku gestiona la exposición de la aplicación a internet. La comunicación se basa en las solicitudes HTTP realizadas por el cliente a las rutas específicas definidas en la API de Flask alojada en Heroku.

4.4. Despliegue

Para desplegar esta API como un servicio web accesible, hemos optado por Heroku, una plataforma en la nube (PaaS) que simplifica la publicación de aplicaciones. A diferencia de un servidor local con un WSGI como Waitress, Heroku se encarga de la infraestructura subyacente, permitiendo que nuestra aplicación Python basada en Flask sea expuesta al mundo.

Heroku funciona mediante Dynos, que son contenedores ligeros donde se ejecuta la aplicación. Al subir nuestra API de Flask a Heroku, la plataforma se encarga automáticamente de proveer un entorno de ejecución, asignar una URL pública y gestionar la escalabilidad básica. Esto elimina la necesidad de configurar servidores web o DNS dinámicos manualmente, ya que Heroku asigna una URL única (por ejemplo, nombre-de-tu-app.herokuapp.com) a cada aplicación desplegada.

Una vez que la API está publicada, es crucial asegurar que pueda ser accedida de forma segura y controlada desde diferentes orígenes, especialmente por aplicaciones cliente alojadas en dominios distintos. Para esto, utilizamos CORS (Cross-Origin Resource Sharing).

CORS es un mecanismo de seguridad en los navegadores que permite a un servidor indicar cualquier origen (dominio, esquema o puerto) diferente al suyo que debería tener permiso para cargar recursos. Sin CORS, las peticiones JavaScript realizadas desde un dominio diferente al de la API serían bloqueadas por el navegador, lo que impediría la comunicación entre la aplicación cliente (por ejemplo, una interfaz de usuario web) y nuestra API.

En nuestra API de Flask, implementamos CORS utilizando la extensión flask-cors. Esta librería facilita la configuración de las cabeceras CORS necesarias en las respuestas de la API, especificando qué orígenes, métodos HTTP (GET, POST, PUT, DELETE, etc.) y cabeceras están permitidos. Por ejemplo, podemos configurar la API para que solo acepte solicitudes de nuestro dominio de frontend, o de cualquier dominio si la API es pública.

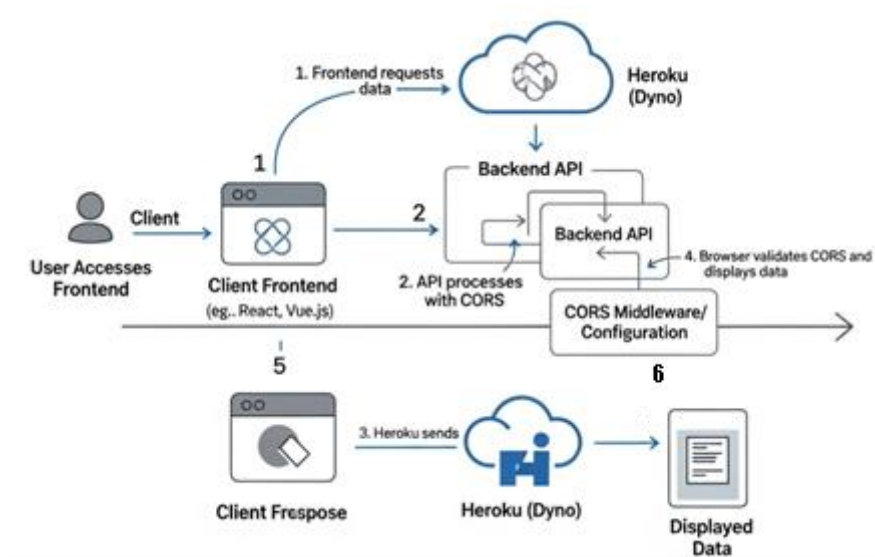


Figura 6.-Flujo completo de consumo (Fuente: Elaboración propia)

Este diagrama visualiza los siguientes pasos:

1. Usuario Accede al Frontend: El usuario inicia la interacción visitando la aplicación cliente (por ejemplo, una página web) en su navegador.
2. Frontend Solicita Datos: La aplicación cliente en el navegador realiza una solicitud HTTP (como GET o POST) a la API de Flask.
3. Heroku Recibe la Solicitud: La plataforma Heroku intercepta la solicitud y la dirige a la instancia activa de tu API de Flask.
4. API Procesa y Aplica CORS: Tu API de Flask procesa la solicitud. Antes de enviar la respuesta, la extensión flask-cors verifica el Origin de la solicitud y añade las cabeceras CORS (Access-Control-Allow-Origin, etc.) necesarias a la respuesta. Esto es crucial para la seguridad entre dominios.
5. Heroku Envía la Respuesta: Heroku devuelve la respuesta generada por la API de Flask de vuelta al navegador del usuario.
6. Navegador Valida CORS y Muestra Datos: El navegador del usuario recibe la respuesta. Si las cabeceras CORS son válidas y la solicitud de origen cruzado es permitida, el navegador entrega los datos a la aplicación cliente, que finalmente los muestra al usuario.

Capítulo 5

5. Pruebas y Resultados

Con el propósito de probar que nuestro servicio backend que ofrece acceso a los modelos de ML se encuentra hosteado correctamente y puede ser consumido.

Se implemento una base de datos PostgreSQL usando la plataforma Heroku, se creó una tabla para almacenar todas las predicciones realizadas a los modelos de IA.

5.1. API

La API fue cargada en Heroku una Plataforma como servicio (PaaS), dicha plataforma permite alojar APIs desarrolladas en Python con Flask, requiriendo poca configuración, además de soportar la integración de bases de datos SQL mediante PostgreSQL, dicho SGBD se encuentra integrado dentro de la plataforma listo para su uso.

URL: <https://conagua-evaluador-nutrientes-370a71552ad5.herokuapp.com/api/process/3>

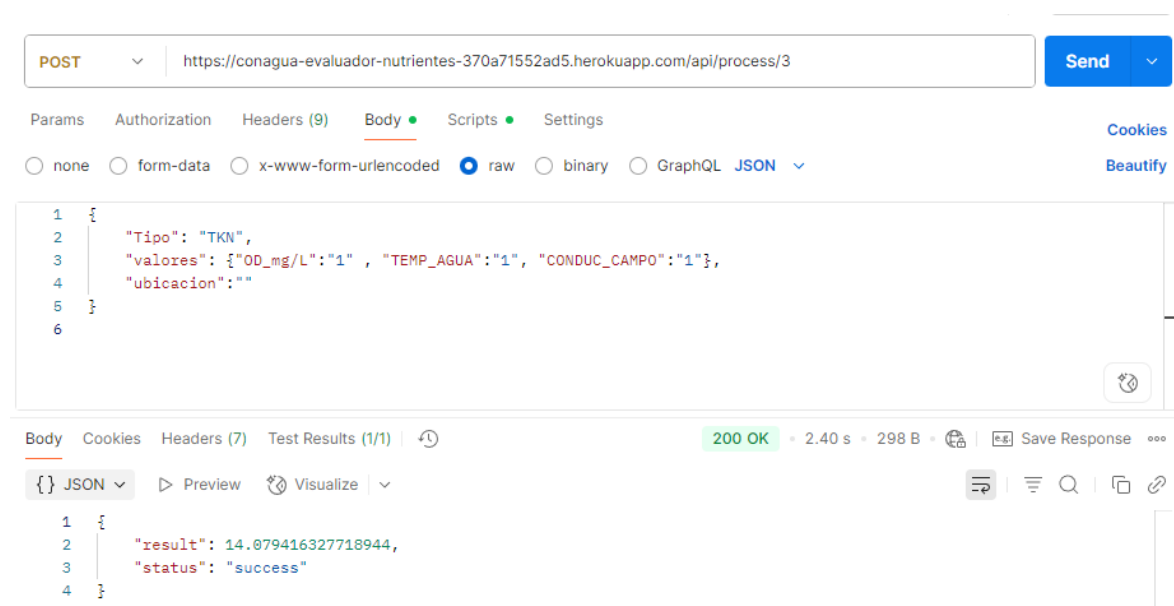


Figura 7.- Prueba de la API en línea, finalizada (Fuente: Elaboración propia)

5.2. Base de datos

Se implementó una base de datos PostgreSQL utilizando la plataforma Heroku, se creó una tabla para almacenar todas las predicciones realizadas a los modelos de IA.

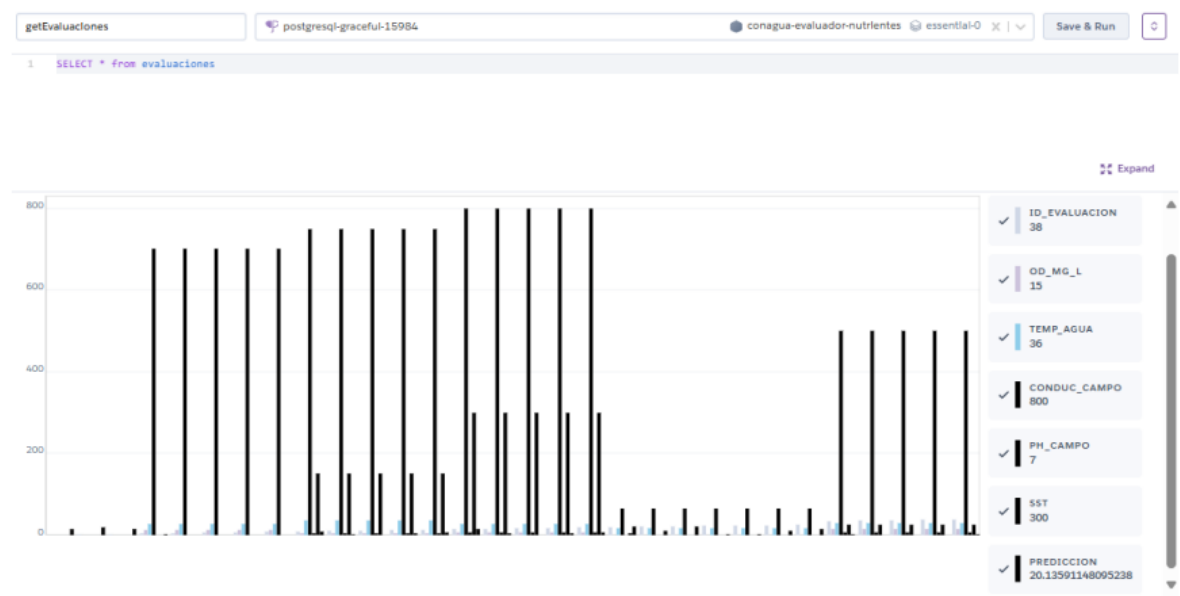


Figura 8.- Grafico de entradas en la base de datos. (Fuente: Elaboración propia)

La base de datos guarda la información del nutriente evaluado, las variables que se usaron para predecir, el resultado de la predicción del modelo y la ubicación, Esta última es opcional y sirviendo como un apoyo al momento de realizar un análisis con la información recabada.

Ubicacion	Nutriente	Od_mg_l	Temp_agua	Conduc_campo	Ph_campo	Sst	Prediccion	Ubicacion
1	TKN	1	1	1	1	1	14.786999818337326	Escuela
2	TKN	1	1	1	1	0	18.646472948354226	EMPTY
3	TKN	1	1	1	0	0	14.879416327718944	EMPTY
4	nitrogeno	12	27	700	0	0	2.389876557988858	
5	fosforo	12	27	700	0	0	0.38167298516789824	
6	ortofosfato	12	27	700	0	0	0.85797494935191847	
7	amoníaco	12	27	700	0	0	0.27499577958698685	
8	TKN	12	27	700	0	0	0.9934721255113891	

Figura 9.- Tabla de predicciones en la base de datos. (Fuente: Elaboración propia)

Se monitoreo del servidor backend en la nube, usando Heroku dashboard para monitorizar el estado del servidor después de ser puesto en marcha.

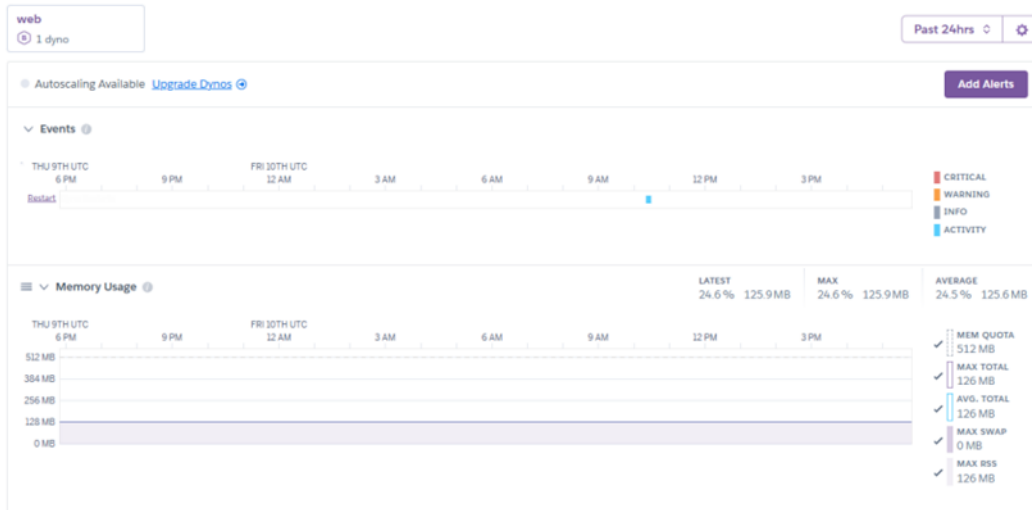


Figura 10.-Panel de control de Heroku (Fuente: Elaboración propia)

5.3. Desarrollo Movil

Durante la decisión de herramientas se planteó el uso de modelos ONNX, al ser más ligeros y multiplataforma, permitiendo la posibilidad de un desarrollo móvil:

```

1  import joblib
2  import skl2onnx
3  from skl2onnx.common.data_types import FloatTensorType
4
5  # Cargar el modelo RandomForestRegressor desde el archivo .joblib
6  modelo = joblib.load("cincoCombinaciones\P_TOT\RF_P_TOT_RS_093\modelo_random_forest.joblib")
7
8  # Definir la forma de entrada (ejemplo con 5 características)
9  n_features = modelo.n_features_in_ # Número de características de entrada
10 entrada = FloatTensorType([None, n_features])
11
12 # Convertir el modelo a ONNX
13 modelo_onnx = skl2onnx.convert_sklearn(modelo, initial_types=[("input", entrada)])
14
15 # Guardar el modelo ONNX en un archivo
16 with open("modelo_rf.onnx", "wb") as f:
17     f.write(modelo_onnx.SerializeToString())
18
19 print("Modelo RandomForestRegressor guardado como modelo_rf.onnx")

```

Figura 11.- Código para convertir modelos de IA a formato ONNX (Fuente: Elaboración propia)

Después de la conversión a modelos ONNX se realizó el proceso de implementación en una app móvil Android con java, donde se realizaron clases, para representar diccionarios de acceso a los modelos de IA, Clase para la entrada de datos.

```

I Enviando los valores
I [OD_mg/L, TEMP_AGUA, CONDUC_CAMPO]
I masivo
I PRIMER PUNTO
I tipo:masivo
I Valores:{OD_mg/L=1, TEMP_AGUA=1, CONDUC_CAMPO=1}
I Valores:PRIMER PUNTO

```

Figura 12.- Prueba de resultado del modelo de IA implementado en la app móvil. (Fuente: Elaboración propia)

Posteriormente de implemento un almacenamiento de información mediante un archivo de “CSV” almacenado localmente en el dispositivo Android.

```

Prediccion: 20.073595
Ubicacion:
Nutriente: N_TOT
OD mg/L: 1.0
Temp. Agua: 1.0
Conduc. Campo: 1.0
pH Campo: 0.0
SST: 0.0
Fecha: 2024-11-14

```

Figura 13.-Datos guardados en archivo CSV (Fuente: Elaboración propia)

Capítulo 6

6. Conclusiones y trabajo futuro

Durante esta sección son presentadas las conclusiones y las posibles recomendaciones para mantener el proyecto desarrollado en un correcto funcionamiento, Son mencionadas tanto las aportaciones de este proyecto como las limitaciones del mismo debido a las herramientas usadas en su desarrollo.

6.1. Conclusiones del proyecto

Este proyecto tiene como objetivo principal ser una herramienta auxiliar para la evaluación de los nutrientes presentes en muestras de agua, ofreciendo una aproximación confiable a los valores obtenidos a través de análisis de laboratorio. Aunque no reemplaza completamente los métodos tradicionales, proporciona resultados rápidos y prácticos basados en un rango de entradas que varía entre 3 y 5 datos fundamentales. Esto lo convierte en una solución eficiente y accesible para el análisis preliminar de calidad del agua, especialmente en escenarios donde el acceso a laboratorios especializados es limitado.

Durante el desarrollo del proyecto, se enfrentaron desafíos técnicos relacionados con la compatibilidad de versiones de software. Fue necesario realizar actualizaciones tanto de Python como de varias bibliotecas utilizadas, debido a problemas de compatibilidad entre las versiones de las herramientas disponibles y las versiones en las que fueron desarrollados originalmente los modelos de inteligencia artificial proporcionados por la institución. Este proceso implicó realizar pruebas exhaustivas para garantizar que los cambios en las versiones no afectaran el rendimiento ni la precisión de los modelos implementados.

Un desafío importante fue la conversión de los modelos de IA a un formato interoperable. Inicialmente, los modelos estaban desarrollados en Scikit-learn, y durante el proyecto se convirtieron al formato ONNX (Open Neural Network Exchange) para garantizar su compatibilidad y facilidad de implementación en diferentes entornos, como dispositivos móviles. Sin embargo, este proceso puede generar discrepancias en la precisión de los

resultados, ya que no todos los lenguajes y formatos manejan los tipos de datos y su precisión de la misma manera. En este caso, los modelos de IA originales utilizaban una precisión de 64 bits para los datos de entrada y salida, mientras que al transformarlos a ONNX la precisión se redujo a 32 bits. A pesar de esto, no se observó una diferencia significativa en los resultados finales. No obstante, es fundamental considerar este aspecto en proyectos futuros, especialmente si la precisión es crítica para las aplicaciones.

La base de datos utilizada en este proyecto fue implementada utilizando un adaptador nativo adecuado para interactuar con la base de datos proporcionada por la plataforma de alojamiento donde se encuentra desplegada la API. Esta configuración asegura una integración fluida entre el backend y la base de datos, optimizando el acceso y la gestión de los datos. La elección de un adaptador nativo también permitió aprovechar al máximo las capacidades de la plataforma de alojamiento, garantizando un rendimiento estable y confiable.

En resumen, el desarrollo de este proyecto no solo implicó la implementación de una solución técnica, sino también la superación de múltiples desafíos relacionados con la compatibilidad de herramientas, la precisión de los modelos y la integración de componentes en un entorno de producción. Estos aprendizajes y consideraciones sientan una base sólida para futuras iteraciones y mejoras en sistemas similares.

6.2. Recomendaciones

6.2.1. Mejorar la infraestructura del sistema

Actualmente, la infraestructura del servidor implementada es suficiente para soportar las necesidades operativas del proyecto en su estado actual. Sin embargo, a medida que el sistema crezca en alcance y popularidad, y si se busca automatizar completamente el proceso en el futuro, será necesario actualizar el servidor para manejar mayores cargas de trabajo. Esto implica la adopción de soluciones más robustas, como la migración a servicios de infraestructura más escalables (por ejemplo, Amazon Web Services, Google Cloud Platform o Microsoft Azure), la implementación de balanceadores de carga, y el uso de contenedores mediante herramientas como Docker y Kubernetes para garantizar la estabilidad y escalabilidad del sistema. Estas mejoras garantizarían que el sistema sea capaz de soportar

múltiples usuarios concurrentes y manejar grandes volúmenes de datos sin comprometer el rendimiento.

6.2.2. Integrar la API con otros sistemas

La API desarrollada en este proyecto ofrece predicciones fiables sobre los nutrientes presentes en muestras de agua. Sin embargo, para maximizar su utilidad, es necesario integrarla con otros sistemas que permitan monitorear, gestionar y controlar la calidad del recurso hídrico de manera integral. Una de las posibles evoluciones de este proyecto es la creación de un gemelo digital, un modelo virtual dinámico que replica el sistema físico real de gestión del agua. Este gemelo digital podría combinar los datos generados por la API con otras fuentes de información, como sensores IoT instalados en el campo, datos meteorológicos en tiempo real y modelos hidrológicos avanzados. Con esta integración, se podrían implementar sistemas predictivos más completos y herramientas de gestión automatizadas, como alertas tempranas para problemas de calidad del agua, optimización de recursos y análisis preventivo de riesgos.

6.2.3. Mejorar los modelos de Inteligencia Artificial

Los modelos de IA desarrollados actualmente ofrecen predicciones confiables basadas en las variables de entrada proporcionadas. Sin embargo, para asegurar la evolución del sistema y su capacidad de adaptarse a nuevos desafíos, será necesario actualizar los modelos periódicamente. Esto incluye buscar nuevas variables de entrada que permitan obtener predicciones más precisas y completas, como datos sobre la contaminación microbiana, presencia de metales pesados o cambios en los patrones estacionales. Además, se recomienda explorar y adoptar tecnologías de IA más avanzadas, como redes neuronales profundas (deep learning) y modelos basados en arquitecturas como Transformers, que han demostrado ser altamente eficientes y precisos en diversas aplicaciones. También sería beneficioso evaluar tecnologías de baja potencia, como modelos ligeros específicamente diseñados para dispositivos móviles o de borde (edge computing), asegurando así que el sistema pueda ser implementado en entornos con recursos limitados y manteniendo un bajo consumo energético.

7.Bibliografía

Ehsan, A., Abuhaliqa, M. A. M., Catal, C., & Mishra, D. (2022). RESTful API testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences*, 12(9), 4369.

Ukpongson, M. P. (2023). REST API for a store management system using Flask.

Lehtonen, O. (2024). On-Device Machine Learning Inference using Cross-Platform Frameworks.

Bourhis, P., Reutter, J. L., & Vrgoč, D. (2020). JSON: Data model and query languages. *Information Systems*, 89, 101478.

Jena, P. K., Rahaman, S. M., Das Mohapatra, P. K., Barik, D. P., & Patra, D. S. (2023). Surface water quality assessment by Random Forest. *Water Practice & Technology*, 18(1), 201-214.

Juba, S., Vannahme, A., & Volkov, A. (2015). *Learning PostgreSQL*. Packt Publishing Ltd.

PsyCOPG – PostgreSQL Database Adapter for Python — PsyCOPG 2.9.10 documentation. (s. f.). Recuperado 4 de octubre de 2024, de <https://www.psycopg.org/docs/>

Dimes, T. (2015). *Conceptos Básicos de Scrum: Desarrollo de software Agile y manejo de proyectos Agile*. Babelcube Inc..

García-Peñalvo, F. J., García-Holgado, A., & Vázquez-Ingelmo, A. (2017). Metodologías de Ingeniería de Software. Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso, 2018.

Pagotto, T., Fabri, J. A., Lerario, A., & Gonçalves, J. A. (2016, June). Scrum solo: Software process for individual development. In *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1-6). IEEE.

Blanco, P., Camarero, J., Fumero, A., Warterski, A., & Rodríguez, P. (2009). Metodología de desarrollo ágil para sistemas móviles. *Introducción al desarrollo con Android y el iPhone*. Dr. en Ing. Sist. Telemáticos, 1-30.

Ortega Arias, C. A. (2015). Elaboración de una herramienta dinámica alimentada de archivos CSV para la generación de gráficos estadísticos de uso libre en la web.

Braga, P. H. S. (2023). Implantação de modelos de aprendizado de máquina no formato onnx utilizando diferentes frameworks.

Jajal, P., Jiang, W., Tewari, A., Kocinare, E., Woo, J., Sarraf, A., ... & Davis, J. C. (2024, September). Interoperability in deep learning: a user survey and failure analysis of ONNX model converters. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 1466-1478).

Thokala, V. S. (2021). Integrating Machine Learning into Web Applications for Personalized Content Delivery using Python. *Int. J. Curr. Eng. Technol*, 11(6), 652-660.

Oladipupo Ayodele, T. (2010). Types of Machine Learning Algorithms. University of Portsmouth. DOI: 10.5772/9385.

Leyva López, N. (2022). Metodología para la implementación de modelos de aprendizaje máquina como servicios [Tesis de maestría, INSTITUTO TECNOLÓGICO DE CULIACÁN]. <https://rinacional.tecnm.mx/jspui/handle/TecNM/6259>

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2021). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, A., ... & Hedges, M. (2015). Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems* 28 (pp. 2503-2511).

Joblib developers. (2024). Joblib Documentation. Recuperado de <https://joblib.readthedocs.io/>

Kreuzberger, D., Auer, S., & Prinz, C. (2023). MLOps: A Survey and Taxonomy. *ACM Computing Surveys*, 55(2), 1-36.

Muller, A. C., & Guido, S. (2018). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32 (pp. 8024-8035).

Scikit-learn developers. (2023). scikit-learn documentation: Model persistence. Recuperado de https://scikit-learn.org/stable/model_persistence.html