Implementación de una plataforma con flujo automatizado para el entrenamiento continuo y validación de modelos de Machine Learning para clasificar enfermedades neurológicas

Emmanuel Arizabaleta Garcés

Proyecto de investigación presentado para optar al título de Bioingeniero

Asesor
John Fredy Ochoa Gómez, Doctor (PhD) en Ingeniería Electrónica

Línea de investigación:
Neurociencias computacionales

Grupo de investigación:
Grupo de Neurociencias de Antioquia

Universidad de Antioquia
Facultad de Ingeniería
Bioingeniería
Medellín
2025

Cita	Arizabaleta Garcés [1]
Referencia Estilo IEEE (2020)	[1] E. Arizabaleta Garcés, "Implementación de una plataforma con flujo automatizado para el entrenamiento continuo y validación de modelos de Machine Learning para clasificar enfermedades neurológicas", Trabajo de grado profesional, Bioingeniería, Universidad de Antioquia, Medellín, Antioquia, Colombia, 2025.

creative commons CC BY NC SA

UNIVERSIDAD DE ANTIOQUIA
Vicerrectoría de Docencia

Sistema de Biblioteca

Centro de Documentación Ingeniería (CENDOI)

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co
Rector: John Jairo Arboleda Céspedes.
Decano/Director: Julio César Saldarriaga Molina.
Jefe departamento: Jenny Kateryne Aristizábal Nieto.

El contenido de esta obra corresponde al derecho de expresión de los autores y no compromete el pensamiento institucional de la Universidad de Antioquia ni desata su responsabilidad frente a terceros. Los autores asumen la responsabilidad por los derechos de autor y conexos.

Dedicatoria

Este trabajo se dedica a mi familia, por su paciencia y compañía durante todo el proceso académico; a mis amigos y colegas, de quienes he recibido aprendizajes y motivación en distintos momentos; y a los profesores del programa, cuyas enseñanzas han nutrido mi formación profesional. A todos ellos va esta obra como reconocimiento del camino recorrido.

Agradecimientos

Quiero expresar mi sincero agradecimiento al profesor John Fredy Ochoa Gómez por su orientación constante y su apoyo durante todo el proceso de este proyecto de investigación; sin su guía este trabajo no habría sido posible. Agradezco al Grupo de Neurociencias de Antioquia (GNA) por proporcionar un entorno académico y los recursos necesarios para el desarrollo del proyecto, así como por las contribuciones colectivas que enriquecieron mi trabajo. Finalmente, agradezco a mi familia, amigos y a los profesores del programa por su apoyo y enseñanzas a lo largo de mi formación.

TABLA DE CONTENIDO

RESUMEN 6
ABSTRACT 7
CAPITULO I 8
I. INTRODUCCIÓN 8
II. OBJETIVOS 9
III. MARCO TEÓRICO Y ESTADO DEL ARTE 10
CAPITULO II: DISEÑO DE INFRAESTRUCTURA DE DATOS 14
I. INTRODUCCIÓN 14
II. METODOLOGÍA 15
A. DEFINICIÓN DE REQUISITOS DEL SISTEMA 15
B. SELECCIÓN Y COMPARACIÓN DE HERRAMIENTAS 16
C. IMPLEMENTACIÓN DE MLFLOW COMO SISTEMA DE TRACKING 17
D.CONTENERIZACIÓN CON DOCKER 18
E. IMPLEMENTACIÓN DE POSTGRESQL COMO SISTEMA DE ALMACENAMIENTO 19
III. RESULTADOS Y DISCUSIONES 20
CAPÍTULO III: IMPLEMENTACIÓN DE UN PIPELINE AUTOMATIZADO DE MACHINE LEARNING 22
I. INTRODUCCIÓN 22
II. METODOLOGÍA 23
A. ANÁLISIS Y REFACTORIZACIÓN DEL MODELO EXISTENTE 23
B. DISEÑO DE UN PIPELINE GENÉRICO Y REUTILIZABLE 26
C. IMPLEMENTACIÓN DEL FLOW CON PREFECT Y MLFLOW 27
D. ORQUESTACIÓN Y EJECUCIÓN DEL PIPELINE 29
E. INTERFACES GRÁFICAS DE ML FLOW Y PREFECT 30
III. RESULTADOS Y DISCUSIONES 33
CAPÍTULO IV: DESARROLLO DE LA INTERFAZ DE APLICACIÓN Y SERVICIOS 35
I. INTRODUCCIÓN 35
II. METODOLOGÍA 36

A. DISEÑO E IMPLEMENTACIÓN DEL BACKEND DE SERVICIOS 36
B. DISEÑO E IMPLEMENTACIÓN DEL FRONTEND 37
III. RESULTADOS Y DISCUSIONES 47
CAPÍTULO V: VALIDACIÓN DE LA PLATAFORMA 49
I. INTRODUCCIÓN 49
II. METODOLOGÍA 49
A. PRUEBAS DE GESTIÓN DE MODELOS 49
B. PRUEBAS DE GENERACIÓN DE NUEVAS PREDICCIONES 50
III. RESULTADOS Y DISCUSIONES 51
CAPÍTULO VI: CONCLUSIONES Y TRABAJO FUTURO 52
I. CONCLUSIONES 52
II. TRABAJO FUTURO 53
REFERENCIAS 54
ANEXOS 56

RESUMEN

El análisis de señales de electroencefalografía (EEG) mediante Machine Learning (ML) para el diagnóstico de enfermedades neurológicas a menudo se ve limitado por flujos de trabajo manuales que dificultan la reproducibilidad y escalabilidad. Este proyecto aborda dicha problemática mediante el desarrollo de una plataforma integral de Machine Learning Operations (MLOps) con el objetivo de implementar un sistema automatizado para el entrenamiento continuo y la validación de modelos de clasificación. Para ello, se diseñó una arquitectura contenerizada con Docker, orquestada con Prefect y monitoreada con MLflow, que gestiona pipelines automatizados de entrenamiento y predicción. La metodología incluyó la implementación de un flujo capaz de procesar datos crudos de EEG en formato BIDS, extraer características relevantes y realizar inferencias con un modelo gestionado y previamente entrenado a través de esta plataforma. La validación funcional demostró el éxito del sistema, ejecutando un ciclo completo desde el entrenamiento de un modelo hasta su uso en una predicción sobre un dataset. Como resultado, se obtuvo una plataforma que automatiza y centraliza el flujo de trabajo de ML, reduciendo la barrera técnica para los usuarios y facilitando la reproducibilidad y escalabilidad de los experimentos en el campo de la neurociencia.

Palabras clave: Machine Learning Operations (MLOps), Electroencefalografía (EEG), Entrenamiento Continuo, Pipeline

ABSTRACT

The analysis of electroencephalography (EEG) signals using Machine Learning (ML) for the diagnosis of neurological diseases is often limited by manual workflows that hinder reproducibility and scalability. This project addresses this issue through the development of a comprehensive Machine Learning Operations (MLOps) platform with the objective of implementing an automated system for the continuous training and validation of classification models. To this end, a containerized architecture was designed with Docker, orchestrated with Prefect, and monitored with MLflow, which manages automated training and prediction pipelines. The methodology included the implementation of a workflow capable of processing raw EEG data in BIDS format, extracting relevant features, and performing inferences with a model managed and previously trained through this platform. Functional validation demonstrated the system's success by executing a complete cycle from training a model to its use in a prediction on a dataset. As a result, a platform was obtained that automates and centralizes the ML workflow, reducing the technical barrier for users and facilitating the reproducibility and scalability of experiments in the field of neuroscience.

Keywords: Machine Learning Operations (MLOps), Electroencephalography, Pipeline, Continuos Training

CAPITULO I

I. INTRODUCCIÓN

Las enfermedades neurológicas representan un problema de salud global con un impacto significativo en la morbilidad y mortalidad. En 2021, aproximadamente 3.40 mil millones de personas presentaban alguna condición que afectaba el sistema nervioso [1]. Estas patologías fueron responsables de 11.1 millones de muertes [1]. Con el avance de las herramientas de investigación en neurociencia, diversas técnicas permiten explorar la función cerebral, entre ellas la resonancia magnética funcional (fMRI), la magnetoencefalografía (MEG) y la electroencefalografía (EEG) [2]. Dentro de estas técnicas, EEG ha sido la más utilizada, debido a su alta resolución temporal y bajo costo en comparación con otras tecnologías [2]. Sin embargo, las señales EEG presentan una alta variabilidad entre individuos y dentro del mismo sujeto en distintos momentos, lo que dificulta la generalización de los análisis. Además, su complejidad y alta dimensionalidad requieren métodos avanzados para la extracción de patrones relevantes. En este contexto, el uso de modelos de Machine Learning (ML) ha emergido como una alternativa prometedora para mejorar la precisión y eficiencia en la clasificación de enfermedades neurológicas [2].

Esta investigación surge como una ampliación de un proyecto en desarrollo titulado "EVALUACIÓN DE LA ACTIVIDAD ELÉCTRICA CEREBRAL EN SUJETOS CON DETERIORO COGNITIVO MEDIANTE ELECTROENCEFALOGRAFÍA PORTABLE Y MODELOS DE MACHINE LEARNING". En dicho estudio, se vienen recopilando señales EEG de diversos sujetos durante la realización de diferentes actividades, y estos datos se han venido utilizando para entrenar y validar un modelo de Machine Learning capaz de clasificar sujetos con deterioro cognitivo.

A pesar de los avances logrados en el estudio precedente, la solución desarrollada aún depende de procesos manuales para la estructuración y entrega de los datos recolectados al modelo, el entrenamiento de modelos y su actualización. Para superar esta limitación y expandir el alcance del trabajo previo, este proyecto propone el desarrollo de una plataforma que integra un flujo automatizado de operaciones de ML, permitiendo el entrenamiento continuo y la validación de modelos a partir de nuevos datos. El modelo inicial y el conjunto de datos base para este desarrollo fueron tomados del trabajo previo mencionado.

El objetivo de esta investigación es desarrollar un sistema que automatice la gestión de datos y el entrenamiento continuo de modelos de ML. Para ello, se desarrolló una infraestructura para almacenamiento de datos, un pipeline automatizado de ML y una interfaz web para la visualización y monitoreo del sistema.

II. OBJETIVOS

A. Objetivo general

Desarrollar una plataforma con un flujo de operaciones en Machine Learning aplicado al ámbito de la salud, con un enfoque específico en la clasificación de enfermedades neurológicas basado en el análisis de índices EEG.

B. Objetivos específicos

Diseñar una infraestructura de datos que permita el almacenamiento y gestión de métricas asociadas a las señales EEG.

Implementar un pipeline automatizado de Machine Learning que integre la recolección, entrenamiento continuo y actualización del modelo basado en nuevos datos EEG.

Construir un backend y un frontend que permita la visualización de datos, el monitoreo del estado del sistema y la realización de nuevas predicciones.

Validar la funcionalidad integral de la plataforma con el flujo de operaciones de Machine Learning implementado, incluyendo sus capacidades de gestión de datos, actualización automática de modelos y generación de predicciones.

III. MARCO TEÓRICO Y ESTADO DEL ARTE

Machine Learning (ML)

Machine Learning (ML) constituye un campo de la inteligencia artificial que ha experimentado un notable desarrollo en la última década. ML se fundamenta en técnicas estadísticas, computacionales y probabilísticas avanzadas para crear sistemas capaces de aprender automáticamente a partir de conjuntos de datos [3].

En la actualidad, el área médica genera un volumen de datos tan grande que resulta imposible analizarlo directamente por humanos. Estos datos incluyen imágenes médicas de alta resolución, métricas continuas de biosensores, historias clínicas electrónicas, entre otros [4]. Esta situación plantea la necesidad de implementar y depender cada vez más de las máquinas para procesar tanta información. Es aquí donde el machine learning se vuelve crucial en el ámbito de la salud, ya que la principal ventaja de estos sistemas radica en su capacidad para analizar grandes volúmenes de información compleja de forma más eficiente [5].

Machine Learning Operations (MLOps)

Machine Learning Operations (MLOps) abarca un conjunto de herramientas, métodos y prácticas recomendadas, cuyo objetivo es estandarizar el desarrollo y operación de sistemas de aprendizaje automático. Su finalidad es automatizar y mejorar la implementación, vigilancia y mantenimiento de modelos de Machine Learning, para que sean sólidos, confiables y que se puedan escalar fácilmente [6].

Los procesos de MLOps pueden clasificarse en diferentes niveles de madurez según el grado de automatización implementado. En el Nivel 0, todas las etapas del pipeline de ML, desde el procesamiento de datos hasta la implementación del modelo, se realizan manualmente, con actualizaciones poco frecuentes y monitoreo limitado. En el Nivel 1, se introduce la automatización en el entrenamiento y la entrega del modelo, permitiendo actualizaciones basadas en nuevos datos y reduciendo la intervención manual. Finalmente, en el Nivel 2, se alcanza una integración y entrega continua, donde cada cambio en los datos o código se refleja automáticamente en la infraestructura de producción, asegurando modelos siempre actualizados y alineados con las condiciones del entorno [6].

MLOps Pipeline

El pipeline de MLOps es un conjunto estructurado de procesos diseñados para gestionar el flujo de trabajo de un sistema de Machine Learning. Abarca desde la selección y preparación de datos hasta el entrenamiento, evaluación y despliegue de modelos, incorporando mecanismos de monitoreo continuo para garantizar su rendimiento. Dependiendo de su estructura, puede ser completamente automatizado para tareas específicas o estar compuesto por múltiples módulos coordinados para gestionar diferentes aspectos del sistema [6].

Componentes de MLOps

Model registry

El Model Registry es un repositorio centralizado diseñado para almacenar, gestionar y versionar modelos de ML. Su función principal es proporcionar un espacio único donde se registran los modelos entrenados, junto con sus metadatos asociados y diferentes versiones [6].

Orquestación de workflow

La orquestación de workflow es un proceso automatizado que gestiona, coordina y controla el flujo completo de las operaciones en un pipeline de aprendizaje automático [6].

Contenerización

La contenerización es una técnica de virtualización ligera que permite empaquetar un modelo de ML, junto con todas sus dependencias, bibliotecas y configuraciones, en una unidad autónoma y portable [6].

Serving

El Serving es el proceso de hosting y exposición de modelos de ML para que sus funciones sean accesibles a otras aplicaciones, típicamente mediante interfaces de programación de aplicaciones (APIs) [6]. Existen dos modalidades de serving:

1. Serving por lotes (Batch serving):

Los modelos se utilizan para procesar grandes volúmenes de datos en trabajos programados. Es ideal para análisis periódicos y procesamiento no urgente. Por ejemplo, generación de informes diarios, análisis estadísticos nocturnos.

2. Serving en tiempo real (Real-time serving):

Procesamiento inmediato de solicitudes individuales. Da Respuesta rápida y dinámica. Por ejemplo: sistemas de recomendación, detección de fraudes, diagnósticos médicos instantáneos.

Electroencefalografía (EEG)

La electroencefalografía (EEG) es una técnica utilizada para registrar la actividad eléctrica del cerebro mediante electrodos situados en el cuero cabelludo. Su principal fortaleza es su alta resolución temporal, que permite capturar cambios en la actividad neuronal en el rango de los milisegundos, lo que la convierte en una herramienta ideal para el estudio en tiempo real de los procesos cerebrales. EEG permite evaluar la sincronización neuronal y el acoplamiento funcional entre distintas áreas del cerebro [7].

En comparación con otras técnicas de neuroimagen, EEG presenta una menor resolución espacial, limitando su capacidad para localizar con precisión el origen de las señales eléctricas en el cerebro. Sin embargo, ninguna de las otras técnicas tiene la precisión temporal que ofrece EEG para analizar la actividad neuronal en distintas bandas de frecuencia [7]. Esta característica es fundamental para el estudio de la actividad cerebral tanto en estado de reposo como en respuesta a estímulos externos.

En el contexto de la enfermedad de Alzheimer (AD), el EEG cuantitativo (qEEG) y los potenciales relacionados con eventos (ERPs) han sido explorados como posibles biomarcadores en etapas tempranas de la enfermedad. Estudios han demostrado que las alteraciones en los ritmos de EEG pueden correlacionarse con el deterioro cognitivo y la progresión de la enfermedad [7].

Trastorno neurocognitivo mayor

El Trastorno neurocognitivo mayor (TNC-M), anteriormente conocido como demencia, es una condición clínica caracterizada por un deterioro significativo y progresivo en una o más funciones cognitivas. Representa un cambio notable respecto al nivel previo de funcionamiento del individuo y afecta dominios cognitivos como la atención compleja, funciones ejecutivas, aprendizaje y memoria, lenguaje, habilidades perceptuales-motoras y cognición social. Es importante destacar que el TNC-M forma parte de un espectro de disfunciones cognitivas y funcionales, que incluye también el Trastorno Neurocognitivo Leve (TNC-L). Mientras que el TNC-M corresponde a lo que anteriormente se denominaba demencia, el TNC-L representa un deterioro menos severo [8].

Un elemento distintivo del TNC-M es que las alteraciones cognitivas impactan en la autonomía del individuo para sus actividades cotidianas. Esto implica que la persona requiere, como mínimo, asistencia para realizar actividades instrumentales complejas de la vida diaria, como la gestión financiera o el cumplimiento de tratamientos médicos [8].

Plataformas existentes

En el campo de la neuroimagen, existen plataformas en línea que han logrado automatizar y democratizar el acceso a análisis complejos. Un ejemplo notable es volBrain [9], un servicio web dedicado al procesamiento de imágenes por resonancia magnética (IRM) cerebral. Esta plataforma permite a los investigadores subir sus datos de IRM y obtener, de manera completamente automática, la segmentación de diversas estructuras cerebrales y el cálculo de sus volúmenes. El valor de volBrain radica en su capacidad para ofrecer resultados estandarizados y reproducibles sin que el usuario final necesite configurar y ejecutar complejos pipelines de software a nivel local, eliminando así una barrera técnica significativa en la investigación clínica y neurocientífica.

A pesar de su gran utilidad, el enfoque de volBrain se centra en ser un servicio de análisis de datos para IRM, y no una plataforma para la gestión del ciclo de vida de modelos de Machine Learning. Mientras que volBrain ejecuta un conjunto de algoritmos predefinidos para entregar métricas volumétricas, no ofrece las capacidades completas de MLOps que son el núcleo de este proyecto.

CAPITULO II: DISEÑO DE INFRAESTRUCTURA DE DATOS

I. INTRODUCCIÓN

La infraestructura de datos constituye el fundamento arquitectónico sobre el cual se sostiene toda la plataforma MLOps desarrollada en este proyecto. Esta infraestructura fue concebida como el componente central que permite el almacenamiento, gestión y acceso eficiente a todas las métricas, metadatos y artefactos asociados a las señales EEG y los modelos de Machine Learning. Su diseño responde a la necesidad de mantener un registro detallado y estructurado de cada fase del ciclo de vida del modelo, desde la ingesta de datos hasta el despliegue en producción.

El diseño de esta infraestructura se fundamentó en principios de portabilidad, escalabilidad y facilidad de implementación. La portabilidad se logró mediante el uso de tecnologías de contenerización, específicamente Docker [13], que permite encapsular todos los componentes del sistema junto con sus dependencias en unidades portátiles y autocontenidas. Esta aproximación permite que la plataforma pueda ser desplegada de manera consistente en diferentes entornos, desde computadores personales (entornos locales) hasta servidores de producción, eliminando los problemas típicos de incompatibilidad de dependencias y configuraciones específicas del sistema [13].

La selección de herramientas de código abierto fue un criterio fundamental en el diseño, no solo por consideraciones económicas, sino por la flexibilidad y transparencia que ofrecen estas tecnologías. MLflow [10], PostgreSQL [11], Prefect[12] y Docker [13] representan tecnologías maduras, lo que garantiza estabilidad a largo plazo y facilita el mantenimiento del sistema.

II. METODOLOGÍA

A. DEFINICIÓN DE REQUISITOS DEL SISTEMA

El análisis de los requisitos del proyecto reveló la necesidad de una infraestructura capaz de manejar múltiples tipos de datos y metadatos de manera simultánea. Específicamente, el sistema debe ser capaz de almacenar y gestionar: métricas de rendimiento de modelos de Machine Learning, metadatos de experimentos y entrenamientos, artefactos de modelos entrenados, datos de predicciones realizadas, y metadatos de workflows de orquestación. Además, el sistema debía proporcionar capacidades de versionado de modelos y trazabilidad completa de experimentos.

La estructura general del proyecto se organizó siguiendo una arquitectura orientada a servicios, donde cada componente tiene responsabilidades claramente definidas. El directorio *backend/* contiene toda la lógica de la infraestructura de datos, incluyendo los archivos de configuración de Docker (*Dockerfile*, *Dockerfile.fastapi*, *Dockerfile.mlflow*), el archivo de orquestación (*docker-compose.yml*), y el código fuente organizado en módulos especializados (*src/flows/*, *src/services/*, *src/routers/*).

B. SELECCIÓN Y COMPARACIÓN DE HERRAMIENTAS

Para la gestión de experimentos y el registro de modelos, se evaluaron múltiples alternativas incluyendo MLflow, y Kubeflow. MLflow fue seleccionado como la solución principal debido a su naturaleza de código abierto, su arquitectura modular que permite personalización, y su amplia adopción [10].

La decisión de utilizar MLflow se fundamentó en varias ventajas técnicas críticas. Primero, su capacidad nativa para integrarse con múltiples frameworks de Machine Learning (scikit-learn, TensorFlow, PyTorch, etc.) sin requerir modificaciones significativas en el código existente. Segundo, su arquitectura flexible que permite el uso de diferentes backends para metadatos (SQLite, MySQL, PostgreSQL) y almacenes de artefactos (sistema de archivos local, S3, Azure Blob Storage). Tercero, su interfaz web integrada que facilita la visualización y comparación de experimentos sin necesidad de herramientas adicionales [10].

Las alternativas evaluadas presentaban limitaciones significativas para los requisitos específicos del proyecto. Kubeflow, si bien es completamente de código abierto, presenta una complejidad de implementación desproporcionada para el alcance del proyecto y requiere conocimientos especializados en Kubernetes.

C. IMPLEMENTACIÓN DE MLFLOW COMO SISTEMA DE TRACKING

La implementación de MLflow se configuró utilizando PostgreSQL [11] como backend store para los metadatos y un sistema de archivos local para el almacenamiento de artefactos. Esta configuración se define en el archivo `entrypoint-mlflow.sh`, donde se especifica la URI de conexión a la base de datos PostgreSQL y se configuran los directorios para el almacenamiento de artefactos. El servidor MLflow se ejecuta con las opciones `--serve-artifacts` habilitada, permitiendo el acceso directo a los artefactos de modelo a través de la interfaz web.

MLflow se integra estrechamente con el pipeline automatizado de entrenamiento implementado en `src/flows/training_flow.py`. Durante cada ejecución del pipeline, MLflow registra automáticamente parámetros de entrenamiento, métricas de rendimiento, y artefactos del modelo. El sistema de tracking captura información detallada incluyendo hiperparámetros utilizados, métricas de validación cruzada, matrices de confusión, y curvas de aprendizaje. Esta información se almacena de manera estructurada, permitiendo la comparación sistemática entre diferentes experimentos.

El Model Registry de MLflow facilita la gestión del ciclo de vida de los modelos mediante un sistema de versionado automático y etapas de promoción. Cada modelo entrenado se registra con un número de versión incremental, y puede ser promovido a producción a través de los aliases de MLFlow según su rendimiento y a selección del usuario.

D.CONTENERIZACIÓN CON DOCKER

La estrategia de contenerización implementada utiliza Docker [13] para encapsular cada servicio del sistema en contenedores independientes. Se desarrollaron tres Dockerfiles especializados: *Dockerfile.mlflow* para el servidor MLflow, *Dockerfile.fastapi* para la API backend, y *Dockerfile* para el servidor Prefect. Cada Dockerfile está optimizado para su función específica, incluyendo solo las dependencias necesarias y utilizando imágenes base ligeras de Python 3.9.

La arquitectura de contenedores ofrece múltiples ventajas técnicas. La consistencia del entorno de ejecución elimina los problemas de "funciona en mi máquina", ya que cada contenedor incluye todas las dependencias necesarias en versiones específicas. El aislamiento de procesos mejora la seguridad y estabilidad del sistema, evitando conflictos entre dependencias de diferentes servicios [13]. La escalabilidad horizontal se facilita mediante la capacidad de ejecutar múltiples instancias de contenedores según la demanda.

El archivo `docker-compose.yml` define la orquestación completa de los contenedores, especificando las dependencias entre servicios, las redes de comunicación, y los volúmenes de datos persistentes. La configuración incluye health checks para PostgreSQL, lo que garantiza que los servicios dependientes solo inicien cuando la base de datos esté completamente operativa. Los volúmenes nombrados (*postgres_data*) aseguran la persistencia de datos incluso cuando los contenedores son reiniciados o recreados.

E. IMPLEMENTACIÓN DE POSTGRESQL COMO SISTEMA DE ALMACENAMIENTO

PostgreSQL fue seleccionado como el sistema de gestión de base de datos principal después de evaluar alternativas de tipo SQL, y soluciones NoSQL como MongoDB. La decisión se fundamentó en las capacidades avanzadas de PostgreSQL para manejar consultas complejas, su soporte nativo para tipos de datos JSON, y su robustez comprobada en entornos de producción. Además, PostgreSQL ofrece excelente compatibilidad con MLflow y Prefect, los dos componentes principales que requieren almacenamiento de metadatos [10], [12].

La base de datos se estructura utilizando múltiples esquemas para organizar lógicamente la información: `mlflow` para metadatos de experimentos y modelos, `predictions` para el registro de predicciones realizadas, `training_data` para datasets de entrenamiento, y `prefect` para metadatos de workflows. Esta separación lógica mejora la organización de los datos y facilita la implementación de políticas de acceso y backup específicas para cada tipo de información.

La integración de PostgreSQL con los demás servicios se gestiona mediante variables de entorno que especifican las credenciales de conexión y los parámetros de configuración. Esta aproximación permite la modificación de la configuración sin necesidad de reconstruir los contenedores, facilitando la adaptación del sistema a diferentes entornos de despliegue.

III. RESULTADOS Y DISCUSIONES

La implementación de la infraestructura de datos resultó en un sistema funcional compuesto por cinco servicios contenerizados interconectados: PostgreSQL como sistema de base de datos, MLflow como plataforma de tracking y registry, Prefect como orquestador de workflows, FastAPI como backend de servicios, y la aplicación frontend en React. Un diagrama que ilustra el sistema de datos resultante y su flujo de información se muestra en la figura 1.

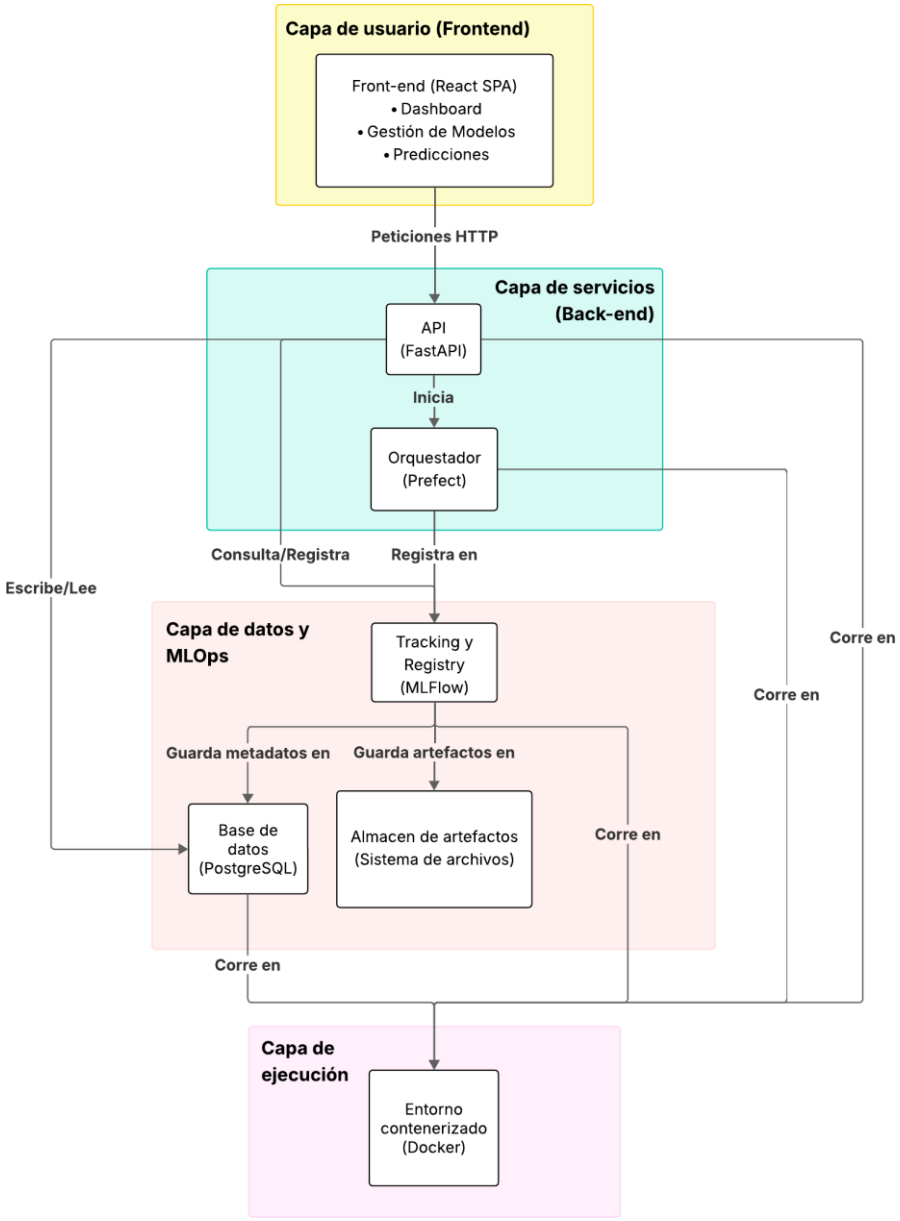


Figura 1. Infraestructura de datos y flujo de información resultante.

La red interna *mlops-network* permite la comunicación entre servicios utilizando nombres de servicio como *hostnames*, lo que simplifica la configuración y mejora la seguridad al aislar el

tráfico interno del host. Las pruebas de conectividad confirman que cada servicio puede acceder a sus dependencias sin exposición innecesaria de puertos al exterior. Los volúmenes persistentes aseguran que los datos de PostgreSQL (*postgres_data*) y los artefactos de MLflow (*./backend/artifacts*, *./backend/mlruns*) se mantienen entre reinicios del sistema.

La integración de MLflow con PostgreSQL como backend store demostró ser efectiva para el almacenamiento y recuperación de metadatos de experimentos. El sistema registra automáticamente más de 15 métricas diferentes por cada entrenamiento, incluyendo accuracy, precision, recall, f1-score, y métricas de validación cruzada. El Model Registry con MLFlow permite el versionado de modelos con metadatos completos, incluyendo parámetros de entrenamiento, métricas de rendimiento, y referencias a datasets utilizados.

Los objetivos de portabilidad se cumplieron exitosamente mediante la containerización completa del sistema. Las pruebas de despliegue en diferentes entornos (macOS y Ubuntu) confirmaron consistencia completa del comportamiento del sistema. El comando único docker-compose up -d es suficiente para inicializar toda la infraestructura en cualquier sistema que soporte Docker, reduciendo significativamente la barrera de cada despliegue. La configuración mediante variables de entorno permite la personalización del sistema sin modificación de código.

A pesar de los resultados positivos, se identificaron algunas limitaciones en la implementación actual. El almacenamiento de artefactos en sistema de archivos local, aunque simple y efectivo para el alcance del proyecto, presenta limitaciones de escalabilidad para implementaciones más grandes. Para entornos de producción a mayor escala, sería recomendable migrar a sistemas de almacenamiento distribuido en la nube como AWS S3 o Azure Blob Storage.

CAPÍTULO III: IMPLEMENTACIÓN DE UN PIPELINE AUTOMATIZADO DE MACHINE LEARNING

I. INTRODUCCIÓN

La transición desde un proceso de experimentación manual hacia un sistema automatizado de entrenamiento de modelos representa el núcleo de la madurez en MLOps [6] y constituye el pilar de este proyecto. En el trabajo de investigación precedente, el ciclo de vida del modelo de Machine Learning, desde la carga de los datos hasta la evaluación del rendimiento, se ejecutaba a través de scripts que requerían intervención manual significativa en cada etapa. Este enfoque, aunque efectivo para la fase de investigación inicial, presenta limitaciones en términos de reproducibilidad, escalabilidad y mantenibilidad. Como señala [14], estos sistemas manuales a menudo se convierten en una compleja red de código de acoplamiento (*glue code*) y procesos frágiles, donde la mayor parte del esfuerzo se dedica al "*plumbing*" en lugar de al modelado. Cada nuevo entrenamiento o ajuste de hiperparámetros implicaba modificaciones manuales en el código, un proceso propenso a errores y difícil de documentar de forma sistemática. Esto es especialmente complejo cuando se trabaja con datos a gran escala, ya que uno de los retos más comunes es lidiar con el enriquecimiento continuo y progresivo de los datos de entrenamiento [15].

Para superar estas limitaciones y alcanzar un nivel de madurez mayor, donde el pipeline de entrenamiento del modelo es automatizado, se propuso el desarrollo de un flujo de trabajo estructurado. El objetivo principal de este capítulo es detallar la implementación de un pipeline automatizado de Machine Learning que integra la ingesta de datos, el preprocesamiento, el entrenamiento continuo y la actualización del modelo basado en nuevos datos. Este pipeline no solo automatiza las tareas repetitivas, sino que también introduce un sistema robusto de seguimiento y versionado de cada componente del proceso, entregando una trazabilidad completa desde los datos iniciales hasta el modelo en producción.

El primer paso para lograr esta automatización fue un análisis exhaustivo del código del modelo existente para deconstruir su lógica en componentes modulares. Este proceso de refactorización fue muy importante para transformar el script de investigación en un conjunto de pasos lógicos y reutilizables que pudieran ser orquestados. La meta no era solo replicar la funcionalidad del modelo original, sino rediseñar el proceso, enfocándose en la modularidad y la reutilización. A través de este enfoque se establece un marco de trabajo genérico para futuros modelos dentro del dominio de las neurociencias y los modelos de clasificación.

II. METODOLOGÍA

A. ANÁLISIS Y REFACTORIZACIÓN DEL MODELO EXISTENTE

El punto de partida fue un análisis del código del modelo de Machine Learning desarrollado en la investigación previa. Este análisis implicó una inmersión en la lógica del modelo, comenzando por la configuración del entorno de desarrollo, la instalación de todas las dependencias de software requeridas, y la ejecución controlada del script para comprender su flujo de operaciones y sus resultados. Durante este proceso se evaluaron las métricas, se visualizaron los artefactos generados (como gráficos y matrices de confusión) y se trazó el flujo de datos desde la entrada hasta la salida.

Una de las observaciones clave durante esta fase fue la identificación de múltiples versiones y variantes del código de entrenamiento dentro del mismo repositorio. Sin embargo, para un sistema de producción, esta práctica representa un riesgo para la mantenibilidad y la reproducibilidad. El script original era un claro ejemplo de los problemas de "código de acoplamiento" y "selvas de pipelines" (pipeline jungles) descritos en [15]. La lógica para la carga, preprocesamiento y entrenamiento estaba intrínsecamente mezclada, haciendo que cualquier modificación fuera riesgosa y difícil de probar. Esta observación reforzó la necesidad de la plataforma MLOps, cuyo objetivo es centralizar el desarrollo en un único pipeline versionado y parametrizable, eliminando la ambigüedad y asegurando que cada modelo entrenado provenga de una fuente de verdad única y controlada.

Consecuentemente, se procedió a la refactorización del script original (En la figura 2 se observa un diagrama de flujo con la lógica del código original). La lógica se descompuso en tres funciones modulares y cohesivas, cada una representando una etapa lógica del ciclo de vida del ML. Estas funciones se consolidaron en un nuevo script, training_script.py, diseñado para la orquestación (ver figura 3):

load_data(): Encapsula toda la lógica de carga de datos, responsable de leer los conjuntos de datos desde su fuente original.

process_data(): Contiene todos los pasos de preprocesamiento y extracción de características, preparando los datos para el entrenamiento.

train_model(): Integra el entrenamiento del clasificador, la validación cruzada, la evaluación de métricas y el registro del modelo final.

Esta refactorización transformó un proceso secuencial y rígido en un conjunto de componentes modulares que pueden ser invocados, probados y mantenidos de forma independiente, sentando las bases para su integración en un pipeline de orquestación.

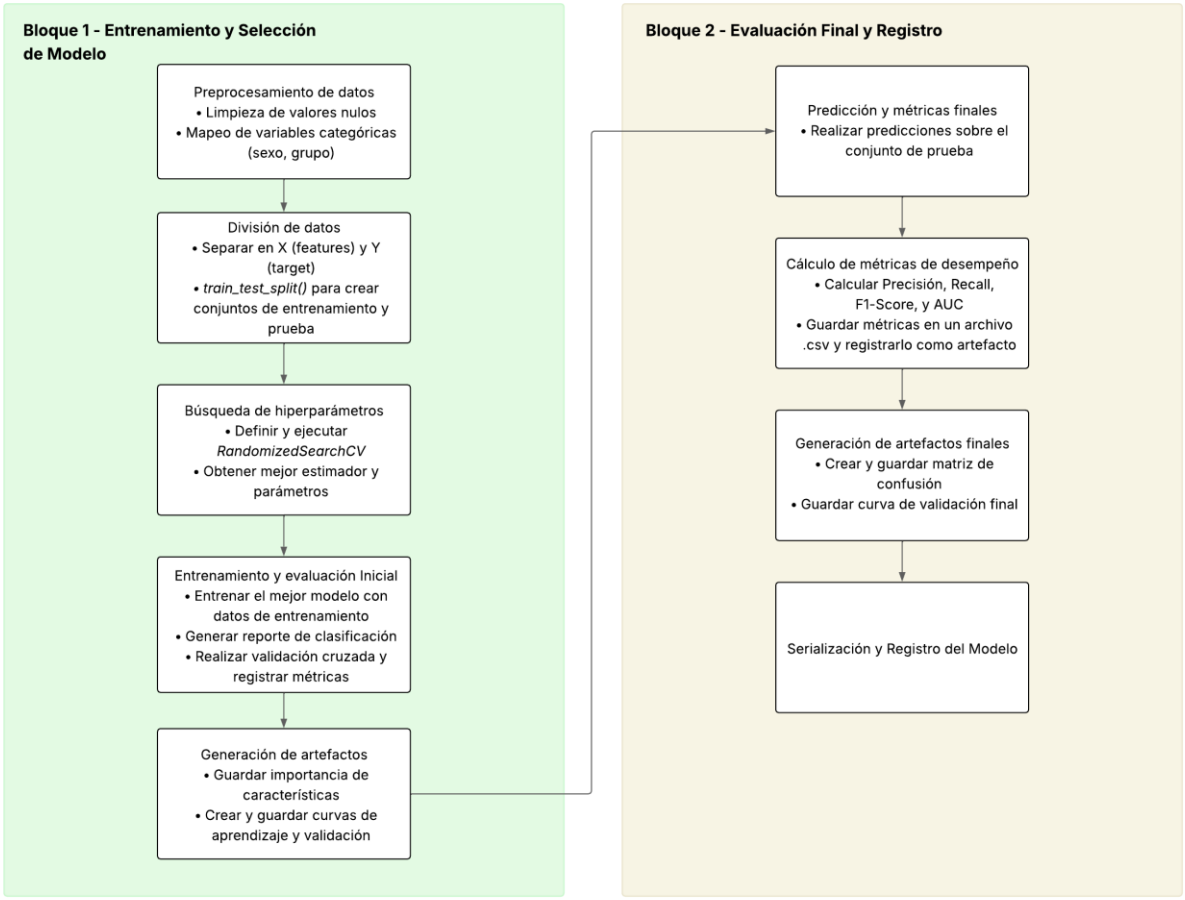


Figura 2. Script original del entrenamiento del modelo.

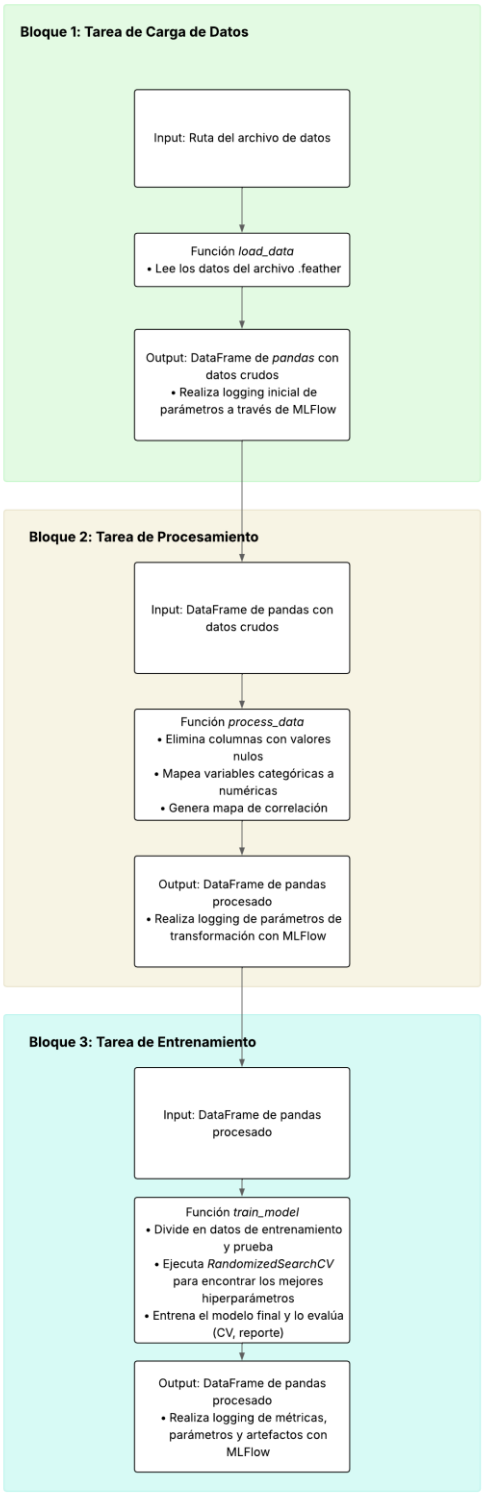


Figura 3. Script del entrenamiento del modelo modificado para la nueva implementación.

B. DISEÑO DE UN PIPELINE GENÉRICO Y REUTILIZABLE

Una vez modularizado el código del modelo, el siguiente paso fue diseñar un sistema de orquestación. En lugar de construir un pipeline específico para este modelo en particular, se tomó la decisión de crear un pipeline de entrenamiento genérico y reutilizable. El objetivo era desarrollar un "meta-pipeline" que pudiera orquestar cualquier modelo de Machine Learning que siguiera la estructura de pasos (cargar, procesar, entrenar) definida en la fase de refactorización. Este enfoque aborda directamente el problema de la "deuda de abstracción" (a*bstraction debt*) [14], donde la falta de interfaces claras y reutilizables dificulta la evolución de los sistemas de ML. La infraestructura desarrollada puede ser fácilmente adaptada para futuros proyectos de investigación sin necesidad de reescribir la lógica de orquestación y seguimiento.

El corazón de este diseño es la función *ml_pipeline_flow*, implementada en el archivo *backend/src/flows/training_flow.py*. Esta función actúa como el orquestador principal, definiendo la secuencia de tareas y manejando la integración con las herramientas de MLOps. Para lograr su naturaleza genérica, ml_pipeline_flow no contiene lógica de negocio específica del modelo; en su lugar, recibe como parámetros las propias funciones que ejecutan cada paso, así como los argumentos necesarios para cada una.

Esta arquitectura, permite que el pipeline sea agnóstico al modelo. La estructura de sus parámetros de entrada (*load_data_func, process_data_func, train_model_func, junto con sus respectivos _args y _kwargs*) permite inyectar las de funciones de carga, procesamiento y entrenamiento, configurando dinámicamente el comportamiento del pipeline para cada caso de uso.

C. IMPLEMENTACIÓN DEL FLOW CON PREFECT Y MLFLOW

La implementación técnica del pipeline genérico se llevó a cabo utilizando Prefect para la orquestación del flujo de trabajo y MLflow para el seguimiento de experimentos y el registro de modelos. Prefect permite definir flujos de trabajo de Python como "flows" y las funciones individuales como "tasks", gestionando automáticamente las dependencias, las reintentos y el estado de cada ejecución. MLflow, por su parte, se integra dentro de cada tarea para registrar de manera sistemática toda la información relevante del proceso.

La función *ml_pipeline_flow* se define como un flow de Prefect utilizando el decorador @flow. Dentro de este flow, se definen tres tareas principales (*load_data_task, process_data_task, train_model_task*), cada una decorada con @task. Estos decoradores permiten a Prefect visualizar, gestionar y ejecutar cada paso de forma independiente. A continuación, se muestra la estructura principal del pipeline.

```
@task(name="Load Data")
def load_data_task(load_data_func: SkipValidation[Callable], parent_run_id: str, *args,
**kwargs):
    # ... Lógica para ejecutar la función de carga y registrar en MLflow ...

@task(name="Process Data")
def process_data_task(process_data_func: SkipValidation[Callable], data: pd.DataFrame,
parent_run_id: str, *args, **kwargs):
    # ... Lógica para ejecutar el preprocesamiento y registrar en MLflow ...

@task(name="Train Model")
def train_model_task(train_model_func: SkipValidation[Callable], data: pd.DataFrame,
parent_run_id: str, *args, **kwargs):
    # ... Lógica para ejecutar el entrenamiento, registrar métricas, artefactos y el modelo en
MLflow ...

@flow(name="ML Training Pipeline")
def ml_pipeline_flow(
    load_data_func: SkipValidation[Callable],
    load_data_args: tuple,
    load_data_kwargs: dict,
    process_data_func: SkipValidation[Callable],
    process_data_args: tuple,
    process_data_kwargs: dict,
    train_model_func: SkipValidation[Callable],
    train_model_args: tuple,
    train_model_kwargs: dict
    ):
    logger = get_run_logger()
    mlflow.set_experiment("EEG_Cognitive_Decline_Classifier")

    with mlflow.start_run() as parent_run:
```

```
        parent_run_id = parent_run.info.run_id
        logger.info(f"MLflow Parent Run ID: {parent_run_id}")

        # Ejecutar las tareas del pipeline
        raw_data = load_data_task(load_data_func, parent_run_id, *load_data_args,
**load_data_kwargs)
        processed_data = process_data_task(process_data_func, raw_data, parent_run_id,
*process_data_args, **process_data_kwargs)
        train_model_task(train_model_func, processed_data, parent_run_id, *train_model_args,
**train_model_kwargs)
```

Código 1. Implementación de pipeline.

Dentro de la tarea *train_model_task*, se realiza una integración más profunda con MLflow. Se inicia un "run" anidado que permite asociar todos los registros a ese paso específico del pipeline. Durante la ejecución, se registran sistemáticamente:

Parámetros (mlflow.log_params): Todos los hiperparámetros del modelo, como el número de estimadores, la profundidad máxima de los árboles, o los criterios de división.

Métricas (mlflow.log_metrics): Los resultados cuantitativos de la evaluación del modelo, tales como la exactitud (accuracy), precisión, recall, F1-score, y los resultados de la validación cruzada (media y desviación estándar de la exactitud).

Artefactos (mlflow.log_artifact): Cualquier archivo generado durante el entrenamiento que sea relevante para el análisis, como la matriz de confusión, el gráfico de importancia de características, y las curvas de aprendizaje.

Modelo (mlflow.sklearn.log_model): El objeto del modelo entrenado se serializa y guarda en un formato estándar. Crucialmente, junto con el modelo se guarda un ejemplo de entrada y el entorno de Conda (conda.yaml) con todas las dependencias y sus versiones exactas, garantizando la reproducibilidad total del entorno de predicción.

Finalmente, el modelo registrado se versiona en el Model Registry de MLflow. Esto crea un historial inmutable de todos los modelos entrenados, permitiendo comparar su rendimiento y gestionar su ciclo de vida desde la experimentación hasta el despliegue en producción.

D. ORQUESTACIÓN Y EJECUCIÓN DEL PIPELINE

Con el pipeline genérico definido y las funciones del modelo refactorizadas, el último paso metodológico fue conectar ambos componentes para crear una ejecución concreta. Esta orquestación se realiza en la función *run_ml_training_pipeline*, ubicada en el archivo *backend/src/services/ml_model_service.py*. Esta función actúa como el punto de entrada para iniciar un nuevo ciclo de entrenamiento.

La función *run_ml_training_pipeline* es responsable de:

Importar las funciones específicas del modelo: Importa *load_data, process_data,* y *train_model* desde *training_script.py*.

Configurar los argumentos: Define los parámetros y configuraciones necesarios para cada una de las funciones del modelo.

Invocar el flow de Prefect: Llama a *ml_pipeline_flow.with_options(name=...)* y le pasa las funciones y sus argumentos, iniciando así la ejecución del pipeline de forma asincrónica.

Cuando un usuario o un sistema externo realiza una llamada a la API para reentrenar el modelo, es esta función la que se ejecuta, actuando como un puente entre la capa de servicios de la aplicación y el sistema de orquestación de MLOps. Prefect recibe la solicitud, programa la ejecución del flow, y comienza a ejecutar cada tarea en la secuencia definida, mientras MLflow registra cada detalle del proceso en la base de datos de PostgreSQL y en el sistema de archivos de artefactos. Este mecanismo completa el ciclo de automatización, permitiendo el reentrenamiento del modelo con un solo clic o una llamada a la API.

E. INTERFACES GRÁFICAS DE ML FLOW Y PREFECT

Las interfaces gráficas de Prefect y MLflow que se describen a continuación son utilidades proporcionadas por las propias librerías; ambas se inician automáticamente cuando sus respectivos contenedores Docker (*prefect* y *mlflow*) se levantan mediante *docker-compose*. La aplicación web desarrollada (que se menciona en el siguiente capítulo) incluye un botón de acceso directo que abre cada interfaz en una nueva pestaña del navegador, de modo que el usuario puede supervisar la ejecución de los flujos y el seguimiento de los experimentos sin salir de la plataforma principal. Aunque son un apoyo fundamental para la trazabilidad y el monitoreo, estas herramientas no representan el foco principal de la parte técnica de este trabajo.

Cada vez que ocurre una ejecución del flow, esta se puede acceder a través de las interfaces gráficas de Prefect y ML Flow. En la figura 4 se observa la vista general de Prefect / Flow Runs, donde se listan todas las ejecuciones del pipeline con su identificador, estado (Completed, Failed o In Progress), duración y hora de inicio; arriba se incluye un gráfico que sintetiza la distribución temporal de las ejecuciones, facilitando la detección de picos de actividad o patrones. Como muestra la figura 5, al ingresar al detalle de una ejecución concreta, Prefect despliega un diagrama interactivo que ilustra la secuencia y el resultado (Success / Fail) de cada task y su duración; debajo del gráfico se presentan los logs cronológicos.

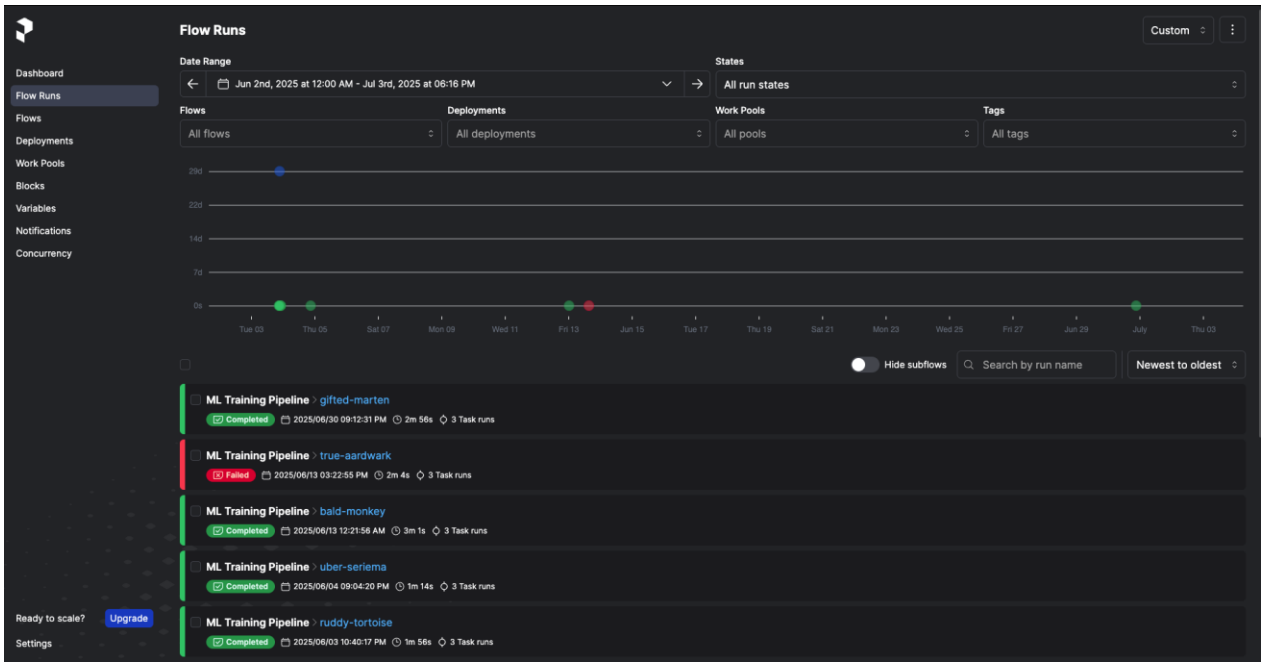


Figura 4. Página Flow Runs de la interfaz gráfica de Prefect.

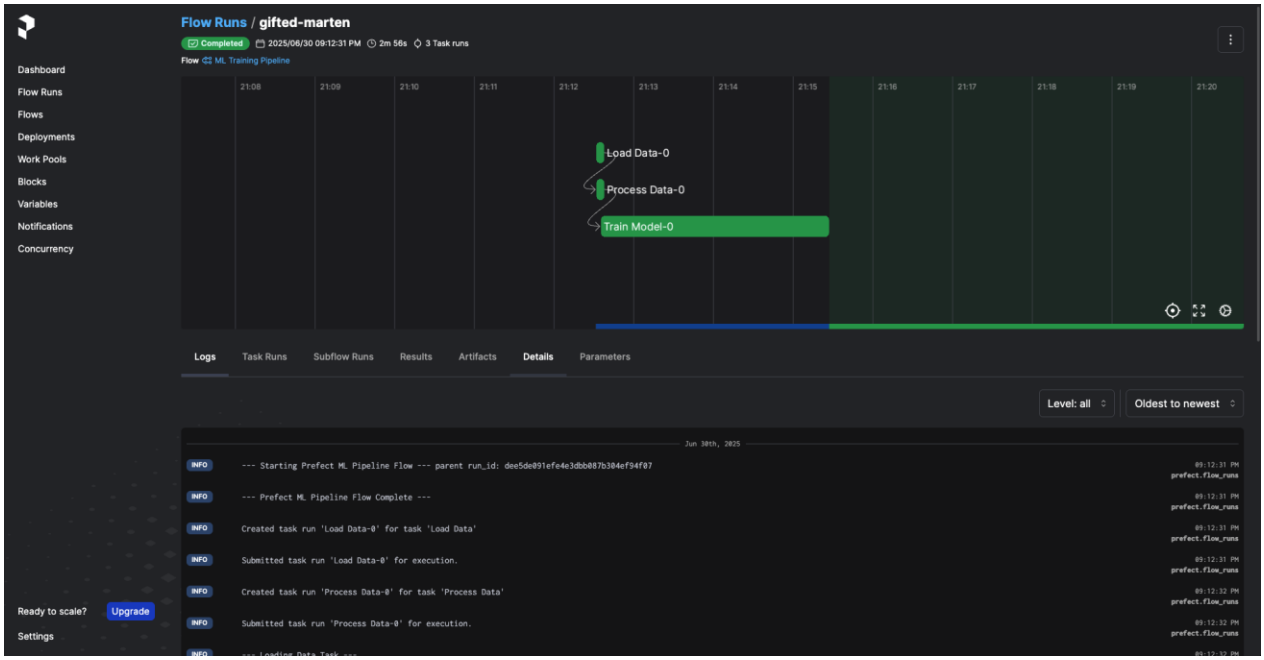


Figura 5. Página de cada Flow Run de la interfaz gráfica de Prefect.

De forma análoga, la interfaz de MLflow —figura 6— presenta una tabla consolidada de todos los experimentos y sus runs (incluyendo los sub-runs de cada task), mostrando métricas clave (accuracy, F1, etc.) y parámetros registrados, lo que permite ordenar y comparar resultados de manera inmediata. La figura 7 profundiza en el detalle de una sub-run (Train Model task), donde se visualizan las métricas finales, los hiperparámetros utilizados, los tags de contexto y los artefactos generados (modelo serializado, matriz de confusión, curvas de aprendizaje).

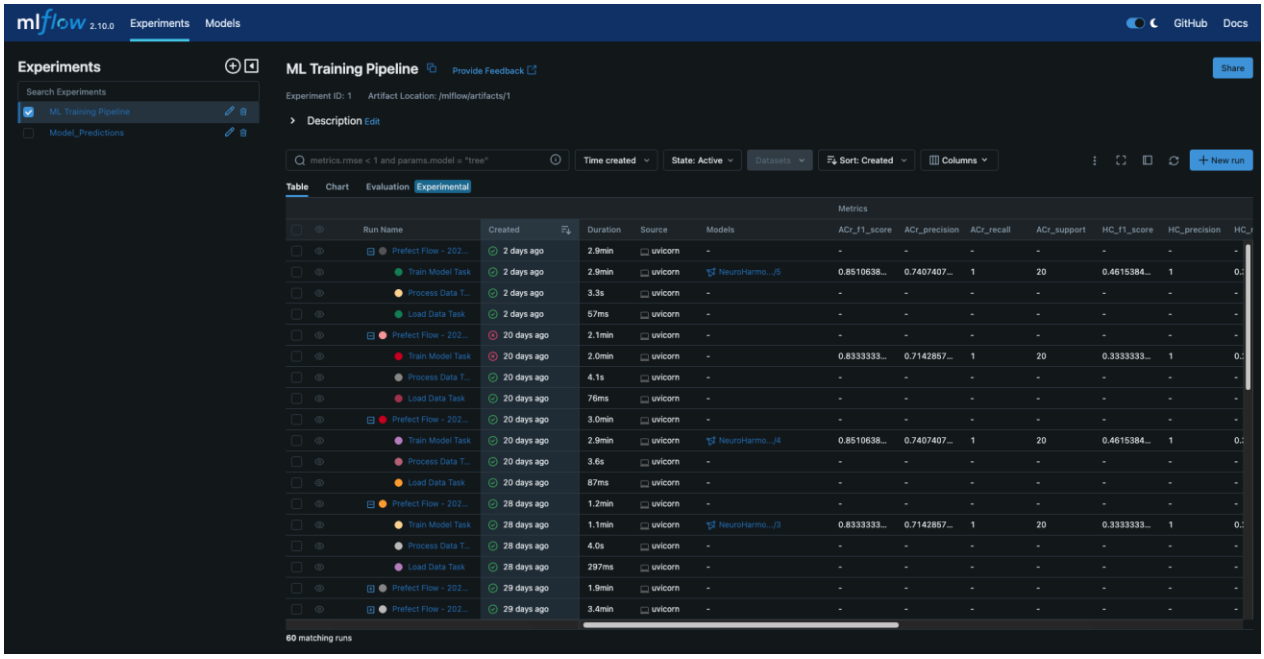


Figura 6. Página de experimentos de la interfaz gráfica de ML FLow.

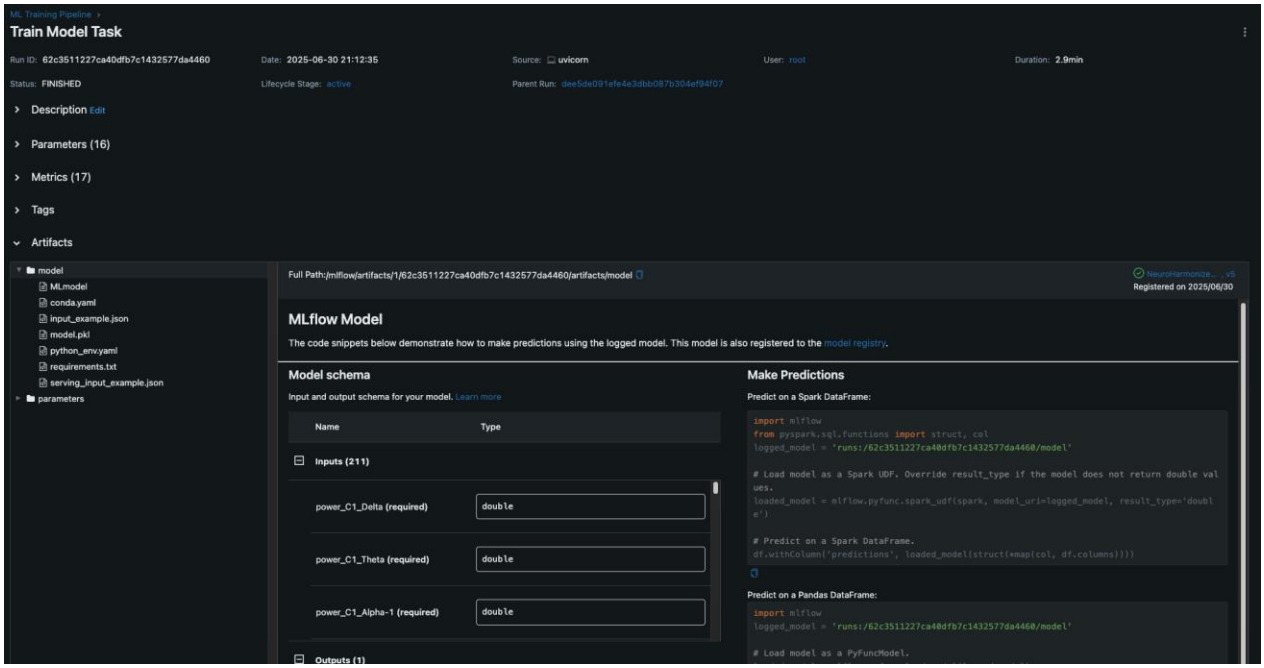


Figura 7. Página de detalles del Sub-Task de Train Model de la interfaz gráfica de ML FLow.

III. RESULTADOS Y DISCUSIONES

La implementación del pipeline automatizado proporcionó un marco reproducible para las pruebas de entrenamiento realizadas durante el desarrollo del proyecto. En lugar de depender de scripts monolíticos, todas las etapas (carga de datos, preprocesamiento, entrenamiento y registro) quedan encapsuladas en tareas independientes orquestadas por Prefect y supervisadas por MLflow, lo que facilita documentar cada decisión técnica y su efecto en el rendimiento del modelo. Este sistema quedó documentado en la figura 8, donde se observa el flujo de información y ejecución del pipeline automático para el entrenamiento del modelo.

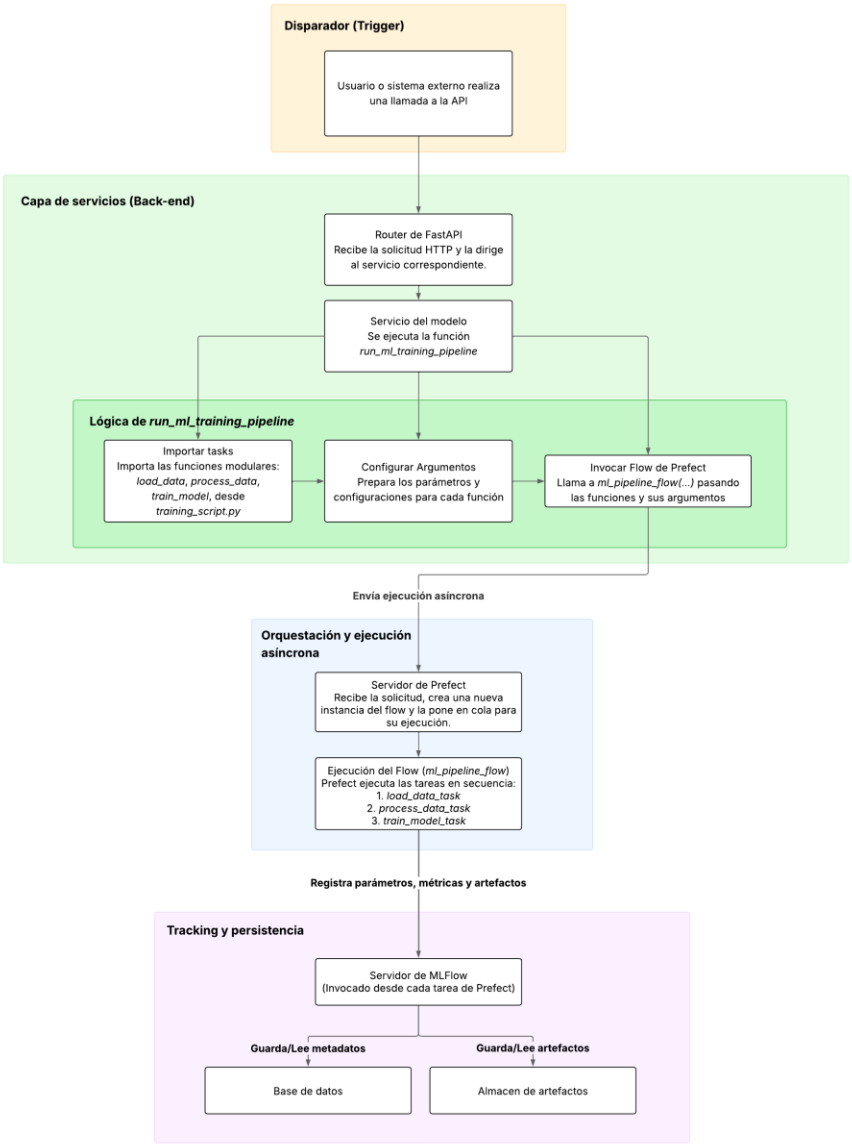


Figura 8. Diagrama de flujo de ejecución del pipeline de entrenamiento.

Desde el punto de vista de la trazabilidad, la combinación Prefect–MLflow resultó determinante. Cada ejecución queda asociada automáticamente al entorno de dependencias (archivo conda.yaml)

y a los hiperparámetros declarados, eliminando ambigüedades sobre la configuración exacta utilizada. La modularidad introducida en training_script.py permite, además, ensayar distintas estrategias de preprocesamiento o variaciones de hiperparámetros sin duplicar código ni ramificar repositorios, favoreciendo una experimentación controlada y sistemática.

La visibilidad que aportan las interfaces gráficas se demostró especialmente valiosa durante el análisis de resultados. El diagrama de Prefect (figura 4) permitió identificar con claridad qué tareas consumen mayor tiempo dentro del flujo, orientando los esfuerzos de optimización hacia la fase de procesamiento de datos. La vista de comparación de runs en MLflow (figura 5) reveló tendencias entre configuraciones de hiperparámetros y métricas finales, convirtiéndose en una guía práctica para seleccionar los candidatos a modelo principal.

CAPÍTULO IV: DESARROLLO DE LA INTERFAZ DE APLICACIÓN Y SERVICIOS

I. INTRODUCCIÓN

Los sistemas de datos y los pipelines de entrenamiento automatizado, detallados en los capítulos II y III respectivamente, constituyen la maquinaria subyacente de esta plataforma de MLOps. Sin embargo, para que esta infraestructura sea verdaderamente útil y accesible para los usuarios finales, es necesario disponer de una capa de abstracción que simplifique la interacción y centralice las operaciones. Este capítulo aborda el diseño y la construcción de dicha capa, hecha a través de un servicio de backend y una interfaz de usuario frontend, componentes que en conjunto entregan una plataforma integral.

El objetivo principal de este desarrollo es unificar las capacidades de la base de datos PostgreSQL, el sistema de tracking MLflow y el orquestador Prefect bajo una única API (Interfaz de Programación de Aplicaciones) y una interfaz gráfica de usuario (GUI). La creación de un backend a medida permite exponer las funcionalidades complejas de la infraestructura MLOps a través de puntos de acceso (endpoints) estandarizados. Este backend fue diseñado no solo para servir como un conducto de datos, sino como un centro de control, orquestando las interacciones entre los distintos microservicios y proveyendo al frontend la información procesada y lista para su visualización.

El diseño de la arquitectura frontend-backend se abordó con un enfoque centrado en el usuario. Se identificaron dos dominios funcionales principales que encapsulan las necesidades del proyecto: la gestión de modelos y la realización de predicciones. El primero abarca todo el ciclo de vida del modelo, desde la iniciación de un nuevo entrenamiento y la visualización de sus métricas, hasta la comparación entre versiones y la promoción de un candidato a producción. El segundo se enfoca en la capacidad de la plataforma para realizar inferencias en tiempo real utilizando el modelo productivo sobre nuevas señales EEG. Si bien estas funcionalidades se entrelazan en la lógica del backend, se presentan como áreas de trabajo separadas y bien definidas en la interfaz de usuario para garantizar una experiencia clara.

II. METODOLOGÍA

A. DISEÑO E IMPLEMENTACIÓN DEL BACKEND DE SERVICIOS

La construcción del backend se fundamentó en la creación de una API RESTful [16], que actúa como la espina dorsal de la plataforma. Esta arquitectura fue seleccionada por su naturaleza *stateless*, su escalabilidad y el uso de estándares HTTP universales, facilitando una integración sencilla y desacoplada con cualquier tipo de cliente, en este caso, la aplicación frontend desarrollada con React.

Selección del Framework y Definición de Arquitectura

Para la implementación de la API se eligió el framework FastAPI [17]. Esta decisión se basó en tres ventajas técnicas clave que se alinean con los objetivos del proyecto. Primero, su alto rendimiento, derivado de su naturaleza asíncrona sobre ASGI (*Asynchronous Server Gateway Interface*). Segundo, su integración nativa con Pydantic [18] para la validación, serialización y deserialización de datos. El uso de los esquemas definidos en src/schemas/model_schemas.py permite declarar los modelos de datos de forma explícita, asegurando la integridad de la información que fluye entre el frontend y el backend y reduciendo los errores de formato. Finalmente, FastAPI genera documentación interactiva de la API de forma automática, una característica muy útil durante el desarrollo.

La estructura del código del backend se organizó siguiendo el principio de separación de responsabilidades para facilitar su mantenimiento y escalabilidad. El archivo src/main.py actúa como punto de entrada de la aplicación, mientras que la lógica de la API se divide en tres capas principales:

Capa de Ruteo (*src/routers/*): Define los endpoints de la API. El archivo models_router.py contiene todas las rutas relacionadas con la gestión de modelos y predicciones.

Capa de Servicios (*src/services/*): Encapsula toda la lógica de negocio. El módulo ml_model_service.py interactúa directamente con MLflow, Prefect y la base de datos, desacoplando la lógica de la API de las implementaciones subyacentes.

Capa de Esquemas (*src/schemas/*): Contiene los modelos Pydantic que definen los contratos de datos para las solicitudes y respuestas de la API.

Implementación de Funcionalidades Clave

El backend expone un conjunto de endpoints diseñados para dar soporte a las dos funcionalidades principales de la plataforma: la gestión de modelos y la inferencia.

Gestión del Ciclo de Vida del Modelo: Esta funcionalidad permite al usuario interactuar con el pipeline de entrenamiento y el registro de modelos.

El endpoint POST */api/models/train* es el punto de entrada para iniciar un nuevo ciclo de entrenamiento. Al ser invocado, este llama a la función run_ml_training_pipeline en la capa de servicios, la cual a su vez despliega el flow de Prefect descrito en el Capítulo III.

Para el seguimiento y la gestión, se implementaron rutas como GET */api/models/available* para listar todos los modelos registrados, GET */api/models/{model_name}/versions* para obtener el historial de versiones de un modelo específico, y GET */api/models/{model_name}/training-history* para consultar los resultados de entrenamientos satisfactorios que fueron registrados como modelos.

Un endpoint crítico es POST */api/models/{model_name}/promote/{version}*, que permite al usuario promover una versión específica de un modelo al alias de "production" en el registro de MLflow. Esta acción es la que determina qué versión del modelo será utilizada para las predicciones.

Servicio de Predicción en Tiempo Real: Esta área funcional se centra en exponer el modelo productivo para realizar inferencias.

El endpoint POST */api/models/{model_name}/predict* recibe nuevos datos de señales EEG en formato JSON. Internamente, carga la versión del modelo marcada como "production" desde MLflow y devuelve la clasificación resultante.

Para facilitar la integración con el frontend, el endpoint GET */api/models/{model_name}/production-input-example* provee un ejemplo del formato de datos que el modelo espera recibir.

Toda predicción realizada es registrada, y el historial puede ser consultado a través de GET */api/models/predictions/history*.

Monitoreo y Agregación de Datos: Para nutrir la interfaz de usuario con información consolidada, se desarrollaron endpoints de monitorización.

La ruta GET */api/models/dashboard* ofrece un resumen de alto nivel del estado de la plataforma, agregando datos como el número total de modelos, la cantidad de predicciones recientes y los entrenamientos del último día.

GET */api/health/services* verifica el estado de salud de los servicios dependientes (MLflow y Prefect), proporcionando un diagnóstico rápido del sistema que puede ser visualizado en el frontend.

B. DISEÑO E IMPLEMENTACIÓN DEL FRONTEND

Mientras que el backend expone las capacidades del sistema a través de una API, el frontend las traduce en una experiencia visual e intuitiva, permitiendo a los usuarios gestionar el ciclo de vida de los modelos y obtener predicciones sin necesidad de interactuar directamente con la infraestructura interna.

La aplicación frontend se diseñó como una Single-Page Application (SPA) utilizando un conjunto de tecnologías modernas y robustas del ecosistema JavaScript. Se eligió React [19] como la biblioteca principal para la construcción de la interfaz por su arquitectura basada en componentes, que promueve la reutilización de código y facilita la gestión de vistas complejas. Para la capa de diseño y componentes UI, se optó por Ant Design [20], una librería de componentes que proporciona un conjunto extenso de elementos pre-construidos (tablas, formularios, modales, etc.), que dan una apariencia profesional y consistente en toda la aplicación, a la vez que acelera significativamente el tiempo de desarrollo. La gestión del estado global de la aplicación se centralizó mediante Redux, utilizando Redux Toolkit [21] como estándar de implementación. Esta elección fue fundamental para manejar de manera predecible y eficiente el estado de la aplicación, incluyendo los estados de carga, los errores de la API y los datos obtenidos del backend, que son compartidos entre múltiples componentes.

La estructura del código fuente, ubicada en el directorio frontend/src, sigue una organización modular por funcionalidad:

pages/: Contiene los componentes de nivel superior que representan cada página principal de la aplicación (Dashboard, ModelManagement, Predictions).

components/: Alberga componentes reutilizables, organizados en subdirectorios según su dominio funcional (ej. model-management/).

store/: Define el store de Redux y los "slices" que gestionan la lógica y el estado de cada entidad (ej. modelSlice.js).

services/: Encapsula la lógica de comunicación con la API del backend.

La aplicación se diseñó con un enfoque "responsive", utilizando hooks personalizados como useIsMobile para adaptar la disposición de los componentes a diferentes tamaños de pantalla, garantizando una experiencia de usuario óptima tanto en dispositivos de escritorio como móviles. Un componente Header persistente en toda la aplicación proporciona una identidad visual y contiene accesos directos a las interfaces de MLflow y Prefect.

La aplicación se organiza en torno a un Dashboard principal que actúa como punto de entrada, ofreciendo una vista resumida del estado del sistema y accesos directos a las dos funcionalidades principales de la plataforma. Estas funcionalidades se presentan al usuario como módulos claramente diferenciados para garantizar una experiencia de usuario intuitiva y ordenada:

Gestión de Modelos (Model Management): Permite supervisar y controlar todo el ciclo de vida de los modelos de Machine Learning, desde el entrenamiento hasta su puesta en producción.

Predicciones (Predictions): Proporciona las herramientas para utilizar el modelo productivo, permitiendo a los usuarios subir nuevos datos de señales EEG, obtener una clasificación y consultar el historial de inferencias pasadas.

A continuación, se detalla la implementación de cada uno de estos módulos.

Módulo de Gestión de Modelos (Model Management)

Esta sección, accesible desde el dashboard principal, constituye el centro de control para la experimentación y la gestión de versiones. Su diseño se enfoca en presentar una vista completa y contextual de un modelo específico, permitiendo al usuario tomar decisiones informadas sobre su evolución y despliegue.

Al ingresar a esta página, el primer elemento con el que interactúa el usuario es un selector desplegable que le solicita elegir uno de los modelos disponibles y registrados en el sistema (ver figura 9).

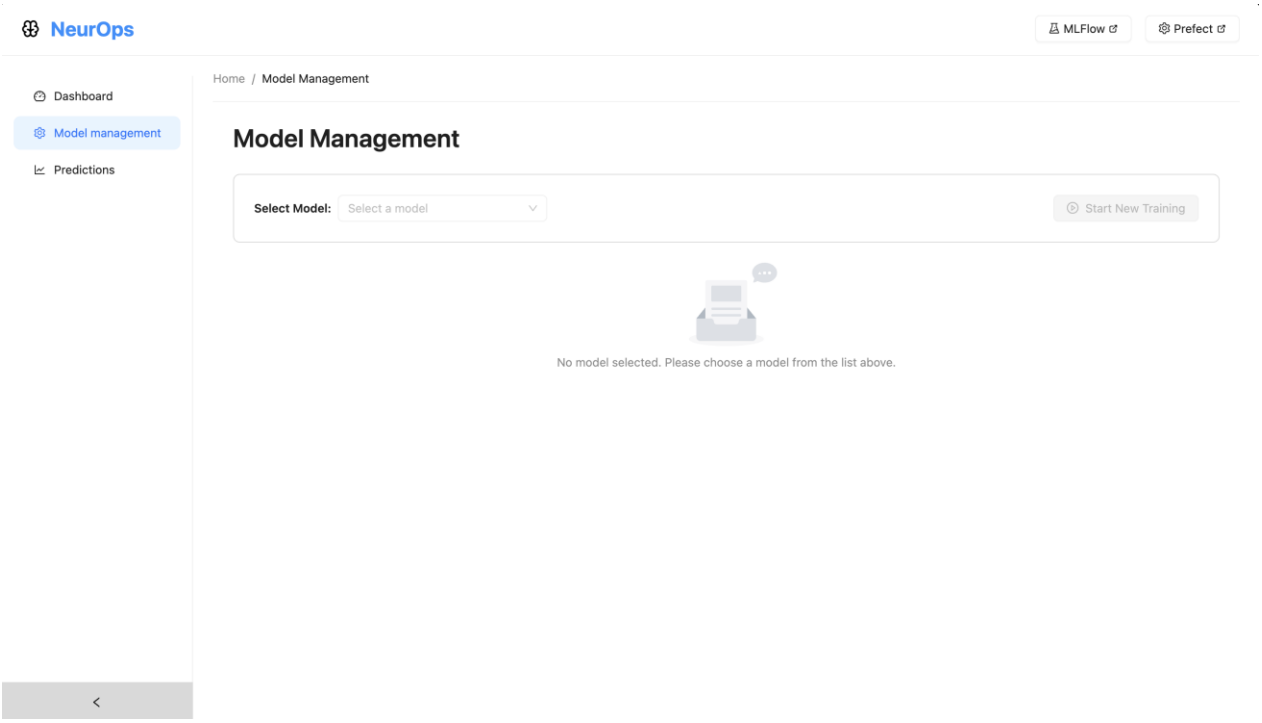


Figura 9. Interfaz gráfica de la sección de Model Management antes de seleccionar un modelo.

En esta selección inicial el nombre del modelo seleccionado se convierte en el contexto para todas las demás consultas y visualizaciones en la página. Una vez seleccionado un modelo, la interfaz se llena dinámicamente con los siguientes componentes informativos:

Información del Modelo en Producción: Este componente de tarjeta (Card) presenta un resumen destacado de la versión del modelo que está actualmente designada como "Production". Muestra información como el nombre del modelo, su número de versión, el ID de la ejecución (Run ID) que lo generó, su estado actual y la fecha de la última actualización (ver figura 10). También despliega una parrilla con las métricas de rendimiento clave (exactitud, F1-score, etc.) obtenidas durante el entrenamiento de esa versión específica, ofreciendo una visión rápida de su eficacia.

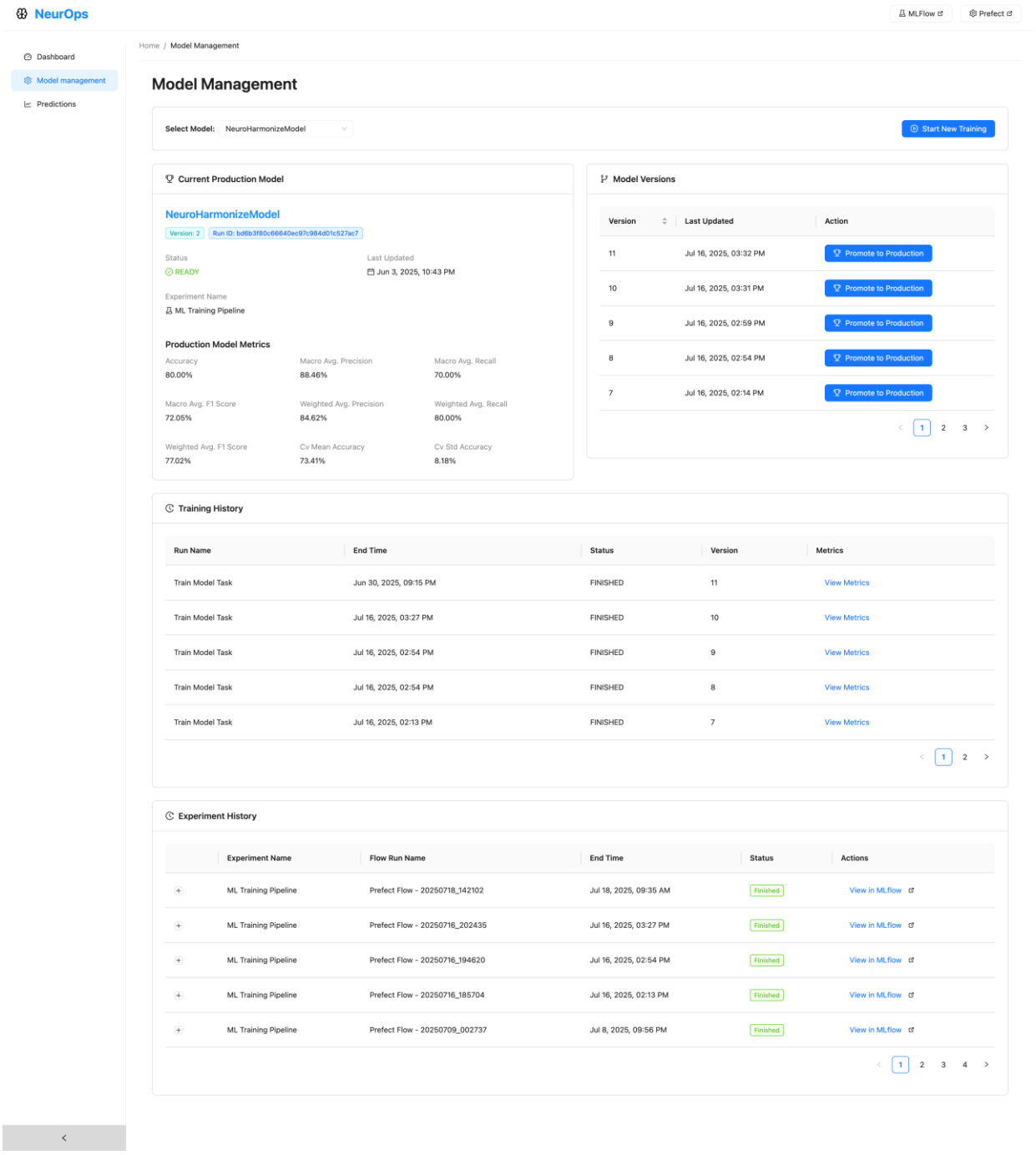


Figura 10. Interfaz gráfica de la sección de Model Management después de seleccionar un modelo.

Historial de Experimentos: Esta tabla proporciona una vista exhaustiva de todas las ejecuciones de flujos de entrenamiento (parent runs de MLFlow) que se han realizado, independientemente de si han sido registradas como un modelo (ver figura 11). Cada fila es expandible para mostrar las tareas anidadas (child runs), permitiendo al usuario ver el estado y la duración de cada paso del pipeline. La funcionalidad más importante de este componente es el botón "Register Model", que

aparece en las ejecuciones finalizadas con éxito. Este botón permite al usuario tomar el resultado de un experimento prometedor y registrarlo formalmente en el Model Registry de MLflow, ya sea como un modelo nuevo o como una nueva versión de uno existente (ver figura 12).

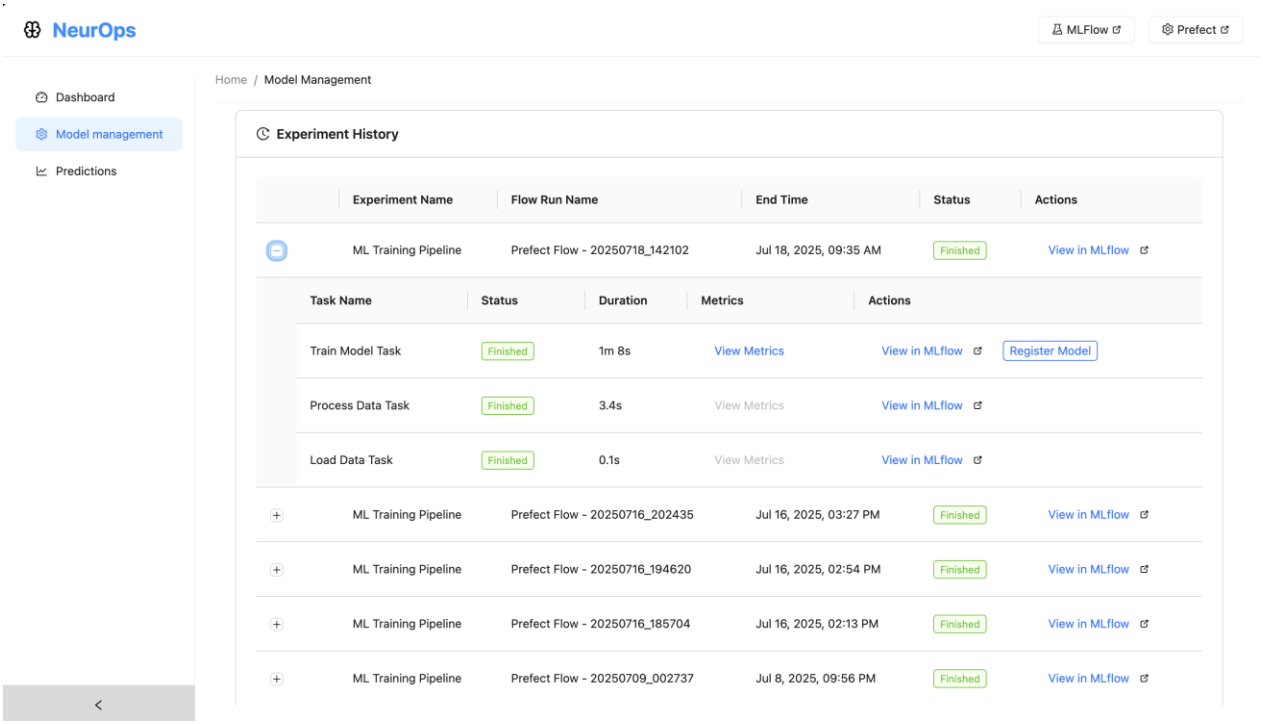


Figura 11. Tabla de Experiment History con un registro expandido por el usuario.

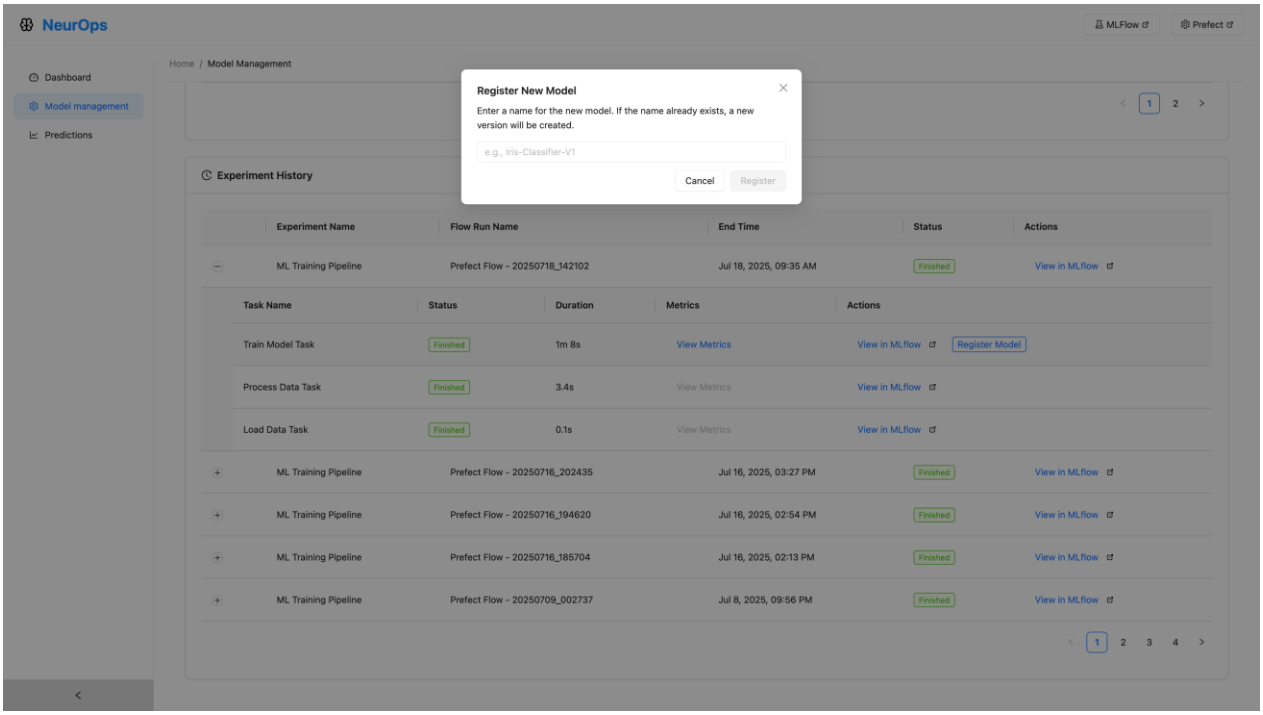


Figura 12. Tabla de Experiment History con un registro expandido por el usuario.

Historial de Entrenamientos del Modelo: De manera complementaria al historial de experimentos, esta sección presenta una tabla filtrada que muestra únicamente las ejecuciones de entrenamiento que han resultado en una versión registrada del modelo actualmente seleccionado (ver figura 10). Permite un seguimiento de la evolución de un modelo específico a lo largo del tiempo. Cada entrada ofrece un botón para visualizar las métricas detalladas en un modal.

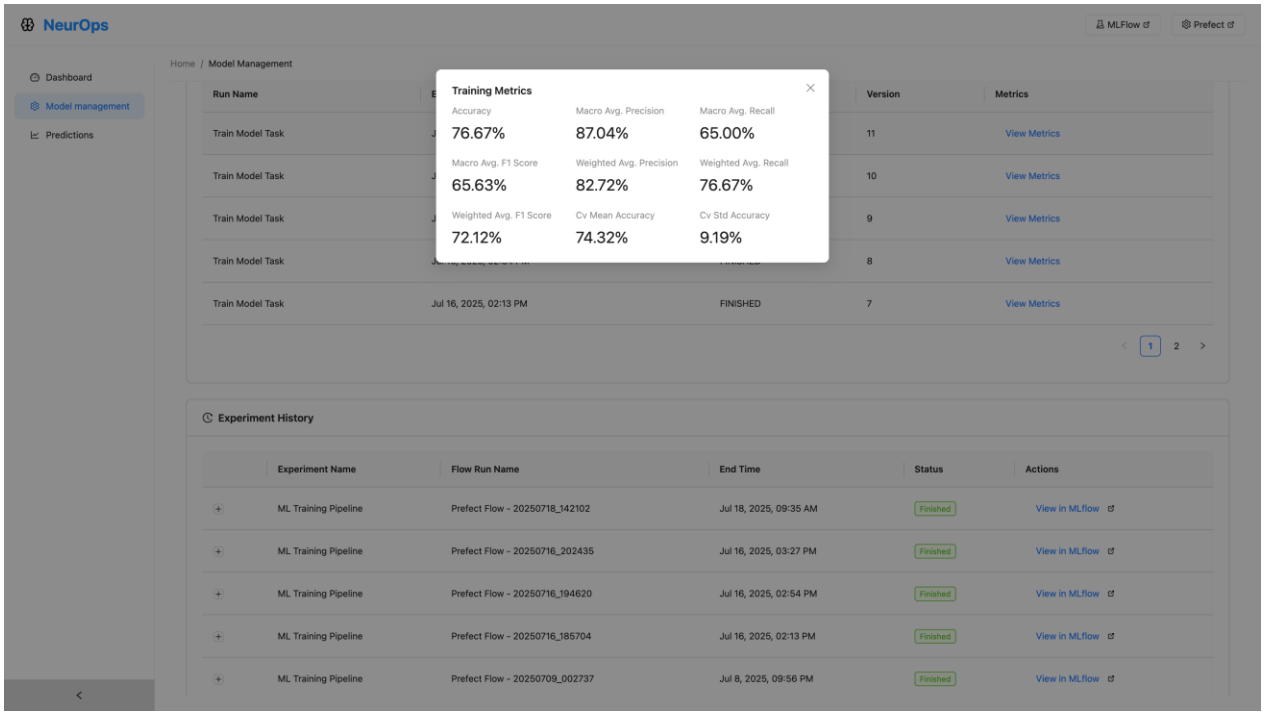


Figura 13. Métricas de una versión del modelo en la tabla de Training History.

Métricas del Modelo en un Modal: Este componente reutilizable se activa desde las tablas de historial. Al hacer clic en "View Metrics", se abre una ventana modal que presenta una visualización de todas las métricas numéricas asociadas a una ejecución de entrenamiento particular, facilitando un análisis más profundo sin sobrecargar la vista principal (ver figura 13).

Versiones del Modelo: Finalmente, este componente presenta una tabla con todas las versiones que han sido registradas para el modelo seleccionado. Muestra el número de versión y su fecha de creación (ver figura 10). La acción principal aquí es el botón "Promote to Production". Al hacer clic, el usuario puede designar una versión específica como el nuevo modelo productivo, completando así el ciclo de gestión que permite la mejora continua y el despliegue controlado de los modelos con mejor rendimiento (ver figura 14).

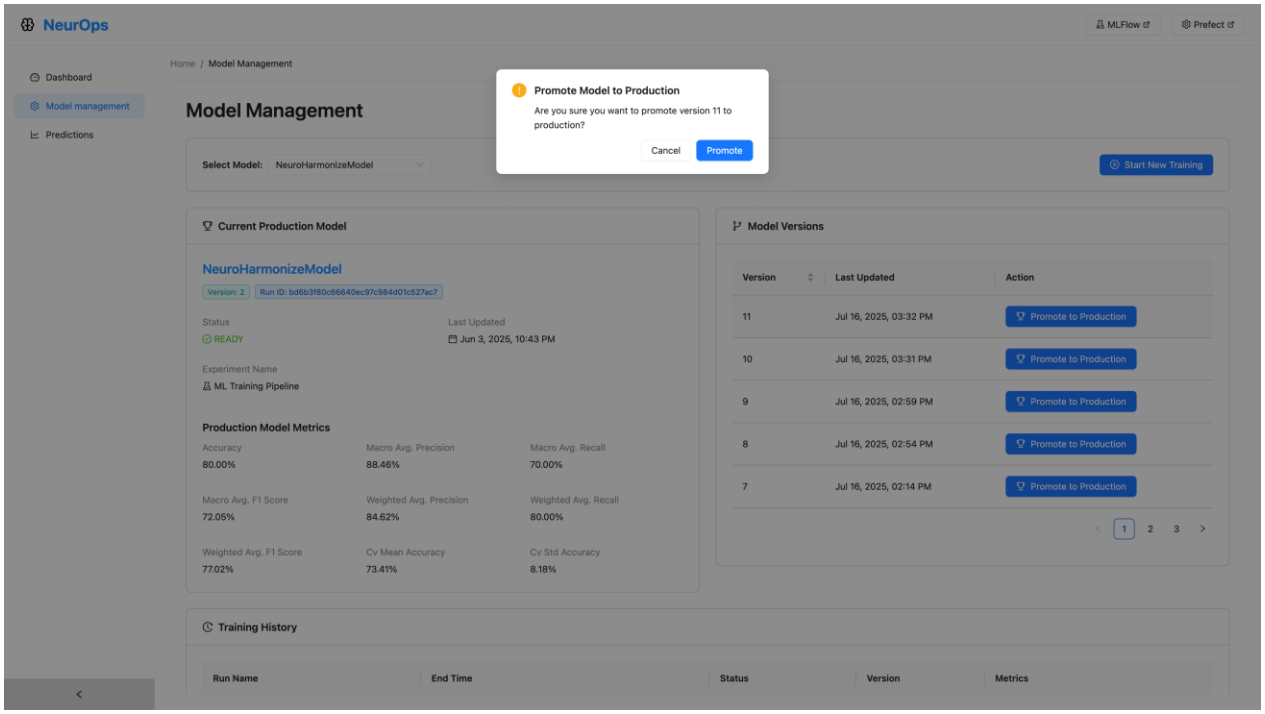


Figura 14. Selección de una versión del modelo como modelo de producción.

Módulo de Predicciones e Inferencia en Tiempo Real

Una vez establecida la capacidad de entrenar, versionar y promover modelos a producción, el siguiente paso lógico y fundamental fue desarrollar la funcionalidad para utilizar dichos modelos. El módulo de Predicciones fue concebido para cerrar el ciclo de MLOps, permitiendo a los usuarios finales interactuar con el modelo productivo para realizar inferencias sobre nuevos conjuntos de datos directamente desde la interfaz web.

El principal reto técnico de este módulo fue abstraer la complejidad del preprocesamiento de datos. El modelo de clasificación fue entrenado con un dataset de características ya extraídas de las señales EEG, en un formato tabular específico (.feather). Sin embargo, para que la plataforma fuera verdaderamente útil, debía ser capaz de aceptar datos en un formato más crudo y estándar, como el formato BIDS (Brain Imaging Data Structure), y orquestar su transformación automática hacia el formato requerido por el modelo.

Para lograr esto, la implementación se dividió en una interfaz de usuario intuitiva en el frontend y un robusto flujo de trabajo asíncrono en el backend, capaz de manejar la carga de archivos, el preprocesamiento de señales y la inferencia.

Interfaz de Usuario para la Gestión de Predicciones

La página de Predicciones fue diseñada para guiar al usuario a través de un proceso lógico y sencillo, desde la carga de datos hasta la visualización de resultados.

Carga y Gestión de Archivos: Se implementó un componente que permite al usuario cargar los datos con los que desea realizar la inferencia. Para optimizar los tiempos de subida y la gestión de múltiples archivos, la interfaz solo acepta archivos comprimidos en formato .zip. Una vez subido, el contenido del archivo se descomprime en el servidor y se presenta en un explorador de archivos interactivo. Esta vista de árbol, implementada en el frontend, permite al usuario navegar por la estructura de directorios y seleccionar las carpetas o archivos específicos que contienen las señales a procesar. Asimismo, se provee la funcionalidad para eliminar archivos o carpetas del servidor, dando al usuario control total sobre sus datos cargados. en la figura 15 se puede observar la carga de un archivo como queda registrada.

Ejecución de la Predicción: Un botón para "Realizar Predicción" se habilita únicamente cuando el usuario ha seleccionado un archivo o carpeta en el explorador. Al activarlo, se despliega un modal que solicita al usuario seleccionar, de una lista desplegable, cuál de los modelos registrados desea utilizar para la inferencia. Esta lista se puebla dinámicamente consultando los modelos disponibles en el registro de MLflow.

Historial de Predicciones: Una vez que la predicción es iniciada, esta se ejecuta de forma asíncrona. Para mantener al usuario informado, se diseñó una tabla de "Historial de Predicciones". Similar al historial de entrenamientos, esta tabla muestra todas las ejecuciones de predicción pasadas, su estado (en progreso, completado, fallido), la fecha de ejecución, y el modelo utilizado. Cada fila puede ser expandida para mostrar detalles adicionales, como los datos de entrada exactos y el resultado de la clasificación obtenido, proporcionando una trazabilidad completa de cada inferencia. Observar en las figuras 16 y 17 como se distribuye la información resultado de la predicción.

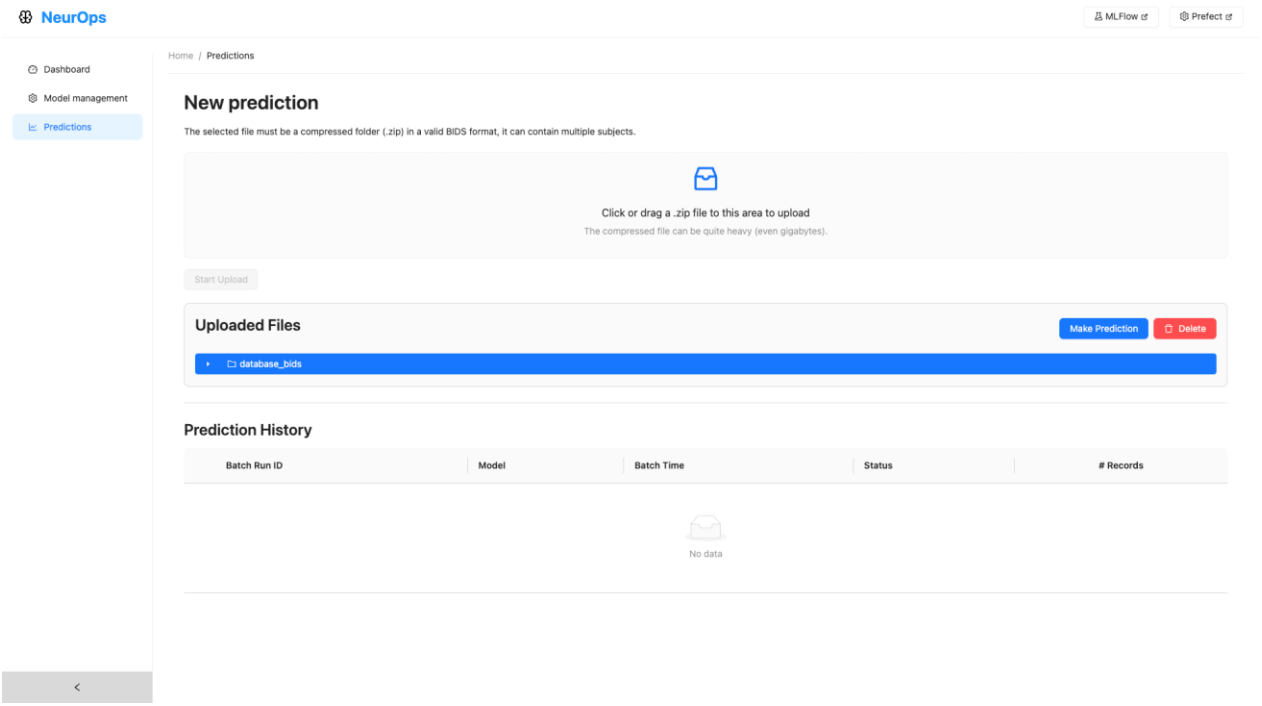


Figura 15. Sección de predicciones, luego de cargar un archivo y antes de generar una nueva prediccion.

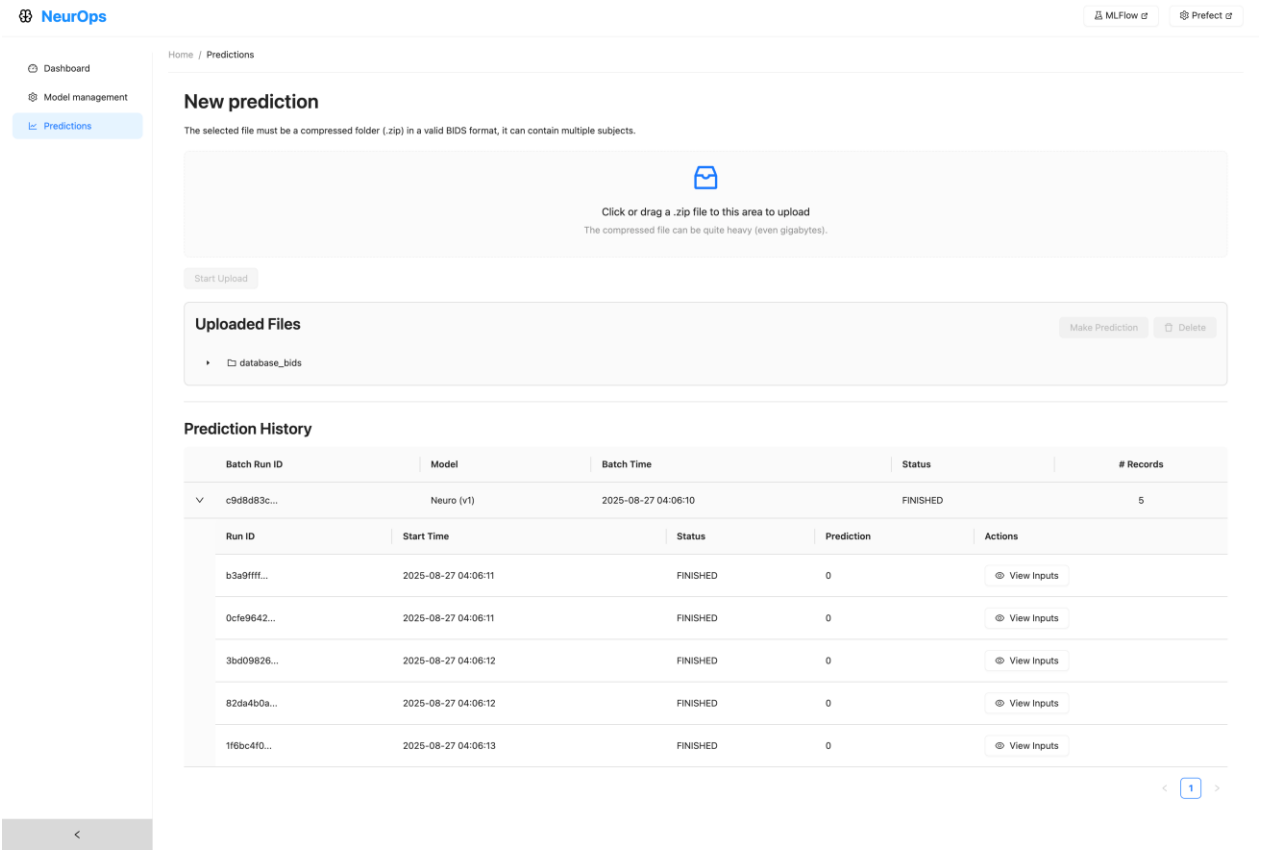


Figura 16. Sección de predicciones. Datos resultados luego de generar una nueva predicción.

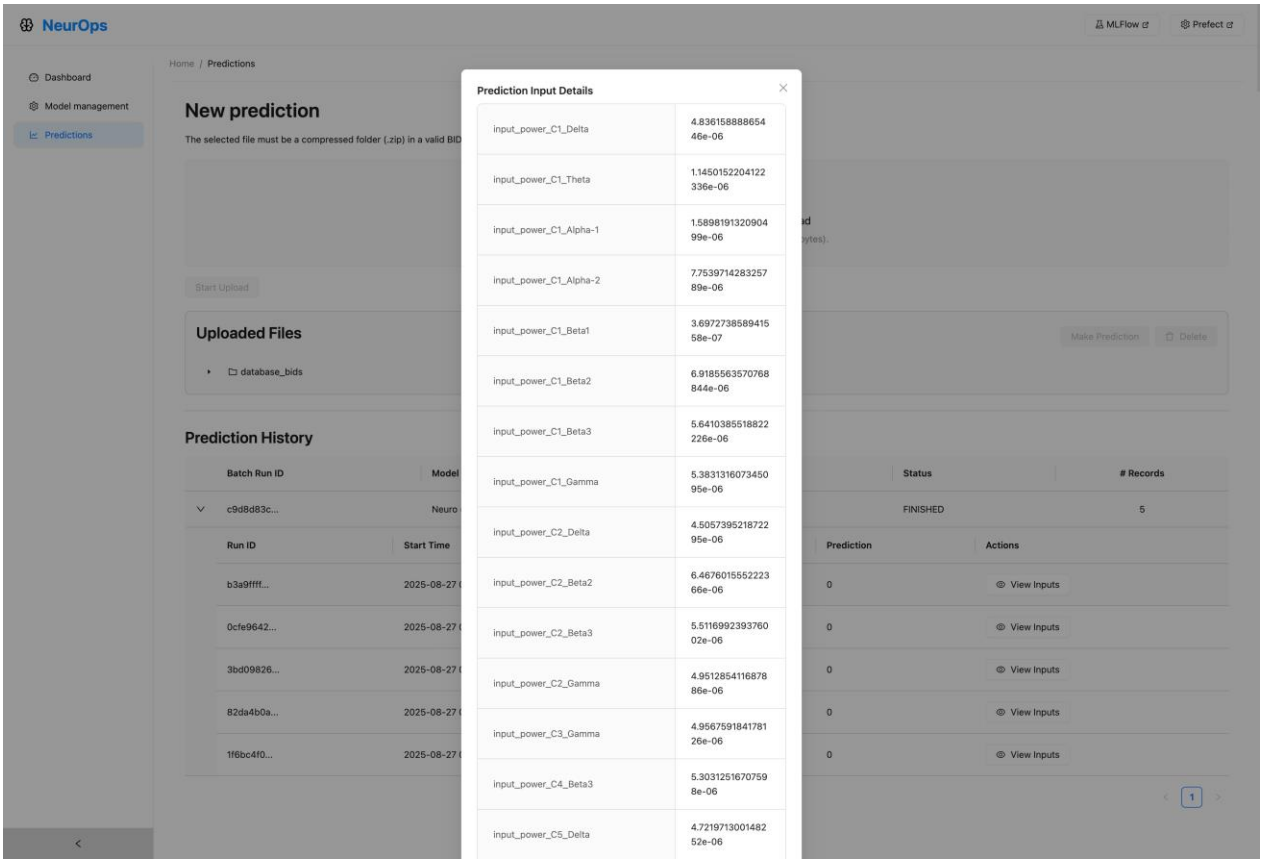


Figura 17. Sección de predicciones. Modal que muestra los datos input preprocesados para la predicción.

Orquestación del Backend para el Pipeline de Predicción

Para dar soporte a la funcionalidad del frontend, el backend fue extendido con nuevos endpoints y un flujo de orquestación dedicado para las predicciones.

Gestión de Archivos en el Servidor: Se desarrolló un nuevo router (files_router.py) con endpoints específicos para manejar la subida, listado y eliminación de archivos. Los ficheros subidos por el usuario se almacenan en un directorio persistente (/app/data/uploads) dentro del contenedor del backend, asegurando que los datos estén disponibles para los procesos posteriores.

Pipeline de Predicción con Prefect y MLflow: El núcleo de la funcionalidad reside en un nuevo pipeline automatizado, ml_prediction_flow, orquestado por Prefect. Cuando un usuario solicita una predicción a través de la API (POST /api/models/{model_name}/predict), el servicio ml_model_service.py invoca este flow. El flow es responsable de ejecutar secuencialmente las siguientes tareas:

Cargar el modelo productivo: Se carga desde el registro de MLflow la versión del modelo seleccionada por el usuario.

Procesar los datos de entrada: Se ejecuta un script de preprocesamiento sobre los datos BIDS seleccionados por el usuario.

Realizar la inferencia: Se aplica el modelo cargado a los datos ya procesados para obtener la clasificación.

Registrar los resultados: La predicción completa (datos de entrada, modelo usado, versión, fecha y resultado) se registra como un nuevo "run" en un experimento dedicado de MLflow llamado Model_Predictions, garantizando un historial auditable.

Integración de sovaharmony para el Procesamiento de Señales

El paso más crítico dentro del pipeline de predicción es la transformación de las señales EEG en formato BIDS a la matriz de características que el modelo espera. Para esta tarea, se integró sovaharmony [22], un proyecto de investigación externo diseñado específicamente para la armonización y el procesamiento de señales de EEG.

sovaharmony es una librería que encapsula una serie de pasos de preprocesamiento estandarizados, incluyendo la aplicación de filtros, la segmentación de la señal (epocaje) y, fundamentalmente, la descomposición mediante Análisis de Componentes Independientes (ICA). A partir de estos componentes, calcula un conjunto de biomarcadores espectrales y de conectividad que sirven como características para el modelo de Machine Learning.

La configuración para el uso de sovaharmony en este proyecto se definió en el script data_preprocessing_script.py. En este, se especifica que el procesamiento debe realizarse utilizando los componentes independientes (IC=True) y se definen las bandas de frecuencia de interés (Delta, Theta, Alpha, etc.) sobre las cuales se calcularán las métricas de potencia espectral.

La integración de sovaharmony presentó un desafío particular. Debido a la complejidad de su árbol de dependencias, la instalación directa generaba conflictos con las librerías existentes en el entorno del proyecto. Para superar este obstáculo sin comprometer la estabilidad del sistema, se optó por una estrategia de empaquetado. El proyecto sovaharmony fue compilado en un archivo de formato Wheel (.whl). Un archivo Wheel es un formato de paquete de Python pre-compilado que permite la distribución de librerías de una manera más eficiente y autocontenida, ya que puede incluir binarios y metadatos, reduciendo la necesidad de compilación en la máquina de destino y simplificando la gestión de dependencias complejas [23]. El archivo sovaharmony-0+untagged.460.g92ce965.dirty-py3-none-any.whl fue incluido directamente en el directorio vendor/ del backend e instalado durante la construcción de la imagen Docker, garantizando que sus funcionalidades estuvieran disponibles para el pipeline de predicción sin generar conflictos en el entorno.

III. RESULTADOS Y DISCUSIONES

El desarrollo de la capa de aplicación y servicios culminó en una plataforma web funcional e integrada, que abstrae con éxito la complejidad de la infraestructura MLOps. La arquitectura

desacoplada, con un backend RESTful implementado en FastAPI y un frontend reactivo construido con React, demostró ser una base sólida.

La interfaz de usuario permite a los usuarios iniciar entrenamientos, monitorear su progreso a través del historial de experimentos y, lo que es más importante, registrar y promover modelos a producción con solo unos clics. Las tablas y modales implementados se comunican con el backend, que a su vez consulta a MLflow y Prefect para obtener y presentar datos de manera consolidada. Esto elimina la necesidad de que el usuario interactúe directamente con las distintas herramientas, cumpliendo el objetivo de unificar la gestión en una única interfaz.

El resultado más significativo se obtuvo en el módulo de Predicciones. La plataforma logró transformar un proceso de múltiples pasos y alta complejidad técnica —la extracción de características de señales EEG mediante sovaharmony— en un flujo de trabajo simple y automatizado para el usuario final. La funcionalidad de carga de archivos .zip y el pipeline de predicción asíncrono operaron según lo diseñado. Las pruebas confirmaron que el sistema es capaz de tomar datos en formato BIDS, orquestar su procesamiento a través del pipeline de Prefect, realizar una inferencia con el modelo productivo y registrar de forma auditable cada paso en MLflow. La capacidad de la plataforma para manejar esta transformación de datos de forma transparente representa un avance clave hacia la usabilidad del sistema en un entorno de investigación real.

A pesar del éxito en la implementación, se identifican posibles áreas de mejora. Si bien la interfaz presenta la información de manera clara y funcional, podría enriquecerse con visualizaciones más avanzadas, como gráficos interactivos que muestren la evolución de las métricas de los modelos a lo largo del tiempo o análisis estadísticos sobre el historial de predicciones. Además, aún no está presente la opción de visualizar los artefactos generados (como gráficas) directamente en la plataforma.

CAPÍTULO V: VALIDACIÓN DE LA PLATAFORMA

I. INTRODUCCIÓN

La fase final de este proyecto se centró en la validación integral de la plataforma, correspondiente al cuarto y último objetivo específico. Esta etapa se enfocó en la verificación sistemática y exhaustiva de todas las funcionalidades desarrolladas dentro del entorno de desarrollo local. El propósito fue demostrar que la plataforma opera como un sistema cohesivo y cumple con los requisitos funcionales, desde la ingesta de datos y el entrenamiento de modelos hasta la generación de predicciones, todo orquestado de forma centralizada y fácilmente accesible a través de la interfaz de usuario.

Esta validación es fundamental para confirmar que la arquitectura MLOps implementada no solo es funcional en sus partes individuales, sino que el flujo de trabajo completo está correctamente integrado. Se simuló el ciclo de vida completo de un modelo de Machine Learning, comenzando desde un estado inicial sin modelos ni experimentos, para verificar cada paso que un usuario seguiría. Se evaluó la capacidad del sistema para entrenar, registrar, versionar y promover modelos, así como su habilidad para procesar datos crudos de señales EEG y utilizarlos para realizar nuevas predicciones, confirmando así la funcionalidad de extremo a extremo de la solución desarrollada.

II. METODOLOGÍA

La metodología de validación se diseñó para evaluar de manera secuencial los dos flujos de trabajo principales de la plataforma: la gestión del ciclo de vida de los modelos y la generación de nuevas predicciones. Las pruebas se ejecutaron en el entorno de desarrollo local, iniciando la aplicación con el comando *docker-compose up -d* para levantar todos los servicios contenerizados.

A. PRUEBAS DE GESTIÓN DE MODELOS

Este conjunto de pruebas tiene como objetivo validar el pipeline de entrenamiento automatizado y la capacidad de la plataforma para gestionar el ciclo de vida de un modelo desde la interfaz de usuario.

Estado Inicial y Lanzamiento del Entrenamiento

La prueba comienza con la plataforma en un estado "limpio", sin ningún experimento previo o modelo registrado. Al acceder a la página de "Gestión de Modelos" (/model-management), se verificó que la interfaz muestre un mensaje indicando que no existen modelos registrados y que el selector de modelos esté vacío. Desde esta página, se procede a iniciar un nuevo entrenamiento. Este proceso utiliza el dataset preprocesado especificado en el script training_script.py, el cual se encuentra en la ruta /app/data/processed dentro del contenedor del backend.

Monitoreo y Registro del Proceso

Una vez iniciado el entrenamiento, se validó que el flujo de trabajo se registre correctamente en las herramientas de MLOps. Se accedió a la interfaz de Prefect para confirmar que un nuevo "flow run" ha comenzado y que sus tareas correspondientes (cargar datos, procesar datos, entrenar

modelo) se ejecutan en la secuencia correcta. Simultáneamente, en la interfaz de MLflow, se verificó que se haya creado un nuevo "parent run" y que cada tarea del pipeline se registre como un "nested run", almacenando sus respectivos parámetros, métricas y artefactos (como la matriz de confusión y las curvas de aprendizaje).

Registro y Promoción del Modelo

Al finalizar el entrenamiento, se revisó que la ejecución aparezca en la tabla "Experiment History" de la interfaz con un estado exitoso ("successful") para todas sus tareas. A continuación, se utilizó el botón "Register Model" para registrar formalmente el resultado del experimento. Se verificó que, una vez registrado, el nuevo modelo aparezca en el menú desplegable "Select model". Al seleccionarlo, la interfaz debe poblarse automáticamente con la información del modelo, incluyendo sus versiones y el historial de entrenamientos. Finalmente, se promovió una versión del modelo al alias "production", dejándolo disponible para el siguiente conjunto de pruebas.

B. PRUEBAS DE GENERACIÓN DE NUEVAS PREDICCIONES

Esta fase de la validación se enfoca en probar el flujo de trabajo de predicción, desde la carga de datos crudos hasta la obtención de un resultado.

Preparación y Carga del Dataset

Para esta prueba se utilizó un dataset de señales EEG en formato BIDS, facilitado por un miembro del Grupo de Neurociencias de Antioquia (GNA). El dataset, contenido en una carpeta llamada database_bids, fue anonimizado, tenía un peso de 2.8 GB y contenía los datos de 5 sujetos de prueba (sub-CTR001 a sub-CTR005). Cada sujeto contaba con sus archivos de metadatos (.json, .tsv) y una carpeta “eeg” que contiene los archivos de la señal cruda (.eeg, .vhdr, etc.). Esta carpeta se comprimió en un archivo .zip y se subió a la plataforma a través de la interfaz de la página de "Predicciones".

Gestión de Archivos y Ejecución de la Predicción

Se verificó que, tras la subida, el archivo .zip se descomprimiera automáticamente en el directorio app/data/uploads del servidor y que su estructura de carpetas fuera visible y navegable en la sección "Uploaded files" de la interfaz. Se probó la funcionalidad de navegación y eliminación de archivos. Posteriormente, se seleccionó la carpeta raíz del dataset y se inició una nueva predicción, seleccionando el modelo que fue previamente promovido a "production".

Monitoreo y Verificación de Resultados

Se monitoreó la ejecución del pipeline de predicción en la interfaz de Prefect. Una vez finalizado el proceso, se confirmó que la ejecución se registrara correctamente en el experimento de predicciones de MLflow. Finalmente, se accedió a la página de "Predicciones" (/predictions) en la interfaz de la aplicación y se verificó que el resultado de la inferencia aparezca en la tabla "Prediction History", mostrando el modelo utilizado, la fecha, y el resultado de la clasificación, completando así la validación del flujo de extremo a extremo.

III. RESULTADOS Y DISCUSIONES

La ejecución del plan de validación funcional permitió confirmar el correcto funcionamiento de la plataforma MLOps. Todas las pruebas se completaron satisfactoriamente, demostrando que los componentes de la arquitectura operan de forma integrada y que los flujos de trabajo de entrenamiento y predicción cumplen con los objetivos planteados.

Los resultados de la validación del flujo de gestión de modelos fueron positivos. Al iniciar la aplicación en su estado inicial, la interfaz de "Gestión de Modelos" mostró correctamente la ausencia de modelos registrados. Se procedió a iniciar un nuevo ciclo de entrenamiento, el cual tuvo una duración total de 1 minuto y 6 segundos. Durante este tiempo, se verificó en las interfaces de Prefect y MLflow que el proceso fue rastreado de manera adecuada, registrando cada tarea del pipeline como un sub-proceso con sus correspondientes métricas y artefactos. Una vez finalizado, el experimento apareció con estado exitoso en el historial de la interfaz de usuario, desde donde fue posible registrarlo como un nuevo modelo y, posteriormente, promover una de sus versiones al alias de "production". Este resultado valida de extremo a extremo el pipeline de entrenamiento continuo, demostrando que la plataforma centraliza y simplifica eficazmente el ciclo de vida del modelo.

En cuanto a la validación del flujo de generación de predicciones, las pruebas también arrojaron resultados exitosos. El archivo .zip de 2.8 GB que contenía el dataset de señales EEG en formato BIDS fue cargado sin inconvenientes a través de la interfaz. El sistema descomprimió automáticamente los datos en el servidor y los presentó de forma navegable en el explorador de archivos de la aplicación. Se inició una predicción utilizando el modelo previamente promovido a producción, un proceso que tomó un total de 10 minutos para completarse. Este tiempo de ejecución es significativo, ya que incluye no solo la inferencia del modelo, sino también el complejo preprocesamiento de las señales crudas realizado por la librería sovaharmony. El seguimiento en Prefect y el registro final en MLflow se realizaron según lo esperado, y el resultado de la clasificación fue correctamente visualizado en el historial de predicciones de la interfaz.

En conjunto, estos resultados demuestran que la plataforma es funcionalmente completa y cumple con su objetivo principal: ofrecer un sistema integrado y automatizado para el entrenamiento y la validación de modelos de Machine Learning aplicados a datos de EEG. La exitosa validación del flujo de predicción, en particular, subraya el valor de la plataforma al abstraer una considerable complejidad técnica de estos procesos.

CAPÍTULO VI: CONCLUSIONES Y TRABAJO FUTURO

I. CONCLUSIONES

Este proyecto de investigación culminó con el desarrollo exitoso de una plataforma MLOps funcional, cumpliendo con el objetivo general de implementar un flujo de operaciones automatizado para el entrenamiento continuo y la validación de modelos de Machine Learning aplicados a la clasificación de enfermedades neurológicas a partir de índices EEG. A continuación, se detallan las conclusiones derivadas de cada uno de los objetivos específicos planteados:

Se diseñó e implementó una infraestructura de datos robusta, portable y escalable. La contenerización de todos los servicios mediante Docker, junto con la orquestación a través de Docker Compose, permitió la reproducibilidad del entorno en cualquier sistema. La integración de PostgreSQL como base de datos centralizada y MLflow como sistema de tracking y registro de modelos proporcionó una base sólida para la gestión sistemática de metadatos, artefactos y versiones, cumpliendo con el primer objetivo específico.

Se implementó con éxito un pipeline automatizado de Machine Learning, superando las limitaciones de los procesos manuales del trabajo precedente. La refactorización del código de entrenamiento en componentes modulares y su orquestación mediante Prefect resultó en un flujo de trabajo genérico y reutilizable. La integración con MLflow dentro de este pipeline aseguró una trazabilidad completa de cada experimento, registrando automáticamente parámetros, métricas y artefactos, lo cual fue necesario para cumplir con el segundo objetivo específico.

Se construyó una aplicación web integral con una arquitectura frontend-backend que abstrae eficazmente la complejidad del sistema subyacente. El backend, desarrollado con FastAPI, expone de manera segura y eficiente las funcionalidades de la plataforma, mientras que el frontend, ofrece una interfaz de usuario para la gestión del ciclo de vida de los modelos y la ejecución de nuevas predicciones. Esto materializó el tercer objetivo específico.

La validación funcional de la plataforma demostró la correcta integración y operación de todospa sus componentes. Las pruebas de extremo a extremo, que abarcaron desde el entrenamiento de un modelo desde cero hasta su uso para predecir sobre un nuevo conjunto de datos en formato BIDS, se completaron satisfactoriamente, cumpliendo así con el cuarto y último objetivo específico.

Se partió de repositorios existentes del Grupo de Neurociencias de Antioquia (GNA) para construir una solución unificada. Al integrar y evolucionar estos desarrollos dentro de una plataforma MLOps, el proyecto no solo valida su utilidad, sino que también los acerca a versiones de producción accesibles para el usuario final, estableciendo una ruta de trabajo para futuros desarrollos y su eventual aplicación en entornos clínicos.

II. TRABAJO FUTURO

Si bien la plataforma actual es funcional y cumple con los objetivos propuestos, su diseño modular y escalable abre la puerta a múltiples líneas de mejora y expansión a futuro. A continuación, se proponen algunas de las más relevantes:

Migración a Almacenamiento de Artefactos en la Nube: Actualmente, los artefactos de MLflow (modelos, gráficos, etc.) se almacenan en el sistema de archivos local del servidor. Para mejorar la escalabilidad y la durabilidad de los datos, un paso futuro sería migrar este almacenamiento a un servicio en la nube como Amazon S3, Azure Blob Storage o Google Cloud Storage.

Implementación de Desencadenadores Automáticos (Triggers): El pipeline de entrenamiento actual se inicia manualmente desde la interfaz. Una mejora significativa sería implementar desencadenadores automáticos. Por ejemplo, se podría configurar el sistema para que inicie un nuevo ciclo de reentrenamiento automáticamente cada vez que se añada una cantidad significativa de nuevos datos al conjunto de entrenamiento, logrando un nivel de madurez MLOps superior (Nivel 2).

Monitoreo Activo del Rendimiento del Modelo (Model Monitoring): Se podría desarrollar un módulo de monitoreo que evalúe continuamente el rendimiento del modelo en producción. Esto implicaría comparar las distribuciones de los datos de entrada de las nuevas predicciones con las de los datos de entrenamiento (detección de data drift) y analizar el rendimiento del modelo a lo largo del tiempo (detección de concept drift). Si se detecta una degradación del rendimiento, el sistema podría alertar al usuario o incluso activar un reentrenamiento automático.

Expansión de la Interfaz Gráfica: La interfaz de usuario podría enriquecerse con visualizaciones más avanzadas. Se podrían añadir dashboards interactivos para comparar el rendimiento de diferentes versiones de modelos, analizar la distribución de las predicciones a lo largo del tiempo o visualizar la importancia de las características del modelo en producción.

Implementación de un Sistema de Autenticación y Gestión de Permisos: Implementar un sistema de gestión de usuarios. Esto incluiría una funcionalidad de inicio de sesión (login) y un sistema de control de acceso basado en roles (RBAC). Se podrían definir perfiles como "Administrador", con control total sobre la plataforma, e "Investigador", con permisos para ejecutar entrenamientos y gestionar sus propios modelos y predicciones.

REFERENCIAS

[1] J. D. Steinmetz et al., "Global, regional, and national burden of disorders affecting the nervous system, 1990–2021: a systematic analysis for the Global Burden of Disease Study 2021," The Lancet Neurology, vol. 23, Apr. 2024.

[2] H. Zhang, Q.-Q. Zhou, and H. Chen et al., "The applied principles of EEG analysis methods in neuroscience and clinical neurology," Military Medical Research, Dec. 2023.

[3] Y. S. Goh, J. Q. Y. Ow Yong, B. Q. H. Chee, J. H. L. Kuek, and C. S. H. Ho, "Machine Learning in Health Promotion and Behavioral Change: Scoping Review," Journal of Medical Internet Research, vol. 24, no. 6, p. e35831, Jun. 2022, doi: 10.2196/35831

[4] E. J. Topol, "High-performance medicine: the convergence of human and artificial intelligence," Nature Medicine, vol. 25, no. 1, Jan. 2019, doi: 10.1038/s41591-018-0300-7.

[5] A. L. Beam and I. S. Kohane, "Big Data and Machine Learning in Health Care," JAMA, vol. 319, no. 13, Apr. 2018, doi: 10.1001/jama.2017.18391

[6] F. K. Khattak, V. Subasri, A. Krishnan, E. Dolatabadi, D. Pandya, L. Seyyed-Kalantari, and F. Rudzicz, "MLHOps: Machine Learning for Healthcare Operations," arXiv preprint arXiv:2305.02474, May 2023

[7] C. Babiloni, R. Lizio, N. Marzano, P. Capotosto, A. Soricelli, A. I. Triggiani, S. Cordone, L. Gesualdo, y C. Del Percio, "Brain neural synchronization and functional coupling in Alzheimer's disease as revealed by resting state EEG rhythms," Int. J. Psychophysiol., vol. 103, 2016. doi: 10.1016/j.ijpsycho.2015.02.008.

[8] American Psychiatric Association, "Manual Diagnóstico y Estadístico de los Trastornos Mentales (DSM-5)," 5ta Edición. Madrid: Editorial Médica Panamericana, 2014.

[9] J. V. Manjón and P. Coupé, "volBrain: An Online MRI Brain Volumetry System," Frontiers in Neuroinformatics, vol. 10, Aug. 2016. [Online]. Available: https://www.volbrain.net/

[10] LF Projects, LLC, "MLflow," 2025. [Online]. Available: https://mlflow.org/.

[11] PostgreSQL Global Development Group, "PostgreSQL 16.3 Documentation," 2025. [Online]. Available: https://www.postgresql.org/docs/16/index.html.

[12] Prefect Technologies, Inc., "Prefect," 2025. [Online]. Available: https://www.prefect.io/.

[13] Docker, Inc., "Docker," 2025. [Online]. Available: https://www.docker.com/.

[14] D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," in Advances in Neural Information Processing Systems 28 (NIPS 2015), 2015.

[15] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data Management Challenges in Production Machine Learning," in Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17), Chicago, IL, USA, May 2017. doi: 10.1145/3035918.3054782.

[16] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Dept. of Information and Computer Science, University of California, Irvine, 2000.

[17] S. Ramírez, "FastAPI," 2025. [Online]. Available: https://fastapi.tiangolo.com/.

[18] Pydantic, "Pydantic," 2025. [Online]. Available: https://pydantic.dev/.

[19] Meta Platforms, Inc., "React," 2025. [Online]. Available: https://react.dev/.

[20] Ant Group, "Ant Design," 2025. [Online]. Available: https://ant.design/.

[21] M. Abramov and L. Johnson, "Redux Toolkit," 2025. [Online]. Available: https://redux-toolkit.js.org/.

[22] GRUNECO, "eeg_harmonization" 2025. [Software].

[23] D. Holth, "PEP 427 – The Wheel Binary Package Format 1.0," Python Software Foundation, 2025. [Online]. Available: https://peps.python.org/pep-0427/.

ANEXOS

Anexo A: Repositorio de GitHub con el código fuente. https://github.com/imEag/MLOps

Anexo B: Manual de usuario en español.
https://github.com/imEag/MLOps/blob/main/MANUAL_USUARIO.md

Anexo C: Manual de usuario en inglés.
https://github.com/imEag/MLOps/blob/main/USER_MANUAL.md

Programa de Bioingeniería

Implementación de una plataforma con flujo automatizado para el entrenamiento continuo y validación de modelos de Machine Learning para clasificar enfermedades neurológicas

Grupo de Neurociencias de Antioquia

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

PRACTICANTE: Emmanuel Arizabaleta Garcés

ASESORES: John Fredy Ochoa Gómez

PROGRAMA: Bioingeniería

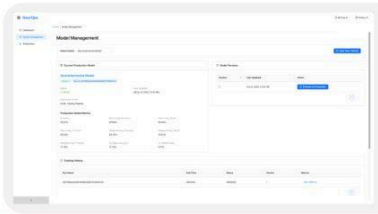
Modalidad de la práctica: Proyecto de investigación

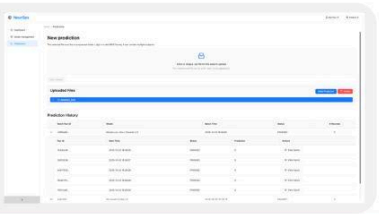
El análisis de señales EEG con Machine Learning para diagnosticar enfermedades neurológicas se ve limitado por flujos manuales que dificultan la reproducibilidad. Este proyecto lo resuelve con una plataforma de Machine Learning Operations (MLOps) que implementa un sistema automatizado para el entrenamiento continuo y la validación de modelos.

Se diseñó una arquitectura contenerizada, orquestada y monitoreada que gestiona pipelines de entrenamiento y predicción. El resultado es una plataforma que centraliza y automatiza el flujo de trabajo de ML, facilitando la reproducibilidad y escalabilidad de los experimentos en neurociencia.

Introducción

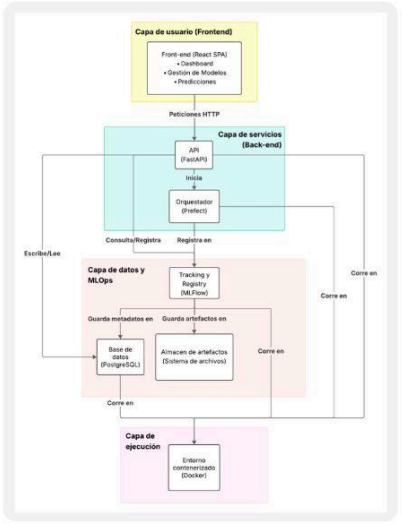
Las enfermedades neurológicas son un problema de salud de nivel global, y la electroencefalografía (EEG) es una técnica clave para explorar la función cerebral. Sin embargo, la complejidad de las señales EEG requiere métodos avanzados como el Machine Learning (ML) para extraer patrones relevantes. Los procesos manuales de entrenamiento limitan la escalabilidad, por lo que este proyecto aborda esa limitación mediante una plataforma con un flujo automatizado que permite la mejora continua de los modelos.





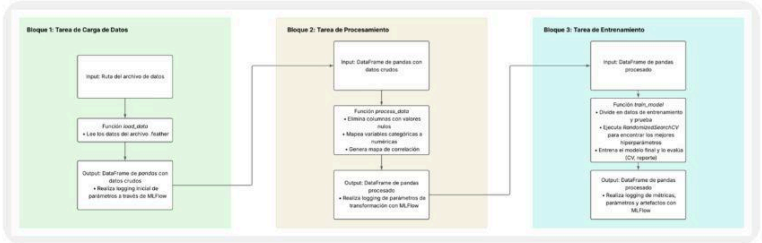
Metodología

Se diseñó una **infraestructura de datos** portable con Docker, PostgreSQL y MLflow. Sobre esta base, se implementó un **pipeline automatizado y reutilizable** con Prefect para el entrenamiento continuo de modelos. Finalmente, se construyó una **interfaz web** (FastAPI y React) que permite al usuario monitorear y gestionar el sistema y ejecutar inferencias de forma centralizada.



Resultados

Se obtuvo una infraestructura de datos robusta que centraliza la gestión de modelos y métricas. El pipeline automatizado demostró ser funcional y reproducible, garantizando la trazabilidad de cada entrenamiento. La interfaz web se validó como una herramienta intuitiva que abstrae la complejidad técnica del sistema, permitiendo al usuario gestionar y utilizar los modelos de forma sencilla. Las pruebas de confirmaron la correcta integración de todos los componentes.



Objetivos

- Desarrollar una plataforma MLOps para la clasificación de enfermedades neurológicas mediante análisis de índices EEG.
- Diseñar una infraestructura de datos para el almacenamiento y gestión de métricas asociadas a las señales EEG.
- Implementar un pipeline de ML automatizado para la recolección, entrenamiento continuo y actualización del modelo.
- Construir una interfaz para la visualización de datos, monitoreo del sistema y la realización de nuevas predicciones.

Conclusiones

- Se desarrolló una plataforma MLOps que automatiza el ciclo de vida de modelos de ML para datos EEG.
- La arquitectura contenerizada con Docker garantiza una completa portabilidad y reproducibilidad del entorno.
- La integración de Prefect y MLflow resultó en una solución robusta para la orquestación y trazabilidad de experimentos.
- La plataforma acerca los desarrollos de investigación a un entorno de producción accesible para el usuario final.

DATOS DE CONTACTO DEL AUTOR:

301 701 0466 Emmanuel.arizabaleta@udea.edu.co http://ca.linkedin.com/in/emmanuelarizabaletagarces

Más información sobre el proyecto