Estandarización y Automatización del Ciclo de Vida de Modelos de Machine learning en Entornos Empresariales mediante Prácticas de MLOps

Juan Esteban Escobar Diaz

Proyecto de grado

Asesor

Edwin Nelson Montoya Múnera

Área de Computación y Analítica

Universidad EAFIT

UNIVERSIDAD EAFIT

Escuela de Ciencias Aplicadas e Ingeniería

Maestría en Ingeniería

Medellín

2025

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

En la actualidad, el desarrollo e implementación de modelos de *Machine learning* (ML) en entornos empresariales enfrenta desafíos relacionados con la reproducibilidad, escalabilidad y automatización de los procesos. La falta de estándares claros en la gestión del ciclo de vida de los modelos puede generar ineficiencias operativas, problemas de gobernanza de datos y dificultades en la integración con los sistemas productivos.

Este trabajo propone la implementación de una arquitectura basada en una metodología estructurada mediante prácticas de MLOps, con el objetivo de estandarizar el proceso de desarrollo, despliegue y monitoreo de modelos de ML en la organización. La metodología y arquitectura diseñada integra herramientas y enfoques modernos para la automatización del pipeline, la gestión del versionado de modelos, el monitoreo en producción y la optimización de los recursos computacionales.

Palabras clave: automatización Machine learning, MLOps, modelo como servicio, CICD en Machine learning, ciclo de vida ML, Kubeflow, Vertex AI.

ABSTRACT

Currently, the development and deployment of Machine learning (ML) models in enterprise environments face significant challenges related to reproducibility, scalability, and process automation. The absence of clear standards for managing the ML model lifecycle can lead to operational inefficiencies, data governance issues, and integration difficulties with production systems.

This work proposes the implementation of an architecture based on a structured methodology following MLOps practices, aiming to standardize the development, deployment, and monitoring of ML models within the organization. The proposed methodology and architecture integrate modern tools and approaches to automate the ML pipeline, manage model versioning, enable production monitoring, and optimize computational resource usage.

Keywords: Machine learning automation, MLOps, model serving, CICD in Machine learning, ML lifecycle, Kubeflow, Vertex AI.

1. INTRODUCCIÓN

En la era actual la implementación de modelos de *Machine learning* (ML) en entornos productivos se ha convertido en un gran desafío dado la demanda en infraestructura y de equipos multi funcionales en diversos ámbitos a lo largo de su ciclo de vida. Para lograr esto, equipos multifuncionales con diferentes roles tales como ingenieros de datos, ingenieros de ML e incluso desarrolladores deben contribuir en varias de las etapas que componen una implementación de un modelo en entornos productivos.[1] Sin embargo, la integración y automatización de cada una de estas etapas representa un desafío a largo plazo debido a la complejidad inherente de los modelos y los datos utilizados en su entrenamiento. Por ello, la adopción de diversas técnicas y herramientas que combinan el *Machine learning* con prácticas de desarrollo (MLOps) ha permitido mitigar esta problemática de manera efectiva. [2]

Para lograr una buena implementación de estas técnicas y no perecer en el intento como a menudo ocurre con muchas industrias que no poseen un grado significativo de madurez [1], se debe considerar segmentar en 3 etapas el flujo de trabajo. La primera, una exploratoria o experimental donde básicamente se realiza una selección, análisis y depuración de los datos que posteriormente se usarán para entrenar el modelo, una segunda de desarrollo que abarca todo el ámbito de la selección, entrenamiento y predicción del modelo, y por último la de operacionalización, que es en palabras menos la automatización de todo lo que respecta ciclo de vida del modelo, incluyendo el despliegue, versionamiento y consumo.[3]

Los factores principales que históricamente han conllevado a que la industrialización de un modelo no sea exitosa radica en que no se tienen en cuenta muchos detalles a la hora de presupuestar los recursos a usar, dado que son modelos iterativos que cada vez se entrenan con un mayor volumen de datos. Además, los roles y segmentación de tareas entre los ingenieros de ML, desarrolladores y científicos de datos pueden generar deudas técnicas y/o conflictos entre dependencias en el largo plazo, lo que puede incurrir en *bugs* e interrupciones en el adecuado funcionamiento. Por otra parte, la necesidad de poseer un sistema que permita monitorear y controlar el versionamiento de los experimentos para mantener la integridad y trazabilidad de los cambios que pueda tener son fundamentales, pues existe la posibilidad inherente de cambios abruptos que afecten el rendimiento del modelo o incluso una alerta en la reestructuración de este.

Mediante la investigación de la conjunción en los ámbitos de operaciones y *Machine learning* y sus buenas prácticas se planea proveer contexto, proponer e implementar una arquitectura que permita la automatización y despliegue de los modelos de ML existentes en la compañía. Estos modelos actualmente se consumen y se reentrenan en gran parte de sus etapas de forma manual lo que implica una probabilidad más alta de introducir errores indeseados, por tanto, tener todo este proceso automatizado podría garantizar la agilidad, confiabilidad y gobierno de estos. Entre los beneficios de

estandarizar el proceso de los modelos mediante MLOps, se encuentra la posibilidad de catalogar y reutilizar sus predicciones y características para alimentar otros a través de un *feature store*, reduciendo reprocesos y optimizando el control sobre los recursos que consumen, con el objetivo de evitar subutilización.

1.1. PLANTEAMIENTO DEL PROBLEMA

En los entornos empresariales, la adopción de modelos de *Machine learning* (ML) ha crecido significativamente; sin embargo, muchas organizaciones enfrentan dificultades para gestionar eficientemente su ciclo de vida. La falta de estandarización en procesos como el entrenamiento, validación, despliegue y monitoreo genera inconsistencias, errores frecuentes en producción y una baja reutilización de modelos. Además, la ausencia de automatización limita la escalabilidad de las soluciones y aumenta los tiempos de entrega.

Teniendo en cuenta el anterior contexto, se vuelve crucial la incorporación de prácticas de MLOps como enfoque sistemático para integrar disciplinas de desarrollo, operaciones y ciencia de datos, con el fin de garantizar entregas continuas, seguras y reproducibles. No obstante, muchas empresas carecen de una guía clara para implementar estas prácticas de forma efectiva, lo que plantea la necesidad de diseñar estrategias que permitan estandarizar y automatizar el ciclo de vida de los modelos de ML dentro de infraestructuras empresariales complejas.

1.2 OBJETIVOS

GENERAL

Evaluar el impacto de una arquitectura diseñada mediante una metodología MLOps para la estandarización del ciclo de vida de los modelos de ML enmarcadas en el uso de herramientas gestionadas y no gestionadas, aplicadas a un caso de uso empresarial.

ESPECÍFICOS

1. Realizar una revisión de la literatura y el estado del arte sobre MLOps: Identificar las mejores prácticas, herramientas y enfoques utilizados en la industria para la implementación de MLOps, así como identificar los beneficios, desafíos y lecciones aprendidas en su adopción en diferentes sectores.

2. Proponer una metodología de desarrollo adaptada a la implementación de MLOps en el caso de uso de la empresa: Desarrollar una metodología para la implementación de una arquitectura MLOps que se ajuste a las necesidades específicas de la organización, teniendo en cuenta sus requisitos, nivel de madurez y objetivos a largo plazo.

3. Evaluar el impacto de la introducción de MLOps en la empresa mediante medición de variables clave: Analizar el impacto de la implementación de MLOps en términos de variables como tiempo, calidad y mantenibilidad, seleccionadas durante el desarrollo del proyecto, con el fin de proporcionar una evaluación objetiva de su efectividad.

2. DISEÑO METODOLÓGICO

A continuación, se presenta la metodología que guiará el proceso de implementación de una arquitectura orientada a la gestión del ciclo de vida de modelos de *Machine Learning* en una empresa, estructurada en varias fases. Cada una aborda aspectos fundamentales del proceso, desde la revisión del estado del arte hasta la evaluación de los resultados obtenidos, con el fin de asegurar una adopción exitosa y adaptada a las necesidades específicas de la organización.

Fase 1: Revisión de Literatura y Estado del Arte

En esta etapa se lleva a cabo una revisión de la literatura académica y aplicada sobre prácticas de desarrollo y operación de modelos de ML en entornos empresariales. El objetivo es identificar herramientas, metodologías y buenas prácticas, así como comprender los beneficios y desafíos que implica su adopción en distintos sectores. También se analizan los lineamientos ofrecidos por distintos proveedores de servicios en la nube.

Fase 2: Diseño de la Metodología de Desarrollo

Aquí se identifican las necesidades, nivel de madurez tecnológica y requisitos particulares de la empresa. Con base en este diagnóstico, se diseña una metodología personalizada que define las fases clave y actividades necesarias para implementar una arquitectura que satisfaga los requerimientos organizacionales, alineada con principios de automatización, monitoreo y escalabilidad.

Fase 3: Selección de Variables Objetivo y Establecimiento de Línea Base

Esta fase se enfoca en identificar los indicadores clave que permitirán evaluar el impacto de la implementación. Para ello, se establece una línea base mediante encuestas o entrevistas a usuarios finales, tomando como referencia un modelo desarrollado sin prácticas automatizadas, a fin de comparar su desempeño frente al nuevo enfoque.

3.1 Definición de variables objetivo: seleccionar variables clave como costo, tiempo, calidad y mantenibilidad, que permitirán medir el impacto de la arquitectura propuesta.

3.2 Establecimiento de línea base: determinar el estado actual de las variables seleccionadas a través de un caso de uso específico y recopilación de información cualitativa con los usuarios.

3.3 Evaluación de madurez y selección del modelo piloto: analizar el nivel de madurez tecnológica de la empresa y elegir un modelo representativo para aplicar la arquitectura y establecer comparaciones posteriores.

Fase 4: Implementación y Recopilación de Datos

En esta fase se diseña e implementa una arquitectura siguiendo la metodología definida en el proyecto piloto. A partir de esta implementación, se recopilan datos sobre las variables previamente establecidas, lo que permitirá evaluar el impacto de la adopción en términos cuantitativos y cualitativos.

> **4.1 Desarrollo del prototipo y evaluación inicial:** se implementa un prototipo utilizando las herramientas y tecnologías más adecuadas, automatizando tareas del ciclo de vida del modelo, y se comparan los resultados obtenidos respecto a la línea base.

Fase 5: Análisis de Resultados y Recomendaciones

Se realiza un análisis comparativo entre el modelo tradicional y el automatizado, tomando como referencia las variables objetivo. Este análisis permite identificar áreas de mejora, así como proponer ajustes para futuras implementaciones en otros proyectos de la organización.

> **5.1 Evaluación del impacto:** recopilación y análisis de resultados mediante encuestas, entrevistas y medición de métricas clave para determinar los beneficios concretos de la implementación.

Fase 6: Conclusiones

Se presentan los principales hallazgos, destacando los beneficios obtenidos y las lecciones aprendidas durante la implementación. Además, se formulan recomendaciones para futuras adopciones en otros contextos empresariales, contribuyendo al desarrollo de una hoja de ruta para la estandarización de procesos en entornos de *Machine Learning* operativos.

3. MARCO TEÓRICO

Los modelos de *Machine learning* comenzaron a ganar relevancia desde la década de los 60, con el objetivo de generar inferencias a partir de modelos estadísticos. Estos modelos, a través de algoritmos, son capaces de aprender y generalizar patrones a partir de los datos, lo que permite realizar predicciones o clasificaciones sobre tareas específicas [4]. Para lograr esto, generalmente se requieren grandes volúmenes de datos y algoritmos complejos, los cuales son entrenados de manera iterativa. Durante este proceso, los parámetros internos del modelo se ajustan con el fin de minimizar una función de pérdida, alcanzando así un estado de convergencia donde las predicciones son consistentes con lo esperado por el usuario [5].

Dado lo anterior, la infraestructura computacional requerida debe estar preparada para satisfacer tanto la demanda de capacidad de procesamiento (CPUs) como de memoria. Esto es fundamental para garantizar estabilidad y consistencia durante la etapa de entrenamiento, especialmente por la gran cantidad de operaciones matriciales involucradas en cada iteración [6]. En modelos más recientes, especialmente aquellos orientados al reconocimiento de voz *(Speech recognition*), las Unidades de Procesamiento Gráfico (GPUs) han reemplazado a las CPUs como principales herramientas para entrenar modelos a gran escala, debido a su mayor eficiencia en el manejo de tareas paralelas [7].

Históricamente, la implementación de modelos de *Machine Learning* han enfrentado distintos desafíos a lo largo de su ciclo de vida, particularmente en las etapas de preprocesamiento, entrenamiento, despliegue y automatización. Estos desafíos pueden intensificarse dependiendo de la complejidad del modelo y del tipo de datos utilizados, que pueden ir desde simples archivos de texto hasta imágenes de alta resolución, lo cual incrementa significativamente la demanda de recursos. Además, el despliegue de modelos en producción ya sea para atender solicitudes en tiempo real o por lotes, representa otro punto crítico si no se dispone de la infraestructura adecuada para gestionar múltiples peticiones simultáneamente [8, 9].

A su vez, la infraestructura no es el único aspecto por considerar. Durante todo el ciclo de vida del modelo es necesario contemplar posibles inconsistencias o dependencias en los datos y en el entorno, como la presencia de valores nulos, datos redundantes o errores en las fuentes de información. Por ello, para llevar a cabo con éxito cada una de las etapas y garantizar su integración efectiva, se requiere una metodología bien estructurada, acompañada de herramientas específicas. Un adecuado pipeline de datos, por ejemplo, debe contemplar subetapas como la selección, procesamiento, validación, almacenamiento y monitoreo de datos [10].

Finalmente, la operacionalización del modelo se logra mediante el uso de MLOps, una práctica que combina principios y tecnologías tanto del *Machine Learning* como del desarrollo ágil de software. Su objetivo principal es reducir la carga operativa sobre los

equipos de datos y fomentar mejoras continuas en aspectos como la calidad del código, la implementación de pruebas y la optimización de procesos [11].

El término MLOps puede tener distintas interpretaciones, ya que en la literatura se emplea tanto para describir buenas prácticas como herramientas tecnológicas. Su popularización se dio entre los años 2010 y 2015, en respuesta a la creciente necesidad de operacionalizar, versionar, integrar y gestionar modelos de *ML* en ambientes productivos. MLOps se basa en los principios de DevOps, adaptados al ciclo de vida de los modelos e integra tres disciplinas clave: ingeniería de datos, *ML* e ingeniería ágil de software (DevOps) [12].

DevOps, por su parte, es una práctica propia de la ingeniería de software que busca cerrar la brecha entre el desarrollo y la operación de sistemas mediante la automatización integral. Esto permite entregas ágiles, continuas y reproducibles, además de reducir los tiempos operacionales. La automatización en DevOps se implementa a través de pipelines, control de versiones, despliegue automatizado y mecanismos de retroalimentación, los cuales incluyen sistemas de monitoreo y observabilidad [13, 14]. La Figura 1 ilustra el flujo típico de MLOps y su integración con los principios de DevOps.

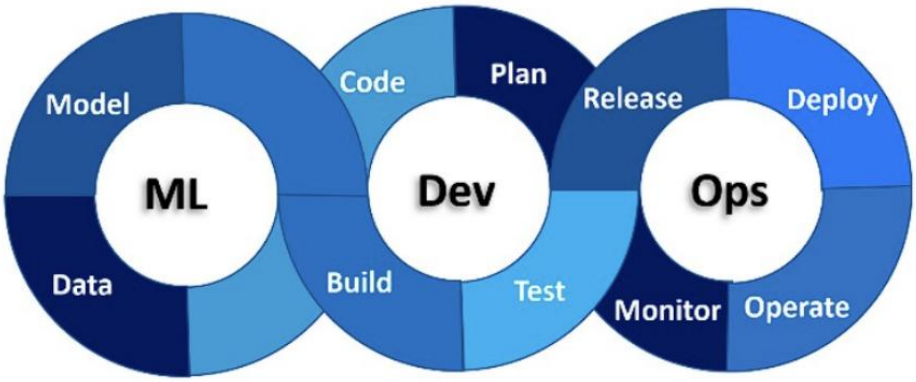

Figura 1. Flujo de trabajo genérico de MLOPs en relación con los componentes de DevOps, fuente [53]

Ciclo de vida de *Machine learning Model*

Idealmente, para lograr un despliegue exitoso de un modelo de *Machine learning*, es fundamental realizar una contextualización inicial de los datos. Esta etapa permite identificar el modelo que mejor se ajusta al objetivo de predicción, ya que, según las características del problema, pueden variar tanto las técnicas a implementar como las herramientas utilizadas a lo largo de su ciclo de vida. Generalmente, los datos provienen de múltiples fuentes, lo que requiere procesos de integración y preprocesamiento para

consolidar un insumo uniforme que posteriormente será utilizado en las fases de entrenamiento y validación.

Una vez consolidada la entrada de datos, se recomienda el uso de un repositorio centralizado de características, conocido como *feature store*, el cual actúa como única fuente de verdad. Este repositorio permite almacenar, gestionar y compartir características ya procesadas, que luego pueden ser reutilizadas entre distintos modelos según el caso de uso [15]. La utilización de un *feature store* favorece las buenas prácticas de automatización, no tanto desde el desarrollo de software, sino desde la gestión de datos. Además, facilita la integración y escalabilidad con diversas herramientas de *Machine learning* como TensorFlow, PyTorch, Jupyter Notebooks, entre otras. Gracias a su trazabilidad y documentación, el mantenimiento de características se vuelve más simple y evita reprocesos innecesarios al construir nuevos modelos sobre atributos previamente calculados [16].

Con los datos centralizados y el modelo a implementar ya definido, se procede a importar las librerías necesarias para desarrollar los distintos scripts, tras lo cual se entrena el modelo utilizando la información almacenada en el repositorio de características. A partir de los resultados obtenidos y de las métricas previamente definidas para evaluar la precisión, se selecciona el modelo más adecuado. Luego, se realiza un proceso de ajuste de hiperparámetros, conocido como "*tuning"* [17], con el objetivo de optimizar el rendimiento. Esta etapa es crucial, ya que una adecuada elección del modelo garantiza mayor consistencia y mantenibilidad a largo plazo, especialmente ante posibles cambios en el contexto de los datos o del negocio.

El despliegue, como etapa final del ciclo de desarrollo, sintetiza los esfuerzos de diferentes frentes. Para ello, la construcción de *pipelines* de integración continua (CI) y despliegue continuo (CD), junto con la definición de la infraestructura necesaria (como microservicios gestionados en clústeres, máquinas virtuales o contenedores), es esencial para facilitar tanto el consumo como el entrenamiento del modelo en producción. La elección de estos componentes dependerá del caso de uso específico y su nivel de complejidad [18].

Finalmente, es imprescindible contar con un sistema de versionamiento y monitoreo adecuado, en función de la arquitectura seleccionada. A lo largo del ciclo de vida del modelo, se generarán múltiples experimentos y reentrenamientos, por lo que es fundamental monitorear tanto la calidad de los datos de entrada como el desempeño del modelo. Este sistema debe registrar eventos anómalos y alertar sobre posibles deterioros en las predicciones [18, 19, 20]. Lo anterior cobra especial importancia debido a la alta sensibilidad de algunos modelos frente a cambios en los datos, como sesgos,

valores faltantes o alteraciones en la distribución. Por esta razón, se recomienda configurar alertas tempranas que informen sobre variaciones en la dinámica o calidad de los datos, facilitando así la intervención oportuna por parte del equipo encargado de la administración del modelo [19, 20].

Benchmarking de Plataformas de MLOps

De acuerdo con un estudio comparativo entre Microsoft Azure ML, AWS SageMaker y Google Vertex AI, se evaluaron aspectos clave como la orquestación de pipelines, la automatización del ciclo de vida de ML y el control de versiones [21]. Azure ML y SageMaker demostraron mayor madurez en tareas de orquestación de datos gracias a sus interfaces visuales y módulos sin código, facilitando la preparación y transformación de datos de forma intuitiva. Vertex AI, por otro lado, presentó una experiencia más técnica, con procesos que requieren mayor codificación manual.

En automatización, Azure ML se destacó por su integración con AutoML y Azure DevOps, permitiendo experimentación automatizada, control de versiones con GitHub y acciones basadas en eventos. SageMaker ofrece una experiencia igualmente robusta mediante Autopilot, automatización de pipelines con plantillas reutilizables y gestión de versiones desde su interfaz. Aunque Vertex AI incluye herramientas como Vertex Vizier y entrenamiento sin código, mostró limitaciones en automatización de aprobaciones y eventos, requiriendo configuraciones más complejas para lograr funcionalidades equivalentes.

Finalmente, el estudio concluyó que Azure ML combinado con DevOps y AWS SageMaker proporcionan robustez en casi todos los ámbitos, sin embargo, Azure se impone en lo que respecta a temas de seguridad, monitoreo y gobierno. Poseer este soporte en estos rubros es fundamental para que los nuevos usuarios puedan alinear todos los recursos usados durante ciclo de vida de MLOps a los estándares y/o políticas empresariales. Es fundamental para una compañía que los usuarios comprendan y apliquen correctamente las funciones de gobernanza que ofrece la plataforma de MLOps, tales como la protección de red y la seguridad de los datos. Estas funcionalidades permiten prevenir escenarios en los que las direcciones IPde las máquinas virtuales (VMs) queden expuestas o sin mecanismos de encriptación adecuados.

Madurez en ciclo de vida de MLOps

Para identificar qué nivel de automatización se requiere en el caso de uso empresarial, se realizó una categorización de los niveles de madurez existentes y como estos podrían impactar todo lo relacionado con la construcción y estandarización de un despliegue en producción. Para ello, se partirá inicialmente por el modelo propuesto [22] en la figura

2, en el que se plantea 4 niveles, siendo el de más bajo nivel la automatización del pipeline de datos (recolección, preprocesado y almacenamiento) y, por último, la automatización completa que involucraría prácticas enmarcadas en el ámbito de DevOps mencionadas anteriormente como lo son CI/CD (Continuos Integration and Continuos Deployment) y CT *(Continuos Training)*.

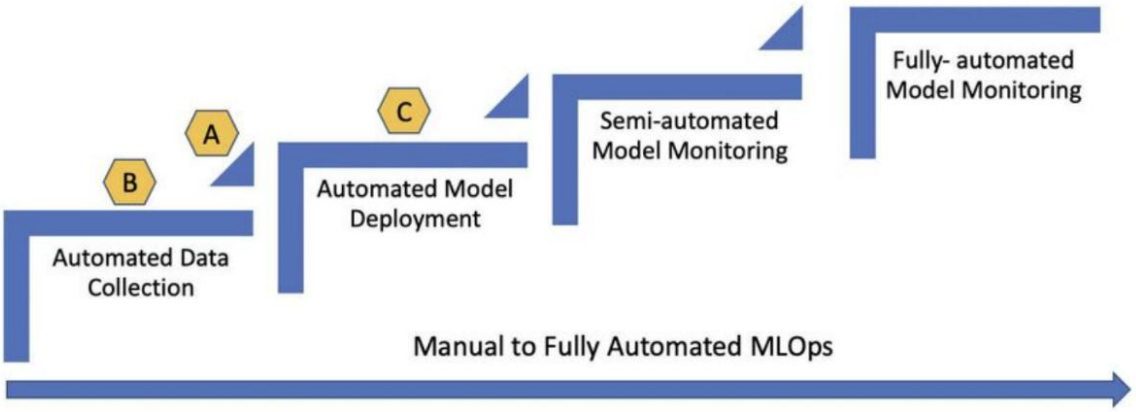

Figura 2. Nivel de madurez en MLOps, fuente [22].

Además de este diagrama de madurez en el ámbito de MLOps, existen propuestas similares desarrolladas por compañías como Google, Azure y AWS, que presentan una visión más sintetizada de los niveles de madurez. Por ejemplo, Google plantea tres niveles de automatización, los cuales son:

- **MLOps nivel 1:** Proceso manual
- **MLOps nivel 2:** Pipeline de *Machine learning* automatizado donde se incluye el componente de CT.
- **MLOps nivel 3**: CI/CD automatización de tubería

Sin embargo, el planteado por Azure se asemeja más al de la figura 2 propuesto por Moore, en él se enmarcan grados de madurez más segmentadas como se muestra a continuación, dado que pasamos de tres niveles a tener cinco como lo indica la figura 3.

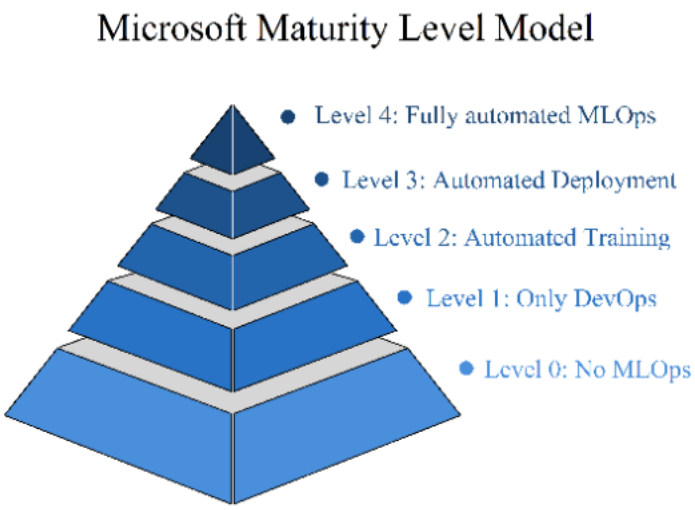

Figura 3. Nivel de madurez Microsoft ML Fuente: [1][12]

Nivel 0 - La falta de integración y comunicación entre los científicos de datos, ingenieros de datos y desarrolladores de software en la organización genera una serie de problemas. La recopilación de datos es manual y descentralizada, lo que dificulta su gestión. La ejecución de experimentos es inconsistente y no se sigue un proceso estandarizado, lo que lleva a resultados dispersos y difíciles de replicar. El desarrollo del script de puntuación es un proceso manual y no se controla mediante versiones, lo que puede generar errores y pérdida de información. Además, la gestión de versiones es un proceso manual y dependiente de la experiencia individual de los científicos de datos y los ingenieros de datos, lo que puede ser un cuello de botella en el proceso de desarrollo.

Nivel 1 – Respecto al nivel 0, las mejoras en este nivel se dan principalmente en las etapas de recolección de datos, donde ya existen pipelines automatizados que facilitan labores de integración y preprocesamiento. Por otro lado, los equipos de data ya no despliegan modelos, sino equipos de desarrollo, que incluyen pruebas de unitarias en el código y algunas otras de integración con el modelo.

Nivel 2 – Respecto al nivel 1, la comunicación entre los equipos de datos y ciencia de datos mejora considerablemente al punto de construir *scripts* que sean reproducibles en el tiempo, de forma tal que los experimentos y los modelos que se generen de forma iterativa tengan trazabilidad mediante diferentes pruebas y versionamiento. Además, para este caso todos los despliegues a producción y la infraestructura subyacente se soportan por un equipo de ingenieros de software.

Nivel 3 y 4: Para las etapas más avanzadas de madurez se contemplan en sus mejoras respecto a las anteriores, una inclusión completa de las prácticas de DevOps, ya que se involucran equipos de software y equipos de ML para automatizar y desplegar los

modelos usando CI/CD. De la misma manera, se incluyen pruebas unitarias de integración y validación durante su ciclo de vida. Sin embargo, esta última etapa a diferencia de la tercera tiene un complemento relevante y es la automatización basada en métricas desempeño u otro factor exógeno del reentrenamiento del modelo.

MLOps *Frameworks*

La operacionalización e integración de múltiples componentes en un flujo de *Machine Learning* pueden contener diferentes dependencias tanto en el rol de quien las desarrolla como en las librerías, herramientas y/o lenguajes usados entre cada etapa, siendo este último un factor determinante para evitar cuellos de botella, errores de compatibilidad e inestabilidad del modelo. Estos *frameworks* generalmente usan herramientas para orquestar estos flujos como lo son Kubeflow, TensorFlow extended, MLflow, Airflow, entre otros, y se complementan con otras herramientas en cada una de las etapas de su ciclo de vida como se puede observar en la figura 4.

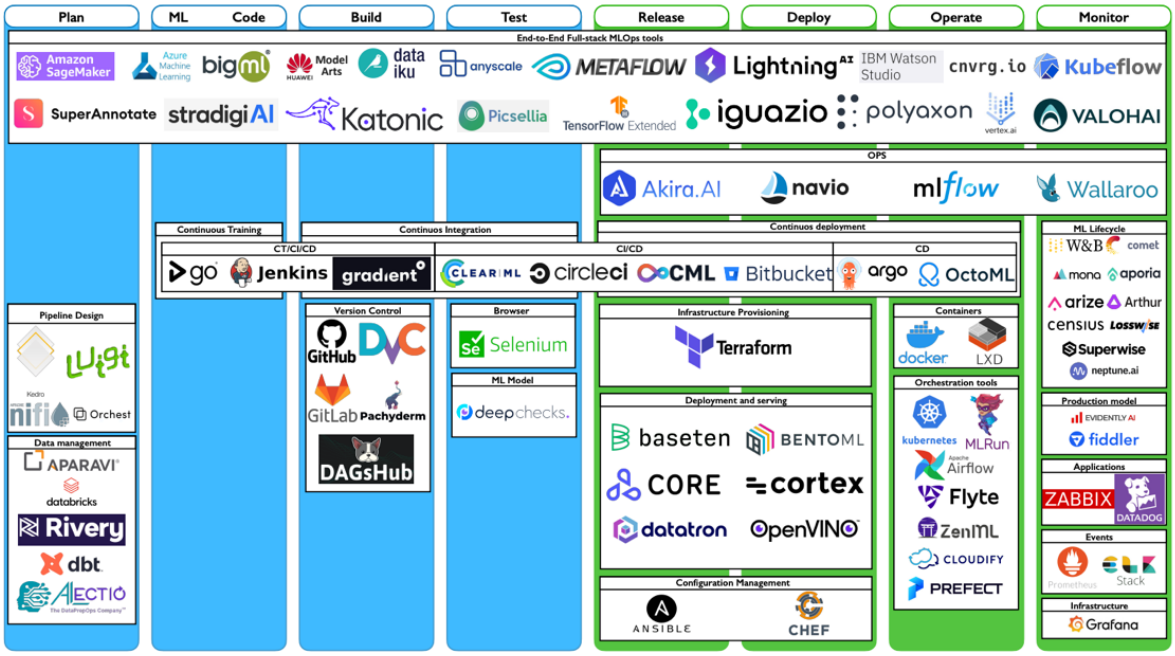

Figura 4. Herramientas identificadas y sus subcategorías para producir pipelines MLOps. Fuente: [54]

Kubeflow

Kubeflow es una plataforma de código abierto para *Machine Learning* y MLOps sobre Kubernetes, introducida por Google en 2018. Actualmente, se considera una herramienta clave para aquellos que buscan desplegar modelos de ML de manera escalable, eficiente y portable [23].

Esta plataforma ofrece una solución modular y flexible para construir, desplegar y gestionar flujos de trabajo de *Machine learning,* aprovechando la infraestructura de Kubernetes. En la figura 5 se muestra cómo Kubeflow puede abarcar de principio a fin las distintas etapas del ciclo de vida de un modelo, mediante componentes de software independientes. Entre ellos se destacan: Kubeflow Notebooks, para el desarrollo de modelos; Kubeflow Pipelines y Kubeflow Training Operator, para el entrenamiento; y KServe, para el despliegue en producción. Además, incluye herramientas como Katib, que permiten realizar ajuste de hiperparámetros, detención temprana y búsqueda de arquitecturas neuronales, aprovechando la escalabilidad de Kubernetes [24, 25, 26].

Kubeflow presenta características fundamentales dentro del contexto de MLOps, tales como escalabilidad, flexibilidad, reutilización de componentes y colaboración entre equipos. A diferencia de herramientas como TensorFlow Extended (TFX), Apache Airflow o AWS SageMaker—las cuales también ofrecen capacidades orientadas al aprendizaje automático desde diferentes enfoques—Kubeflow se distingue por su enfoque nativo en Kubernetes y su orientación modular y extensible, lo que lo convierte en una opción sólida y abierta para la gestión completa del ciclo de vida de modelos ML.

Asimismo, la plataforma promueve la colaboración entre científicos de datos, ingenieros de ML y profesionales DevOps, facilitando así el trabajo en equipo en entornos multidisciplinarios, lo cual es crucial en proyectos modernos de ciencia de datos y desarrollo de soluciones basadas en *Machine learning* [27].

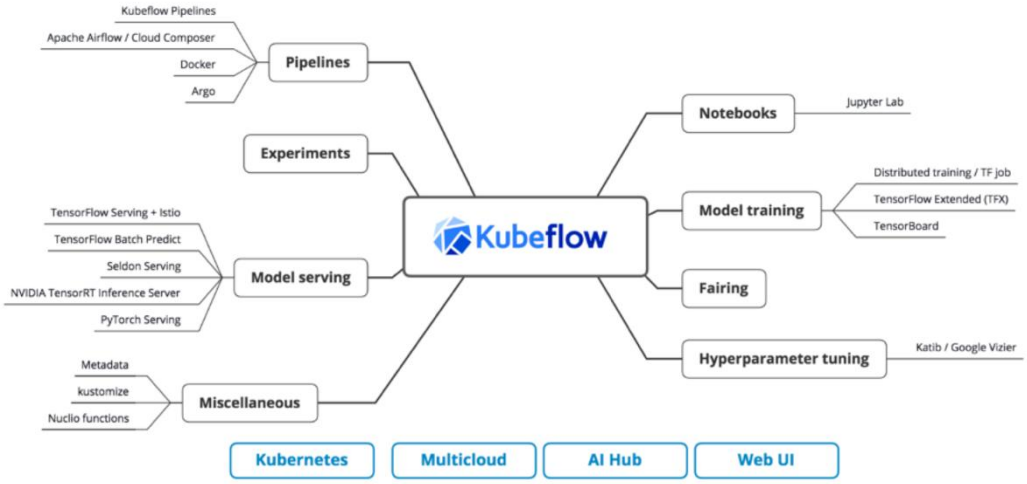

Figura 5. Componentes de Kubeflow y su arquitectura. Fuente: [55]

MLflow

MLflow es una plataforma de código abierto para la gestión del ciclo de vida de modelos de *Machine learning*, desarrollada por Databricks en 2018. Su objetivo inicial fue facilitar

la creación de modelos más confiables, seguros y escalables en contextos empresariales. Actualmente, MLflow incorpora capacidades innovadoras orientadas a la inteligencia artificial generativa (GenAI) y a la gestión de grandes modelos de lenguaje (LLMs), consolidándose como una herramienta relevante dentro del ámbito de LLMOps [28]. La plataforma se estructura en cuatro componentes principales: seguimiento de experimentos (Tracking), registro de modelos (Model Registry), despliegue de modelos (Model Deployment), y recetas de ML (MLflow Recipes). Estas funcionalidades permiten registrar parámetros, métricas, artefactos y configuraciones de entorno, asegurando la trazabilidad y reproducibilidad de los experimentos a lo largo del tiempo [29].

Uno de los puntos fuertes de MLflow es su carácter agnóstico respecto a bibliotecas, permitiendo su integración con herramientas como TensorFlow, PyTorch, Scikit-learn, entre otras. En comparación con otras plataformas como Kubeflow o TensorFlow Extended (TFX), MLflow se distingue por su enfoque simplificado en la gestión de experimentos y modelos, lo que lo convierte en una opción accesible para científicos de datos que buscan una solución liviana sin la complejidad de entornos más robustos.

Sin embargo, esta simplicidad también implica que MLflow necesita apoyarse en herramientas externas, como Apache Airflow o Prefect, para la orquestación de flujos de trabajo complejos. Mientras que Kubeflow está optimizado para arquitecturas sobre Kubernetes y orientado a perfiles DevOps, MLflow ofrece una experiencia más directa para equipos de ciencia de datos. Por su parte, TFX presenta una integración más cerrada dentro del ecosistema TensorFlow, a diferencia de MLflow que prioriza la interoperabilidad entre diferentes bibliotecas y entornos [30].

TensorFlow Extended

TensorFlow Extended (TFX) es una plataforma integral para desplegar pipelines de *Machine learning* en producción, desarrollada por Google en 2019. Un pipeline de TFX es una secuencia de componentes modulares que conforman un flujo de trabajo diseñado específicamente para tareas de ML escalables y de alto rendimiento. Estos componentes se basan en bibliotecas desarrolladas por TFX, algunas de las cuales también pueden utilizarse de forma independiente fuera del entorno del pipeline, como es el caso de herramientas para validación de datos, transformación de características o análisis de modelos [31].

La plataforma TensorFlow ofrece múltiples niveles de abstracción, lo que permite elegir el nivel adecuado para las necesidades específicas de cada proyecto. Por ejemplo, es posible construir y entrenar modelos utilizando la API de alto nivel Keras, que facilita el inicio con TensorFlow en *Machine learning* [31]. Si necesitas más flexibilidad, *eager execution* te permite probar y depurar el modelo de forma interactiva y rápida. Y si el entrenamiento se realiza a gran escala, puedes utilizar la API de estrategia de distribución (*distribution strategy*) para ejecutarlo en múltiples servidores o GPUs, sin necesidad de modificar la estructura del modelo. [32]

TFX también soporta múltiples librerías aparte de las ya contenidas en el Tensor Hub y algunas de Keras, como lo son Dopamine y Sonnet, que permiten construir avanzados algoritmos de refuerzo y redes neuronales. Además, TFX cuenta con un *toolkit* propio dentro de su ecosistema para la creación de componentes en el *pipeline*, como TensorFlow Data Validation (TFDV), TensorFlow Metadata (TFMD) y TensorFlow Model Analysis (TFMA). Esta última herramienta permite analizar y validar los datos de entrada, un paso fundamental para garantizar un entrenamiento adecuado del modelo. Además, TFX tiene dos componentes fundamentales, principalmente el *Validator* e *infraValidator*, juntos garantizan la calidad, desempeño y adecuado funcionamiento antes de desplegarlo, dado que el *infraValidator* verifica que el modelo este configurado correctamente y que no haya errores de configuración o problemas técnicos que puedan afectar su funcionamiento en el ecosistema que se va a desplegar, como por ejemplo compatibilidad, dependencias, sistema operativo, etc. [33]

Inferencia por lotes y en tiempo real

La inferencia en línea permite realizar predicciones en tiempo real mediante APIs, utilizando tecnologías como REST, gRPC y WebSockets, siendo ideal para aplicaciones que requieren baja latencia. En contraste, la inferencia por lotes es más eficiente desde el punto de vista computacional y resulta adecuada para tareas que no demandan respuestas inmediatas, ya que procesa grandes volúmenes de datos de forma agrupada, usando herramientas como Apache Spark y TensorFlow Extended (TFX) [22, 34, 35].

Para implementar estas modalidades de inferencia, el entorno de despliegue juega un papel fundamental. Las plataformas en la nube como Google Vertex AI, AWS SageMaker o Microsoft Azure ML ofrecen ventajas como escalabilidad automática, integración con pipelines de CI/CD y reducción en los costos de infraestructura. No obstante, cuando existen restricciones regulatorias o requerimientos específicos de gobernanza de datos, los despliegues *on-premises* (entornos locales) siguen siendo una alternativa viable. En estos escenarios, herramientas como Kubeflow (KServe), TensorFlow Serving y Seldon Core permiten gestionar y escalar modelos de manera eficiente dentro de infraestructuras privadas [3, 12].

Estrategias de Modelo como Servicio en el Ciclo de Vida de ML

El *Model serving* (servicio de inferencia o modelo como servicio) es un componente esencial del ciclo de vida del *ML* ya que permite utilizar los modelos entrenados para generar inferencias en distintos escenarios, desde predicciones en tiempo real hasta procesos por lotes [34]. Su implementación puede realizarse tanto en entornos *on-premises* como en la nube, cada uno con ventajas y desafíos particulares. Estas implementaciones pueden apoyarse en despliegues *serverless*, contenedores completamente gestionados (*fully managed*), máquinas virtuales o arquitecturas basadas en microservicios. Estas últimas han potenciado la flexibilidad y escalabilidad

del despliegue, facilitando una integración más eficiente con infraestructuras existentes [36].

Tecnologías como Docker y Kubernetes han impulsado el uso de arquitecturas basadas en microservicios, transformando el paradigma del *Model serving* al ofrecer modularidad, reutilización y escalabilidad. Docker permite encapsular los modelos junto con sus dependencias, garantizando portabilidad y reproducibilidad en distintos entornos; mientras que Kubernetes actúa como orquestador, gestionando el despliegue, la escalabilidad y la automatización de estos contenedores [37]. En lugar de desplegar modelos monolíticos, cada componente del flujo de inferencia puede encapsularse como un microservicio independiente, facilitando la gestión de versiones y el desacoplamiento entre el procesamiento de datos y el modelo.

Finalmente, Kubernetes ofrece un conjunto integral de mecanismos para la orquestación de servicios, tales como el descubrimiento de servicios, balanceo de carga y políticas de gobernanza. Estas capacidades son fundamentales en arquitecturas de microservicios, especialmente en aplicaciones de inteligencia artificial donde múltiples servicios deben interactuar de forma coordinada para asegurar un procesamiento eficiente y escalable [42].

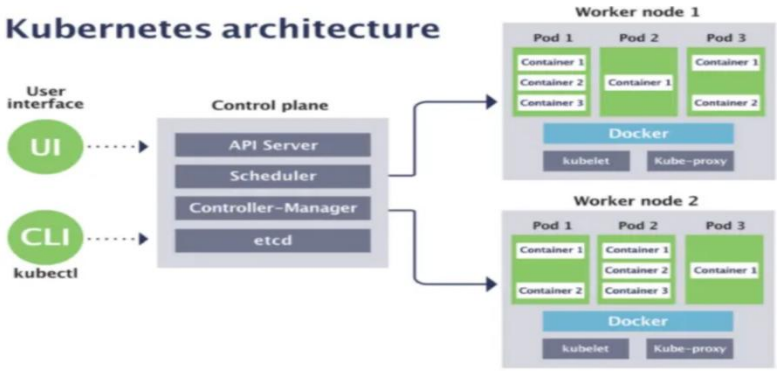

Figura 6. Arquitectura de Kubernetes. Fuente: [56]

Estrategias de integración de modelos de *Machine learning* dentro de entornos empresariales usando microservicios

La incorporación de modelos de *Machine learning* en sistemas empresariales exige arquitecturas que aseguren escalabilidad, mantenimiento eficiente y fácil integración con componentes existentes. Los microservicios se presentan como una solución efectiva, ya que permiten desplegar modelos de forma modular, desacoplada y reutilizable. En este contexto, han surgido tres distintas estrategias arquitectónicas que abordan este desafío desde distintas perspectivas, cada una con propuestas complementarias que

responden a las necesidades, retos técnicos y operativos de llevar modelos de ML a producción.

Machine learning como un microservicio reutilizable

El enfoque MLRM (Machine Learning as a Reusable Microservice) propone una arquitectura que separa la configuración de la ejecución del modelo. en dos componentes principales: el servicio de ML, encargado de encapsular y exponer los modelos como microservicios, y el conjunto de configuración (*configuration set*), que define los parámetros de personalización del modelo sin necesidad de modificar su lógica interna. Al adoptar este enfoque, se mejora la mantenibilidad de los modelos en producción y se facilita su integración en sistemas empresariales complejos, particularmente en arquitecturas basadas en microservicios [38].

Minerva para SaaS

Una arquitectura relevante en el contexto de ML en producción es Minerva, diseñada específicamente para facilitar la integración de soluciones de Machine learning en entornos SaaS empresariales mediante el uso de microservicios. Este enfoque aborda desafíos como la reutilización de componentes, la escalabilidad de los modelos y la seguridad en el manejo de datos, permitiendo incorporar modelos predictivos sin necesidad de reestructurar completamente el software existente. Minerva promueve una arquitectura modular en la que los modelos se integran como servicios desacoplados, reutilizables y versionables, facilitando su mantenimiento y evolución. Esta propuesta es particularmente útil para organizaciones que están migrando desde arquitecturas monolíticas u *on-premises* hacia entornos cloud o híbridos, ya que permite adoptar *Machine learning* de forma progresiva sin comprometer la estabilidad del producto base [39].

Machine learning en arquitecturas de microservicios

Esta arquitectura MLMA (Machine learning Microservice Architecture) ofrece una estructura modular orientada a la inferencia en producción, complementando enfoques como MLRM y Minerva. En este modelo, componentes como el *data orchestrator* y el *Predict/Classification Service* se encargan de tareas específicas como la ingesta de datos, la extracción de características y la generación de predicciones. Estos servicios se integran de forma fluida y aislada en pipelines empresariales ya existentes [40]. Potenciado por Kubernetes, este enfoque permite una asignación eficiente de recursos computacionales incluyendo GPUs y proporciona un marco escalable y desacoplado para desplegar modelos de ML a gran escala [41].

4. DESARROLLO DE LA METODOLOGÍA MLOPS

4.1 METODOLOGÍA DE MLOPS

En esta sección nos enfocaremos en la construcción de una metodología enmarcada en prácticas MLOps para solucionar el caso de uso empresarial y como este puede apalancar una arquitectura que estandarice su proceso a través de herramientas gestionadas y no gestionadas con el fin solucionar las brechas operacionales entre los equipos de científicos de datos, ingeniero de datos y software al momento de desplegar un modelo a producción.

1. Definir viabilidad y costos: La viabilidad se calcularía mediante una estimación integral y precisa de los costos del proyecto, teniendo en cuenta todos los requisitos establecidos anteriormente. Esta evaluación abarca aspectos como los gastos de infraestructura y procesamiento de datos y/o calidad de los mismo, recursos humanos y otros factores que inciden en el despliegue y automatización del ciclo de vida del modelo.

2. Definir el dominio de las fuentes de datos: En esta fase nos encontramos las diferentes fuentes de datos que pueden estar en distribuidas en diferentes formatos y pueden pertenecer a diferentes líneas de negocios, de tal forma que se evaluarían las integraciones con diferentes fuentes que se podrían soportar.

3. Definir el propósito del modelo: En esta fase se evalúa la problemática que ayudara a mitigar el modelo de acuerdo con tipo de negocio, cual podría ser su impacto (ganancias, retención de clientes, detecciones preventivas, etc) [43] considerando las limitaciones y las dinámicas que puedan afectar su viabilidad en el tiempo.

4. Identificar plantillas de configuración y procesamiento de datos y flujo de *Machine learning:* Esta fase estaría dividida en 3 plantillas principales, y se centra en la estandarización de un modelo que pueda tanto proveer flexibilidad al momento de personalizar el modelo como también de limitarlo respecto a ciertos aspectos en la infraestructura a usar, así mismo en los requisitos de desempeño mínimos de aceptación para un modelo.

La creación de plantillas estandarizadas para proyectos de ciencia de datos puede mejorar significativamente la gestión del proyecto, al facilitar la automatización, reducir errores y fomentar una mejor colaboración entre los equipos. Esta estandarización permite segmentar tareas y asignarlas a equipos especializados, lo que contribuye a una mayor eficiencia, eficacia y madurez en los procesos [44].

4.1 Plantilla para definición del modelo:

Esta plantilla incluirá la configuración del modelo de ML a implementar, junto con los parámetros personalizables según el enfoque adoptado. Distinguiremos entre enfoques *low-code* y *no-code*, donde el segundo hace referencia al uso de AutoML (*Automated Machine Learning)*. No obstante, AutoML también permite cierto grado de personalización, como la configuración de etapas específicas o la integración con otros servicios mediante scripts mínimos.

Low-code: De esta forma la plantilla contendrá información como el lenguaje en que se desarrollará, los activadores para un posible reentrenamiento (si es por un Cron Job o una alerta recibida por una cola de mensajes debido a algún cambio en la dinámica de los datos de entrenamiento). Además, los datos a los que accederá para su posterior procesamiento y entrenamiento (*Buckets* de almacenamiento o bases de datos).

No-code: Si se opta por esta alternativa el usuario debería configurar únicamente dentro de una plantilla tipo JSON o YAML el tipo de modelo a usar (Imagen, tabular, texto, video), las rutas de los datos a usar (Cloud Storage o importa desde BigQuery) y por último los hiperparámetros.

4.2 Plantilla de procesamiento de datos y entrenamiento:

Especificar la integración con las fuentes de datos y los métodos y/o ajustes a realizar, antes de ser usados como insumo para entrenar el modelo. Además, en caso de usarse una solución *low-code*, se incorporaría el modelo a usar, la división de los datos de entrenamiento/prueba, estrategias altamente personalizables como parámetros de *tunning*, *epochs*, regularización, entre otros.

4.3 Plantilla para definir la orquestación de ciclo de vida del modelo:

Mediante herramientas como Kubeflow, MLflow, TFX e incluso Airflow, podremos definir un pipeline que cree un flujo de trabajo automático para el ciclo de vida de un modelo. Este pipeline orquestara una serie de pasos como la ingesta de datos, el preprocesamiento, el entrenamiento del modelo, la evaluación y el despliegue.

4.4 Plantilla para orquestación de CT (*Continues training*):

Esta plantilla es clave para ejecutar los flujos de trabajo de reentrenamiento, ya sea a para una ejecución periódica, a raíz de cambios en el repositorio/código fuente o una alerta proveniente de sistemas externas/internos integrados que identifiquen posibles patrones de *data drifiting*, *bias* o deterioro del performance con una nueva versión del modelo.

5. Definición de alternativas de despliegue:

La selección de la alternativa más adecuada para el despliegue dependerá del contexto técnico y operativo de cada organización. Entre las opciones más comunes implementadas están el *edge deployment* para aplicaciones con baja latencia y procesamiento local; máquinas virtuales (VMs) para mayor control sin complejidad excesiva; microservicios sobre Kubernetes, reconocidos por su escalabilidad y modularidad, aunque con una mayor carga operativa; y servicios completamente gestionados como Cloud Run o Azure ML, ideales para implementaciones rápidas y de bajo mantenimiento.

6. Definición de alternativas de Inferencia (*Streaming and Batch*)

Es fundamental para la inferencia del modelo tener soporte para ambas modalidades, tanto en tiempo real como por lotes. La modalidad *streaming* a diferencia de los lotes (*batch predictions*) funciona mediante solicitudes hechas en tiempo real a un API REST endpoint en el cual se entrega la muestra como entrada y se recibe una inferencia (salida) de forma continua con la predicción del modelo [22, 45]. Esto generalmente se logra con enfoques de *data stream* y conllevan a diferentes retos entre ellos determinar la concurrencia, el tiempo de respuesta, el manejo de errores y por último el uso de *chaos engineering* con el fin de mostrar la fortaleza y/o debilidad de nuestro sistema ante fallas críticas. Esta prueba, es conocida como ingeniería del caos, mostrará la resiliencia de nuestro sistema [46].

7. Definición de pipeline de integración CI/CD

La tendencia hacia la automatización en los procesos de desarrollo empresariales hace que sea cada vez más importante contar con un seguimiento fiable de CI y CD, en esta fase se procedería a identificar cual de estas herramientas puede ser la más útil para integrar el flujo de las plantillas y cuál es la más adecuada en términos de compatibilidad. Para ello identificamos las particularidades tanto para CI como CD.

CI (Continuos Integration).

La Integración Continua (CI) en MLOps consiste en automatizar la reconstrucción del pipeline de *Machine learning* cada vez que se realiza un cambio en el código. Este proceso incluye validaciones automáticas de datos y modelos, ajuste de hiperparámetros y reentrenamiento del modelo. Además de garantizar que los cambios en el código sean integrados de forma controlada, también se ejecutan pruebas de calidad, como análisis estático mediante herramientas de linting (por ejemplo, *flake8*, *mypy*, *pylint*), detección de dependencias circulares (por ejemplo, *SonarQube*), entre otros. Del mismo modo, se contemplan aspectos de infraestructura y aprovisionamiento

de recursos, asegurando así la calidad, consistencia y confiabilidad del modelo a lo largo de todo su ciclo de vida.

CD (Continuos Deployment)

La entrega continua (CD) es un enfoque que permite incorporar cambios y mejoras en una aplicación de *Machine learning* de manera rápida y segura, mediante la entrega de pequeños y seguros incrementos de código, datos y modelos, que pueden ser reproducidos y lanzados en cualquier momento, garantizando la calidad y la confiabilidad del sistema [47]. En esta etapa se considerarían integrar pruebas pre despliegue a producción, centrado principalmente en los *smoke tests* y *canary deployments*.

8. Monitoreo y explicación

En el ámbito de monitoreo y seguridad tiene que ofrecer visión completa y en tiempo real del rendimiento y la salud del modelo, monitoreando aspectos como la calidad de los datos de entrenamiento y prueba, el desempeño del modelo en producción, la estabilidad y la escalabilidad del sistema, así como la detección de problemas y anomalías en el flujo de datos y en la salida del modelo [48]. Esto incluye el monitoreo de métricas como la precisión, la exactitud, la cobertura y la velocidad de procesamiento, así como la detección de problemas en la distribución de las clases predichas ligadas al concepto de deriva de datos (*data drift* —variación en la distribución de los datos de entrada respecto a los datos utilizados durante el entrenamiento, que puede afectar negativamente el rendimiento del modelo—) y la calidad de los datos [49]. Un sistema de monitoreo eficaz también debe ser capaz de detectar automáticamente problemas y anomalías, y alertar a los equipos correspondientes para que puedan tomar medidas correctivas de manera oportuna.

9. Consideraciones éticas y seguridad

La seguridad de los datos y el cumplimiento de los sistemas involucrados en el ciclo de vida deben ser una prioridad, especialmente al manejar información sensible. Es esencial implementar soluciones de autenticación como OAuth y JWT para evitar accesos no autorizados, y proteger los datos tanto en reposo como en tránsito para prevenir brechas de seguridad. Además, automatizar procesos éticos que verifiquen el cumplimiento normativo, como el GDPR, HIPAA y CCPA, mediante herramientas que monitoricen continuamente la privacidad de los datos y el cumplimiento de las regulaciones locales e internacionales, es clave. Las auditorías de seguridad, evaluaciones de riesgos y pruebas de penetración también son fundamentales para detectar amenazas y asegurar que los datos se mantengan protegidos, garantizando así la privacidad y el cumplimiento de las leyes vigentes.

5. IMPLEMENTACIÓN DE LA METODOLOGÍA MLOPS

Se procede a realizar con base a la documentación y las tecnologías disponibles tanto de código abierto como privadas (proveedores de nube por ejemplo) una arquitectura que permita realizar una sinergia entre herramientas gestionas y no gestionadas en combinación con CI/CD para la automatización del ciclo de vida de diferentes modelos de ML. Para ello se toman en consideración ciertas restricciones ya definidas de antemano como lo es un proveedor de nube. Inicialmente se propuso un enfoque híbrido donde se buscará integrar herramientas de código abierto como Kubeflow y MLflow con algún proveedor de nube, sin embargo, la compañía no cuenta con el personal capacitado para gestionar y administrar la infraestructura subyacente que esto pueda conllevar, dado que no es su enfoque de negocio. Teniendo en cuenta lo anterior se investigó acerca de soluciones gestionadas que ya existen como lo son AWS Sagemaker, AzureML y Vertex AI la cuales sirvieron como referencia para realizar una arquitectura que pudiera cubrir aspectos de seguridad, escalabilidad, operabilidad y gobierno, sin perder de vista un balance entre la relación costo/beneficio.

Teniendo en cuenta lo anterior se consideró dividir la automatización en diferentes etapas, partiendo de una línea base y pensándolo desde una solución objetiva que más adelante permitiera evolucionar e integrarse con el actual ecosistema de la nube.

5.1 Línea base

El modelo base servirá como referencia para evaluar los cambios al implementar la metodología de MLOps en la estandarización del despliegue y automatización de modelos de ML, el cual tiene como objetivo predecir la probabilidad de que un usuario adquiera un préstamo de libre inversión, para poder contrastar con el proceso previo semiautomatizado que se destalla en la figura 7. Este modelo ha desempeñado un papel crucial desde sus primeras implementaciones, al acelerar significativamente el proceso de evaluación de solicitudes mediante la provisión de respuestas rápidas sobre la viabilidad del préstamo leasing, basadas tanto en variables externas, como el comportamiento del mercado, así como factores intrínsecos de cada solicitante.

El modelo fue construido usando un *framework* de *lightGBM classifier* y usualmente se reentrena de forma manual cuando hay suficiente información en determinadas ventanas de tiempo, usualmente cada mes, sin embargo, dependiendo de cambios en el negocio o drásticos en las distribuciones de los datos se considera la opción de reentrenamiento bajo la aprobación de las partes involucradas.

Las variables principales en las que se apoya este modelo son variables cualitativas y cuantitativas como la edad, sexo, ingresos mensuales, estabilidad de los ingresos, patrimonio, relación deuda/ingreso, tipo de contrato laboral (trabajo formal o prestación de servicios), nivel educativo, estado civil. En el ámbito crediticio y de seguros del

usuario se toman en cuenta variables como el historial y puntaje crediticio, tasa de impagos, si el usuario esta activo con una deuda o no, morosidad promedio, si está actualmente moroso, el informe de seguro habitacional, las propuestas de pago de consumidores, entre otros.

Dado que el modelo actualmente funciona en entornos productivos la efectividad y viabilidad del modelo han sido comprobadas previamente mediante rigurosos análisis que evaluaron su impacto y el equilibrio entre costo/beneficio para la organización. Por tanto, nuestro enfoque no se centrará en la naturaleza del modelo a nivel de explicabilidad, selección de variables, análisis, etc., En su lugar, nos centraremos en las fases del marco metodológico que serán clave para estandarizar este y otros modelos futuros hacia las prácticas automatizadas que ofrece el enfoque MLOps.

Madures actual MLOps en caso de estudio

Una vez establecida la línea base, así como la de los modelos anteriormente desplegados en la compañía, se observa que algunas etapas están semiautomatizadas, mientras que otras siguen siendo predominantemente manuales, como se ilustra en la figura 5. Inicialmente parten de una necesidad de negocio que requiere analítica para su solución. Para ello el equipo de científico de datos realiza una exploración manual de los datos en un proyecto *sandbox* (proyecto en entorno de pruebas) donde realizan diferentes pruebas e inferencias estadísticas para determinar la consistencia y la distribución de los datos. Con base a al análisis de los datos y la complejidad de lo que se requiera predecir, el equipo de científicos de datos determina un conjunto de modelos candidatos para llevar a cabo el entrenamiento y posteriormente su evaluación con los datos de prueba.

Una vez definido los modelos el equipo de científico de datos se encarga de escribir el código usando el lenguaje qué mejor se adapte al caso de uso. Allí ejecutaran el entrenamiento del modelo(s) de forma manual (ya que se trata de un caso en demanda). Concluido el entrenamiento continúa la etapa de evaluación mediante muestras nunca antes vistas por el modelo y determina de acuerdo a unas métricas de desempeño (las cuales pueden variar de acuerdo al caso de uso) si la predicción es aceptable y cumple con los estándares inicialmente definidos.

El modelo binario resultante es usualmente un archivo *.pkl* y es almacenado en un Storage Bucket para luego ser desplegado en un servicio que permita recibir nuevas muestras y realizar predicciones. La infraestructura y servicios/microservicios usados para el despliegue son creados y configurados manualmente cada vez que hay un modelo nuevo.

Este es en resumen el nivel de madurez actual el cual se encuentra la compañía, la cual entraría en un nivel 1 si lo comparamos con el estándar usado por Microsoft. A

continuación, encontramos en color rojo claro los procesos manuales y en verde claro los semiautomatizados que los usuarios realizan para llevar a producción un modelo ML.

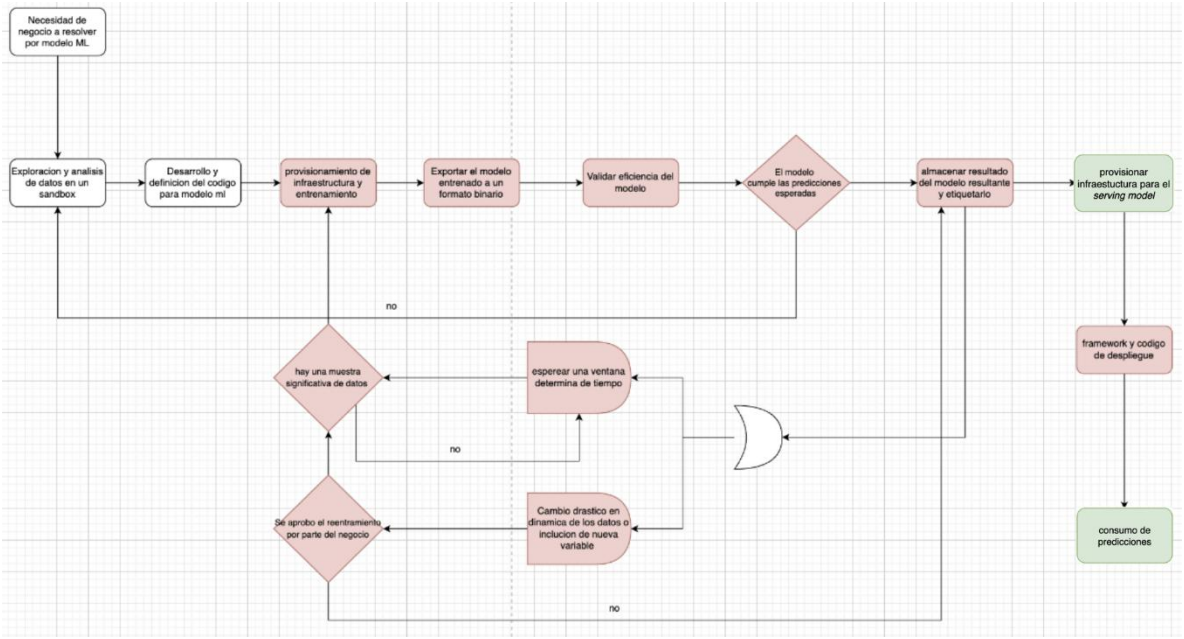

Figura 7. Madurez actual de los modelos de ML en su ciclo de vida.

El modelo base que se utilizará para probar el prototipo de MLOps fue desarrollado siguiendo el flujo anterior para cada una de sus etapas. Según la información recopilada del equipo de científicos de datos, el tiempo estimado invertido en cada fase del proceso —incluyendo la creación de los scripts del modelo y del servicio que expone las predicciones, la configuración de la infraestructura, la integración de pruebas unitarias y la validación de resultados— se resume en la siguiente tabla:

Tabla 1. Tiempos en cada etapa del ciclo de vida de ML antes de su automatización y estandarización.

Fase	**Duración**
Provisionamiento de infraestructura y ambiente de entrenamiento	1 día (esto tenía que ser con aprobación del equipo de *data platform*)
Entrenar y Validar eficiencia del modelo y aprobación	1 hora
Exportar el modelo a un formato binario	30 minutos
Gobierno de los modelos y experimentos (creación de artefactos,	No tenían establecido ninguna estrategia clara, lo hacían de forma

almacenar métricas, almacenar versionado de datos, *feature storing,* almacenar modelos)	manual y tomaba alrededor de 3 a 6 horas
Trigger de reentrenamiento del modelo	30 minutos (se ejecuta manualmente por demanda y se debía preparado el ambiente)
Estrategias de reentrenamiento con base a cambios en la dinámica de datos o degradación de las predicciones en el modelo	Realizaban análisis manuales de forma periódica mediante un *notebook*, este proceso tomaba alrededor de 3 horas dado que debía crear las pruebas para el análisis y ejecutar el modelo para el reentrenamiento.
Construcción del *framework* de la API para exponer el modelo	2 días

El tiempo total invertido por los equipos, incluyendo el de científicos de datos y el de *Data platform* (encargados de administrar la infraestructura en el ámbito de ML y datos), era de aproximadamente 3 días y 4 horas. Una vez consolidadas estas fases, cada experimento podía ejecutarse con los cambios requeridos. Sin embargo, era necesario realizar pruebas de integración y unitarias, lo que tomaba entre una o dos horas adicionales para garantizar el correcto funcionamiento del sistema.

5.2 Implementación de MLOps

Todas las tareas repetitivas en rojo claro de la figura 8 pueden automatizarse con herramientas gestionadas, no gestionadas e hibridas. Para este caso uso empresarial y luego estudiar diferentes alternativas que mejor se integraran con el ecosistema existente (Google Cloud Platform*)*, se optó por una solución hibrida en la cual se permitiera a los equipos de ciencia de datos flexibilidad a la hora de definir su modelo, variables de desempeño y hardware específicos con base a los requerimientos de entrenamiento. En la figura 7 se ilustra la arquitectura propuesta e implementada para el caso de uso enmarcado en la metodología listada en la fase 2. En esta sección se profundizará en cada en uno de los componentes de ella y se explicará el funcionamiento y el porqué de sus elecciones.

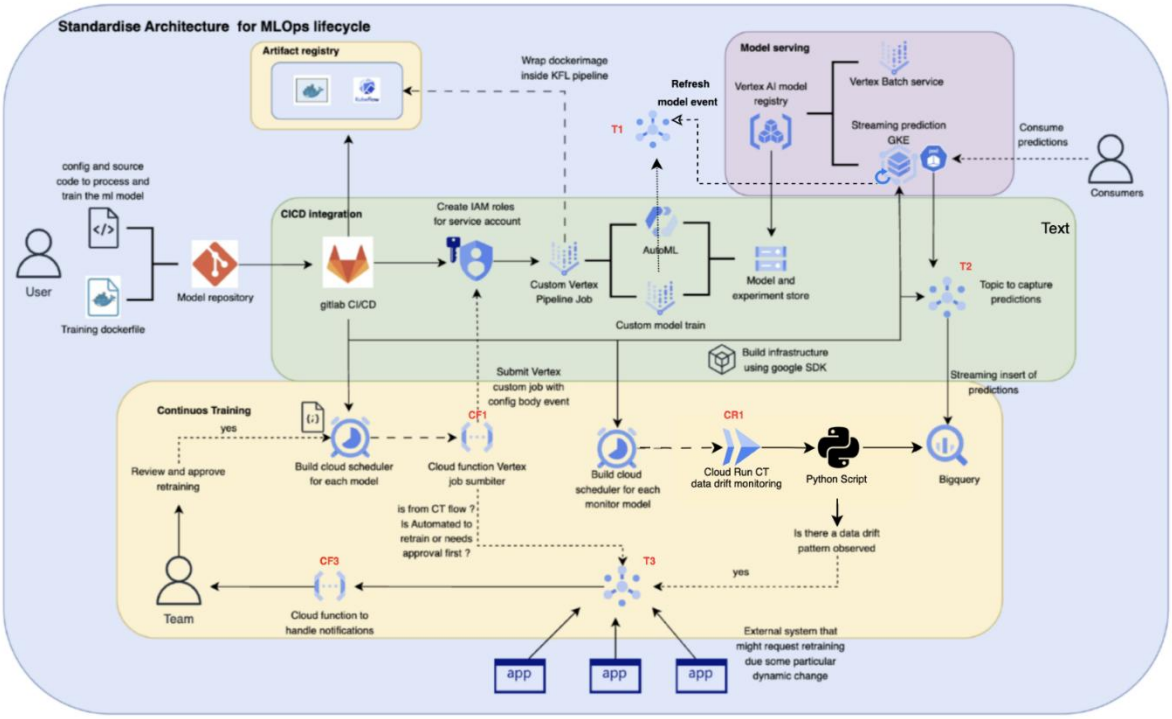

Figura 8. Arquitectura implementada acorde a la metodología propuesta de MLOps.

Plantilla de configuración del modelo

La configuración inicial se utilizará un archivo en formato YAML o JSON, el cual seguirá los lineamientos establecidos en el numeral 4.1 de la metodología propuesta (*low-code* como *no-code),* en los cuales incluirá parámetros adicionales como la métrica de monitoreo, la frecuencia de evaluación, los umbrales de sensibilidad y *bless model,* la máquina para el training y recursos para el *Model serving,* entre otros. Es importante destacar que toda la información definida en este archivo será utilizada por los pipelines de CI/CD para construir, mediante Google SDK (*Software Development Kit*), los recursos que habilitarán la automatización del ciclo de vida del modelo, principalmente a través de los Cloud Schedulers —servicios que actúan como disparadores del flujo, para este caso las Cloud Functions—, el *Model serving* y los tópicos (T1, T2 y T3).

A diferencia de las Cloud Functions (CFs) y ciertos accesos o roles predefinidos, estos recursos son aprovisionados de manera estática, antes de la ejecución de los pipelines de CI/CD, utilizando Terraform. Esta decisión responde a su rol fundamental como componentes encargados de gestionar las peticiones generadas por los distintos tópicos y *Schedulers*, ya sean creados de manera síncrona o asíncrona, permitiendo la ejecución de flujos personalizados a partir de los argumentos dinámicamente generados, como se ilustra en la figura 8.

- CF1: Gestiona y ejecuta los Custom Vertex Job.
- CF2: Gestiona y ejecuta las pruebas de *data drift.*
- CF3: Gestiona y ejecuta la lógica para generar alertas con base a peticiones externas o patrones de *data drift* para el reentrenamiento.

Para el resto de las plantillas base del código fuente, el *lenguaje* elegido fue Python por su alta popularidad y uso en el ámbito de *Machine learning*. Este incluye las librerías, definición de las clases y métodos para implementar el modelo juntos con los datos de entrenamiento y etapas de preprocesamiento, de tal forma que el usuario debe completarlo añadiendo su fuente de datos y las etapas de preprocesado que requiera. Todo está plantilla se construye junto a un "*Dockerfile"* para su posterior contenerizacion dentro del pipeline de Vertex AI.

Pipeline framework

Para automatizar el pipeline de entrenamiento, se seleccionó Kubeflow Pipelines (KFP) como la opción más adecuada para el proyecto, dado que al ser un lenguaje específico de dominio (DSL) permite a los científicos de datos definir los pasos necesarios para entrenar un modelo de ML, incluyendo pruebas, aprobación *(bless model)* e implementación. Una vez definido el pipeline, una herramienta gestionada de Google Cloud, conocida como Vertex AI Pipelines se encargará de provisionar los recursos de hardware necesarios para ejecutar cada etapa de forma aislada, simplificando la complejidad de configurar y administrar un clúster de Kubernetes nativo.

En comparación con otras herramientas de operacionalización, Kubeflow se destacó como la opción más adecuada para el proyecto debido a su integración nativa con Kubernetes y su capacidad para automatizar el pipeline de entrenamiento de modelos de ML. Aunque Airflow *y* Argo son herramientas populares para la orquestación de flujos de trabajo, no están diseñadas específicamente para *Machine learning* y no ofrecen la misma escalabilidad y control que Kubeflow. En particular, Airflow se centra en la gestión de flujos de trabajo generales, mientras que Argo es un motor de flujo de trabajo nativo de contenedores que, aunque es de código abierto, no está diseñado para el *Machine learning* [50].

La elección de Kubeflow se vio reforzada por su estrecha familiaridad con Google Cloud y su capacidad para proporcionar un alto control a los científicos de datos sobre el pipeline de entrenamiento. Además, Kubeflow utiliza Katib [26], un proyecto nativo de Kubernetes para el *Machine learning automated* (AutoML), que admite el ajuste de hiperparámetros, la detención temprana y la búsqueda de arquitectura neuronal (NAS) lo cual permite a los científicos de datos ajustar los hiperparámetros en múltiples marcos de *Machine learning*, incluyendo TensorFlow, MXNet, PyTorch y XGBoost, entre otros [51].

Ejecución de Entrenamiento

Finalmente, la plantilla definida para la ejecución periódica o el reentrenamiento se divide en dos componentes: una orientada al orquestador, compuesta por una Cloud Function (CF1) específica para cada dominio de negocio, y otra correspondiente al Cloud Scheduler.

La Cloud Function contiene la lógica encargada de crear y enviar la nueva tarea de reentrenamiento al servicio de Vertex AI, siempre y cuando exista una cantidad suficiente de nuevos datos disponibles —o al menos una cantidad cercana a la utilizada durante el entrenamiento inicial—. Esta función opera a partir de los parámetros recibidos en formato JSON, los cuales pueden ser enviados a través de una suscripción a una cola de mensajes (Pub/Sub) o mediante una invocación directa a través del Cloud Scheduler o un servicio externo.

Estos parámetros pueden ser introducidos como variables de ambiente en un trabajo personalizado de Kubeflow para luego ser usados con diferentes propósitos por el usuario final. Este enfoque ofrece flexibilidad en el proceso de reentrenamiento, ya que puede ejecutarse tanto bajo demanda como de forma periódica. Además, permite la integración con servicios externos que publiquen eventos en el tópico (T3) al que está suscrita la Cloud Function (CF1), con el objetivo de detectar comportamientos inusuales en los datos o cambios netamente de negocio. Ante tales eventos, se puede activar automáticamente un proceso de reentrenamiento del modelo, el cual inicia en un entorno no productivo y según los resultados obtenidos y la configuración definida, el modelo podrá avanzar directamente a producción o pasar previamente por una etapa de pruebas y validación por parte del equipo antes de su despliegue final (apartado *continuos training* figura 8).

Una vez construidas las plantillas, se integran para formar lo que posteriormente sera un pipeline MLOps que ejecutará un trabajo personalizado *(custom job)*, ya sea en demanda o periódica mediante agenda o eventos. Esta función combina el Kubeflow Pipeline y el “Dockerfile” en un Vertex Custom Job*,* permitiendo una ejecución eficiente y escalable del proceso de entrenamiento.

Pipeline de Entrenamiento

El usuario utilizará un Dockerfile que incluye todas las dependencias y bibliotecas necesarias para ejecutar cada etapa del proceso de entrenamiento. Este archivo ya ha sido estandarizado, utilizando una imagen base especializada en *Machine learning* e integrando comandos que permiten su construcción de manera automatizada. La imagen resultante será registrada en el repositorio de Artifact Registry, lo cual garantiza su disponibilidad, trazabilidad y consistencia.

Posteriormente, se definirán las etapas del entrenamiento utilizando el lenguaje de dominio específico (DSL, por sus siglas en inglés) de Kubeflow. Como resultado, se genera un pipeline compilado de Kubeflow, también almacenado en Artifact Registry en formato KFP (Kubeflow Pipelines). Cada pipeline se representa mediante un archivo YAML, lo que facilita su gestión, versionamiento y ejecución dentro del flujo automatizado de MLOps.

Una vez que se encuentren listos los dos artefactos clave, (la imagen de Docker y el pipeline compilado de Kubeflow) la Cloud Function se encargará de iniciar la ejecución del pipeline en la plataforma, utilizando los argumentos recibidos a partir del evento generado por el Cloud Scheduler o por un servicio externo. Estos argumentos incluyen principalmente la ruta de la última versión disponible tanto de la imagen como del pipeline (KFP). A partir de ese momento, Vertex AI se encargará de gestionar automáticamente todos los recursos de infraestructura necesarios para ejecutar las distintas etapas del pipeline, permitiendo así que los científicos de datos se concentren en el desarrollo y la mejora de los modelos.

Además, la facturación está basada en el tiempo efectivo de ejecución, lo que ofrece una mayor flexibilidad para escalar o reducir recursos según la demanda, sin necesidad de administrar manualmente un clúster de Kubernetes ni realizar reservas anticipadas. Cabe resaltar que, internamente, Vertex AI opera sobre una infraestructura basada en Kubernetes, pero completamente gestionada, lo cual simplifica significativamente su administración.

En la figura 9 se puede se puede observar diferentes experimentos ejecutados y propiedades de cada uno, del proceso descrito anteriormente una vez es ejecutado.

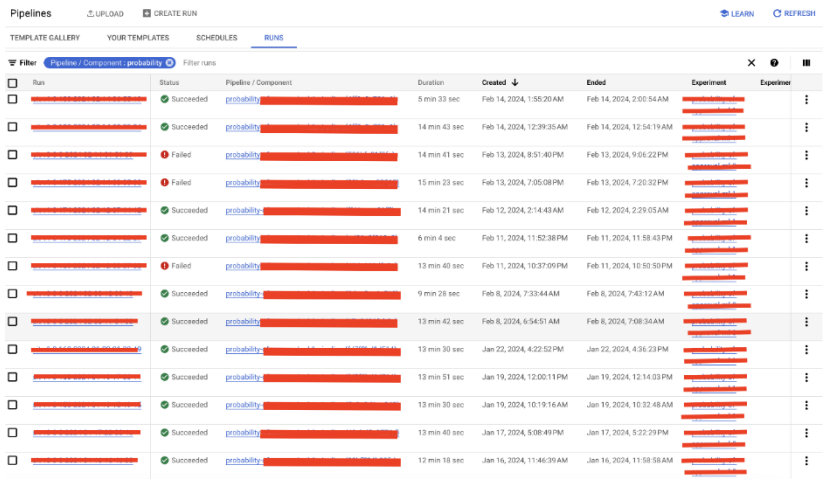

Figura 9. Lista de experimentos realizados con el modelo base.

Para cada experimento, es posible visualizar en detalle cada componente creado en el pipeline de Kubeflow y su estado de ejecución, como se muestra en la figura 10. En estos experimentos en particular, se ha configurado la ejecución de tres modelos en el archivo de configuración, lo que da lugar a tres etapas de entrenamiento que se ejecutan en paralelo. Cada una de estas etapas incluye procesos como la construcción de la *metadata*, la validación del modelo aprobado (*blessed model*), una etapa denominada "*explain predictions"* —centrada en la equidad y explicabilidad de las predicciones —, y, finalmente, una fase que abarca la actualización del modelo, tanto en los artefactos como en el *Model serving* encargado de disponibilizar la nueva versión.

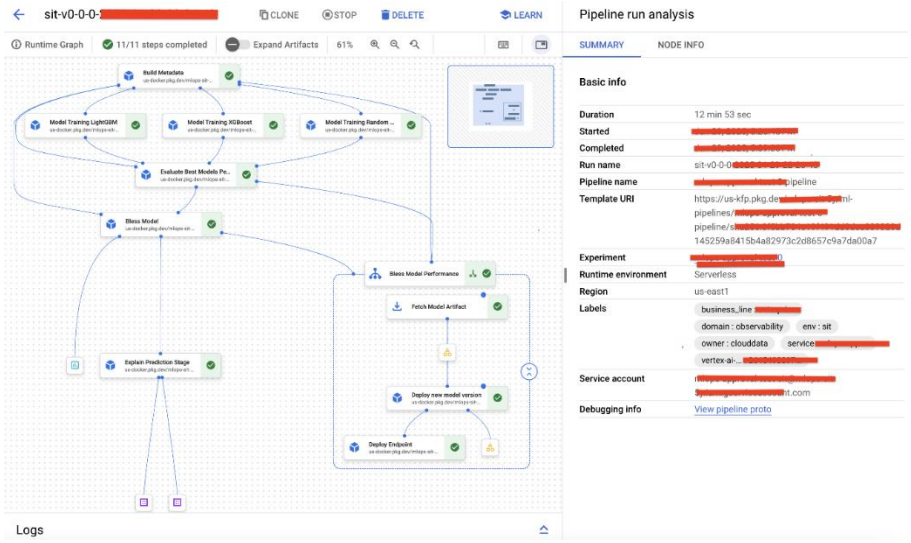

Figura 10. Kubeflow pipeline estandarizado para los modelos que se ejecuten (la etapa de entrenamiento varia con base a la cantidad de modelos configurados.

Model serving

Cuando el trabajo de entrenamiento finaliza, se genera automáticamente una nueva versión del modelo binarizado en el Vertex AI Model Registry. Posteriormente, este modelo, junto con los datos utilizados en el entrenamiento, se almacena en Google Cloud Storage (GCS) para garantizar su trazabilidad, consumo y distribución.

Una vez almacenado en la nueva ruta, se ejecuta la fase final del Vertex Job indicada en la figura 10, denominada *"Deploy Endpoint"*, la cual publica un mensaje en un tópico (T1) de Pub/Sub con la nueva ruta y su *metadata*. A este tópico estará suscrito el servicio donde el modelo estará desplegado en Google Kubernetes Engine (GKE) y listo para ejecutar un proceso en caso de recibirlo. Sin embargo, si es la primera vez que se realiza el despliegue, el modelo usará la versión inicial por defecto y esta etapa se omitirá.

En caso de recibir el evento, el servicio invocara internamente un método que carga la nueva versión del modelo para su uso inmediato en predicciones en tiempo real. Lo anterior se puede apreciar a detalle en la figura 11. Por otra parte, las predicciones en lotes, en cambio, utilizará la funcionalidad de Vertex AI Batch Predictions.

La decisión de utilizar una arquitectura basada en microservicios para el despliegue de predicciones en tiempo real respondió principalmente a dos factores. En primer lugar, la organización requería una solución robusta, con potencial de escalar hacia el uso de GPU en el futuro, y que además se integrara adecuadamente con el ecosistema de Kubernetes, considerando que gran parte de los servicios existentes se encontraban en proceso de migración hacia dicha plataforma, mientras que los modelos de *Machine learning* aún se ejecutaban en Cloud Run.

En segundo lugar, se priorizó la portabilidad y la optimización de costos. Aunque Vertex AI ofrece un servicio para predicciones en línea, este implicaba un alto costo operativo, ya que el cobro se basa en tres factores: el tiempo de actividad del endpoint, la máquina utilizada, y la cantidad de predicciones realizadas. En contraste, el uso de GKE permite pagar únicamente por el uso efectivo de recursos (RAM y CPU) mientras se reciben solicitudes, lo que permite que los recursos no utilizados puedan ser aprovechados por otros servicios dentro del mismo clúster. Esta diferencia se vuelve crítica cuando se manejan múltiples modelos, ya que en Vertex AI sería necesario habilitar un endpoint por cada uno, lo cual resultaría insostenible económicamente a largo plazo para la organización.

Predicciones en tiempo real

Para el caso de predicciones en tiempo real, un microservicio es desplegado para cargar el modelo mediante el CI/CD pipeline en un entorno de Kubernetes en la nube (Google Kubernetes Engine), el cual expondrá un endpoint a través de una de una red privada para garantizar la seguridad y bloqueo ante peticiones externas. Estos microservicios se construyen con FastAPI, utilizando una plantilla que adapta el entorno de ejecución a la tecnología en la que fue desarrollado el modelo a servir, lo que asegura la instalación de las dependencias y librerías necesarias para su correcto funcionamiento. Cabe mencionar que las plantillas construidas en el prototipo tendrían soporte únicamente para modelos que sean desarrollados en Python, usando librerías como skit-learn, TensorFlow y Keras.

De esta manera, cada nuevo modelo desplegado funcionaría como un microservicio independiente, permitiendo la generación de predicciones en tiempo real y aislamiento de cargas. Este contendrá tres métodos principales como se aprecia en la figura 10, el primero seleccionar una versión de modelo diferente —*rollback, s*olo puede ser realizado por un administrador en entornos productivos—; el segundo para actualizar con la última versión cuando hay un entrenamiento nuevo, y por último uno "*get"* para obtener las predicciones. No obstante, en una arquitectura de inferencia en tiempo real, surgen desafíos adicionales en términos de concurrencia y rendimiento. Si bien Kubernetes permite la escalabilidad automática mediante HPA (Horizontal Pod Autoscaler), el prototipo desarrollado no incorporó un mecanismo para preservar el estado en cada pod (unidad básica de ejecución en Kubernetes).

Esto significa que, al escalar dinámicamente dentro del *namespace*, las nuevas réplicas no conservarían en memoria la versión del modelo ya cargado, lo que podría generar fallos en la respuesta a múltiples solicitudes simultáneas. Para abordar esta limitación, sería necesario un enfoque *stateful*, en el cual los pods compartieran el estado del modelo mediante volúmenes persistentes o estrategias de sincronización entre réplicas.

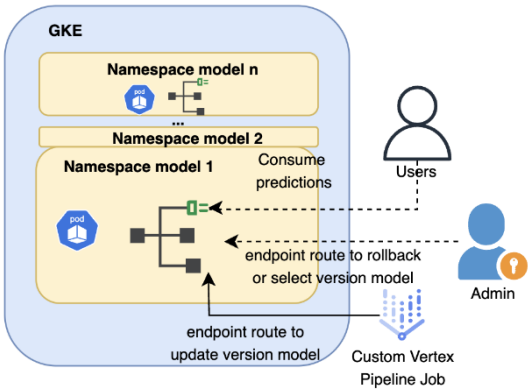

Figura 11. Componentes funcionales del Model serving.

Predicción por lotes

Por otro lado, para las predicciones en lotes, la funcionalidad de predicción por lotes ofrecida por la plataforma de Vertex AI garantiza el manejo de grandes volúmenes de datos, dado que, al ser completamente gestionado se encarga de aprovisionar toda la infraestructura, endpoints a exponer, realizar las predicciones y garantizar la eliminación de los recursos de infraestructura una vez finalizado el trabajo de predicción; es decir, solo se generan cargos por el tiempo que tarde en generarlas [52].

Para habilitar esta funcionalidad de Vertex, se puede especificar el tipo de máquina en la que se ejecutará el trabajo, así como los aceleradores opcionales (GPUs o TPUs) para optimizar el rendimiento en inferencias a gran escala. Basta con proporcionar un *bucket* (GCS) con las muestras a predecir o una tabla en BigQuery, y Vertex AI se encargará de gestionar el procesamiento en segundo plano. Al finalizar la tarea, las predicciones estarían disponibles en el mismo *bucket* o en una nueva tabla de BigQuery, permitiendo su integración con otros procesos analíticos o de reportes. Asimismo, esta arquitectura facilita la escalabilidad, ya que permite manejar grandes volúmenes de datos sin necesidad de administrar directamente la infraestructura subyacente.

Integración con CI/CD/CT

El proceso descrito anteriormente es posible mediante la integración de un pipeline de CI/CD, el cual se encargará de actualizar de forma iterativa todos los cambios que reciba tanto el modelo como el archivo de configuración, partiendo del código fuente, pruebas,

paquetes y los artefactos que estos generan al crear nuevas versiones de las imágenes en el Google Artifact Registry, posteriormente serán usadas por los pipelines de Vertex AI y *Model serving.*

En la Figura 12 se detalla la etapa de Integración Continua (CI), donde se aplican diversas validaciones internas. Estas incluyen la verificación de parámetros clave como el modelo aprobado (*bless model*), el tipo de máquina utilizada y la frecuencia de entrenamiento, asegurando que se ajusten a los valores permitidos para el entorno de despliegue y las necesidades del modelo.

Además, en esta fase se generan automáticamente las variables por defecto del entorno, las plantillas de los Cloud Schedulers según la información suministrada por el usuario, y la imagen que será utilizada tanto para el entrenamiento como para la inferencia. Dicha imagen queda registrada en Google Artifact Registry, para su posterior uso en las etapas del Vertex Custom Job y en el proceso de entrega continua (CD), donde el modelo es desplegado como servicio en Google Kubernetes Engine (GKE).
.

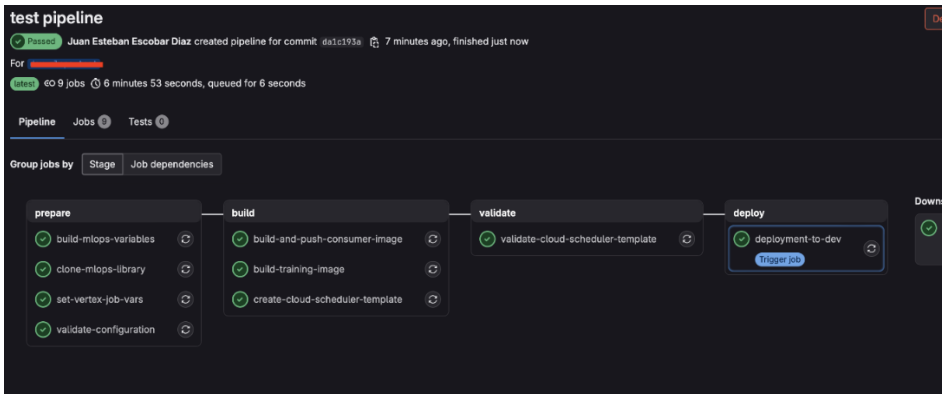

Figura 12. CI pipeline para arquitectura propuesta.

La Figura 13 muestra la integración con la etapa de Entrega Continua (CD), que abarca el despliegue de los artefactos generados durante la fase de CI y almacenados en Artifact Registry. Estos artefactos se implementan sobre una infraestructura previamente definida según el entorno objetivo, incluyendo servicios como:

- Microservicio de inferencia desplegados en GKE.
- Agendamiento de tareas mediante Cloud Scheduler.
- Tópicos de mensajería utilizados para monitoreo, actualización, recepción de eventos externos y ejecución de reentrenamientos.
- El nuevo Kubeflow Pipeline que será consumido por Vertex AI al momento de iniciar el trabajo de entrenamiento.

Finalmente, se ejecuta una etapa de verificación (*health-check*) para asegurar que los servicios desplegados existan y se encuentren operando correctamente.

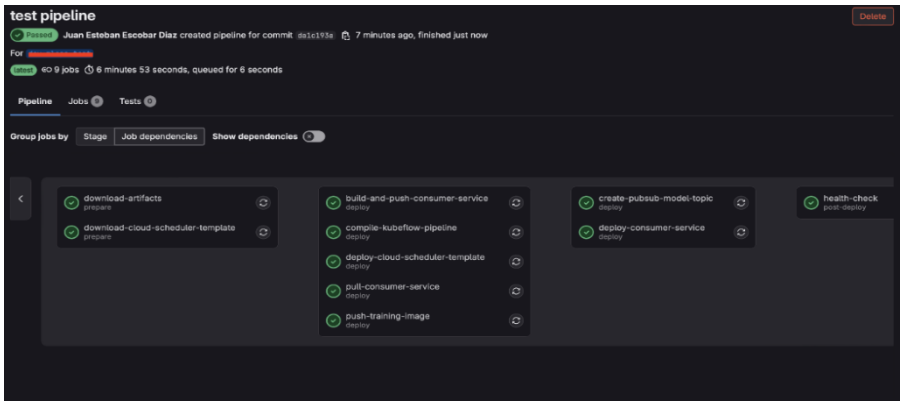

Figura 13. CD pipeline para arquitectura propuesta.

Para el entrenamiento continuo (CT), se optó por un enfoque diferente al que ofrece Vertex AI Monitoring, ya que no se utilizó la funcionalidad de predicciones en línea de Vertex, la cual permite habilitar el monitoreo directamente sobre el endpoint del modelo y acceder a métricas relacionadas con patrones de *bias* o *data drift*. En su lugar, dado que los modelos se encuentran expuestos como microservicios en GKE, se implementó una solución similar utilizando Cloud Functions, BigQuery y Pub/Sub.

El proceso comienza cuando un modelo como servicio (microservicio) en GKE recibe solicitudes de inferencia y publica las características de entrada y la predicción en un tópico (T2) de Pub/Sub. Estos datos se cargan automáticamente en una tabla de BigQuery mediante una suscripción configurada, lo que permite su análisis posterior. Para evaluar la estabilidad del modelo, se emplea una plantilla usando BigQuery y Python para comparar la distribución de los datos actuales con los datos de entrenamiento, tomando en cuenta que esta versión inicial incluirá métricas simples como *media* y *desviación estándar*, *Jensen–Shannon Divergence (JSD)* y *Kolmogorov–Smirnov (KS)* para identificar posibles patrones de *data drift*.

Adicionalmente, una Cloud Function (CF2) ejecuta de manera periódica este flujo para calcular la magnitud de la desviación y, en caso de detectar cambios significativos respecto al umbral especificado por el usuario y el tipo de métrica, generará un evento que podría o no ser procesado para un reentrenamiento automático (dependiendo de la configuración del usuario), o, en su defecto, generar una alerta (CF3) y pasar a un estado de cuarentena donde el equipo a cargo evaluará y decidirá si aprobar el reentrenamiento de forma manual, como se aprecia en la figura 8 bajo el apartado de "*continuous training"*.

La elección de Cloud Run en lugar de Cloud Functions para ejecutar esta tarea de monitoreo se debe a la naturaleza más intensiva y prolongada del proceso, tanto en términos de cálculo como de volumen de datos a procesar. Cloud Run ofrece mayor

flexibilidad al escalar automáticamente y permitir ejecuciones de mayor duración, lo que lo hace adecuado para tareas como el análisis periódico de desviaciones o la lectura de grandes volúmenes históricos. En contraste, Cloud Functions está optimizado para flujos más livianos y de corta duración —como disparar un flujo ante un evento puntual o enviar una notificación—, dado su límite máximo de ejecución de 9 minutos.

6. RESULTADOS

La implementación de esta arquitectura basada en prácticas de MLOps tuvo un impacto significativo en aspectos clave como la eficiencia del despliegue de modelos, la reducción de errores y la aceleración del tiempo de comercialización. En el caso del modelo base utilizado en el prototipo, el análisis se centró específicamente en la reducción del tiempo necesario para ejecutar y desplegar cambios en los distintos entornos. Como se documenta en la Tabla 1, el tiempo estimado para realizar un primer despliegue era anteriormente de aproximadamente tres días. Posteriormente, los despliegues subsiguientes podían tardar entre dos y tres horas adicionales.

Tras automatizar el ciclo de vida de ML y realizar varios experimentos con el modelo base, se identificaron las fases en las que se resumieron los pasos previamente ejecutados de forma manual, los cuales se detallan en la Tabla 1 y fueron posteriormente automatizados. Es importante señalar que estos valores fueron estimados para el modelo base y pueden variar dependiendo del modelo utilizado y de las dependencias o requisitos específicos. Los valores aproximados se presentan en la Tabla 2.

Tabla 2. Tiempos en cada etapa del ciclo de vida de ML después de la automatización y estandarización

Fase	**Duración**
Configuración de parámetros de entrada	5 minutos
CI	6-10 minutos (varía dependiendo de los requerimientos del modelo que se esté construyendo)
CD	7-10 minutos
Vertex AI Pipeline (Kubeflow) incluye training y la todo los que abarca la sección de gobierno de modelos y experimentos	12 - 20 minutos (varia con la cantidad de datos de entrenamiento, modelos usados e hiperparámetros)
Data drift validator para reentrenamiento y generación de alerta	2-3 minutos
Recursos Estáticos (Terraform)	Una sola vez, por tanto, no se incluye

Al comparar los tiempos totales de *time-to-market* antes y después de la implementación de prácticas MLOps, se observa una mejora significativa. En el proceso tradicional, la etapa inicial de configuración de todo el entorno y componentes requería aproximadamente tres días. Una vez completada, la ejecución de posteriores experimentos de principio a fin tomaba alrededor de 3 horas y 30 minutos.

Con la nueva arquitectura basada en MLOps, el proceso de configuración inicial se ha reducido drásticamente: ahora basta con definir un archivo de configuración, lo que toma entre 5 y 10 minutos. A partir de allí, los experimentos subsecuentes pueden ejecutarse en un promedio de 48 minutos, gracias a la automatización del flujo de trabajo.

Esto representa una reducción sustancial en los tiempos operativos, especialmente en la fase de experimentación, con una disminución de aproximadamente un 77% en la duración de cada ciclo experimental.

Aunque la única variable cuantitativa documentada en la fase práctica del modelo base fue la reducción del tiempo operativo, esta limitación se debió principalmente a la ausencia de registros históricos sobre otras variables clave, como la cantidad de despliegues fallidos, los tiempos de entrenamiento y los recursos computacionales utilizados por cada modelo. Esta situación fue resultado de una gestión deficiente de los registros de entrenamiento (*logs*) y de la falta de etiquetado en los recursos provisionados para los experimentos. Si bien las métricas de costos eran almacenadas, no era posible identificar con precisión qué recursos fueron utilizados por cada modelo ni durante cuánto tiempo, lo que dificultó el cálculo de métricas relacionadas con el consumo y los costos operativos. Por esta razón, se optó por incluir una evaluación cualitativa a través de la percepción de los usuarios, con el objetivo de evidenciar el impacto de los cambios implementados en su experiencia cotidiana durante el desarrollo y despliegue de modelos a producción.

Percepción de la comunidad de científicos de datos

A continuación, se presentan los principales hallazgos del estudio, los cuales se complementan con la perspectiva del usuario final mediante una encuesta aplicada a diez miembros del equipo de científicos de datos de la organización, de los cuales ocho respondieron.

La Figura 14 presenta la distribución de los usuarios según el grado de automatización en las distintas fases del ciclo de vida del modelo. Se identifican tres niveles: ejecución manual, ejecución parcialmente automatizada y ausencia total de un proceso implementado. Es importante destacar que, aunque algunas etapas fueron reportadas como parcialmente automatizadas, no existe un proceso estandarizado adoptado por toda la comunidad de científicos de datos. En la práctica, un nuevo integrante del equipo debe crear manualmente su propio flujo o replicar partes de los existentes antes de alcanzar una fase semiautomatizada.

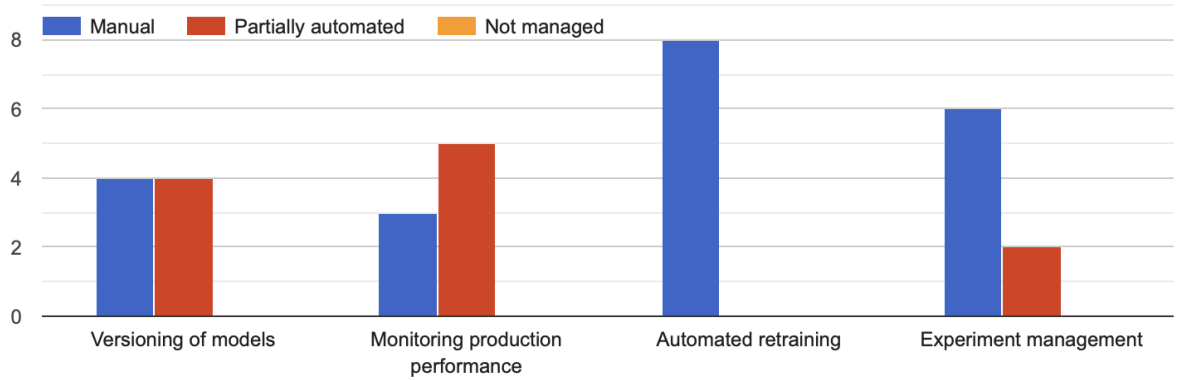

Figura 14. Distribución de los usuarios según el grado de automatización en algunas fases para el desarrollo y despliegue de modelo de Machine learning.

Tiempo de desarrollo y despliegue de modelos

Antes de la estandarización, el 80% respondió que el tiempo promedio para desarrollar, entrenar y desplegar un modelo en producción era alrededor de entre 2 semanas y un mes, mientras que el 20% menciono que le tardaba más de 1 mes considerando etapas manuales como validación y pruebas de integración. Tras la implementación del proceso estandarizado, este tiempo según la perspectiva del de la mayoría de los usuarios disminuyó entre un 50% y 75% el tiempo dado que les estaba tomando alrededor de menos de una semana y máximo dos semanas en contraste con el mes, gracias a la automatización de los pipelines y la adopción de herramientas como Kubeflow y Vertex AI. Cabe mencionar que tiempo que experimentado desde lo practico fue de aproximadamente de 3 días teniendo que ya se tenían definidos un pool de datos de entrada del modelo ya se estaba realizando *feature engineering* la cual puede ocupar gran parte del tiempo a la hora de construir un modelo) y se procedía a realizar únicamente experimentos y posteriormente el despliegue.

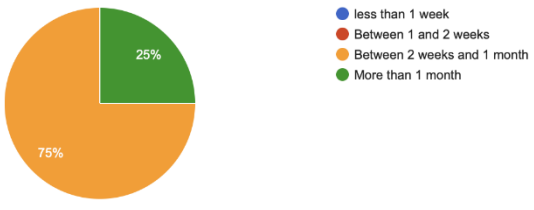

Figura 15. Tiempo que tomaba desarrollar y desplegar un modelo antes de MLOps.

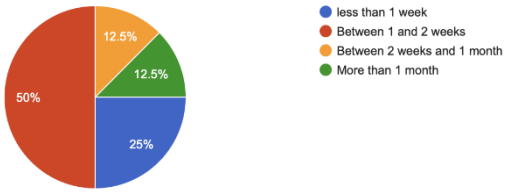

Figura 16. Tiempo que tomaba desarrollar y desplegar un modelo después de MLOps

De forma más granular para cada una de las etapas más críticas que se automatizaron durante la estandarización evaluamos el tiempo de que les solía tomar cada una de ellas siendo la automatización del reentrenamiento la más crítica junto con el manejo de

experimentos la cual está ligada en general el gobierno y adecuación de los ambientes y máquinas usadas para cada uno de ellos. La mayoría de los usuarios encuestados en la figura 17 respondieron que les tomaba alrededor de 3 a 6 horas establecer el flujo de reentrenamiento, y más un de un día el flujo para un adecuado manejo de experimentos, sin embargo, estos flujos debían ser ejecutados de forma manual.

Estas etapas evidenciaron una reducción significativa —principalmente en el manejo de experimentos y la automatización del reentrenamiento, como se muestra en la figura 18— con una disminución de más del 75 % en el tiempo invertido en estas tareas. Sin embargo, algunos usuarios no experimentaron tal impacto en esta última, lo cual podría deberse a que, en modelos anteriores, debieron implementar procesos externos muy personalizados —como *data drift*, análisis estadísticos u otros— que no son cubiertos por este prototipo.

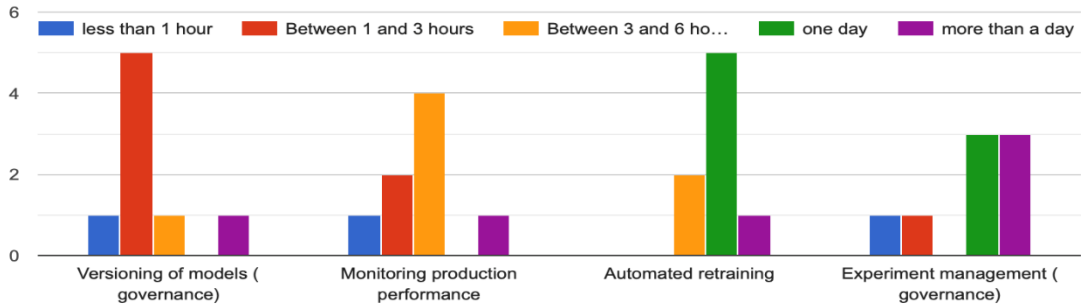

Figura 17. Tiempo que tardaban los equipos de Científicos de datos antes de estandarizar MLOps.

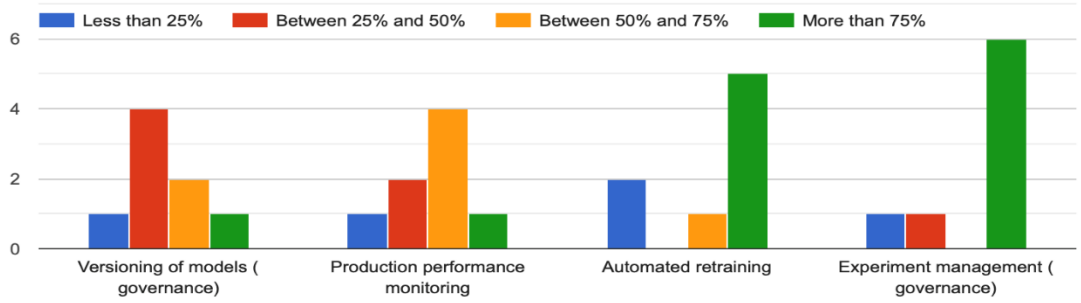

Figura 18. Porcentaje de reducción del tiempo que les toma para cada etapa luego de estandarizar MLOps.

Frecuencia de errores

Otro aspecto relevante, como se muestra en la figura 19, es la notable disminución en la frecuencia de errores durante el desarrollo y despliegue de modelos a producción. Según los resultados obtenidos, el 75 % de los usuarios percibió una reducción de aproximadamente el 50 % en la incidencia de fallos. En términos prácticos,

anteriormente entre 7 y 8 de cada 10 despliegues presentaban algún tipo de error, mientras que actualmente esa cifra ha disminuido a solo 3 o 4 por cada 10. Esta mejora se atribuye principalmente a la integración de validaciones automáticas en las etapas del flujo CI/CD, las cuales permiten detectar errores de forma anticipada. Los fallos más comunes estaban relacionados con configuraciones incorrectas, problemas en la integración con APIs de consumo, falta de recursos para cargar los modelos o incompatibilidades entre dependencias.

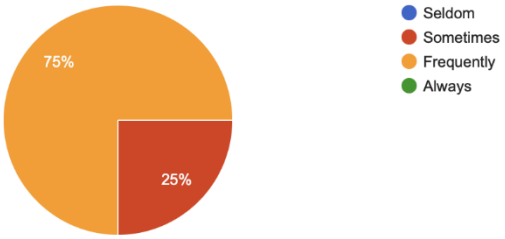

Figura 19. Percepción de errores durante el desarrollo y despliegue antes de implementar MLOps.

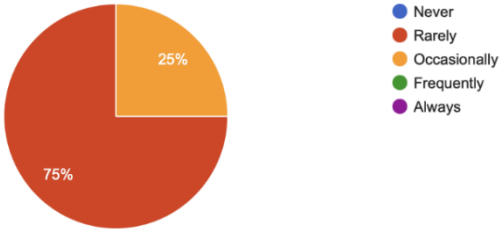

Figura 20. Percepción de errores durante el desarrollo y despliegue después de implementar MLOps.

La disminución de errores se refuerza con el hecho de que la detección de estos se realiza de forma más rápida, dado que todo el sistema cuenta con adecuados *logs* y alertas al momento de ejecución del pipeline y estado del modelo desplegado. Anteriormente tardaban más de 3 horas y con el sistema de monitoreo el 75% de los usuarios se tarda menos de 1 hora y un 12.5% entre 1 y 3 horas como se aprecia en la figura 21. Además, Vertex AI proporciona para cada experimento métricas sobre el estado del modelo, los parámetros y los datos de entrada, lo que brinda trazabilidad en cada fase del ciclo de vida del modelo.

Esta trazabilidad permite identificar de manera más eficiente posibles problemas en la calidad de los datos, el ajuste de los hiperparámetros o incluso patrones de *data drifts* del modelo en producción. Así mismo, al estandarizar el flujo de MLOps, se optimiza la detección de fallos en la inferencia y en el consumo de los recursos, asegurando que cualquier degradación en el desempeño del modelo pueda ser corregida rápidamente. Todo esto se traduce en un sistema más confiable, con menos incidentes y una mayor capacidad de respuesta ante cambios en los datos en entornos de producción.

En la figura 22 se muestra el grado de satisfacción con el sistema de monitoreo. Si bien el 37.5% de los encuestados manifestó estar parcialmente satisfecho —lo que indica una mejoría, aunque aún quedan aspectos por mejorar como la integración de tableros como Looker o PowerBI—, el 62.5% expresó estar completamente satisfecho, lo que refleja una buena recepción por parte de los científicos de datos encuestados

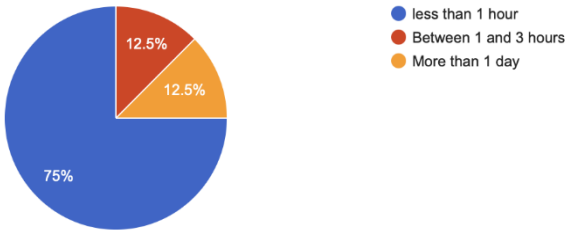

Figura 21. Tiempo que toma actualmente rastrear e identificar errores.

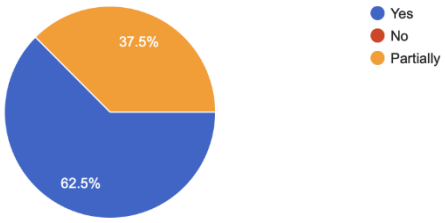

Figura 22. Porcentaje de satisfacción respecto al actual sistema de monitoreo para prevenir errores.

Impacto en el equipo y flujo de trabajo

Se observó una mejora en la percepción de los usuarios sobre la calidad del modelo y el desempeño de las predicciones, con un 62.5% de los usuarios reportando mejoras significativas, mientras que un 25% notó mejoras moderadas respecto a versiones anteriores. Esto podría estar relacionado con la facilidad para generar nuevos experimentos utilizando herramientas como AutoML, así como la eficiencia de Kubeflow para realizar ajustes de hiperparámetros. Sin embargo, estos resultados requieren un análisis más detallado para confirmar su impacto real en el desempeño del modelo.

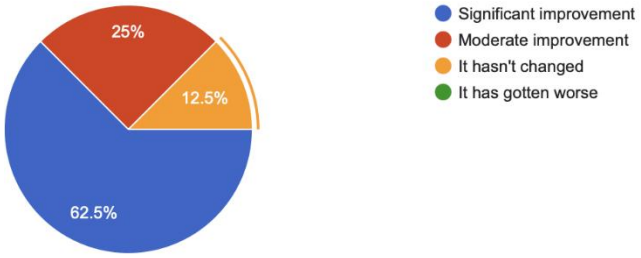

Figura 23. Percepción de mejora en la calidad de los modelos de Machine learning.

7. CONCLUSIONES

La automatización del ciclo de vida completo de los modelos de *machine learning*, implementada mediante Google Cloud Platform (GCP), Vertex AI y Kubeflow, ha permitido optimizar y estandarizar de manera significativa los procesos de entrenamiento y despliegue. La integración de Kubeflow ha facilitado una gestión eficiente de los *pipelines* de entrenamiento, garantizando coherencia en cada etapa del ciclo de vida del modelo. Asimismo, la adopción de un enfoque basado en microservicios para el *model serving* ha mejorado la disponibilidad de los modelos, al tiempo que abre posibilidades de escalabilidad futura. Por otro lado, el uso del servicio de *batch predictions* de Vertex AI ha incrementado la eficiencia en el procesamiento de grandes volúmenes de datos, gracias a su naturaleza completamente gestionada y sin necesidad de administración de infraestructura.

En línea con estos resultados, y a partir de las diversas arquitecturas revisadas en la literatura, se concluyó que la combinación de Vertex AI con Kubeflow ofrece una solución escalable adecuada para gestionar diferentes modelos. Sin embargo, en contextos donde los casos de uso requieren configuraciones altamente personalizadas, es más conveniente optar por soluciones *on-premises* o IaaS, ya que proporcionan mayor control sobre la infraestructura y los recursos utilizados. Esta alternativa resulta especialmente útil cuando se emplean recursos intensivos como TPUs o GPUs, cuyo uso continuo en la nube puede representar un costo considerable a largo plazo.

Por otra parte, al estandarizar el proceso para todos los científicos de datos, se identificó una variabilidad en algunas respuestas de la encuesta, particularmente en la forma en que se abordaban ciertas etapas del ciclo de vida de los modelos, como se muestra en la figura 14. Este hallazgo sugiere que no toda la comunidad de científicos de datos estaba alineada con los procesos de desarrollo y despliegue, existiendo silos operativos entre equipos. En este sentido, incluso sin procesos completamente automatizados, es crucial establecer estándares o lineamientos compartidos para facilitar la reutilización de componentes y reducir fricciones en el trabajo colaborativo.

A pesar de haber logrado una estandarización desde una perspectiva metodológica, la correcta adopción y sostenibilidad de los modelos a largo plazo sigue dependiendo, en gran medida, de la cultura organizacional y de los mecanismos que permitan identificar sesgos o modelos desactualizados. Aunque se cuente con herramientas avanzadas, una configuración incorrecta puede derivar en la obsolescencia temprana del modelo, afectando su rendimiento en producción.

Adicionalmente, la implementación del nuevo flujo de trabajo estandarizado favoreció una colaboración más efectiva entre los equipos de desarrollo y operaciones. Aunque este punto no fue evaluado directamente en la encuesta, se observó una notable reducción en las solicitudes de soporte debido a la eliminación de tareas manuales por parte del equipo de operaciones, como la creación de recursos para cada usuario o

equipo. Además, se evidenció una disminución de errores tanto en entornos productivos como de desarrollo, percepción que sí fue capturada por los usuarios encuestados. Esto se relaciona con el hecho de que el equipo encargado del nuevo proceso obtuvo un mayor control y gobernanza, facilitando así la gestión de excepciones y corrección de errores.

No obstante, cabe destacar que si bien se utilizó Kubernetes (GKE) como plataforma de despliegue en el prototipo, esta opción no es necesariamente la más adecuada para todos los contextos. Su implementación implica un nivel de complejidad técnica considerable, que puede representar una carga adicional para equipos con experiencia limitada en infraestructura y orquestación de contenedores.

Por ello, se sugiere considerar alternativas más accesibles y flexibles como Cloud Run (en Google Cloud), AWS ECS (en Amazon Web Services) o Azure App Service (en Microsoft Azure) para las primeras fases de adopción. Estas opciones serverless o gestionadas permiten acelerar la puesta en marcha de servicios sin comprometer la escalabilidad futura, reduciendo así las barreras técnicas iniciales y facilitando un enfoque iterativo y evolutivo hacia arquitecturas más complejas.

Finalmente, si bien la arquitectura propuesta ha mostrado ser eficaz para modelos tabulares y de procesamiento de texto, una de las principales lecciones aprendidas ha sido la necesidad de mantener la flexibilidad para expandirse hacia nuevos dominios de aplicación. En este contexto, se recomienda explorar la incorporación de agentes personalizados en futuras implementaciones, permitiendo especializar modelos en áreas como atención al cliente u otros sectores con requerimientos específicos.

Esta visión a futuro se refuerza con la evolución de ecosistemas como los *foundation models* y herramientas como Gemini AI, que posibilitan el uso de modelos preentrenados de gran escala. Su integración puede representar una ventaja competitiva al reducir tiempos de desarrollo, aumentar la adaptabilidad de los modelos y facilitar la personalización en múltiples contextos empresariales.

8. REFERENCIAS

[1] G. Symeonidis, E. Nerantzis, A. Kazakis and G. A. Papakostas, "MLOps - Definitions, Tools and Challenges," in 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2022, pp. 0453–0460, doi: 10.1109/CCWC54503.2022.9720902.

[2] S. Alla and S. Adari, Beginning MLOps with MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure. Springer, 2021, doi: 10.1007/978-1-4842-6549-9.

[3] O. Spjuth, J. Frid and A. Hellander, “The Machine learning life cycle and the cloud: implications for drug discovery,” Expert Opinion on Drug Discovery, vol. 16, pp. 1–9, 2021, doi: 10.1080/17460441.2021.1932812.

[4] "Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming," in Artificial Intelligence in Design '96, Dordrecht: Springer, 1996, pp. 151–170, doi: 10.1007/978-94-009-0279-4_9.

[5] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning. MIT Press, 2016, ch. 8.
[6] J. Dean et al., “Large-Scale Machine learning on Heterogeneous Distributed Systems,” in Adv. Neural Inf. Process. Syst., vol. 28, 2012.

[7] "GPUs Continue to Dominate the AI Accelerator Market for Now," InformationWeek, Dec. 2019. [Online]. Available: https://www.informationweek.com

[8] A. Paleyes, R.-G. Urma and N. D. Lawrence, "Challenges in deploying Machine learning: a survey of case studies," 2021.

[9] A. R. Munappy, J. Bosch, H. H. Olsson, A. Arpteg and B. Brinne, "Data management for production quality deep learning models: Challenges and solutions," *Journal of Systems and Software*, vol. 191, p. 111359, 2022, doi: 10.1016/j.jss.2022.111359.

[10] E. Raj, D. Buffoni, M. Westerlund and K. Ahola, "Edge MLOps: An Automation Framework for AIoT Applications," in 2021 IEEE International Conference on Cloud Engineering (IC2E), San Francisco, CA, USA, 2021, pp. 191–200, doi: 10.1109/IC2E52221.2021.00034.

[11] A. Raj et al., “From Ad-Hoc Data Analytics to DataOps,” 2020, doi: 10.13140/RG.2.2.36807.93604.

[12] D. Kreuzberger, N. Kühl and S. Hirschl, “Machine learning Operations (MLOps): Overview, Definition, and Architecture,” IEEE Access, vol. 11, pp. 31866–31879, 2023, doi: 10.1109/ACCESS.2023.3262138.

[13] C. Ebert, G. Gallardo, J. Hernantes and N. Serrano, “DevOps,” IEEE Software, vol. 33, no. 3, pp. 94–100, May 2016, doi: 10.1109/MS.2016.68.

[14] B. B. N. de França, H. Jeronimo and G. H. Travassos, “Characterizing DevOps by Hearing Multiple Voices,” in Proceedings of the 38th International Conference on Software Engineering Companion, 2016, doi: 10.1145/2973839.2973845.

[15] Google, “MLOps: Continuous delivery and automation pipelines in Machine learning,” [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[16] “The importance of having a feature store,” Towards Data Science, [Online]. Available: https://towardsdatascience.com/the-importance-of-having-a-feature-store-e2a9cfa5619f

[17] D. Sculley et al., “Hidden Technical Debt in Machine learning Systems,” in Adv. Neural Inf. Process. Syst., vol. 28, 2015.

[18] A. Crivellaro et al., “Model serving: A Machine learning Perspective,” 2019.

[19] T. Schlossnagle, “Monitoring in a DevOps world,” Commun. ACM, vol. 61, no. 3, pp. 58–61, Mar. 2018, doi: 10.1145/3168505.

[20] M. Treveil et al., “Introducing MLOps: how to scale Machine learning in the enterprise,” Accessed: Feb. 26, 2023.

[21] W. E. Moutaouakal and K. Baïna, "Comparative Experimentation of MLOps Power on Microsoft Azure, Amazon Web Services, and Google Cloud Platform," in 2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech), Marrakech, Morocco, 2023, pp. 1–8, doi: 10.1109/CloudTech58737.2023.10366138.

[22] M. John, H. Olsson and J. Bosch, "Towards MLOps: A Framework and Maturity Model," 2021, doi: 10.1109/SEAA53835.2021.00050.

[23] Kubeflow, “About Kubeflow,” Kubeflow Documentation. [Online]. Available: https://www.kubeflow.org/docs/started/introduction/

[24] —, “Installing Kubeflow,” Kubeflow Documentation. [Online]. Available: https://www.kubeflow.org/docs/started/installing-kubeflow/

[25] —, “Kubeflow Architecture,” Kubeflow Documentation. [Online]. Available: https://www.kubeflow.org/docs/started/architecture/

[26] —, “Katib,” Kubeflow Documentation. [Online]. Available: https://www.kubeflow.org/docs/components/katib/overview/

[27] —, “Kubeflow Pipelines,” Kubeflow Documentation. [Online]. Available: https://www.kubeflow.org/docs/components/pipelines/

[28] Databricks, “Managed MLflow.” [Online]. Available: https://www.databricks.com/br/product/managed-mlflow

[29] MLflow, “Introduction.” [Online]. Available: https://mlflow.org/docs/latest/introduction/index.html

[30] —, “Why use MLflow?” [Online]. Available: https://mlflow.org/docs/latest/introduction/index.html#why-use-mlflow

[31] TensorFlow, “Keras Guide.” [Online]. Available: https://www.tensorflow.org/guide/keras

[32] —, “Run functions eagerly.” [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/config/run_functions_eagerly

[33] —, “TFX Guide.” [Online]. Available: https://www.tensorflow.org/tfx/guide

[34] C. Lehmann, L. Goren Huber, T. Horisberger, G. Scheiba, A. C. Sima, and K. Stockinger, "Big Data architecture for intelligent maintenance: a focus on query processing and Machine learning algorithms," Journal of Big Data, vol. 7, no. 1, 2020.

[35] Á. López García et al., "A Cloud-Based Framework for Machine learning Workloads and Applications," IEEE Access, vol. 8, pp. 18681–18692, 2020, doi: 10.1109/ACCESS.2020.2964386.

[36] P. Pääkkönen and D. Pakkala, "Extending reference architecture of big data systems towards Machine learning in edge computing environments," Journal of Big Data, vol. 7, no. 1, 2020. [Online]. Available: https://doi.org/10.1186/s40537-020-00303-y

[37] I. Karamitsos, S. Albarhami and C. Apostolopoulos, “Applying devops practices of continuous automation for Machine learning,” Information, vol. 11, no. 7, 2020, doi: 10.3390/info11070363.

[38] M. Pahl and M. Loipfinger, “Machine learning as a reusable microservice,” in 2018 IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–7, doi: 10.1109/NOMS.2018.8406165.

[39] V. Duvvuri, “Minerva: A portable Machine learning Microservice Framework for Traditional Enterprise SaaS applications,” arXiv preprint, arXiv:2005.00866, 2020.

[40] J. L. Ribeiro et al., “A microservice based architecture topology for Machine learning deployment,” in 2019 IEEE International Smart Cities Conference (ISC2), 2019, pp. 426–431, doi: 10.1109/ISC246665.2019.9071708.

[41] C. Carrión, “Kubernetes scheduling: Taxonomy, ongoing issues and challenges,” ACM Computing Surveys, vol. 55, no. 7, pp. 1–37, 2022.

[42] V. Medel et al., “Characterising resource management performance in Kubernetes,” Computers & Electrical Engineering, vol. 68, pp. 286–297, 2018.

[43] M. Pohl, C. Haertel and K. Turowski, “Value Creation from Data Science Applications - A Literature Review,” in 22nd Int. Conf. on Perspectives in Business Informatics Research, 2023 (in press).

[44] C. Haertel, C. Daase, D. Staegemann, A. Nahhas, M. Pohl and K. Turowski, "Toward Standardization and Automation of Data Science Projects: MLOps and Cloud Computing as Facilitators," in *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023)*, pp. 294–302, 2023, doi: 10.5220/0012235100003598.

[45] L. Faubel-Teich and K. Schmid, “MLOps: A Multiple Case Study in Industry 4.0,” 2024, doi: 10.48550/arXiv.2407.09107.

[46] Microsoft, “Introducing chaos engineering to Machine learning deployments,” Microsoft Industry Blogs - India, Mar. 29, 2022. [Online]. Available: https://www.microsoft.com/en-in/industry/blog/machinelearning/2022/03/29/introducing-chaos-engineering-to-machinelearning-deployments/

[47] B. M. A. Matsui and D. H. Goya, “MLOps: Five Steps to Guide its Effective Implementation,” in Proc. - 1st Int. Conf. on AI Engineering - Software Engineering for AI, CAIN 2022, pp. 33–34, doi: 10.1145/3522664.3528611.

[48] Great Expectations, “Data validation workflow,” [Online]. Available: https://docs.greatexpectations.io/docs/guides/validation/validate_data_overview

[49] A. Bodor, M. Hnida and D. Najima, “Machine learning Models Monitoring in MLOps

Context: Metrics and Tools," Int. J. of Interactive Mobile Technologies (iJIM), vol. 17, pp. 125–139, 2023, doi: 10.3991/ijim.v17i23.43479.

[50] "Airflow vs Luigi vs Argo vs MLflow vs Kubeflow," Medium, [Online]. Available: https://medium.com/data-science/airflow-vs-luigi-vs-argo-vs-mlflow-vs-kubeflow-b3785dd1ed0c

[51] K. S. Gill et al., "Utilization of Kubeflow for Deploying Machine learning Models Across Several Cloud Providers," in 2023 3rd Int. Conf. on Smart Generation Computing, Communication and Networking (SMART GENCON), Bangalore, India, 2023, pp. 1–7, doi: 10.1109/SMARTGENCON60755.2023.10442069.

[52] Google, "Getting batch predictions," [Online]. Available: https://cloud.google.com/vertex-ai/docs/start/predictions-guide#get_batch_predictions

[53] R. Subramanya, S. Sierla, and V. Vyatkin, "From DevOps to MLOps: Overview and Application to Electricity Market Forecasting," *Applied Sciences*, vol. 12, no. 19, 2022. doi: 10.3390/app12199851

[54] S. Moreschini, G. Recupito, V. Lenarduzzi, F. Palomba, D. Hästbacka, and D. Taibi, "Toward End-to-End MLOps Tools Map: A Preliminary Study based on a Multivocal Literature Review," *arXiv preprint arXiv:2304.03254*, 2023. doi: 10.48550/arXiv.2304.03254

[55] M. Bryś, "Kubeflow – A Machine learning Toolkit for Kubernetes," *Medium*, May 3, 2019. [Online]. Available: https://medium.com/@michal.brys/kubeflow-a-machine-learning-toolkit-for-kubernetes-d8686f6c91b6

[56] SecureSQL, "Kubernetes Basics," SecureSQL.info, Nov. 19, 2020. [Online]. Available: https://securesql.info/cloud-security/2020/11/19/kubernetes-basics