Maestría en Inteligencia Artificial

MLOps2

Proyecto Final: Sistema de Búsqueda de Productos por Similitud

Grupo N° 5

Alumnos: Alex Barria, Clara Bureu, Maximiliano Torti

Fecha: 19 de Agosto de 2025

Link al Repositorio de GitHub

Tabla de Contenidos

1. Introducción

En este proyecto se desarrolló un sistema de búsqueda multimodal de productos de moda, capaz de recibir consultas tanto en formato textual como visual, y retornar los resultados más relevantes en función de la similitud semántica. El trabajo constituye una evolución de un sistema anterior, inicialmente ejecutado de manera local, que ahora fue rediseñado bajo un enfoque MLOps para garantizar escalabilidad, reproducibilidad y facilidad de mantenimiento.

La nueva arquitectura integra un conjunto de herramientas y servicios dockerizados que permiten automatizar todo el ciclo de vida del modelo, desde la ingesta de datos hasta su despliegue en producción. Entre las principales mejoras implementadas se incluyen la orquestación de procesos con Apache Airflow, el almacenamiento distribuido de datos e imágenes en MinIO (S3-compatible), la persistencia de metadatos y embeddings en PostgreSQL con la extensión pgvector, la exposición de endpoints para consulta a través de APIs REST y GraphQL, la disponibilidad de una interfaz de usuario basada en Streamlit para acelerar la integración, el registro y versionado de modelos mediante MLflow y su puesta en servicio a través de gRPC, como así también el uso de Kafka para procesamiento en streaming.

Este rediseño no solo facilita la escalabilidad y la ejecución repetible de experimentos, sino que también permite incorporar buenas prácticas de ingeniería, como la separación modular de componentes, la trazabilidad completa de los modelos y la posibilidad de actualizar automáticamente el modelo en producción cuando se detecta una mejora en su rendimiento.

2. Dataset

El dataset utilizado corresponde a Fashion Product Images (Small), disponible públicamente en la plataforma Hugging Face. Este conjunto de datos cuenta con un total de 44.072 imágenes de productos de moda, acompañadas de metadatos relevantes tales como identificador único, género, categoría principal (masterCategory), subcategoría, tipo de artículo (articleType), color predominante (baseColour), descripción comercial (productDisplayName), temporada (season), año de lanzamiento y uso previsto.

En esta versión MLOps, la descarga del dataset se realiza de forma automática dentro de los DAGs de Airflow, utilizando la librería datasets. El flujo de procesamiento contempla la descarga y almacenamiento de los datos en bruto en MinIO, su posterior transformación, y la inserción de metadatos en PostgreSQL. Este esquema permite centralizar el acceso a la información, facilitar su reutilización y asegurar que todas las etapas posteriores del pipeline trabajen sobre datos consistentes.

Figura 1: Imágenes de productos del dataset utilizado.

3. Metodología y diseño del sistema

La arquitectura implementada se basa en un ecosistema de servicios desplegados mediante Docker Compose, lo que permite que cada componente pueda desarrollarse, actualizarse y escalar de manera independiente. La figura siguiente muestra los componentes principales y la interacción entre ellos.

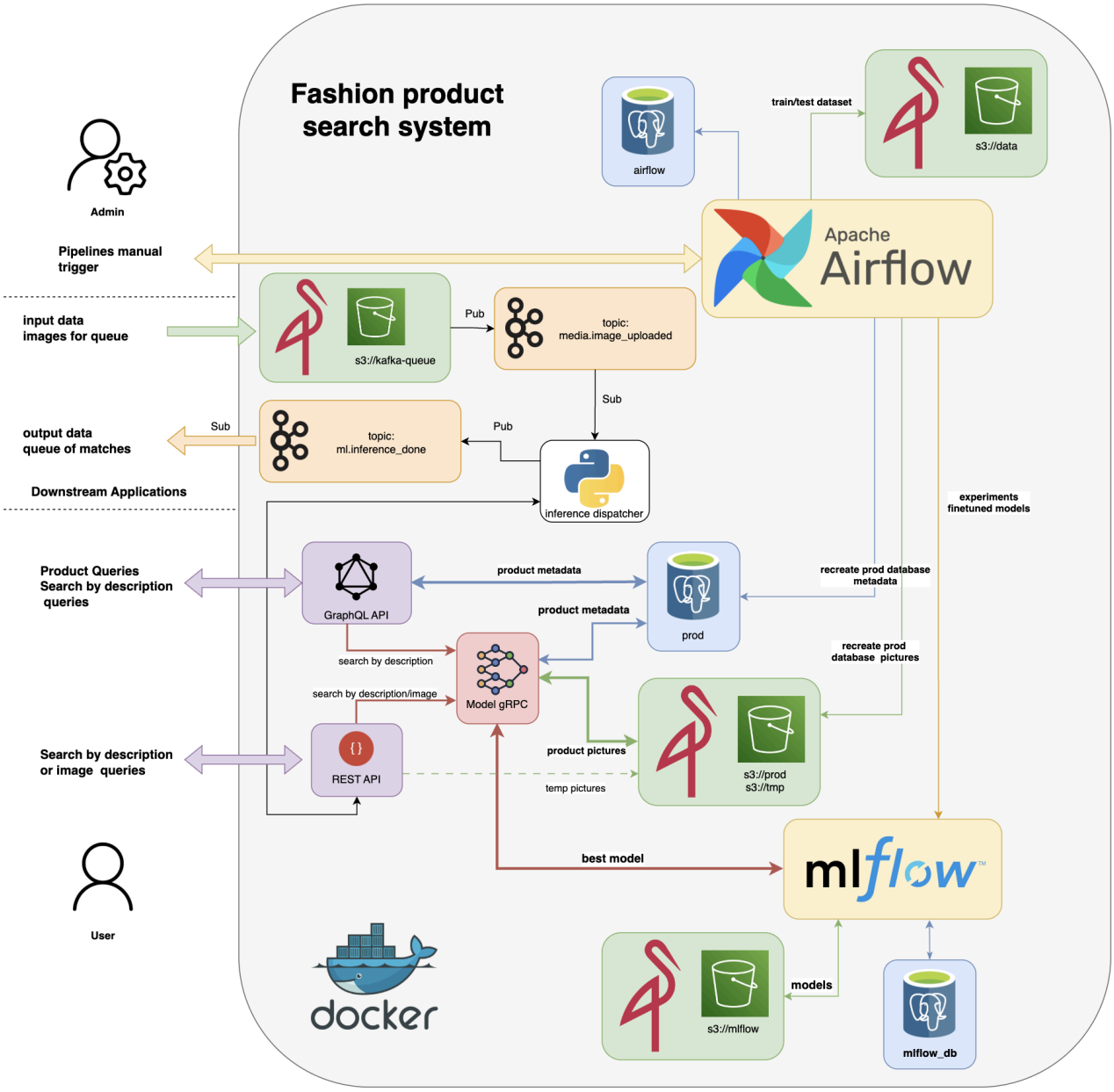

Figura 2: Componentes principales del sistema y la interacción entre ellos.

El núcleo del flujo de trabajo está orquestado por Apache Airflow, que coordina las tareas de ingesta de datos, procesamiento, entrenamiento de modelos y evaluación. Se implementaron tres DAGs principales en Airflow:

- *recreate_prod_database*: descarga y procesa el dataset, almacenando imágenes y metadatos en MinIO y PostgreSQL respectivamente. Adicionalmente utiliza el modelo vigente en producción para generar los embeddings semánticos de los productos descargados y los almacena junto a los metadatos en PostgreSQL.

- *process_train_test_dataset*: prepara el dataset para un nuevo experimento de entrenamiento de modelo, descargando los datos y realizando la separación en conjuntos de entrenamiento y prueba. Los datos procesados son guardados y la información del experimento es registrada en MLflow, de manera de tener todo preparado para el entrenamiento.

- *finetune*: entrena un nuevo modelo (*challenger*) sobre los datos del experimento creado por el dag anterior, evalúa su rendimiento comparándolo con el modelo en producción y, en caso de superar las métricas definidas, promueve automáticamente al *challenger* a *champion* (modelo de producción).

El almacenamiento de datos e imágenes se realiza en MinIO, una solución compatible con el protocolo S3 que asegura disponibilidad y redundancia de la información. Los metadatos y embeddings de los productos se almacenan en PostgreSQL, enriquecido con la extensión pgvector para soportar búsquedas vectoriales eficientes.

El registro, versionado y promoción de modelos se lleva a cabo mediante MLflow, que permite comparar métricas de rendimiento y conservar el historial de experimentos. Un servicio de ejecución de modelos carga el modelo de producción desde MLflow y lo pone en servicio utilizando protocolo gRPC.

La interacción del usuario con el sistema se realiza a través de APIs REST y GraphQL, que exponen funcionalidades de búsqueda y recuperación de información para clientes y aplicaciones externas. Estos servicios se comunican directamente con la base de datos PostgreSQL para recuperar información o con el servicio de ejecución de modelos para las funcionalidades de búsqueda. A fin de acelerar la integración con las APIs, el proyecto provee una UI basada en Streamlit que ejemplifica las comunicaciones y facilita el uso del sistema en forma visual.

También se incorporó un proceso totalmente automatizado y asíncrono, basado en una arquitectura de microservicios orientada a eventos. Usando Kafka, MinIO y gRPC, el sistema reacciona en tiempo real a las nuevas imágenes, asegurando un procesamiento eficiente y escalable sin que el usuario tenga que esperar por el resultado final. Para ello, se crearon dos tópicos de Kafka: uno que recibe notificaciones de un bucket de S3 cada vez que se carga una nueva imagen, y que es consumido por el servicio dispatcher, el

cual se encarga de realizar las llamadas a las APIs para ejecutar la inferencia. Una vez completada la inferencia, el dispatcher publica las predicciones en un tópico de salida, al que cualquier aplicación aguas abajo puede suscribirse para hacer uso de los resultados.

4. Flujo de Datos

El flujo de datos del sistema contempla dos recorridos principales. El primero se inicia con la ingesta de datos, que parte de la descarga del dataset desde Hugging Face, su almacenamiento en formato crudo en MinIO, el procesamiento y normalización de imágenes y metadatos, y la generación de embeddings que se insertan en PostgreSQL con soporte pgvector. Este pipeline concluye con la indexación y disponibilidad de los datos para entrenamiento y búsqueda.

El segundo flujo corresponde a la interacción de los usuarios con el sistema. Una consulta, ya sea textual o visual, es recibida por la API REST o GraphQL, la cual procesa el input y se comunica con el servicio que ejecuta el modelo de producción mediante gRPC. Este último genera el embedding correspondiente, lo compara contra los vectores almacenados en la base de datos, recuperando los productos más similares, y retorna (primero a la API REST o GraphQL y finalmente al origen de la consulta) los resultados junto con sus metadatos y, opcionalmente, las imágenes asociadas.

5. Evaluación

El sistema se evalúa utilizando métricas que permiten medir tanto la precisión en la recuperación exacta del producto como la coherencia de los resultados en términos de similitud semántica. Las principales métricas empleadas son exact match accuracy@k, que mide la proporción de consultas en las que el producto exacto se encuentra entre los k primeros resultados, y group accuracy@k, que evalúa si los elementos recuperados pertenecen a la misma categoría o grupo semántico que la consulta.

La evaluación se encuentra automatizada en el DAG de fine-tuning. Al finalizar el entrenamiento del *challenger*, se calcula su rendimiento en el conjunto de prueba y se compara con el *champion* vigente. Si el *challenger* obtiene mejores resultados en las métricas definidas, es promovido automáticamente a producción. Además, se mide el tiempo de respuesta promedio para consultas de texto e imagen, asegurando que el sistema sea apto para un uso interactivo en contextos de e-commerce.

6. Conclusiones

La transición del sistema a una arquitectura MLOps dockerizada ha permitido alcanzar un alto grado de escalabilidad, reproducibilidad y trazabilidad en todo el ciclo de vida del modelo. La integración de Airflow, MinIO, PostgreSQL con pgvector, MLflow, Kafka y APIs REST/GraphQL/gRPC conforma un ecosistema robusto que facilita tanto el mantenimiento como la mejora continua del sistema.

Entre las principales ventajas se destacan la automatización de pipelines complejos, la capacidad de realizar despliegues controlados y versionados, y la posibilidad de integrar nuevas fuentes de datos y modelos sin interrumpir el servicio. No obstante, persisten oportunidades de mejora, como optimizar las búsquedas vectoriales, incorporar mecanismos de retraining automático ante la llegada de nuevos datos, y explorar estrategias de *hard-negative mining* o feedback humano para ajustar el ranking de resultados.

El sistema desarrollado no solo mantiene la funcionalidad y precisión del enfoque inicial, sino que además incorpora una infraestructura que lo prepara para operar en entornos reales de alta demanda, con capacidad de evolución continua y administración eficiente de recursos.