Científico de Datos: Andrés Mauricio Castro Llanos
26/Febrero/2025

1. **Explica cómo implementarías un modelo de machine learning que debe ser explicable y auditable en un entorno regulado (ej. sector financiero o salud).**

Utilizar modelos interpretables como árboles de decisión, Random Forest o XGBoost con SHAP o LIME para explicabilidad local y global. Optar por Regresión Lineal o Regresión Logística (depende de su variable desenlace) cuando la transparencia sea una prioridad regulatoria, con estos modelos Transparencia en coeficientes y relación directa entre variables y predicciones.

Para asegurar auditabilidad se realiza el Versionamiento de Datos y códigos del Modelo con DVC (Data Version Control) o MLflow para rastrear cambios en datos y modelos **Git** para el control de versiones del código. También podríamos realizar un Registro de Experimentos MLflow Tracking o Weights & Biases para registrar hiperparámetros, métricas y resultados.

2. **¿Cómo diseñarías un experimento de A/B Testing para evaluar el impacto de un nuevo modelo de machine learning en producción?**

El objetivo de este A/B testing es evaluar el impacto de un nuevo modelo de machine learning en la tasa de conversión y la tasa de clics en experiencias recomendadas dentro de una plataforma de bienestar. Se utilizarán métricas clave como la **Tasa de Conversión**, la **Tasa de Calificaciones** y la **Tasa Combinada de Clics, Calificaciones y Compras** para medir el comportamiento del usuario y determinar el éxito del modelo.

Definición de Hipótesis

- **Hipótesis Nula (H0):** El nuevo modelo no tiene impacto significativo en las tasas de conversión o clics.
- **Hipótesis Alternativa (H1):** El nuevo modelo mejora significativamente las tasas de conversión o clics.

Por Ejemplo: "El nuevo modelo aumenta el CTR en un 5% en comparación con el modelo actual."

Cambios en el Front para A/B Testing

- **Diseño de la Interfaz:** Colores, tipografía, y tamaños de botones.
- **Ubicación de Elementos:** Reorganización de experiencias recomendadas.
- **Formato de Imágenes:** Tamaño, calidad y estilo de las imágenes de productos.
- **Personalización de Recomendaciones:** Orden de experiencias basadas en el historial del usuario.
- **Pop-ups y Banners:** Mensajes promocionales o recordatorios de productos.
- **Tiempo de Carga:** Optimización del tiempo de respuesta para una mejor experiencia.
- **Interacciones Sociales:** Botones de compartir o calificar experiencias

Métricas a Evaluar

En el A/B testing, las métricas clave ayudan a medir el impacto del nuevo modelo, tres principales son las siguientes:

- Tasa de Conversión (CVR): Mide el porcentaje de usuarios que realizan una acción deseada tras ver una experiencia recomendada.

Tasa de clics en experiencias recomendadas:

$$\frac{\text{Número de clics en experiencias recomendada}}{\text{Total Experiencias mostradas /recomendadas}}$$

- Tasa de Calificaciones: Evalúa cuántos usuarios califican las experiencias mostradas.

Tasa calificaciones:

$$\frac{\text{Número de Calificaciones}}{\text{Total experiencias mostradas}}$$

- Tasa Combinada: Suma de clics, calificaciones y compras, reflejando el compromiso general del usuario.

Tasa de clics + calificaciones + compras:

$$\frac{\text{Clics +Calificaciones + Compras}}{\text{Total de clics (Filas Reporte)}}$$

Tamaño de Muestra

Determina la cantidad de usuarios necesarios para detectar cambios significativos en las métricas evaluadas. Un cálculo adecuado evita resultados falsos y garantiza decisiones basadas en datos confiables. Se considera el nivel de confianza, el poder estadístico y el efecto mínimo detectable para definir el tamaño de muestra ideal.

$$n = \frac{(Z_{\alpha/2} + Z_{\beta})^2 \cdot (p_1(1-p_1) + p_2(1-p_2))}{(p_1 - p_2)^2}$$

- Zα/2 : es el valor crítico para el nivel de confianza (1.96 para 95%).
- Zβe: Es el valor crítico para el poder estadístico (0.84 para 80%)
- p1, p2: Tasas de conversión esperadas en los grupos.
- p1-p2: El efecto mínimo detectable

Diseño Experimental y Asignación de Grupos

1. **Filtrar Usuarios Elegibles:**

Se seleccionan usuarios que cumplen criterios de comportamiento según disponga el negocio o los mismos datos, clasificándolos según su actividad en los últimos 30, 60, 90 días ya sea por consulta o compras

- Pocos clics: Menor probabilidad de interacción.
- Más clics: Mayor probabilidad de interacción.

2. **División en Grupos:**

- Estratificación: Basada en el comportamiento de clics.
- Grupo Control: Recibe el modelo actual.
- Grupo Experimental: Prueba el nuevo modelo.
- Método: Uso de train_test_split con stratify en Python.

Una vez definidos los usuarios elegibles y calculado el tamaño de muestra, se asigna un producto diferente a cada usuario. Se establece una ventana de 30 días para observar su comportamiento e interacción con el producto, garantizando datos suficientes para evaluar el impacto del experimento.

Análisis Estadístico

Una vez transcurrido el periodo de recolección del tamaño de muestra de clics, se procede al análisis estadístico. Se realizan pruebas de hipótesis para evaluar el impacto del experimento:

- Test t de Student: Compara las medias de métricas continuas como la tasa de conversión.
- Prueba de Chi-Cuadrado: Analiza las distribuciones de eventos, como clics vs. no clics. Estos análisis permiten determinar si las diferencias observadas son estadísticamente significativas.

Los resultados finales del experimento se pueden monitorear de manera continua utilizando un A/B Test Dashboard en herramientas como Power BI o Tableau. Esto permite visualizar las métricas clave en tiempo real, facilitando la toma de decisiones basadas en datos y el seguimiento del impacto del modelo en producción.

3. En un modelo basado en redes neuronales profundas, ¿cómo manejarías el problema de vanishing/exploding gradients y qué soluciones aplicarías?

Soluciones para Vanishing Gradients

1. Funciones de Activación Adecuadas:
   Utilizar funciones de activación como ReLU (Rectified Linear Unit) y sus variantes, como Leaky ReLU o ELU, ayuda a mitigar el problema de vanishing gradients. Estas funciones permiten que los gradientes se mantengan positivos y no se anulen durante la retropropagación, mejorando la convergencia del modelo.

2. Inicialización de Pesos:
   Una inicialización adecuada de los pesos es esencial para evitar la desaparición de gradientes. Técnicas como He Initialization (ideal para ReLU) y Xavier Initialization (para funciones sigmoides) ajustan la escala de los pesos iniciales, asegurando que los gradientes permanezcan en un rango saludable y evitando que se vuelvan demasiado pequeños.

3. Batch Normalization:
   Batch Normalization normaliza las activaciones de cada capa durante el entrenamiento, estabilizando la distribución de los gradientes. Esto permite que el modelo aprenda de

manera más eficiente al reducir la sensibilidad a la inicialización de pesos y permitiendo tasas de aprendizaje más altas sin riesgo de vanishing gradients

Soluciones para Exploding Gradients

1. Gradient Clipping establece un límite máximo para el valor de los gradientes, evitando que se vuelvan excesivamente grandes y causen inestabilidad en el entrenamiento. Esta técnica permite que el modelo aprenda de manera controlada sin que los gradientes exploten, lo que mejora la estabilidad y la convergencia.
2. La regularización, como el uso de L2 Regularization, ayuda a controlar la magnitud de los pesos, evitando que se vuelvan demasiado grandes. Esto contribuye a estabilizar los gradientes y a prevenir el overfitting, mejorando la generalización del modelo.

4. ¿Cómo diseñarías un sistema de recomendaciones en producción que pueda actualizarse dinámicamente con feedback del usuario?

Para evaluar el impacto de un nuevo modelo de machine learning en una plataforma de bienestar, se propone un sistema de recomendaciones dinámico que se actualiza en tiempo real utilizando el feedback del usuario. El enfoque es híbrido, combinando Filtrado Colaborativo, Filtrado Basado en Contenido y Modelos con Embeddings, lo que asegura recomendaciones personalizadas y altamente relevantes.

Enfoque Híbrido y Actualización Dinámica

- Filtrado Colaborativo: Utiliza comportamientos similares entre usuarios para generar recomendaciones.
- Filtrado Basado en Contenido: Recomienda experiencias basadas en las características de productos previos.
- Embeddings y Aprendizaje Incremental: Capturan relaciones complejas y se actualizan en tiempo real con feedback del usuario, mejorando continuamente la precisión.

Pipeline de Datos y Despliegue en Producción

- Pipeline en Tiempo Real: Captura interacciones del usuario (clics, calificaciones, compras) utilizando Apache Kafka o AWS Kinesis.
- Entrenamiento y Actualización del Modelo: Se aplican Batch Training diario y Online Learning para adaptarse dinámicamente al comportamiento del usuario.
- Despliegue Escalable: Se utiliza Docker y Kubernetes para asegurar alta disponibilidad y escalabilidad del sistema en producción.

Ejemplo: En una plataforma de bienestar, utilizar un modelo híbrido con Matrix Factorization para feedback implícito y BERT para contenido textual. TensorFlow Serving para despliegue en tiempo real.

5. ¿Cómo aplicarías modelos de machine learning en un entorno con datos altamente heterogéneos y no estructurados (ej. imágenes, texto y datos tabulares)?

Para implementar modelos de machine learning en entornos con datos heterogéneos y no estructurados como imágenes, texto y datos tabulares, se utiliza un enfoque multimodal que integra diferentes tipos de datos en un espacio vectorial conjunto. Esto permite aprovechar al máximo la información disponible, mejorando la precisión y relevancia de las predicciones.

Enfoque Multimodal y Representación de Datos

- Imágenes: Se procesan con Redes Neuronales Convolucionales (CNNs) o Vision Transformers (ViT) para obtener embeddings representativos.
- Texto: Se utiliza NLP avanzado con Transformers como BERT o GPT para generar embeddings contextuales de alta calidad.
- Datos Tabulares: Se representan mediante XGBoost, LightGBM o TabNet para capturar relaciones complejas y garantizar interpretabilidad.

Fusión de Datos y Modelado Multimodal

- Vectorización y Fusión de Embeddings: Se combinan mediante concatenación o atención multimodal para crear representaciones conjuntas.
- Modelos Multimodales: Se utilizan CLIP y Multimodal Transformers (MMT) para relacionar imágenes, texto y datos tabulares de forma eficiente y escalable.

Pipeline de Datos y Despliegue en Producción

- Pipeline en Tiempo Real: Se utiliza Apache Kafka o AWS Kinesis para ingesta de datos en tiempo real, almacenando en Data Lakes en la nube.
- Entrenamiento y Despliegue: Se aplica Batch Training e Incremental Learning para mantener el modelo actualizado. Se despliega con Docker y Kubernetes para escalabilidad y alta disponibilidad.
- MLOps Automatizado: Se implementa un pipeline de CI/CD con Kubeflow o MLflow para orquestar el entrenamiento, validación y despliegue continuo del modelo.

Monitoreo y Validación Continua

- A/B Testing Continuo: Se evalúan las recomendaciones en tiempo real con Feature Flags y Dashboards en Power BI o Tableau.
- Monitoreo de Desempeño: Se utilizan Prometheus y Grafana para métricas de rendimiento y EvidentlyAI para detectar deriva de datos.

6. ¿Cuáles son las ventajas y desventajas de desplegar un modelo de machine learning como un microservicio en comparación con integrarlo directamente en una aplicación monolítica?

Desplegar un modelo de machine learning como microservicio ofrece alta escalabilidad, ya que se puede ajustar su capacidad de manera independiente sin afectar al resto de la aplicación. Además, permite actualizaciones continuas y mayor flexibilidad tecnológica, utilizando herramientas

especializadas para machine learning. Sin embargo, requiere una infraestructura distribuida más compleja y mayores costos operativos debido a la necesidad de gestionar múltiples servicios y su comunicación en red.

Por otro lado, integrarlo en una aplicación monolítica reduce la latencia al realizar inferencias localmente y simplifica el despliegue y mantenimiento, ya que toda la aplicación comparte recursos. Además, los costos operativos son más bajos al no requerir infraestructura adicional. No obstante, limita la escalabilidad independiente y la flexibilidad tecnológica, ya que todo el sistema debe escalarse en conjunto, lo que afecta la resiliencia y disponibilidad si el modelo falla.

En conclusión, la decisión depende de las necesidades de escalabilidad, flexibilidad y complejidad operativa. Se recomienda el microservicio en entornos escalables y dinámicos, mientras que el monolítico es ideal para aplicaciones más simples y controladas.

Característica	Microservicio	Monolítico
Escalabilidad	Independiente y flexible	Limitada, se escala toda la aplicación
Latencia	Mayor debido a llamadas de red	Menor, procesamiento local
Despliegue y CI/CD	Continua e independiente	Completo y acoplado
Mantenimiento	Separado y modular	Complejo y dependiente del monolito
Resiliencia	Aislada, falla no afecta al sistema	La falla impacta toda la aplicación
Costo Operativo	Alto (infraestructura distribuida)	Bajo (recursos compartidos)
Flexibilidad Tecnológica	Alta, permite stack especializado	Limitada al stack de la aplicación
Complejidad Operativa	Alta (orquestación y monitoreo)	Baja, menos componentes a gestionar

7. ¿Cómo manejarías la monitorización y logging de un modelo en producción para detectar degradaciones en su rendimiento?

Para monitorear el modelo y gestionar el logging de manera eficiente, se recomienda un enfoque integral que permita detectar degradaciones en el rendimiento en tiempo real. Se deben utilizar métricas clave como Precision@K, Recall@K y NDCG para evaluar la precisión y relevancia de las recomendaciones, así como métricas de negocio como CTR (Click-Through Rate) y Conversion Rate

para monitorear el impacto en el comportamiento del usuario. Para detectar Data Drift y Model Drift, se pueden aplicar pruebas estadísticas como KS-test (Kolmogorov-Smirnov) o Jensen-Shannon Divergence para comparar la distribución de datos de entrada con la de entrenamiento, identificando cambios en el comportamiento de los usuarios.

El logging debe capturar detalles precisos y relevantes para permitir una detección temprana de degradaciones en el rendimiento, especialmente en cuanto a latencia y tiempos de procesamiento del modelo. Un enfoque efectivo es configurar logs que registren el tiempo de inferencia (latencia de predicción) y el tiempo de respuesta completo (incluyendo la preparación de datos y la entrega de resultados). Por ejemplo, si el modelo normalmente responde en 500 ms en promedio, se puede configurar una alerta si la latencia promedio supera los 800 ms durante un periodo continuo de 5 minutos. Además, si se detectan picos de latencia superiores a 1500 ms en más del 5% de las solicitudes, se debe generar una alerta crítica, ya que esto podría indicar problemas de rendimiento o cuellos de botella en la infraestructura.

Para implementar este enfoque, se pueden utilizar herramientas open source como Prometheus y Grafana para la recolección de métricas en tiempo real y la visualización de datos mediante dashboards personalizados. Elastic Stack (ELK), compuesto por Elasticsearch, Logstash y Kibana, es ideal para el análisis de logs, permitiendo un logging estructurado y consultas avanzadas. OpenTelemetry se puede integrar para rastrear trazabilidad distribuida en sistemas complejos. En entornos en la nube, Azure Monitor y Application Insights ofrecen una integración nativa para monitorear el rendimiento y detectar degradaciones en tiempo real, utilizando inteligencia artificial para identificar anomalías y generar alertas automáticas. Este enfoque integral permite una detección temprana de degradaciones, garantizando la relevancia continua de las recomendaciones y optimizando el rendimiento del modelo en producción.

8. Explica cómo diseñarías una arquitectura de CI/CD para modelos de machine learning en producción.

Para diseñar una arquitectura de CI/CD (Continuous Integration/Continuous Deployment) para modelos de machine learning en producción, se debe considerar un enfoque modular y escalable que permita automatizar el ciclo de vida del modelo, desde el desarrollo hasta el despliegue en producción, garantizando reproducibilidad, trazabilidad y calidad del modelo.

El flujo de CI/CD para machine learning incluye las siguientes etapas:

1. Control de Versiones y Gestión de Código: Utilizar Git para el versionamiento del código fuente, scripts de entrenamiento y configuración de experimentos. Se recomienda GitHub, GitLab o Azure Repos como repositorio central. Además, se deben versionar los datos y modelos utilizando DVC (Data Version Control) o MLflow para garantizar reproducibilidad y trazabilidad.
2. Continuous Integration (CI): Al hacer un push al repositorio, un pipeline de CI se activa automáticamente para:

- Validación del Código: Ejecutar pruebas unitarias y de integración utilizando Pytest o Unittest.
- Verificación de Calidad de Código: Utilizar Pylint o Flake8 para mantener estándares de calidad.
- Entrenamiento del Modelo: Entrenar el modelo en un entorno controlado utilizando MLflow para rastrear hiperparámetros, métricas y versiones del modelo.
- Validación de Modelo: Evaluar el modelo en un conjunto de validación y verificar que las métricas cumplan con los umbrales definidos, utilizando NDCG, Precision@K, Recall@K o métricas de negocio como CTR y Conversion Rate.

3. Continuous Deployment (CD): Si el modelo pasa todas las validaciones, se activa un pipeline de CD para:
   - Versionamiento del Modelo: Almacenar el modelo en un Model Registry utilizando MLflow Model Registry o Azure Machine Learning Model Registry.
   - Pruebas de Integración y Validación: Realizar pruebas de integración en un entorno staging para asegurar la compatibilidad con otros servicios y validar el rendimiento del modelo.
   - Despliegue en Producción: Implementar el modelo en un entorno de producción utilizando:
     - Docker y Kubernetes para empaquetar y escalar el modelo en contenedores.
     - Azure Kubernetes Service (AKS) o Azure Machine Learning Endpoint para el despliegue escalable en la nube.
     - CI/CD Automation con Azure DevOps o GitHub Actions para gestionar el flujo de despliegue de forma automatizada.
     - Canary Deployment o Blue-Green Deployment para minimizar riesgos al introducir cambios en producción.
4. Monitorización y Logging: Una vez desplegado el modelo, se debe implementar un sistema de monitorización y logging para:
   - Monitorear Métricas de Rendimiento: Utilizar Prometheus y Grafana para rastrear métricas en tiempo real como latencia de predicción, throughput, y métricas de precisión (Precision@K, NDCG, CTR).
   - Logging Estructurado: Implementar Elastic Stack (ELK) para capturar logs detallados de predicciones, interacciones de usuario y métricas en tiempo real.
   - Data Drift y Model Drift: Utilizar EvidentlyAI o Azure Monitor para detectar cambios en la distribución de datos de entrada (Data Drift) y degradación en el rendimiento del modelo (Model Drift).

5. Automatización y Orquestación: La arquitectura se puede automatizar y orquestar utilizando:
    - Azure DevOps Pipelines, GitHub Actions o Jenkins para CI/CD.
    - MLflow o Kubeflow para la orquestación de experimentos y pipelines de entrenamiento.
    - Terraform o Azure Resource Manager (ARM) para la infraestructura como código (IaC).

Este enfoque integral garantiza un ciclo de vida continuo y automatizado para los modelos de machine learning, alineado con las mejores prácticas de DevOps y MLOps, asegurando calidad, escalabilidad y mantenimiento eficiente en producción.

9. ¿Cómo optimizarías el tiempo de inferencia de un modelo en producción sin comprometer su precisión?

Tomando como base que el tiempo de inferencia es el tiempo que tarda un modelo en generar una predicción después de recibir los datos de entrada. Si el modelo se demora mucho, la experiencia del usuario se ve afectada. La idea es hacerlo más rápido sin sacrificar la precisión de las predicciones.

Se podría mejorar esto de dos formas principales:

1. Optimizando el modelo en sí (haciéndolo más ligero y rápido).

2. Mejorando la infraestructura donde se ejecuta el modelo (utilizando hardware y software más eficientes).

1. Optimización del Modelo

a. Cuantización: Cuantización es una técnica que consiste en convertir los números utilizados por el modelo para hacer cálculos (como pesos y activaciones) de punto flotante (FP32) a enteros (INT8) o punto flotante de menor precisión (FP16). Esto hace que el modelo use números más pequeños, lo que reduce la memoria necesaria y acelera el tiempo de inferencia. La pérdida de precisión suele ser mínima o casi imperceptible.

Para aplicar cuantización, se pueden utilizar herramientas como TensorFlow Lite para modelos en TensorFlow, ONNX Runtime para modelos exportados en formato ONNX (compatible con PyTorch y TensorFlow), y PyTorch TensorRT para optimizar modelos en PyTorch en GPUs.

b. Pruning (Poda): Pruning (Poda) consiste en eliminar partes del modelo que tienen poco impacto en su precisión, como neuronas o conexiones que apenas contribuyen a las predicciones. Al hacerlo, el modelo se vuelve más pequeño y ligero, lo que acelera el tiempo de inferencia y reduce el consumo de memoria. Si se aplica de manera adecuada, la pérdida de precisión es mínima o prácticamente imperceptible.

Existen dos tipos principales de poda:

- Magnitude-based Pruning: Se eliminan conexiones cuyos pesos son cercanos a cero, ya que aportan muy poco al resultado final.
- Structured Pruning: Se eliminan neuronas completas o incluso capas enteras de forma organizada, optimizando así la arquitectura del modelo.

Esta técnica ayuda a simplificar el modelo manteniendo su precisión y, al mismo tiempo, mejora significativamente el rendimiento en producción.

c. Vectorización y Batch Processing consiste en procesar varias solicitudes simultáneamente en lugar de hacerlo una por una. Esto se logra agrupando las solicitudes en batches (lotes), aprovechando que las GPUs y TPUs están diseñadas para realizar cálculos en paralelo. Al hacerlo, se acelera significativamente el tiempo de inferencia sin afectar la precisión del modelo.

El principal beneficio es la optimización del tiempo de procesamiento, ya que se aprovechan al máximo las capacidades de procesamiento paralelo del hardware. Además, esta técnica no afecta la precisión del modelo, solo mejora la velocidad de respuesta.

2. Optimización de Infraestructura

a. Hardware Acelerado consiste en ejecutar el modelo en dispositivos especialmente diseñados para procesamiento paralelo, como GPUs (Graphics Processing Units), TPUs (Tensor Processing Units) y Inferentia (AWS). Estos hardware están optimizados para realizar operaciones matemáticas de manera simultánea, lo que acelera significativamente el tiempo de inferencia.

- GPUs son ampliamente utilizadas en deep learning debido a su capacidad para procesar miles de operaciones en paralelo.
- TPUs son procesadores de Google optimizados específicamente para TensorFlow, ofreciendo un rendimiento superior en modelos de machine learning.
- Inferentia es un chip de AWS diseñado exclusivamente para tareas de inferencia, proporcionando una baja latencia y alta eficiencia en costos.

b. Model Serving y Caching son técnicas clave para optimizar el tiempo de inferencia en producción.

- Model Serving consiste en desplegar el modelo en servidores especialmente optimizados para recibir solicitudes y devolver predicciones de manera rápida y eficiente. Esto asegura tiempos de respuesta cortos, incluso bajo alta demanda.
- Caching implica almacenar las respuestas de solicitudes frecuentes para reutilizarlas en lugar de recalcularlas en cada solicitud, lo que reduce significativamente la carga de procesamiento en el modelo.

Estas técnicas se complementan para mejorar el rendimiento y la escalabilidad de los modelos en producción.

Herramientas para Model Serving:

- TensorFlow Serving: Ideal para modelos desarrollados en TensorFlow, optimizado para escalabilidad y rendimiento en producción.

- TorchServe: Diseñado específicamente para modelos en PyTorch, permite un despliegue eficiente y flexible.
- ONNX Runtime: Funciona con modelos en formato ONNX, compatible con TensorFlow y PyTorch, y está optimizado para múltiples plataformas de hardware.

La combinación de Model Serving y Caching garantiza tiempos de respuesta rápidos y un uso eficiente de recursos, mejorando la experiencia del usuario final.

10. Explica cómo manejarías la orquestación de workflows de ML en la nube y qué herramientas utilizarías.

Para manejar la orquestación de workflows de Machine Learning en la nube, es necesario coordinar todas las etapas del ciclo de vida del modelo, como el preprocesamiento de datos, el entrenamiento, la evaluación, el despliegue y la monitorización, asegurando que todo se ejecute en el orden correcto y de forma automatizada. Utilizaría Kubeflow para crear pipelines modulares y escalables en Kubernetes, ya que permite experimentar y desplegar modelos fácilmente en la nube. También usaría Apache Airflow para programar tareas complejas y gestionar dependencias de manera flexible mediante DAGs (grafos acíclicos dirigidos). Para el tracking de experimentos y la gestión de versiones de modelos, emplearía MLflow, que además facilita el despliegue en entornos en la nube como Azure o AWS. Finalmente, integraría Azure ML Pipelines para automatizar el CI/CD de los modelos, aprovechando su integración nativa con otros servicios de Azure como AKS y Azure DevOps.