Generative AI Solution Development con Databricks

1. ¿Qué es una solución GenAI contextual?

- **Objetivo**: Generar respuestas personalizadas y precisas usando información específica del contexto (documentos, bases de datos, etc.).
- Se evita depender solo del conocimiento preentrenado del modelo (que puede estar desactualizado o ser genérico).

---

2. ¿Qué es RAG (Retrieval-Augmented Generation)?

- **Arquitectura RAG = Retrieval + Generation**
    - **Retrieval**: Se busca contexto relevante (documentos, fragmentos) desde una base de datos vectorial.
    - **Augmented Generation**: Ese contexto se pasa al LLM junto al prompt para generar una respuesta más precisa.
- Ventaja: no necesitas *fine-tune*, reduces costos y mantienes control sobre la información usada.

---

3. Mosaic AI Playground (Databricks)

- Herramienta visual para explorar prompts y resultados de modelos LLM.
- Permite experimentar con entradas, ver cómo responde el modelo, e integrar herramientas como embeddings y vector search.

---

4. Preparación de datos para soluciones GenAI

- **Limpieza y segmentación**: Separar el texto en fragmentos lógicos (chunking).

- **Embeddings**: Convertir texto en vectores numéricos usando modelos como `text-embedding-ada-002`.

- **Almacenamiento**: Se guardan en un vector store para hacer búsquedas semánticas.

---

5. Vector databases y Vector Search

- **Vector Store**: Base de datos especializada para hacer búsquedas por similitud semántica (no por palabras exactas).

- **Ejemplo en Databricks**: Mosaic AI Vector Search.

- Puedes usar **Delta Lake** como almacenamiento subyacente, aprovechando su capacidad para manejar grandes volúmenes de datos y cambios incrementales.

---

6. Context Embedding y Recuperación

- Cada fragmento de información se convierte en un vector.

- Cuando un usuario hace una pregunta, esta también se convierte en vector → se recuperan los vectores más similares → se construye el contexto.

---

7. Prompt Engineering (Ingeniería de prompts)

- Diseñar prompts claros, con instrucciones específicas.

- Ejemplo: “Responde usando solo la información proporcionada en el contexto, sin inventar.”

- Técnicas útiles: Few-shot prompting, chain-of-thought, instrucciones directas.

---

8. Fine-tuning vs. RAG

Característica	Fine-tuning	RAG
Costo	Alto	Bajo
Tiempo de ajuste	Lento	Rápido (sin entrenar el modelo)
Flexibilidad	Menor (modelo fijo)	Alta (puedes cambiar el contexto)
Ideal para	Datos estructurados, tareas específicas	Documentos, respuestas contextuales

---

9. Evaluación de modelos GenAI

- **Evaluación humana**: Revisión manual de respuestas.
- **Evaluación automática**: Métricas como relevancia, factualidad, toxicidad.
- Se puede integrar con **MLflow** para registrar experimentos y comparar variantes de prompts, modelos o contextos.

---

10. Herramientas clave en Databricks para GenAI

Herramienta	Uso principal
Mosaic AI Playground	Experimentación con LLMs y prompts.
Mosaic AI Vector Search	Recuperación semántica de documentos.
MLflow	Trazabilidad, comparación de resultados.
LangChain	Orquestación de flujos LLM, integración con APIs.
Delta Lake	Almacenamiento confiable para datos y vectores.

The importance of evaluating genAI applications

1. ¿Por qué es importante evaluar aplicaciones de GenAI?

- Porque las aplicaciones generativas pueden generar errores, respuestas dañinas o usar datos ilegales.
- Evaluarlas asegura que sean **seguras, legales y útiles** para los usuarios.

---

2. ¿Qué se evalúa en una aplicación GenAI?

- Cada parte del sistema puede necesitar una evaluación distinta:
    - **Entradas** (input): ¿El prompt está claro y bien diseñado?
    - **Salidas** (output): ¿La respuesta es útil, correcta y segura?
    - **Comportamiento del sistema**: ¿Cómo responde a abusos o casos inesperados?

---

3. ¿Qué riesgos se deben tener en cuenta?

- **Legales**: Uso de datos sin licencia, privacidad, derechos de autor.
- **Éticos**: Respuestas tóxicas, discriminación, desinformación.
- **De seguridad**: Mal uso por parte de usuarios (por ejemplo, para hacer trampas, spam, etc.).

---

4. ¿Por qué es difícil evaluar GenAI?

- No siempre hay una respuesta correcta.
- Las salidas pueden variar mucho.

- La evaluación automática (por ejemplo, con otra IA) no siempre es confiable.
- Se deben controlar riesgos sin frenar la utilidad del sistema.

---

5. ¿Cómo se manejan estos problemas?

- **Licencias de datos**: Asegurarse de que los datos usados estén legalmente permitidos.
- **Guardrails (barreras de seguridad)**: Reglas o filtros para evitar respuestas dañinas o peligrosas.
- **Monitoreo y feedback**: Evaluar continuamente con métricas, usuarios y testers.

Securing and governing genAI applications

1. ¿Por qué es importante asegurar y gobernar aplicaciones GenAI?

- Para evitar que la IA sea usada de forma maliciosa.
- Para controlar **quién accede a qué datos** y cómo se usan.
- Para cumplir con normas legales, proteger la privacidad y evitar riesgos.

---

2. ¿Por qué es difícil asegurar y gobernar GenAI?

- Las salidas generativas son **impredecibles**.
- Hay muchos puntos de riesgo: el modelo, los datos, los prompts, el usuario.
- Se necesita control en tiempo real y trazabilidad de cada acción.

---

3. ¿Cuál es el rol del equipo técnico (data scientists y AI developers)?

- Asegurar que las aplicaciones cumplan normas éticas, legales y de seguridad.
- Implementar barreras, filtros, permisos y auditoría en el flujo de desarrollo.
- Trabajar junto al equipo de seguridad y gobernanza para aplicar buenas prácticas.

---

4. ¿Qué herramientas de Databricks ayudan con esto?

🔐 **Unity Catalog**

- **Permissions**: Controla quién puede ver, usar o modificar datos o modelos.
- **Lineage**: Muestra el recorrido de los datos (de dónde vienen y a dónde van).
- **Audit**: Registra quién hizo qué y cuándo, para tener trazabilidad.

🛡️ **Herramientas de seguridad específicas**

- **Safety Filter**: Evita que se generen respuestas inseguras o inadecuadas.
- **Llama Guard**: Marco de seguridad para proteger interacciones con modelos LLM.

Gen AI evaluation techniques

1. ¿Cómo se compara la evaluación de LLM con la de ML tradicional?

- En ML tradicional, se usan métricas claras y objetivas (ej: precisión en clasificación).
- En LLMs, las salidas son lenguaje natural, así que:
    - **Más difícil de medir**.
    - **Más subjetivo** (una misma tarea puede tener varias respuestas correctas).
    - Se necesitan **métricas nuevas y adaptadas**.

---

2. Evaluar un LLM vs evaluar todo el sistema

- Evaluar un **LLM solo**: Mide la calidad de respuestas individuales.
- Evaluar un **sistema de IA completo**: Incluye todo el flujo (prompts, interacciones, herramientas, decisiones finales).

---

3. Métricas genéricas para LLM

- **Accuracy (precisión)**: ¿Responde correctamente?
- **Perplexity**: Mide qué tan bien el modelo predice la siguiente palabra (bajo = mejor).
- **Toxicity**: Evalúa si el modelo genera contenido ofensivo o dañino.

---

4. ¿Por qué se necesitan métricas específicas por tarea?

- Porque cada tipo de tarea (traducción, resumen, generación creativa) requiere criterios distintos para medir si está bien hecha.

---

5. Métricas específicas por tarea

- **BLEU**: Evalúa calidad en traducción comparando con frases de referencia.
- **ROUGE**: Evalúa resúmenes, comparando si aparecen las mismas palabras/frases clave que en un resumen ideal.

---

6. LLM-as-a-judge

- Es una técnica donde **otro modelo LLM evalúa las respuestas** de un modelo.
- Se usa para medir calidad, relevancia o corrección de respuestas generadas.
- Ventaja: Escalable.
- Riesgo: El evaluador también puede equivocarse o tener sesgos.

Evaluating an entire AI system

1. ¿Por qué evaluar todo el sistema GenAI y no solo el modelo?

- Porque el **rendimiento general** depende de más que el modelo: prompts, datos, herramientas, lógica de negocio, etc.
- También hay que considerar el **costo total**: uso de recursos, tiempo de respuesta, eficiencia, escalabilidad.

---

2. ¿Cómo es la arquitectura de un sistema GenAI?

- Tiene **varios componentes**:
    - Modelo LLM
    - Prompts
    - Herramientas externas (APIs, bases de datos, etc.)
    - Controladores o cadenas de lógica (como en LangChain)
    - Filtros de seguridad

---

3. ¿Cómo se mejora el rendimiento general?

- **Evaluando cada componente por separado**:
    - ¿El prompt está bien diseñado?
    - ¿El modelo genera respuestas útiles?
    - ¿Las herramientas se usan correctamente?
- Luego se ajustan esos componentes para mejorar el resultado final.

---

4. ¿Qué son métricas personalizadas?

- Son métricas diseñadas específicamente para un **componente o tarea concreta**, por ejemplo:
    - Tiempo de respuesta de una API externa.
    - Relevancia del output del modelo según la intención.
    - Costo por llamada a modelo o servicio.

---

5. ¿Qué es la evaluación online?

- Evaluar mientras el sistema está en uso real (**en producción**).
- Permite ver **cómo funciona con usuarios reales** a lo largo del tiempo.
- Clave para mejorar continuamente a **gran escala** (con muchos usuarios y datos).

Model deployment fundamentals

1. Tipos de despliegue de modelos

📦 Batch (por lotes)

- El modelo procesa muchos datos a la vez, en intervalos.
- Ideal para tareas **no urgentes**, como generar reportes diarios o análisis históricos.

💧 Stream (en flujo)

- El modelo procesa datos en tiempo casi real, conforme llegan.
- Útil para **eventos continuos**, como monitorear sensores o redes sociales.

⚡ Real-time (tiempo real)

- El modelo responde inmediatamente a cada solicitud.
- Es ideal para **interacciones rápidas** como chatbots, motores de recomendación o sistemas de fraude.

---

2. ¿Cuándo usar cada tipo?

Tipo	Ideal para...
Batch	Procesos programados o periódicos
Stream	Datos en constante flujo (telemetría)
Real-time	Respuestas instantáneas al usuario

---

3. Batch vs Real-Time en Databricks

- **Batch**: Se integra bien con trabajos de Spark y pipelines agendados.
- **Real-Time**: Requiere endpoints activos y baja latencia, pero más costoso y complejo de mantener.

---

4. MLflow y despliegue

- **Model Flavors**: Diferentes formatos compatibles (e.g., sklearn, pytorch, tensorflow).
- **Deploy Client**: Herramienta para desplegar modelos en múltiples plataformas desde MLflow.

---

5. Beneficios de usar Unity Catalog para registrar modelos

- **Seguridad**: Control de acceso centralizado.
- **Organización**: Mejora la gestión y descubrimiento de modelos.
- **Trazabilidad**: Puedes ver quién registró o cambió un modelo y cuándo.

Batch deployment

1. ¿Qué es el batch deployment?

Es una forma de usar un modelo entrenado para hacer inferencias **sobre un conjunto de datos grande**, **en intervalos definidos** (por ejemplo: cada hora, diariamente o semanalmente).

---

2. ¿Cuándo usar batch deployment?

- Cuando **no necesitas resultados en tiempo real**.
- Ejemplos:
    - Generar reportes de riesgo financiero al final del día.
    - Clasificar grandes volúmenes de texto en un pipeline nocturno.
    - Predecir demanda semanal de productos.

---

3. Ventajas del batch deployment

✅ Más **eficiente en costos** que el tiempo real.
✅ **Escalable** para grandes volúmenes de datos.
✅ Fácil de automatizar y programar (ej. con jobs en Databricks).

---

4. Desventajas

❌ No sirve para tareas que requieren **respuestas inmediatas**.
❌ Puede tener **retrasos** entre el momento del dato y la inferencia.

---

5. Workflow típico en Databricks (batch deployment)

1. Entrenar y registrar el modelo en MLflow.

2. Crear un job programado en Databricks (usando notebooks o workflows).
3. Cargar el modelo desde **Model Registry**.
4. Usarlo para hacer inferencia por lotes (batch inference).
5. Guardar los resultados (por ejemplo, en una tabla Delta o archivo CSV).

Real-time deployment

1. ¿Qué es real-time deployment?

Es cuando el modelo se despliega para **responder de inmediato** a cada solicitud individual del usuario o sistema.
Ejemplo: un modelo de recomendación que responde instantáneamente cuando navegas en una app.

---

2. ¿Cuándo es necesario el despliegue en tiempo real?

- Cuando se requiere **respuesta inmediata**, como en:
    - Motores de recomendación.
    - Chatbots o asistentes virtuales.
    - Detección de fraude.
    - Aplicaciones interactivas.

---

3. Desafíos del despliegue en tiempo real

❌ **Baja latencia**: debe responder en milisegundos.
❌ **Alta disponibilidad**: debe estar siempre activo.
❌ **Escalabilidad**: debe soportar muchos usuarios al mismo tiempo.
❌ **Costo**: mantener endpoints activos puede ser caro.

---

4. ¿Qué ofrece Databricks Model Serving?

✅ Crear endpoints en pocos clics desde la interfaz.
✅ Escalado automático.
✅ Monitoreo, logging y métricas de rendimiento.
✅ Soporte para modelos registrados en MLflow.

---

5. Flujo típico de real-time deployment en Databricks

1. Entrenar y registrar el modelo en MLflow.
2. Activar Model Serving desde la UI o API.
3. Publicar el modelo como un **endpoint REST**.
4. Enviar solicitudes con datos de entrada y obtener respuesta inmediata.
5. Monitorear el rendimiento desde el panel.

AI system monitoring

1. ¿Por qué es importante monitorear un sistema de IA?

Porque un modelo que funcionaba bien al principio puede:

- **Perder precisión** con el tiempo (data drift).
- Responder mal ante nuevos tipos de datos o usuarios.
- Generar respuestas inadecuadas o costosas.

El monitoreo permite:
✅ Detectar fallos antes de que afecten al usuario.
✅ Mejorar continuamente el sistema.
✅ Cumplir con normativas y seguridad.

---

2. ¿Qué métricas se deben monitorear?

🔧 Por componente del sistema

- **Modelo (LLM)**: precisión, latencia, tasa de error, costo por consulta.
- **Prompt / Cadena de lógica**: calidad de respuesta, coherencia, llamadas encadenadas.
- **API o herramientas externas**: tiempos de respuesta, disponibilidad.

📊 A nivel general del sistema

- Tiempo de respuesta total.
- Consumo de recursos (CPU, memoria, tokens).
- Satisfacción del usuario (medida por retroalimentación o clics).
- Porcentaje de respuestas rechazadas por filtros de seguridad.

---

3. ¿Qué es Lakehouse Monitoring?

Es una herramienta de Databricks que permite:

- Monitorear el rendimiento **en línea (online)** de tu aplicación GenAI.
- Visualizar métricas clave en tiempo real.
- Detectar anomalías automáticamente.
- Unificar el monitoreo de datos y modelos en una misma plataforma.

LLMops concepts

1. ¿Qué es LLMOps?

LLMOps (Large Language Model Operations) es la práctica de **gestionar, desplegar y mantener modelos de lenguaje grande (LLMs)** en producción, de forma **segura, escalable y eficiente**.

Es similar a MLOps, pero con retos únicos por el tamaño, complejidad y comportamiento impredecible de los LLMs.

---

2. ¿En qué se diferencia LLMOps de MLOps?

Aspecto	MLOps	LLMOps
Tipo de modelo	Modelos tradicionales (ML)	Modelos fundacionales (LLMs)
Datos de entrenamiento	Generalmente internos y controlados	Usualmente externos y masivos
Evaluación	Métricas tradicionales (accuracy, etc.)	Evaluaciones más subjetivas y complejas
Riesgos	Moderados	Altos (alucinaciones, toxicidad, sesgos)
Costos	Relativamente bajos	Altos (por tokens y cómputo)

---

3. Arquitectura recomendada de LLMOps

- **Entrenamiento/Fine-tuning** (opcional).

- **Evaluación y pruebas continuas**.
- **Despliegue (serving)** con endpoints controlados.
- **Monitoreo constante** de rendimiento, seguridad y costos.
- **Control de versiones** de modelos, prompts y chains.
- **Ciclo cerrado de feedback** para mejora continua.

---

4. Ciclo de vida típico en LLMOps

1. **Experimentación**: prompt engineering, pruebas con LLMs.
2. **Evaluación**: con usuarios, métricas automáticas o LLM-as-a-judge.
3. **Despliegue**: mediante Model Serving u otras APIs.
4. **Monitoreo**: latencia, costo, calidad, errores.
5. **Iteración**: se ajustan prompts, se cambia el modelo, se mejora el pipeline.