DATABRICKS GENERATIVE AI ENGINEER ASSOCIATE · CHEAT SHEET · CERTIFYDECK

Databricks Certified Generative AI Engineer Associate

Dominios, pesos oficiales y habilidades clave del temario de Databricks Generative AI Engineer Associate, para repasar antes del examen.

Nivel: Intermediate (Associate) Duración: 90 min Fuente: guía de estudio oficial de Microsoft Learn

1. Design Applications

14–14%

Design Applications

- Para obtener un formato concreto, combina instrucciones claras con ejemplos few-shot del resultado deseado y, cuando la salida debe ser JSON, usa structured outputs: response_format de tipo json_schema en Foundation Model APIs o responseFormat en ai_query. Los esquemas simples y planos dan mejores resultados, y un paso de post-procesado puede validar el formato y aplicar una respuesta de reserva.
- Traduce cada requisito de negocio a una tarea de modelo: clasificación con etiquetas fijas, extracción de campos con esquema, resumen, traducción, análisis de sentimiento, enmascarado de entidades o embeddings para similitud semántica. Databricks recomienda empezar por las AI Functions específicas de cada tarea y usar ai_query cuando haga falta controlar el prompt, el modelo o consultar un modelo propio o ajustado.
- Una cadena RAG combina comprensión de la consulta (reescritura, extracción de filtros), recuperación (semántica, por palabras clave o híbrida, con reranking y filtros de metadatos), aumento del prompt, el LLM y post-procesado con guardrails. Cada componente se elige según la entrada y la salida deseadas, y conviene usar modelos más pequeños en pasos sencillos como clasificar la intención.
- Antes de construir, define con los interesados qué entradas recibirá el sistema y quién las envía, cómo debería ser la respuesta ideal, qué criterios marcan una buena respuesta (precisión, tono, citas, longitud) y qué restricciones de coste, latencia y escala hay. De ahí salen la forma de entrada y salida, por ejemplo citas de fuentes, campos personalizados o un proceso por lotes sobre tablas.
- Los agentes usan herramientas para recopilar información (consultas a tablas, recuperadores vectoriales, Genie) y para actuar (APIs externas, funciones de Unity Catalog). El orden lógico es recopilar datos, razonar y solo después actuar, con aprobación humana en acciones de riesgo. Las descripciones claras de cada herramienta guían su selección, y los límites de iteraciones evitan bucles.

2. Data Preparation

14–14%

Data Preparation

- El fragmentado debe seguir la estructura del documento (cabeceras Markdown o HTML, párrafos, funciones de código) para crear fragmentos coherentes y con metadatos útiles. El tamaño equilibra detalle y contexto, el solapamiento evita perder información en los cortes y cada fragmento debe caber en el límite de tokens del modelo de embeddings, medido con su propio tokenizador.
- Antes de fragmentar, limpia el texto de ruido como cabeceras, pies de página, números de página o menús de navegación HTML, elimina duplicados y casi duplicados (por ejemplo con MinHash) y filtra documentos irrelevantes, obsoletos, tóxicos o con datos sensibles usando metadatos y clasificadores. ai_parse_document etiqueta cada elemento por tipo, lo que permite descartar los que no aportan.
- La herramienta de extracción depende del formato: PyPDF2 o unstructured para PDF con texto, BeautifulSoup o lxml para HTML y OCR (por ejemplo, Tesseract) para escaneados e imágenes. En Databricks, ai_parse_document analiza PDF, Office e imágenes leídos como binario y devuelve elementos tipados (texto, tablas en HTML, encabezados), lo que permite limpiar ruido como encabezados y pies de página.
- El orden habitual es: leer los archivos de un volumen con binaryFile, analizarlos, trocearlos y expandirlos a una fila por fragmento con un ID único y la URI del documento, guardar la tabla Delta en Unity Catalog, activar el change data feed y crear el índice Delta Sync. Para cambios incrementales, MERGE INTO aplica altas, cambios y bajas en una sola operación.
- El corpus correcto depende del caso de uso: conviene implicar desde el principio a expertos del dominio para elegir fuentes y consultas probables. Antes de indexar hay que eliminar duplicados (por ejemplo con MinHash), filtrar documentos obsoletos, poco fiables, tóxicos o con PII, resolver contradicciones y añadir de forma incremental las fuentes que falten.

3. Application Development

30–30%

Application Development

- Databricks AI Bridge ofrece paquetes por framework: databricks-langchain (ChatDatabricks, DatabricksEmbeddings, DatabricksVectorSearch y VectorSearchRetrieverTool) y databricks-openai para el SDK de OpenAI. Los índices con embeddings autogestionados requieren indicar text_column y modelo de embeddings. Con cualquier framework, la interfaz ResponsesAgent de MLflow integra trazas, evaluación y despliegue.
- La calidad se valora con jueces de MLflow: RetrievalGroundedness detecta alucinaciones, Safety contenido dañino y Guidelines reglas de tono o cumplimiento en lenguaje natural, sin datos de referencia; Correctness requiere hechos esperados. La Review App recoge la opinión de expertos. Evaluar y monitorizar mide, pero bloquear respuestas dañinas exige barreras de seguridad en tiempo de ejecución.
- Elegir el chunking es iterativo y depende de los datos: el tamaño fijo es rápido pero rara vez da chunks coherentes; por párrafos, por estructura (encabezados Markdown o HTML) o semántico respetan mejor el sentido. Los chunks pequeños localizan mejor y los grandes aportan contexto; el solapamiento evita perder información en los cortes y el esquema padre-hijo combina ambas cosas. Cualquier cambio se valida con una evaluación de recuperación.
- Antes de recuperar conviene transformar la consulta: reescribirla para reflejar la intención (por ejemplo, con el historial del chat), extraer filtros como fechas, productos o países y clasificar la intención en llamadas separadas al LLM. Los filtros solo funcionan si el pipeline guarda esos campos como metadatos y el recuperador los aplica. La plantilla del prompt se parametriza con datos de la petición, como la fecha o el contexto del usuario.
- Para llevar la respuesta de un LLM de una base a la salida deseada se itera sobre el prompt: ejemplos few-shot para fijar formato y estilo, chain-of-thought para preguntas de varios pasos e instrucciones claras y específicas. Los prompts no siempre se transfieren entre modelos. Si hace falta un JSON fijo, mejor structured outputs con response_format, y mlflow.genai.optimize_prompts() mejora prompts con datos y scorers.

4. Assembling and Deploying Applications

22–22%

Assembling and Deploying Applications

- Un modelo pyfunc personalizado hereda de mlflow.pyfunc.PythonModel: load_context carga una sola vez artefactos como tokenizadores o pesos, y predict ejecuta en cada petición el preprocesado, la llamada al modelo y el posprocesado. El código local compartido se incluye con code_paths, y mlflow.models.predict valida dependencias y entradas antes de desplegar.
- Para recursos externos, Model Serving inyecta credenciales con variables de entorno que referencian secretos ({{secrets/scope/key}}); quien crea el endpoint necesita READ sobre el secreto. Para recursos de Databricks, la autenticación automática se activa declarando resources al registrar el agente; la autenticación en nombre del usuario usa scopes y se inicializa en predict.
- Una cadena sencilla de LangChain compone pasos con el operador |, por ejemplo una ChatPromptTemplate con variables y un modelo ChatDatabricks que envuelve un endpoint de Model Serving; invoke recibe un diccionario con esas variables. Los prompts pueden cargarse del Prompt Registry por alias, y pasos como la reescritura de consultas mejoran la recuperación en chats multiturno.
- Una aplicación RAG se registra con un flavor como langchain o pyfunc usando Models from Code; las consultas usan el mismo modelo de embeddings que los documentos; el retriever necesita text column y modelo de embeddings si estos son autogestionados; las dependencias omitidas se añaden con extra_pip_requirements; un input example válido permite inferir la firma que exige Unity Catalog.
- Para registrar en Unity Catalog se usa un nombre de tres niveles (catálogo.esquema.modelo) con mlflow.register_model o registered_model_name, y set_registry_uri("databricks-uc") si el catálogo por defecto no es de Unity Catalog. Crear el modelo exige CREATE MODEL y USE SCHEMA; añadir versiones, ser propietario o CREATE MODEL VERSION. Los alias desacoplan despliegues del código.

5. Governance

8–8%

Governance

- Enmascarar en lugar de bloquear permite proteger la PII sin rechazar peticiones: el guardrail de PII de AI Gateway admite Block o Mask, y la detección determinista por patrones añade poca latencia frente a los jueces LLM, que se reservan para nombres y texto libre. En batch, ai_mask oculta entidades antes de indexar; los column masks de Unity Catalog rinden mejor con funciones SQL sencillas y reutilizadas.
- Contra entradas maliciosas se combinan capas: guardrails integrados de AI Gateway (Jailbreak en peticiones, Unsafe Content en entrada o salida), políticas personalizadas en SQL para reglas propias y modelos guardián en Model Serving. Un agente es vulnerable cuando une datos privados, entradas no fiables y salida al exterior; limitar el tráfico saliente o los permisos rompe esa cadena. DetectJailbreak evalúa las entradas sin juez LLM.
- Antes de ingerir una fuente hay que leer su licencia: en Hugging Face está en los metadatos de la dataset card y en Databricks Marketplace en las condiciones de uso del proveedor. Los modelos servidos en Databricks tienen sus propias condiciones, y su cumplimiento es responsabilidad del cliente. Las governed tags etiquetan cada tabla con su licencia y el lineage de Unity Catalog localiza lo que hay que borrar si se retira un permiso.
- El texto problemático se trata en el pipeline de datos, no en el prompt: se guarda el origen en bruto para trazabilidad, se analizan los PDF con ai_parse_document, se quita el texto repetitivo conservando la estructura, se eliminan casi duplicados con MinHash y se filtran documentos dañinos con un clasificador de toxicidad. Lo que deba conservarse, como versiones antiguas, se excluye con filtros de metadatos al recuperar.

6. Evaluation and Monitoring

12–12%

Evaluation and Monitoring

- Se elige el LLM evaluando todos los candidatos con el mismo dataset y los mismos scorers, descartando los que no cumplen los requisitos obligatorios y comparando después coste y velocidad. La latencia total es el time to first token más el time per output token por los tokens generados; en lotes manda el throughput. Un MoE usa solo parte de sus parámetros por entrada, y cuantizar exige repetir la evaluación.
- Elige métricas según el riesgo del caso: calidad de recuperación (precisión y recall de fragmentos), calidad de respuesta (groundedness, relevancia, safety, correctness) y rendimiento (tokens como coste y latencia). En RAG conviene medir recuperación y respuesta para localizar la causa de un fallo; las métricas de salud del endpoint cubren errores, latencia y uso de CPU y memoria.
- mlflow.genai.evaluate() ejecuta el agente (predict_fn) sobre un conjunto de evaluación, captura trazas y aplica scorers; también puntúa salidas ya existentes (answer sheet) o trazas obtenidas con mlflow.search_traces(). Las claves de inputs deben coincidir con los argumentos de predict_fn, y los jueces de recuperación necesitan un span de tipo RETRIEVER en la traza.
- Las inference tables registran automáticamente peticiones y respuestas de un endpoint en una tabla Delta de Unity Catalog; en agentes guardan también trazas de MLflow y deploy() las activa por defecto. Sirven para depurar, auditar y crear conjuntos de evaluación uniéndolas con etiquetas. No se puede usar una tabla existente ni cambiar su esquema o nombre, y algunos errores pueden no registrarse.
- Para controlar costes: pago por token en pilotos con tráfico irregular y provisioned throughput cuando hacen falta garantías; scale to zero en endpoints poco usados (con arranque en frío); ai_query con modelos alojados para inferencia batch; etiquetas en los endpoints, que llegan a system.billing.usage, para atribuir el gasto; y tasas de muestreo menores para los scorers caros en producción.