Solución de Prueba Técnica Desarrollador IA

Ana Valentina Lopez Chacon

08 de Enero, 2026

1 Descripción del problema

Según lo planteado en la prueba técnica [1], una organización recibe diariamente una gran cantidad de documentos en formato PDF, como contratos, resoluciones administrativas, quejas, oficios e informes técnicos, que deben ser clasificados manualmente para su gestión interna. Este proceso presenta múltiples problemas operativos, siendo la necesidad principal es **automatizar la clasificación inicial de documentos**, el objetivo funcional de la solución es diseñar un sistema basado en Modelos de Lenguaje de Gran Escala (LLMs) que sea capaz de:

- Extraer texto desde documentos PDF.
- Analizar semánticamente el contenido.
- Clasificar el documento en una categoría definida.
- Generar una justificación legible de la clasificación.
- Ejecutarse de forma reproducible, ya sea localmente o en Microsoft Azure.

Las categorías de clasificación consideradas son: Contrato, Queja o Reclamo, Resolución administrativa, Informe técnico o Comunicación interna.

2 Descripción de la solución propuesta

La solución implementada responde directamente a las actividades y criterios de evaluación establecidos en la prueba técnica, cubriendo los aspectos funcionales, técnicos y operativos.

2.1 Selección del ambiente de ejecución

Se optó por una solución **basada en Microsoft Azure (Azure OpenAI)**, complementada con una implementación **local en Python**, con el fin de garantizar:

- Reproducibilidad de los resultados.
- Facilidad de validación.
- Separación clara entre infraestructura y lógica de negocio.

2.2 Diseño funcional de la solución

Desde un punto de vista funcional, el flujo de la solución es el siguiente:

1. Carga de uno o más documentos PDF.
2. Extracción del texto completo del documento.

3. Normalización básica del texto.

4. Envío del contenido al LLM.

5. Recepción de la categoría y justificación.

6. Presentación del resultado de forma estructurada.

Este flujo permite automatizar completamente la clasificación inicial sin intervención humana.

2.3 Diseño técnico y arquitectura

La separación entre *recurso* y *deployment* permite centralizar credenciales, cambiar de modelo sin modificar la lógica del cliente y facilitar escalabilidad futura. Como primer paso, se creó un recurso de tipo **Azure OpenAI** dentro del portal de Microsoft Azure, el cual actúa como punto central de acceso a los modelos de lenguaje desplegados. Este recurso define el *endpoint* público y las credenciales necesarias para autenticar las solicitudes realizadas desde el cliente Python.

Una vez creado el recurso, Azure genera automáticamente dos claves de acceso (API Keys) asociadas al mismo. Estas claves permiten autenticar peticiones al servicio sin necesidad de exponer credenciales de usuario final, siguiendo un modelo de autenticación basado en secreto compartido. En esta solución se utiliza exclusivamente una de estas claves, almacenada en un archivo separado y nunca embebida directamente en el código fuente.

Es importante destacar que las credenciales y el endpoint están asociados al *recurso Azure OpenAI*, y no al modelo en sí. El modelo (`gpt-4o-mini`) se expone a través de un *deployment* lógico (`doc-classifier-gpt4o-mini`), el cual puede ser referenciado desde el código cliente sin necesidad de regenerar credenciales. Esta separación permite modificar o sustituir el modelo desplegado sin impactar la lógica de consumo ni los mecanismos de autenticación.

Desde el punto de vista de seguridad, esta aproximación garantiza:

- Aislamiento de credenciales respecto al código.
- Posibilidad de rotación de claves sin cambios en la implementación.
- Cumplimiento de buenas prácticas básicas de gestión de secretos.

2.4 Selección y uso del LLM

Se seleccionó el modelo `gpt-4o-mini` desplegado en Azure OpenAI debido a que tiene un buen desempeño en tareas de clasificación semántica al igual que un bajo costo relativo frente a modelos más grande, además de soporte nativo para respuestas estructuradas. El deployment fue nombrado explícitamente como `doc-classifier-gpt4o-mini` para facilitar su referencia directa desde código.

2.5 Configuración del entorno

El entorno de ejecución se configuró en Python utilizando:

- `openai` (AzureOpenAI SDK).
- `pydantic` para validación estructurada.
- `langchain-core` para parseo de salidas.

Las credenciales se gestionan exclusivamente mediante variables separadas que no se incluyen en la entrega final, sin embargo el video demo prueba su funcionamiento correcto. Esto cumple con los criterios de seguridad básica exigidos.

2.6 Diseño y desarrollo del prompt

El prompt fue diseñado de forma estructurada para definir claramente las categorías, incluir definiciones semánticas y forzar una salida estrictamente en formato JSON. La temperatura se fijó en `0` para garantizar comportamiento determinista y reproducible. Durante el desarrollo de la solución se adoptó un enfoque incremental y pragmático, priorizando la validación funcional y la reproducibilidad sobre configuraciones complejas o dependientes de la interfaz gráfica de Azure.

Inicialmente se evaluó el uso de Prompt Flow dentro de Azure Machine Learning como mecanismo de orquestación del prompt. Sin embargo, debido a limitaciones prácticas del entorno (compatibilidad entre modos de *completion* y *chat*, visibilidad de deployments y dificultad para depurar errores en etapas tempranas), se optó por trasladar el núcleo de la lógica a una implementación local en Python.

Esta decisión respondió a los siguientes criterios técnicos:

- Mayor control sobre el formato de entrada y salida del modelo.
- Facilidad para depurar errores y validar comportamiento determinista.
- Reproducibilidad total sin depender de configuraciones específicas de la interfaz web.
- Posibilidad de integrar rápidamente preprocesamiento, validación y protección de datos sensibles.

Una vez validado el comportamiento del modelo y estabilizado el prompt, la solución queda preparada para ser reintegrada a servicios gestionados como Prompt Flow o Azure ML si se requiere una orquestación más avanzada o despliegue a gran escala. Asimismo, se tomó la decisión explícita de fijar la temperatura del modelo en `0`, eliminando cualquier aleatoriedad en la generación de respuestas. Esto es particularmente importante en escenarios administrativos o legales, donde la trazabilidad y la consistencia de resultados son requisitos fundamentales.

Finalmente, se incorporó un proceso de pseudonimización de datos sensibles previo al envío de la información al modelo. Esta estrategia permite preservar la estructura semántica del documento sin exponer información personal identificable, alineándose con prácticas estándar de protección de datos y mitigando riesgos asociados al tratamiento de información sensible mediante modelos de lenguaje.

2.7 Clasificación y generación de resultados

El LLM devuelve una respuesta estructurada con: Categoría asignada y Justificación textual basada en evidencia del documento. La salida es validada automáticamente mediante un esquema Pydantic, garantizando consistencia y verificabilidad.

2.8 Ejecución y evidencia

La solución ejecutada puede verse desde el notebook `TechTest.ipynb` o ejecutando en local la app de streamlit de `app.py` (ambas formas se muestran en el video). Los resultados pueden ser fácilmente auditados y reproducidos utilizando el mismo conjunto de documentos. Dado el alcance de la prueba técnica y el objetivo de demostrar la validez funcional y arquitectónica de la solución, se optó por un modo de ejecución local controlado, el cual permite validar el pipeline completo sin requerir un despliegue productivo permanente.

3 Conclusión

La solución desarrollada cumple con todos los objetivos planteados en la prueba técnica, ya que aplica LLMs a un problema real, presenta un diseño arquitectónico claro y justificable usando Azure AI y produce resultados verificables, reproducibles y documentados.

Esta solución presenta no solo el resultado de la clasificación, sino también la justificación de por qué se llegó a esa conclusión. Esto proporciona explicabilidad al código implementado, dado que los LLM pueden ser difíciles de gestionar y pueden alucinar mucho, lo que proporciona cobertura legal y trazabilidad para el modelo en escenarios de alto riesgo.

Para un análisis más detallado, si quisiéramos procesar una gran cantidad de consultas, sería necesario paralelizar este proceso, ya que cada entrada es independiente de las demás. Si se ejecuta localmente, se utilizaría un hilo para utilizar diferentes CPU. Siguiendo esta línea, el pipeline RAG sería también la forma más natural de abordar esta solución con más tiempo, almacenando diferentes cantidades de ejemplos y bases de datos vectoriales para que el proceso de recuperación sea más fluido y rápido y evitar realizar demasiadas llamadas a la API.

References

[1] Prueba Técnica – Desarrollo de Proyecto de IA (Linux o Microsoft Azure), Procuraduría General de la Nación / DAE, 2025.