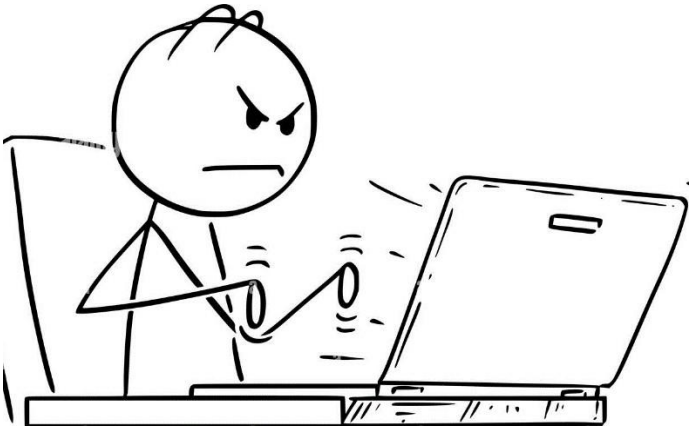
Como parte del proceso de selección para incorporarte a Fenie Energía nos gustaría conocer cómo analizas y te enfrentas a problemas parecidos a los que te podrás encontrar en un día trabajando con nosotros, es por esto por lo que te proponemos el siguiente reto.

Feníe Challenge | GEN-AI AL RESCATE!



Jesús, trabajador de una empresa de telecomunicaciones se encuentra frustrado porque dedica demasiado tiempo a hacer una tarea muy repetitiva, ¿su misión?, etiquetar emails que le llegan a su correo electrónico como emails de quejas, sugerencias o peticiones de servicio.
Tú como parte del equipo de sistemas te has enterado del problema y quieres ayudarlo. Te pones manos a la obra para poder echarle una mano a Jesús y decides utilizar un **modelo encoder preentrenado tipo BERT/Roberta** para **clasificación en inferencia** (sin reentrenar), generando directamente una etiqueta por correo con **semilla fija**.

Para poder ayudar a Jesús él ha exportado un fichero con todos los emails que tiene pendientes en la bandeja de entrada y los registros tienen la siguiente estructura:

Fecha llegada email	Email remitente	Texto del correo electrónico

Clasificación (sin reentrenar): usar un modelo encoder preentrenado tipo BERT/Roberta para asignar una etiqueta por correo en inferencia (sin fine-tuning), con semilla fija y probabilidad/confianza.

Revisión/consultas (RAG): usar otro LLM conversacional, únicamente en el front (Streamlit), apoyado (opcionalmente) en una base de datos vectorial para búsqueda semántica y preguntas en lenguaje natural. Para ello te sugerimos que sigas los siguientes pasos:

1. Lo primero que tendremos que hacer será desplegar nuestra infraestructura en nuestro entorno local, para lo que te recomendamos que uses Docker. En nuestro caso la única pieza que necesitaremos a nivel de servicio será una base de datos vectorial, en nuestro caso te recomendamos que utilices qdrant, haciendo uso de este contenedor: https://qdrant.tech/documentation/quick-start/. Aunque puedes usar la que te sea más cómoda, como Pinecone o PGVector, etc.

2 Una vez que tengas listo tu BD vectorial empezamos con el código. A nosotros nos encanta usar git, así que si tu código acaba en un repositorio de Github privado nos encantará ver tus *commits* el día que nos muestres la prueba"El código debe utilizar un modelo encoder preentrenado (p. ej., BERT/Roberta en Hugging Face) para la clasificación en inferencia (sin reentrenar). Además, podrás usar un LLM conversacional separado solo para el RAG del front. Nosotros, para buscar este tipo de modelos solemos buscarlos en Huggingface, te aconsejamos que tú también lo hagas para que te sea más sencillo. Este proceso de Python deberá realizar los siguientes pasos:

1) Leer el fichero csv
2) Pasar cada texto por el modelo y etiquetarlo con la etiqueta que más se acerque de las siguientes:
    a. Quejas
    b. Petición de servicio
    c. Sugerencia de mejora
3) Insertar las cuatro columnas en qdrant en una colección llamada 'emailclassification' o lo escribes en un fichero csv.

3 Jesús no sabe usar qdrant ni sabe lo que es un embedding, así que le encantará que montes un front con la librería Streamlit de Python para poder hacer preguntas directas sobre la base de datos vectorial de manera trasparente. Añade todas las funcionalidades extras que consideres interesantes para Jesús, así como posibilidades de búsqueda en su base de datos de manera semántica (aprovechando la base de datos vectorial), su escalado: ¿Cómo manejarías la inserción de nuevos correos o la actualización de etiquetas? ¿Sería útil filtrar por tipo de queja o fecha?

Aclaraciones

1) Enfoque de modelo (clasificación vs RAG)

- **Clasificación automática (sin reentrenar):** utiliza un modelo encoder preentrenado tipo BERT/Roberta únicamente en modo inferencia para asignar una etiqueta por correo (*queja*, *petición de servicio*, *sugerencia de mejora*). No se requiere fine-tuning.
- **Revisión y consultas (RAG):** de forma separada, puedes usar un LLM conversacional en el front (Streamlit) para hacer preguntas en lenguaje natural y revisar resultados, apoyándote en recuperación semántica.

2) Determinismo y reproducibilidad

- Fija una **semilla** (ej.: 42) y usa el encoder en modo inferencia sin dropout (es decir, sin aleatoriedad) para que, a igualdad de entrada, la predicción sea reproducible.
- Se recomienda registrar la **confianza/probabilidad** junto con la etiqueta.

3) Base de datos vectorial (alcance)

- La **BD vectorial** (p. ej., Qdrant/PGVector) es **para el módulo de RAG en el front** (búsqueda/recuperación semántica y preguntas).
- La **clasificación** no depende de la BD vectorial; si lo prefieres, puedes entregar solo el **CSV** con las predicciones y usar la BD vectorial como **opcional**.

4) Formato mínimo de entrada/salida

- **Entrada (CSV):** campos equivalentes a *fecha de llegada*, *email remitente*, *texto del correo*.
- **Salida (CSV):** añadir **etiqueta_predicha** y **confianza**. Si usas BD vectorial, crea una colección (p. ej., emailclassification) con esos mismos campos.

5) Front (Streamlit) – funcionalidades esperadas

- Tabla con **etiqueta** y **confianza**, filtros básicos (tipo/fecha).
- RAG con un LLM para preguntas en natural y recuperación desde la BD vectorial.
- **Opcional:** permitir **corregir etiquetas** y **exportar** un CSV revisado.

6) Alcance y evaluación

- Los **datos son limitados**; no buscamos un sistema de producción.
- Valoraremos principalmente:
    1. **Arquitectura y separación de responsabilidades** (encoder para clasificación vs LLM para RAG).
    2. **Reproducibilidad** (semilla/determinismo).
    3. **Calidad y claridad del código** (README breve con instrucciones).
    4. **Usabilidad del front** y la explicación de decisiones.

¡Mucha suerte! ¡Estamos impacientes de ver lo que eres capaz de hacer!

Nota:

Esto es solo una sugerencia de implementación, siempre que los requisitos estén cubiertos y puedas explicar cómo has creado tu sistema, será totalmente válido.