Eden prueba técnica reporte

Enrique Marquez

January 2026

Notas generales

- Para todos los ejercicios se utilizo Gemini 3.0 pro. Este modelo fue usado vía API y via UI para consultas. Pensé en utilizar ChatGPT como comparación pero no me dio el tiempo de continuar con esas comparaciones.
- Todos los ejercicios son Minimum Value Exercises realizados durante aproximadamente 10 horas de trabajo
- Se utilizo el IDE antigravity, sin embargo la gran mayoría del código fue hecho a mano con poca interacción con el asistente de antigravity. Pocos métodos fueron generados usando Gemini UI website
- Las conclusiones y decisiones fueron totalmente propias con poco tiempo de research. Mi objetivo con esta decisión es demostrar mis capacidades sin la utilización de información externa. De manera evidente, utilizando papers o discutiendo mas con alguna IA pudiese llegar a conclusiones o vías mas refinadas
- No se utilizo AI para refinar textos ni corregir errores ortográficos en notebooks, scripts o este reporte. Apologies por tildes o errores que puedan estar faltando
- Se utilizo **uv** como lib & env manager.
- No hubo particionamiento de la data entre training, validation y test debido al tiempo de desarrollo que tenia para invertir en este desafío.
- Por simplicidad no decidí usar JSON Schema. Todos los JSON outputs los estoy manejando explicados dentro del mismo system instruction.
- El procesamiento de casos se realizan en un solo hilo, esto puede beneficiarse de paralelismo o batch processing para agilizar el proceso de análisis
- El código de los escenarios puede ser abstraído de mejor manera para evitar repetir secuencias de código una y otra vez. Sin embargo, por cuestiones de tiempo esta abstracción no se implemento.

- Todos los análisis de este reporte pueden beneficiarse de atributos característicos del paciente. Enriqueciendo de esta forma el contexto de la LLM sobre el caso de estudio.

Escenario 1

Para este ejercicio se genero el notebook escenario1-baseline/notebooks/general-notebook.ipynb. El notebook tiene info de los analisis. A su vez, esta en modo bitácora de ejecuciones. De esta manera busque plasmar mi chain of thought. En las siguientes sub-secciones describo findings y análisis hechos dentro del notebook

Análisis exploratorio del dataset

En la siguiente itemización muestro la cadena de pensamientos y ejecuciones del análisis de la metadata

- Lo primero que realice fue leer el df y ver los campos proporcionados, entre ellos se observo varios identificadores y el study description (el cual define el tipo de estudio, metadata relevante). Se leyeron los campos **field_value** para entender el formato y posible estructura. Se observo que esta en html con el texto embebido dentro de este html

- Se tomo la decisión de limpiar el html e ignorar los html-tags. Esta decisión puede ser perjudicial (debido a que puede haber información relevante en los html-tags [por ejemplo negritas o cursivas sobre palabras relevantes]), sin embargo por motivos de tiempo se prefirió usar lo que llamé *cleaned_text*

- Este cleaned text lo genero con un método generado por Gemini, el cual permite limpiar el html text usando BeautifulSoup. Este método puede verse utilizado en el notebook

- De manera curiosa se observo que todos los textos estaban divididos en dos. Una primera sección sin header y otra que empieza con IMPRESIÓN DIAGNÓSTICA. Determinamos que en los 19 casos existía esta división de información.

- A su vez, se observo que la segunda sección era típicamente mas corta que la primera

- En la siguiente celda veo la distribución de study description. En la cual se ve que la mayoría de los estudios son RX DE TÓRAX POSTERO-ANTERIOR. Se pudiese analizar mejor si existe una relación entre el tipo de estudio y la cantidad de palabras o incluso tipo de diagnostico, sin embargo se decidió continuar con el ejercicio y no buscar correlacionar este tipo de variables.

Métricas

Pensando en métricas que considere sean invariantes a la semántica del medico o invariante al wording (e.g categorizar "normal" y "sin anomalías" como la misma categoría), se me ocurrieron dos particulares. Cabe destacar que no se utilizo ninguna métrica de NLP debido a la ineficacia que puede tener hacer matcheos de palabras o letras. Intentar vectorizar el diagnostico (utilizando Bag Of Words u otro método) también quedo descartado principalmente por la sensibilidad a palabras que cambien totalmente el significado (ejemplo, "No se encuentra flema" con "Se encuentra flema")

Debido a esto solo voy a estar utilizando dos métricas que permitan comparaciones semánticas y no necesariamente exactas. Particularmente utilizando LLMs como juez de comparación de algun u otra manera

Importante mencionar que estas métricas buscan comparar diagnósticos sin importar la semántica o el wording. Buscan poder objetivizar el diagnostico.

Los system instructions utilizados tanto para la generación de reportes como para la comparación fueron hechos partiendo de un prompt y sirviendo a gemini como prompt engineer.

Sobre la forma de comparar generated vs ground truth se pueden calcular métricas como F1 Score, recall, precisión entre otras. Va a depender mucho del objetivo del producto que métrica específicamente se quiere tunear. Por ejemplo, si se quisiera que el sistema sirva dando asesorías directas a pacientes, tendríamos que maximizar tanto el recall como la precisión. Sin embargo, si solo es para asistir a radiólogos, buscaría que el sistema tenga un recall mas alto y no necesariamente una precisión tan alta (debido a que prefiero que el sistema diga todo lo que piensa posible así existan falsos positivos [de esta forma el radiólogo puede funcionar como filtro y la AI evitaría que se le puedan escapar cosas])

Para este ejercicio no se calculo ninguna métrica de Machine Learning. Se hizo énfasis en analizar semántica y objetivamente los errores criticos de la generación de reportes. Cabe mencionar de nuevo que las métricas pueden ser misleading sin algún forense que determine el error / missing / alucinaciones de los radiólogos, de esta forma tener un porcentaje de error de las métricas calculadas.

Diagnostic JSON normalization

La idea de esta métrica es tener un JSON universal que permita describir las características de un diagnostico. De esta manera si para cualquier texto de diagnostico puedo generar este JSON, pudiese comparar key a key dos diagnosticos "diferentes" y determinar la similaridad. De esta manera también pudiese utilizar un experto para que pondere cada key del JSON y tener un métrica laxa. Esto para que por ejemplo, si no es tan relevante para la comparación el (e.g) volumen de flema, pueda permitir diferencias en esta dirección y poder categorizarlo aun como verdadero positivo. Sin embargo, para este análisis vamos a asumir que todas las keys tienen la misma relevancia. Para esto se iteró desde gemini UI la generación del system instruction y puede ser encontrado en

src.system_instructions.escenario1_baseline.system_instruction_text_json_normalizer

Comparación text vs text con system instruction especializado

Esta métrica le comparte a la LLM ambos textos y utiliza un System Instruction que le hace saber el tipo de comparación semántica que tiene que hacer. El system instruction busca hacer énfasis en comparación medica semántica mas que palabra a palabra como mencionado anteriormente. Para esto se iteró desde gemini UI la generación del system instruction y puede ser encontrado en *src.system_instructions.escenario1_baseline.system_instruction_text_to_text_comparison*

Output de las comparaciones

Ambas "métricas" o comparaciones sacan el mismo output (se construyo el system instruction de esta manera para facilitar comparativas entre las métricas y reducir el trabajo individual de cada forma de comparación):

Output description

```
{
  "score": <integer 0-100>,
  "clinical_accuracy_status": "<CORRECT | MINOR_ERROR
     | CRITICAL_ERROR>",
  "missed_findings": ["<lista de hallazgos presentes
     en el Ground Truth pero ausentes en el generado>"
     ],
  "hallucinations": ["<lista de hallazgos inventados
     por el generado que no estan en el Ground Truth>"
     ],
  "reasoning": "<Explicaci n concisa de 1 o 2
     oraciones justificando el puntaje basandose en la
      comparaci n>"
}
```

Como se observa el output tiene score entre 0 y 100, definiendo la exactitud de generado vs el ground truth; un valor categórico definiendo el clinial accuracy, los hallazgos faltantes, las alucinaciones y el reasoning de las conclusiones. El score puede utilizarse para colocar un threshold y tuner alguna en función de alguna métrica, sin embargo para los analices que estaré mostrando solo nos quedamos con el valor categórico.

Para generar los reportes, métricas y comparaciones se creó el script generate_reports_and_metrics.py. El cual toma como input la carpeta de imágenes, el csv de metadata, las métricas a calcular, el output csv path y la temperatura del modelo.

Algunos análisis detallados pueden encontrarse en el notebook.

Conclusión sobre métricas, generación de reportes y resultados

La generación de reportes en ambas métricas fue pobre e ineficiente en esta primera iteración. Se observa que la mayoría de errores están en falta de información y no tanto en alucinaciones en el diagnostico. Esto me da a entender que el recall es mas bajo que la precisión, asumiendo que una alucinación es un falso positivo y un atributo faltante en el diagnostico es un falso negativo.

Las métricas cumplieron su objetivo y determinaron que la generación y diagnostico automático usando Gemini 3.0 esta lejos de ser perfecto.

Anexo tabla comparativa de los resultados de ambas comparaciones:

	json_normalizer	text_to_text_comparison
CRITICAL_ERROR	10	16
MINOR_ERROR	6	0
CORRECT	2	2

En el notebook buscó conseguir algún patrón en los errores pero no encontré ningún patrón bien definido mas que missings attributes are greater than hallucinations

Mejoras del pipeline de generación de reportes

Listo las mejoras que considero pueden aportar a la generación de reportes, considerando que pienso las métricas son las correctas. Es importante definir el objetivo de la generación de reportes y de esta manera buscar optimizar en alguna dirección, bien sea recall, precisión o f1 (u otra compuesta métrica asociada a análisis clínicos). Se asume un volumen mas grande de imágenes:

1. El system instruction fue realizado a manera de one shot, una sola interacción con gemini la cual llevo a los resultados mostrados en este reporte. Por lo que, optimizar este system instruction es crucial. Para ellos propongo utilizar un agente que tenga las tools de comparación, de esta forma realizar un pipeline agentico que maximice la métrica deseada. La salida de este pipeline es un system instruction optimizado. Algorithm 1 muestra pseudocodigo del proceso agentico iterativo.

2. Aplicar técnicas de finetuning como LoRa sobre patrones de errores. En mi experiencia hacer converger estos pipelines es complejo sin que caigan sobre cathastrophic forgetting. Sin embargo, si los patrones están bien identificados tiene sentido aplicar finetuning de la VLM sobre estos errores. Siempre considerando un set de control donde ya el pipeline esta funcionando correctamente para evitar el que "el remedio" sea peor que "la enfermedad"

3. Juez multi agentico. Utilizar varios mixture of experts con system instructions (potencialmente optimizado como en el punto uno) para las distintas etapas de diagnósticos. De esta manera, no tener una sola pegada a la LLM que tenga que observar todo los aspectos relevantes del

Algorithm 1 Iterative System Instruction Optimization

Require: Initial One-Shot System Instruction S_0, Training Set D_{train}, Test Set D_{test}, Iterations N

Ensure: Optimized Instruction S_{final}, Final Test Metrics M_{final}

$S_{current} \leftarrow S_0$
for $i \leftarrow 1$ **to** N **do**
    $R_{gen} \leftarrow \text{GenerateReports}(S_{current}, D_{train})$ ▷ Generate reports from images
    $Metric \leftarrow \text{CalculateMetric}(R_{gen}, D_{train})$
    $Errors \leftarrow \text{IdentifyErrorPatterns}(R_{gen}, D_{train})$ ▷ e.g., False Positives, False Negatives
    $S_{current} \leftarrow \text{GeminiRefine}(S_{current}, Errors)$ ▷ Update instruction to fix error patterns
end for
$S_{final} \leftarrow S_{current}$
$R_{test} \leftarrow \text{GenerateReports}(S_{final}, D_{test})$
$M_{final} \leftarrow \text{CalculateMetric}(R_{test}, D_{test})$
return S_{final}, M_{final}

diagnostico. Sino tener pegadas en paralelo y que cada pegada se encarge de algo especifico. Por ejemplo, una pegada se encarga de bone densitiy, otra de cancer detection, otra de apertura de espacios intercostales etc. Esta técnica ha demostrado en otras áreas alta efectividad, ya que cada "pegada" a la LLM es bastante objetiva, evitando confusiones internas y problemas adyacentes debido a esto. Luego de todas estas pegadas a la LLM tendría que haber una llamada final para hacer el merge de todos los sub-diagnósticos. Potencialmente, los sub-diagnósticos estén relacionados entre ellos y se tengan que mergear de manera acorde. La interacción y los system instructions pueden ser optimizados de manera similar al punto 1.

4. Conjunto de Jueces debatiendo. Tener un set de agentes "doctores" especializados en distintas áreas (system instructions orientados a diferentes áreas) (potencialmente distintas LLMs) que debatan entre ellas hasta llegar a una conclusión final. Potencialmente, se necesite un juez final que genere el veredicto. Este punto tiene relación con el punto 3. La interacción y los system instructions pueden ser optimizados de manera similar al punto 1.

Escenario 2

Para este ejercicio se genero el notebook escenario2-baseline/notebooks/general-notebook.ipynb. El notebook tiene info de los análisis.

Análisis de las diferencias de estilo entre médicos

Para entender estas diferencias se utilizo el script escenario2-baseline/generate_difference_in_reports.py el cual compara "one vs all" (uno a uno) diagnósticos de diferentes médicos. Este script utiliza el system instruction *src.system_instructions.escenario2_personalizacion_por_medico.system_instru* el cual retorna un JSON comparativo entre estilos:

```
{
  "style_comparison": {
    "verbosity_level": {
      "doctor_a": "<Alto/Medio/Bajo - Descripci n breve>",
      "doctor_b": "<Alto/Medio/Bajo - Descripci n breve>"
    },
    "structure_type": {
      "doctor_a": "<Ej: Estructurado por secciones / Narrativa libre>",
      "doctor_b": "<Ej: Estructurado por secciones / Narrativa libre>"
    },
    "tone_analysis": "<Comparaci n de 1 oraci n sobre la asertividad o formalidad (ej. A es m s acad mico, B es m s pragm tico)>"
  },
  "vocabulary_preferences": {
    "negative_findings_phrasing": {
      "doctor_a_example": "<C mo dice A que algo no est (ej. 'Sin alteraciones')>",
      "doctor_b_example": "<C mo dice B que algo no est (ej. 'Libre')>"
    },
    "anatomical_references": {
      "note": "<Diferencia en c mo nombran partes del cuerpo (Latinismos vs. Espa ol com n)>"
    }
  },
  "semantic_patterns": [
    {
      "pattern_name": "<Ej: Uso de 'Hedging' (Incertidumbre)>",
      "observation": "<Ej: El M dico A usa frecuentemente 'podr a corresponder a', mientras el M dico B diagnostica directamente.>"
    },
    {
```

```
28        "pattern_name": "<Ej: Estilo Telegr fico>",
29        "observation": "<Ej: El M dico B omite
              art culos y verbos (ej. 'Coraz n normal'),
              M dico A usa oraciones completas.>"
30      }
31    ]
32  }
```

Cabe destacar que este JSON se genero de manera universal pero se puede modificar en función de los atributos que se consideren mas importantes. De esta manera pedirle a la LLM la extracción de semántica/estilo en la dirección mas deseada. Luego de extraído los jsons comparativos, en el notebook sample 10 aleatorios y construyo un prompt simple para pedirle a Gemini que haga un resumen de las diferencias. Esto es lo obtenido por Gemini:

GEMINI JSONS DIFFERENCES ANALYSIS - START

Basado en el análisis de los JSONs proporcionados, se observa una clara dicotomía en los estilos de redacción médica, que puede resumirse en un contraste entre un enfoque **Académico-Protocolario (predominante en el Médico A)** y un enfoque **Pragmático-Funcional (predominante en el Médico B)**.

Aquí presento un resumen detallado de los hallazgos:

Resumen General de Estilos

El análisis revela que la diferencia fundamental no radica solo en la longitud del texto, sino en la **intención comunicativa**. Mientras que uno busca la precisión descriptiva y la protección legal mediante la cautela (hedging), el otro prioriza la velocidad de lectura, la categorización inmediata de la patología y la asertividad.

* **El Médico A (El Académico/Cauteloso):** Tiende a utilizar un lenguaje más denso, con una sintaxis compleja basada en la voz pasiva ("se observa", "se aprecia") y construcciones subordinadas. Su enfoque es defensivo y educativo; justifica las limitaciones técnicas ("no valorable por...") y utiliza modificadores de incertidumbre ("aparente", "sugiere"). Estructuralmente, prefiere la narrativa fluida y titula sus conclusiones como "COMENTARIO", lo que implica una interpretación subjetiva. * **El Médico B (El Pragmático/Administrativo):** Adopta un estilo más directo, a menudo telegráfico o en formato de lista. Prioriza la jerarquización de la información, utilizando mayúsculas sostenidas para las "CONCLUSIONES" y separando visualmente los hallazgos. Su lenguaje es más absoluto ("Normal", "Libre", "No hay") y menos dubitativo. En ocasiones, incluye descargos de responsabilidad legal explícitos en lugar de justificaciones clínicas.

Detalles Específicos y Patrones Lingüísticos

1. Sintaxis y Verbosidad * **Voz Pasiva vs. Activa/Elipsis:** El Médico A depende excesivamente de la voz pasiva refleja y el uso de gerundios para

conectar oraciones ("presentando", "mostrando"), creando párrafos largos. El Médico B, en cambio, oscila entre un estilo activo ("Llama la atención") y un estilo telegráfico con elipsis verbales (omite el verbo "es/está" para decir "Silueta cardíaca normal"). * **Muletillas de Normalidad:** El Médico A utiliza repetidamente la palabra **"Conservado"** o **"Respetado"** para validar estructuras. El Médico B prefiere términos de estado absoluto como **"Íntegros"**, **"Libres"** o simplemente **"Normales"**.

2. Manejo de Hallazgos Negativos (La "No Patología") * **Negación Compuesta vs. Simple:** El Médico A tiende a usar frases de "mantenimiento de estado" o negaciones suaves: *"Sin evidencia de alteraciones"*, *"No se constatan cambios"*. El Médico B es más categórico y económico: *"No hay"*, *"Libres"*, *"Sin hallazgos"*. * **Exhaustividad:** En algunos casos (como en el JSON 4), el Médico B se vuelve paradójicamente más verboso para documentar legalmente la ausencia de múltiples patologías específicas, mientras que el Médico A agrupa todo en una "normalidad general".

3. Estructura y Nomenclatura * **Micro vs. Macro Anatomía:** El Médico A suele desglosar la anatomía en componentes técnicos finos (interfases, líneas paratraqueales, densidad), mientras que el Médico B agrupa por sistemas funcionales grandes (partes blandas, óseo, pleuropulmonar). * **Cierre del Informe:** Existe una diferencia marcada en cómo etiquetan el resumen final. El Médico A prefiere **"COMENTARIO"**, sugiriendo una nota clínica reflexiva. El Médico B prefiere **"CONCLUSIONES"** (a menudo en mayúsculas), sugiriendo un dictamen definitivo, o incluye disclaimers legales sobre la responsabilidad del médico tratante.

4. Tono y Actitud * **Incertidumbre (Hedging):** El Médico A se protege clínicamente usando palabras como *"aparentes"* o *"sugieren correlación"*. * **Validación Técnica:** El Médico B suele iniciar validando explícitamente la calidad de la técnica radiológica ("buena técnica", "centrada"), un paso que el Médico A a menudo omite o integra implícitamente en la descripción.

GEMINI JSONS DIFFERENCES ANALYSIS - END

Leyendo las conclusiones de gemini uno puede entender las diferencias que pueden existir en los strings de diagnósticos. A su vez, se puede observar que Gemini es capaz de capturar patrones en distintas diferencias como lo es en sintaxis, verbosidad, manejo de hallazgos (negativos y positivos), formalidad, uso de lenguaje académico, tono actitud y nomenclatura. Es importante considerar que este analysis es dependiente del JSON de entrada, por lo que estas diferencias pueden cambiar considerablemente si se utilizara otra filosofía para la extracción de semántica, preferencia y/o estilo.

Extensión de pipeline para generar reportes en función del estilo del medico

Se genero el script escenario2-personalizacion-por-medico/generate_reports_and_metrics.py el cual tiene bastante similaridades con el script principal del ejercicio anterior. La mayor diferencia es la extracción "on demand" del estilo. Para disponibilización es recomendable extraer estos estilos a priori y no on demand, de esta manera solo buscarlos a medida se necesiten (no calcularlos cada pegada)

Una potencial mejora seria incluir few shot samples del diagnostico. Incluso se podría tener un dynamic few shot con embeddings de las imágenes, de esta manera buscar hacer un retrieval de diagnósticos potencialmente similares al caso actual que permitan hacer un few shot mas relevante para el caso en cuestión.

Análisis de reportes utilizando estilos similares

Se utilizo las métricas del ejercicio 1 para evaluar los reportes generados. Cabe destacar que los resultados pueden ser un poco misleading debido a que el estilo fue extraído del reporte de la misma imagen donde se genera el reporte (un poco de data leak). Idealmente, se tuvieran varios diagnósticos por medico lo que permitiría evitar este leak de información.

A continuación la tabla de categorización de las comparaciones para la métrica 1 y 2:

	json_normalizer	text_to_text_comparison
CRITICAL_ERROR	10	9
MINOR_ERROR	4	3
CORRECT	10	12

Se puede observar que involucrar el estilo / preferencia ayudo a generar reportes mas similares al ground truth. Esto me hace pensar que el system instruction "universal" utilizado para la generación de reporte puede estar escaso de información de manera no adrede. Esta conclusión se apoya también en la conclusión del escenario uno en la cual la mayoría de los errores era "missing fields" y no alucinaciones. La extracción de este estilo puede apoyar a que la LLM se enfoque en la misma info que usualmente el medico observa en vez de un diagnostico universal. Uno pensaría que los diagnósticos deberían ser universales, sin embargo, me parece intuitivo que los médicos estén biased en función del conocimiento que puedan tener o las circunstancias en las cuales se encuentran.

Diseño o planificación de cómo se podría llevar esta solución a producción, y criterios a considerar para un buen funcionamiento

La siguiente figura muestra un diagrama de como disponibilizaria este servicio. A continuación listo detalles de la implementación:

- El input al servicio backend seria la imagen del scan y el id del medico a simular. La imagen al ser XRAy puede ser de alta resolución por lo que podría ser problemático enviarlo al backend via el HTTP body. Por lo que seria preferible enviar directamente los bytes usando una técnica multi-part. El ID del medico si puede ir dentro del body.
- Luego, el servicio busca el JSON de style & preference para armar el prompt con el cual se consultará a la LLM.
- Finalmente, la LLM responde con el reporte (similar al escenario 1) y se almacena en la base de datos necesaria o se muestra en el front end necesario.
- En paralelo, cada vez que venga un nuevo reporte ground truth es importante regenerar el estilo con el set de nuevo de diagnósticos
- (optional) RAG para involucrar casos y diagnósticos similares al prompt. Esto podría permitir anexar al prompt no solo el estilo sino también casos con similaridad. Es importante que la generación de este embedding sea driven por un modelo que conozco sobre x-rays, otherwise, los embeddings no van a aportar al retrieval y el dynamic few shot va a ser mas parecido a un "random few shot". Esto requeriría tener una base de datos vectorial de la cual se pueda hacer las comparaciones. A su vez, también necesitaríamos disponibilizar (potencialmente en una maquina con GPU) el modelo visual (e.g CNN, ViT) que permita generar estos embeddings on demand (idealmente en una API separada.
- El diagrama no incluye observabilidad o trazabilidad de la utilización de los servicio. Tampoco involucra monitorización de las instancias que hostean el servicio.
- Este servicio se puede disponibilizar utilizando tecnologías como fastAPI o Flask (en caso de usar python con lenguaje de backend)

Propuesta teórica de entrenamiento para que el sistema genere el reporte generado clinicamente correcto (similar al Escenario 1), pero condicionándolo a que siga el estilo de reporte de diferentes médicos. Considera un volumen de datos mucho mayor pero del mismo tipo (múltiples médicos, una imagen por estudio)

Mi propuesta seria bastante similar al escenario 1, con el cambio evidente de agregar los system instructions la posibilidad de compartir en el prompt los

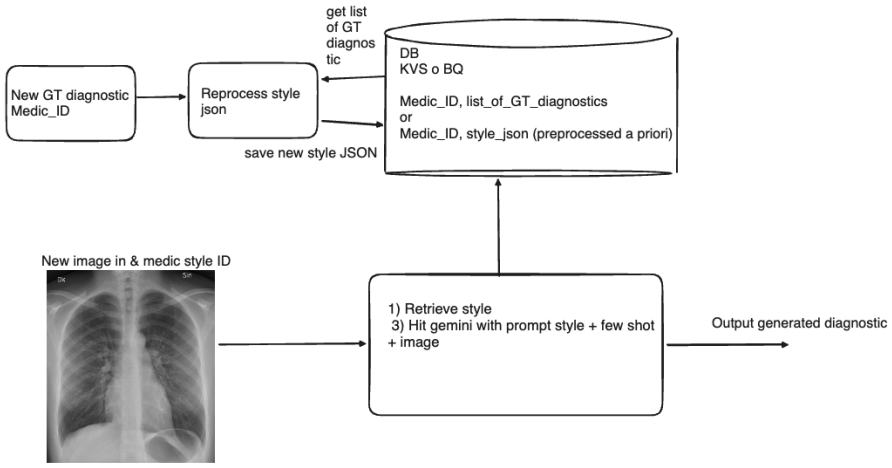

Figure 1: Diagrama de sistema para la generación de reportes online

JSON styles & preferences. También, podríamos involucrar el esquema de few shot

Todos las técnicas de finetuning, system instruction optimizer & judge systems son aplicables en este entorno del escenario 2.

Escenario 3

En el caso de gemini, tiene la capacidad de recibir múltiples imágenes a la vez. Esto simplifica el trabajo y hace que el pipeline sea muy parecido al escecnario 2. La mayor diferencia es que ahora se leen todas las imágenes que tiene un report_id. Otra diferencia es que el system instruction ahora le hace saber a la LLM que vienen varias imágenes y no una sola. El for loop pasa de ser $.iterrows()$ a un $.groupby("report\_id")$. El código ejecutado para generar reportes dado multiples imágenes puede ser encontrado en escenario3-multiples-imagenes-por-estudio/generate_reports_and_metrics.py. Una mejora seria darle mas info en el prompt sobre cada imagen, de esta manera contextualizar a la LLM de que angulo o tipo de estudio se le esta compartiendo. Si son todas del mismo tipo no sería relevante esta información.

Extensión del pipeline a producción. Puedes apoyarte de tu diseño productivo del Escenario 2 y hacer supuestos sobre consideraciones productivas

El core del pipeline permanecería igual. El cambio principal es que ahora tiene que soportar múltiples imágenes. En caso de que enviar todas estas imágenes por

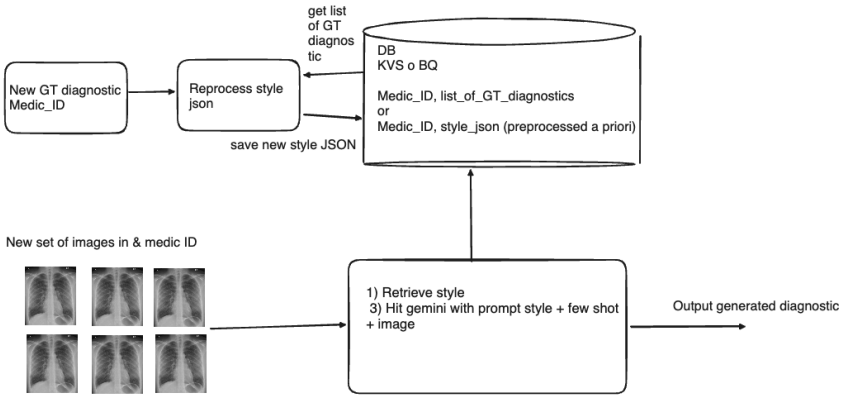

Figure 2: Diagrama enviando imágenes via POST multi-part

el tunnel HTTP sea un problema, alternativamente se puede subir a un bucket las imagenes (en paralelo) y del lado del backend descargar las imágenes. De esta forma el backend solo recibe los ids de imágenes a descargar del bucket. De no ser problema el ancho de banda, el pipeline no cambiaría en nada. Se ilustran las imágenes de ambos escenarios:

Propuesta teórica de entrenamiento con inputs de longitud variable, considerando un volumen de datos mucho mayor pero del mismo tipo (múltiples imágenes por estudio)

De la misma manera que en el escenario 2, las propuestas de finetuning y mejoras del escenario 1 también son aplicables en este escenario. Sin embargo, menciono de nuevo los cambios de propuesta teórica para que estas técnicas de mejora utilicen múltiples imágenes:

1. El system instruction actual fue realizado a manera de one shot, una sola interacción con gemini la cual llevó a los resultados mostrados. Dado que ahora el input consta de múltiples imágenes (e.g., PA y Lateral), la complejidad del prompt aumenta exponencialmente. Por lo que, optimizar este system instruction es crucial para garantizar que el modelo correlacione correctamente los hallazgos entre las distintas vistas. Para ello propongo utilizar un agente que tenga las tools de comparación, de esta forma realizar un pipeline agéntico que maximize la métrica deseada evaluando la coherencia entre proyecciones. La salida de este pipeline es un system instruction optimizado para ingesta multi-imagen. Algorithm 1 muestra pseudocódigo del proceso agéntico iterativo.

2. Aplicar técnicas de finetuning como LoRa sobre patrones de errores específicos de la interpretación multimodal. En mi experiencia hacer con-

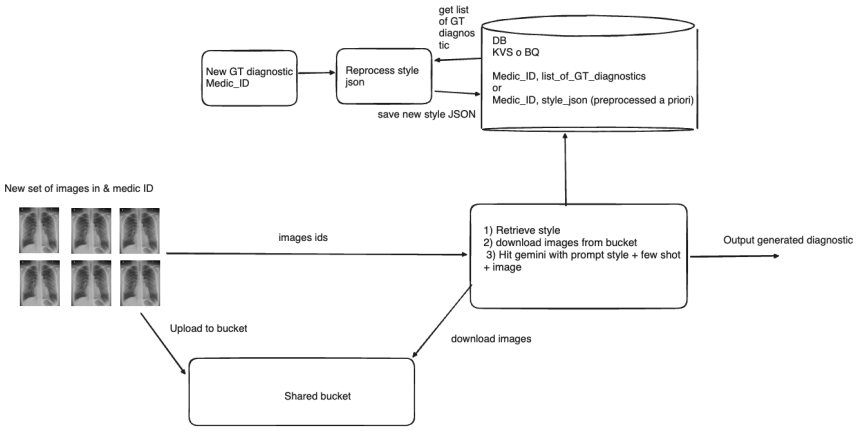

Figure 3: Diagrama subiendo imágenes a bucket y descargando en backend

verger estos pipelines es complejo sin que caigan sobre catastrophic forgetting, más aún al introducir la variable de múltiples inputs visuales. Sin embargo, si los patrones de discrepancia entre imágenes están bien identificados, tiene sentido aplicar finetuning de la VLM sobre estos errores para mejorar la integración espacial. Siempre considerando un set de control donde ya el pipeline está funcionando correctamente para evitar que "el remedio" sea peor que "la enfermedad".

3. Juez multi agéntico. Utilizar varios mixture of experts con system instructions (potencialmente optimizado como en el punto uno) para las distintas etapas de diagnósticos y para el análisis cruzado de imágenes. De esta manera, no tener una sola pegada a la LLM que tenga que observar todos los aspectos relevantes y todas las imágenes simultáneamente. Sino tener pegadas en paralelo donde cada una se encargue de algo específico o de validar un hallazgo en una proyección contra la otra. Por ejemplo, una pegada se encarga de bone density, otra de triangulación de nódulos usando ambas vistas, otra de apertura de espacios intercostales, etc. Esta técnica ha demostrado alta efectividad, ya que cada "pegada" a la LLM es bastante objetiva, evitando confusiones internas. Luego de todas estas pegadas tendría que haber una llamada final para hacer el merge de todos los sub-diagnósticos, sintetizando la información volumétrica inferida.

4. Conjunto de Jueces debatiendo. Tener un set de agentes "doctores" especializados en distintas áreas (system instructions orientados a diferentes áreas o vistas específicas) que debatan entre ellas hasta llegar a una conclusión final. Esto es particularmente útil cuando una proyección sugiere una patología (falso positivo por superposición) y la otra la descarta; el debate permite resolver esta ambigüedad. Potencialmente, se necesite un juez final que genere el veredicto integrando la evidencia de todas las

	json_normalizer	text_to_text_comparison
CRITICAL_ERROR	2	4
MINOR_ERROR	4	1
CORRECT	4	5

Table 1: Resultados múltiple images prompts

	json_normalizer	text_to_text_comparison
CRITICAL_ERROR	6	7
MINOR_ERROR	1	1
CORRECT	3	2

Table 2: Resultados one random image prompt

imágenes. La interacción y los system instructions pueden ser optimizados de manera similar al punto 1. Si se utilizan distintos tipos de estudios, es posible que cada imagen (de estudios distintos) vaya a una LLM distinta (o un system instruction distinto) que permita utilizar LLMs especializadas para cada imagen en especifico.

Evaluación cuantitativa y/o cualitativa del uso de la información de múltiples imágenes en los reportes generado, i.e. que seas capaz de mostrar que con el approach realmente el modelo está usando la información de múltiples imágenes.

Para demostrar que se esta usando la información de múltiples imágenes, se ejecuto el pipeline sobre la misma data utilizando todas las imágenes y seleccionando una imagen random. Seria esperable que si esta efectivamente utilizando todas las imágenes, los resultados sean superiores en este caso.

Los resultados de ambas ejecuciones fueron:

Observando las tablas se puede apreciar que usando una única imagen compete mas errores críticos que cuando se le comparten varias imágenes. De manera análoga, es mas acertado cuando tiene info de varias imágenes.

Debido al tamaño del dataset puede que los resultados estén un poco noisy.

Conclusiones sobre limitaciones y trade-offs de tus soluciones

La principal desventaja es la utilización de modelos no propios. Al utilizar Gemini o ChatGPT se esta anclado que estén up, en caso de no estarlo el pipeline no tiene posibilidades de responder. Aparte de esta desventaja principal se señalan otras:

- Métricas fueron utilizadas sobre un dataset muy pequeño. Lo cual puede que no sea representativo para que la métrica estabilice.

- La extracción de estilo entre otras cosas fueron pensadas de forma universal. Esto no necesariamente ataca las necesidades de negocio o medicas del momento. Este producto tiene que ser mas driven by las necesidades reales y no un generador de reportes "universal"
- No se utilizo dataset de test, lo cual puede generar afectaciones en las metricas en caso de usarlo en un pipeline real.
- Se podría trabajar modelos VLMs full in-house si se tiene suficiente data para entrenar estos modelos. Sin embargo, la cantidad de data necesaria es bastante grande para lograr llevarlo a generalización. De existir alguna VLM entrenada masivamente en XRays, uno podria simplemente aplicar LORA en esta re pre-entrenada
- Se pensó en usar SAM o Dyno para generar mascaras de segmentación y de esta forma asisitir mejor a la VLM con segmentaciones finas de cada una de las partes del XRay. Esto puede ser como una imagen extra al modelo.
- Los system instructions son universales y no están focalizados a las distintas partes del cuerpo o distintos tipos de observaciones
- Estos estudios no hicieron analisis profundo sobre el patrón de los errores. Por lo que, seguramente con un mejor error analysis tweaks pueden ser ejecutados para boost en todos los performances.
- La velocidad de respuesta del pipeline esta atada a la velocidad de Gemini. El thinking budget siempre fue -1 por lo que los tiempos de respuesta son bastante altos.
- No se incluyo código dela implementación, sin embargo utilizando los métodos y clases creadas es bastante straight forward crear un backend e.g FastAPI para disponibilizar este servicio.
- El estudio no incluye un EDA extensivo sobre las imágenes. Se hubiese podido visualizar la diferencia entre embeddings de texto generados y embeddings de las imágenes. Esto para entender mejor las diferencias entre cada reporte (tanto textual como visualmente)

Elaboración de propuesta teórica sobre el escalamiento del servicio de generación de reportes

De referirse a escalamiento en instancias, uno podría simplemente escalar el numero de recursos en función de los RPSs o CPU usage. Esto debido a que los requests van a ser bound a gemini. Podría establecerse un esquema async con colas con retries, para evitar que se pierdan requests debido a overhead en la espera de respuesta de gemini.

De referirse a escalamiento en volumen, utilizando correctamente los servicios de la nube para almacenar los JSON styles, las imágenes, los reportes generados, y los ground truths, este esquema de microservicios debería aguantar todos los requests que fueran necesarios.