UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS

TRABAJO FIN DE MÁSTER

APLICACIÓN DE LLM PARA LA INTERPRETACIÓN DE SEÑALES BIOMÉDICAS

MARIO ALFONSO CLAVIJO MOJICA

Dirigido por

JESÚS GIL RUIZ

CURSO 2024 - 2025

TÍTULO: APLICACIÓN DE LLM PARA LA INTERPRETACIÓN DE SEÑALES BIOMÉDICAS

AUTOR: MARIO ALFONSO CLAVIJO MOJICA

TITULACIÓN: MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS

DIRECTOR DEL PROYECTO: JESÚS GIL RUIZ

FECHA: Octubre de 2025

RESUMEN

Este proyecto aborda la generación automática de informes de electrocardiograma (ECG) combinando modelos de lenguaje de gran tamaño (LLM) con datos públicos. Se parte del problema práctico de transformar hallazgos de ECG en descripciones clínicas coherentes y estandarizadas, reduciendo el esfuerzo manual y la variabilidad entre redactores. Para ello se diseña una metodología reproducible sobre el dataset PTB-XL: preparación de datos con una única columna `text` (*input* clínico textualizado $\rightarrow$ *report*), ajuste fino de cuatro LLM biomédicos (BioGPT, BioMedLM, PMC-LLaMA y BioMistral-7B) y evaluación homogénea con BLEU, ROUGE y METEOR en un conjunto de prueba fijo.

Los principales resultados muestran que la variante **PMC-LLaMA (7B) con QLoRA/LoRA y plantilla instruccional** "`### Input / ### Report`" ofrece el mejor rendimiento global, produciendo informes más fieles y concisos que las alternativas probadas. BioMistral-7B y BioGPT logran un desempeño intermedio; BioMedLM, en mi configuración, resulta más verboso y penalizado por métricas léxicas. Por tanto se concluye que, para recursos limitados de GPU, la combinación *plantilla + LoRA/QLoRA* es eficaz y viable.

Como líneas futuras, propongo integrar un etiquetador multietiqua desde señal cruda (tipo "ECG-BERT" o CNN) para automatizar los hallazgos y reforzar la evaluación clínica con especialistas, cerrando el ciclo señal $\rightarrow$ hallazgos $\rightarrow$ informe.

Palabras clave: ECG; LLM; LoRA/QLoRA; PMC-LLaMA; PTB-XL;Fine-tuning.

ABSTRACT

This project addresses the automatic generation of electrocardiogram (ECG) reports by combining large language models (LLMs) with public biomedical data. It starts from the practical challenge of transforming raw ECG findings into coherent and standardized clinical descriptions, aiming to reduce manual effort and inter-observer variability. To this end, a reproducible methodology is designed using the *PTB-XL* dataset: data preparation into a single `text` column (*textualized clinical input* $\rightarrow$ *report*), fine-tuning of four biomedical LLMs (BioGPT, BioMedLM, PMC-LLaMA, and BioMistral-7B), and consistent evaluation with BLEU, ROUGE, and METEOR metrics on a fixed test set.

The main results show that the **PMC-LLaMA (7B) model with QLoRA/LoRA and an instructional template** "`### Input / ### Report`" achieves the best overall performance, producing more faithful and concise reports compared to other tested alternatives. BioMistral-7B and BioGPT achieve intermediate performance, while BioMedLM, under the present configuration, tends to generate overly verbose outputs and is penalized by lexical metrics. Therefore, it can be concluded that, under limited GPU resources, the combination of *instructional template + LoRA/QLoRA* is both effective and computationally feasible.

As future work, I propose integrating a multi-label classifier from raw signals (e.g., "ECG-BERT" or CNN-based) to automatically extract findings, and to strengthen clinical evaluation with expert feedback, thus closing the loop from *signal* $\rightarrow$ *findings* $\rightarrow$ *report*.

Keywords: ECG; LLM; LoRA/QLoRA; PMC-LLaMA; PTB-XL; Fine-tuning.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1. RESUMEN DEL PROYECTO

1.1 Contexto y justificación

El desarrollo de modelos de lenguaje avanzados (LLM, Large Language Model) ha revolucionado diversas áreas gracias a su notable capacidad para realizar tareas complejas, tales como responder preguntas, generar resúmenes precisos, o traducir textos entre múltiples idiomas. Sin embargo, su potencial en la interpretación de señales biomédicas, particularmente electrocardiogramas (ECG), es todavía un campo emergente con enormes oportunidades por explorar. Tradicionalmente, el análisis de señales ECG depende considerablemente de la experiencia de especialistas clínicos, lo que conlleva limitaciones importantes como la disponibilidad restringida de expertos, tiempos prolongados en la interpretación y dificultades en la identificación temprana de patrones complejos asociados a patologías cardíacas. En este contexto, la integración de modelos de lenguaje con redes neuronales especializadas mediante técnicas de ajuste fino (fine-tuning) presenta un enfoque innovador que podría incrementar notablemente la precisión diagnóstica, reducir tiempos de análisis y automatizar eficazmente la generación de informes clínicos. Por tanto, investigar y desarrollar metodologías basadas en LLM aplicadas a señales ECG constituye un paso clave hacia una atención médica más ágil, precisa y eficiente.

1.2 Planteamiento del problema

En el ámbito clínico actual, la interpretación de electrocardiogramas (ECG) continúa siendo una tarea crítica y altamente especializada. Si bien existen algoritmos automatizados que ofrecen análisis básicos de las señales cardíacas, estos sistemas presentan limitaciones importantes: baja capacidad para generar descripciones clínicas completas, escasa adaptabilidad a distintos contextos clínicos y una limitada comprensión del lenguaje médico. Esta situación dificulta el uso autónomo de estos sistemas en entornos reales y genera dependencia continua del criterio humano para interpretar hallazgos relevantes.

Paralelamente, el auge de los modelos de lenguaje de gran escala (Large Language Models, LLM) ha abierto nuevas posibilidades en la generación automática de texto, incluyendo aplicaciones biomédicas. Modelos como BioGPT, PMC-LLaMA o BioMedLM, preentrenados con grandes volúmenes de literatura científica o notas clínicas, ofrecen la capacidad de generar descripciones complejas y contextualizadas. Sin embargo, su aplicación directa a datos numéricos como señales ECG aún no está suficientemente explorada, especialmente cuando se requiere transformar esa señal en lenguaje clínico interpretativo.

La pregunta motriz que guía este proyecto es la siguiente:

> *¿Pueden los modelos avanzados de lenguaje (LLM), combinados con redes neuronales especializadas en el análisis de señales ECG mediante técnicas de fine-tuning, transformar la interpretación automática de electrocardiogramas y optimizar la detección temprana y clasificación precisa de anomalías cardíacas?*

Este trabajo se enmarca dentro del ámbito científico-técnico, con un enfoque innovador orientado a investigar la viabilidad del uso combinado de representación computacional de señales

biomédicas y modelos generativos de lenguaje para simular informes clínicos. A través de la experimentación con datos reales (como PTB-XL) y el ajuste fino de modelos biomédicos preentrenados, se pretende evaluar si es posible obtener descripciones diagnósticas coherentes, comprensibles y clínicamente válidas a partir de señales codificadas.

La propuesta representa un avance en la intersección entre procesamiento de señales, inteligencia artificial biomédica y generación automática de lenguaje natural, con potencial impacto en el desarrollo de herramientas clínicas asistidas por IA.

1.3 Objetivos del proyecto

El objetivo de este trabajo es evaluar el uso de modelos de lenguaje (LLM) ajustados mediante fine-tuning para generar informes clínicos a partir de señales ECG, comparando diferentes enfoques y modelos con el fin de mejorar la interpretación automática en contextos médicos.

1.4 Resultados obtenidos

En este proyecto se ha implementado y evaluado cuatro LLM biomédicos (BioGPT, BioMedLM, PMC - LLaMA y BioMistral-7B) para la generación de informes de ECG a partir de entradas clínicas textualizadas del dataset **PTB-XL**. Utilicé el mismo *test set* ($n = 150$) y la misma configuración de decodificación (greedy, `max_new_tokens`=128), midiendo **BLEU**, **ROUGE-1/2/L** y **METEOR**.

- **Mejor modelo**: **PMC-LLaMA** (7B, QLoRA/LoRA, plantilla instruccional) obtuvo el mejor desempeño global en todas las métricas (BLEU, ROUGE-1/2/L y METEOR), véase Tabla 6.1.
- **Segundo bloque**: **BioMistral-7B** y **BioGPT** lograron resultados intermedios; BioMistral tiende a producir textos más largos, lo que penaliza métricas lexicográficas, mientras que BioGPT recupera más 1-gramas (ROUGE-1) pero rinde algo peor en BLEU/METEOR.
- **Por detrás**: **BioMedLM** quedó claramente por debajo y generó salidas más extensas, lo que afectó negativamente a BLEU/ROUGE y tampoco mejoró METEOR.
- **Metodología reproducible**: consolidé una canalización de *fine-tuning* y evaluación homogénea (columna `text` con *prompt* y referencia), adaptada a recursos de Colab (LoRA/QLoRA).
- **Recomendación**: selecciono **PMC-LLaMA** como candidato preferente para la fase generativa y dejo como línea futura el acoplamiento con un etiquetador *multietiqueta* desde señal cruda para cerrar el flujo señal $\rightarrow$ hallazgos $\rightarrow$ informe.

1.5 Estructura de la memoria

A continuación describo brevemente la organización del documento:

- **Introducción**: contextualización del problema, motivación y objetivos (generales y específicos).
- **Estado del arte**: revisión de la literatura sobre LLM biomédicos y modelos aplicados a ECG, así como los conjuntos de datos públicos más relevantes.
- **Datos y preparación**: descripción del conjunto de datos PTB-XL, generación del fichero de trabajo con la columna `text`, criterios de limpieza y particionado.

- **Metodología y herramientas**: marco metodológico, decisiones de diseño (plantillas, separadores) y *stack* tecnológico (Transformers, PEFT, LoRA/QLoRA, Colab).
- **Desarrollo del proyecto**: planificación (cronograma), actividades realizadas y justificación de los hitos.
- **Modelos y configuración**:
  - *BioGPT*: *full fine-tuning* como modelo de lenguaje causal.
  - *BioMedLM*: QLoRA con máscara de pérdida aplicada tras el separador.
  - *PMC-LLaMA*: QLoRA/LoRA con plantilla instruccional `### Input / ### Report`.
  - *BioMistral-7B*: QLoRA/LoRA con plantilla instruccional.
- **Evaluación**: definición del conjunto de prueba, protocolo común de inferencia y métricas (BLEU, ROUGE-1/2/L, METEOR).
- **Resultados del proyecto**: presentación de los resultados cuantitativos (véase Tabla 6.1) y análisis comparativo; selección del modelo óptimo.
- **Viabilidad**: análisis coste–beneficio, sostenibilidad técnica y riesgos con sus respectivas mitigaciones.
- **Recursos y presupuesto**: relación de recursos empleados y desglose económico.
- **Conclusiones personales**: aprendizajes, aportaciones y relevancia del trabajo realizado.
- **Líneas futuras**: desarrollo de un etiquetador *multietiqueta* a partir de señal cruda, control de longitud y estilo de salida, evaluación clínica con expertos y exploración de técnicas RAG.
- **Bibliografía**: referencias bibliográficas utilizadas.

Capítulo 2. ANTECEDENTES / ESTADO DEL ARTE

2.1 Estado del arte

En los últimos años, la aplicación de modelos de lenguaje de gran escala (LLM) en el ámbito biomédico ha experimentado un crecimiento significativo, especialmente en la interpretación automatizada de señales clínicas como el electrocardiograma (ECG). Esta tendencia refleja la búsqueda de soluciones innovadoras para mejorar la precisión diagnóstica y optimizar los procesos clínicos mediante inteligencia artificial y procesamiento del lenguaje natural. Un avance relevante es el estudio de Pang et al. (2024), en el que se exploran los beneficios y limitaciones de los LLM para el análisis avanzado de electrocardiogramas. Los autores demuestran que los modelos de lenguaje pueden identificar patrones complejos en las señales ECG, facilitando así el análisis automático y apoyando a los profesionales de la salud en la evaluación diagnóstica. De manera complementaria, Safranek et al. (2024) presentan una investigación sobre la automatización de la determinación del HEART score utilizando ChatGPT, proponiendo un marco de desarrollo iterativo de prompts para optimizar la precisión de las respuestas generadas por el modelo. Este estudio evidencia la capacidad de los LLM para sintetizar información médica compleja y sugiere nuevas formas de integración de inteligencia artificial en procesos clínicos rutinarios. Ambos trabajos constituyen una base sólida para el presente proyecto, validando la viabilidad y actualidad de aplicar modelos de lenguaje en el análisis de señales biomédicas, y destacando desafíos y oportunidades que este TFM busca abordar en el contexto de la interpretación automatizada de señales ECG.

2.2 Contexto y justificación

El creciente volumen de datos biomédicos generados en contextos clínicos, como los registros de electrocardiogramas (ECG), ha impulsado el interés por soluciones basadas en inteligencia artificial que permitan una interpretación más eficiente, precisa y accesible. A pesar de los avances en algoritmos de análisis de señales, muchos sistemas actuales siguen limitados a detecciones numéricas o clasificación binaria, sin lograr emular la capacidad de síntesis diagnóstica que caracteriza a los profesionales sanitarios.

En paralelo, los modelos de lenguaje de gran escala (LLM) han demostrado un potencial notable para generar textos coherentes y especializados a partir de datos estructurados. En el ámbito médico, modelos como BioGPT, PMC-LLaMA o BioMedLM, preentrenados con literatura científica y notas clínicas, han abierto nuevas posibilidades para automatizar la redacción de informes médicos. Sin embargo, su aplicación específica al dominio de señales ECG —especialmente mediante *fine-tuning* adaptado— aún representa un campo incipiente y con amplias oportunidades de desarrollo.

Este trabajo se justifica por la necesidad de explorar enfoques innovadores que integren representación computacional de señales biomédicas y generación automática de lenguaje clínico. En concreto, se busca investigar si es posible generar descripciones médicas de calidad a partir de señales ECG representadas de forma simbólica, lo que permitiría:

- Reducir la carga de trabajo de los profesionales sanitarios.
- Facilitar la interpretación de señales para personal no especializado.
- Ofrecer una herramienta complementaria para la detección temprana de anomalías cardíacas.

Desde una perspectiva investigadora, este proyecto aporta nuevo conocimiento en la intersección entre aprendizaje automático, procesamiento de lenguaje natural y análisis de señales médicas. Supone un paso hacia la creación de sistemas híbridos que combinan capacidades de percepción (análisis de señales) y generación (lenguaje clínico), con posibles aplicaciones en diagnóstico asistido por IA y formación médica.

Además, la experimentación con modelos preentrenados y la exploración de técnicas de *fine-tuning* sobre datos reales posicionan este trabajo como una contribución relevante tanto al desarrollo técnico como al debate ético y metodológico en torno al uso de IA en medicina.

2.3 Planteamiento del problema

La interpretación de señales electrocardiográficas (ECG) es una tarea esencial pero compleja dentro del ámbito clínico. Si bien existen algoritmos tradicionales que permiten detectar anomalías en las señales, su capacidad para generar informes clínicos interpretativos sigue siendo limitada. Estas soluciones suelen centrarse en la clasificación de patologías concretas, careciendo de comprensión contextual y generativa del lenguaje clínico.

El estado del arte evidencia que los modelos de lenguaje de gran escala (LLM), como BioGPT o PMC-LLaMA, tienen un gran potencial para la comprensión y generación de lenguaje médico. Estudios recientes demuestran que estos modelos pueden integrarse en procesos clínicos para asistir en tareas como el análisis de ECG, la interpretación de escalas de riesgo o la redacción de notas médicas. Sin embargo, la mayoría de estos enfoques siguen sin abordar directamente la fusión efectiva entre las señales biomédicas estructuradas y la generación automática de informes diagnósticos completos.

Actualmente, no existe una metodología estandarizada y validada que permita combinar representaciones computacionales de señales ECG con modelos generativos de lenguaje biomédico para simular descripciones clínicas precisas y coherentes. Tampoco hay consenso sobre qué modelo LLM es más adecuado para esta tarea, ni sobre qué técnicas de *fine-tuning* resultan más eficaces según el tipo de entrada (simbólica, estructurada, texto libre) y la calidad de los datos disponibles.

Por tanto, el problema principal que este trabajo pretende abordar es la ausencia de un marco metodológico que permita transformar señales ECG en informes médicos útiles mediante el uso de modelos de lenguaje ajustados a través de técnicas de transferencia de aprendizaje. Esto incluye tanto el diseño del pipeline de procesamiento como la validación del resultado frente a criterios clínicos y expertos humanos.

Este planteamiento tiene una orientación investigadora y de innovación, ya que busca aportar nuevo conocimiento sobre cómo integrar modelos de lenguaje con datos biomédicos estructurados, evaluando su aplicabilidad en contextos clínicos reales y estableciendo recomendaciones sobre su uso futuro.

Capítulo 3. OBJETIVOS

3.1 Objetivos generales

Desarrollar y evaluar una metodología integral que combine modelos de lenguaje avanzado (LLM) con redes neuronales especializadas en el análisis automatizado de señales electrocardiográficas (ECG), con el fin de mejorar la interpretación clínica y automatizar la generación de informes diagnósticos, analizando además sus limitaciones y desempeño comparativo frente a especialistas médicos.

3.2 Objetivos específicos

1. Explorar el uso de LLM en la interpretación de señales electrocardiográficas (ECG).
2. Identificar y recopilar bases de datos públicas de señales electrocardiográficas (ECG) que permitan evaluar modelos LLM.
3. Desarrollar una metodología para integrar modelos de lenguaje con redes neuronales especializadas en el análisis de señales electrocardiográficas (ECG).
4. Aplicar técnicas de *fine-tuning* (ajuste fino o reentrenamiento especializado) sobre cuatro modelos LLM diferentes para la tarea específica de interpretación de ECG.
5. Evaluar la capacidad de los modelos LLM para generar informes clínicos basados en señales biomédicas (ECG) procesadas.
6. Comprobar la interpretación del LLM con informes generados por especialistas médicos (cardiólogos).
7. Proporcionar una recomendación fundamentada, teniendo en cuenta las limitaciones y desafíos, sobre qué modelo LLM y método de *fine-tuning* es el más adecuado para la interpretación de señales ECG en entornos clínicos reales.

3.3 Beneficios del proyecto

Este proyecto contribuye significativamente al avance del procesamiento automático de señales biomédicas mediante la aplicación de modelos de lenguaje (LLM) ajustados con técnicas de *fine-tuning*. Aporta beneficios en varios niveles: desde el desarrollo de una metodología reproducible para la generación automática de informes clínicos a partir de señales ECG, hasta la comparación crítica entre distintos modelos especializados en lenguaje biomédico.

Además, proporciona una evaluación del rendimiento y limitaciones de estas herramientas frente a especialistas humanos, lo cual permite reflexionar sobre su aplicabilidad real en entornos clínicos. El trabajo también facilita futuras investigaciones al ofrecer una estructura clara y recursos reutilizables (datos, modelos, criterios de evaluación), favoreciendo así la innovación en el cruce entre inteligencia artificial, lenguaje natural y salud.

Capítulo 4. MARCO TEÓRICO

En esta sección se expondrán los conceptos básicos necesarios para contextualizar el trabajo propuesto. Por un lado, se explicarán los fundamentos clave de las señales ECG, que son esenciales para nuestro estudio, así como su representación y las diferentes bases de datos que utilizaremos durante el proceso de fine-tuning. Por otro lado, se hará una explicación más específica sobre los LLM, destacando las características de los modelos seleccionados para este estudio, sus ventajas, desventajas, y finalmente, se describirán las métricas que nos permitirán evaluar qué tan buenos son estos modelos.

4.1 Procesamiento de señales biomédicas: fundamentos y desafíos

4.1.1. Fundamentos del ECG

Un electrocardiograma (ECG) o electrocardiografía es una prueba médica que mide las señales eléctricas que controlan el ritmo cardíaco. El ECG muestra cómo viajan estos impulsos eléctricos a través del músculo cardíaco a medida que se contrae y se relaja. Cuando hablamos de un ECG, es fundamental identificar las partes clave del trazado, que se describen a continuación:

1. **Onda P**: representa la actividad eléctrica que atraviesa las cavidades superiores del corazón, denominadas aurículas.
2. **Complejo QRS**: refleja el movimiento de los impulsos eléctricos a través de las cavidades inferiores, conocidas como ventrículos.
3. **Segmento ST**: indica el momento en que el ventrículo está contraído, pero no hay flujo eléctrico activo. Suele visualizarse como una línea recta horizontal entre el complejo QRS y la onda T.
4. **Onda T**: señala el proceso en el que los ventrículos se restablecen eléctricamente y se preparan para la siguiente contracción.

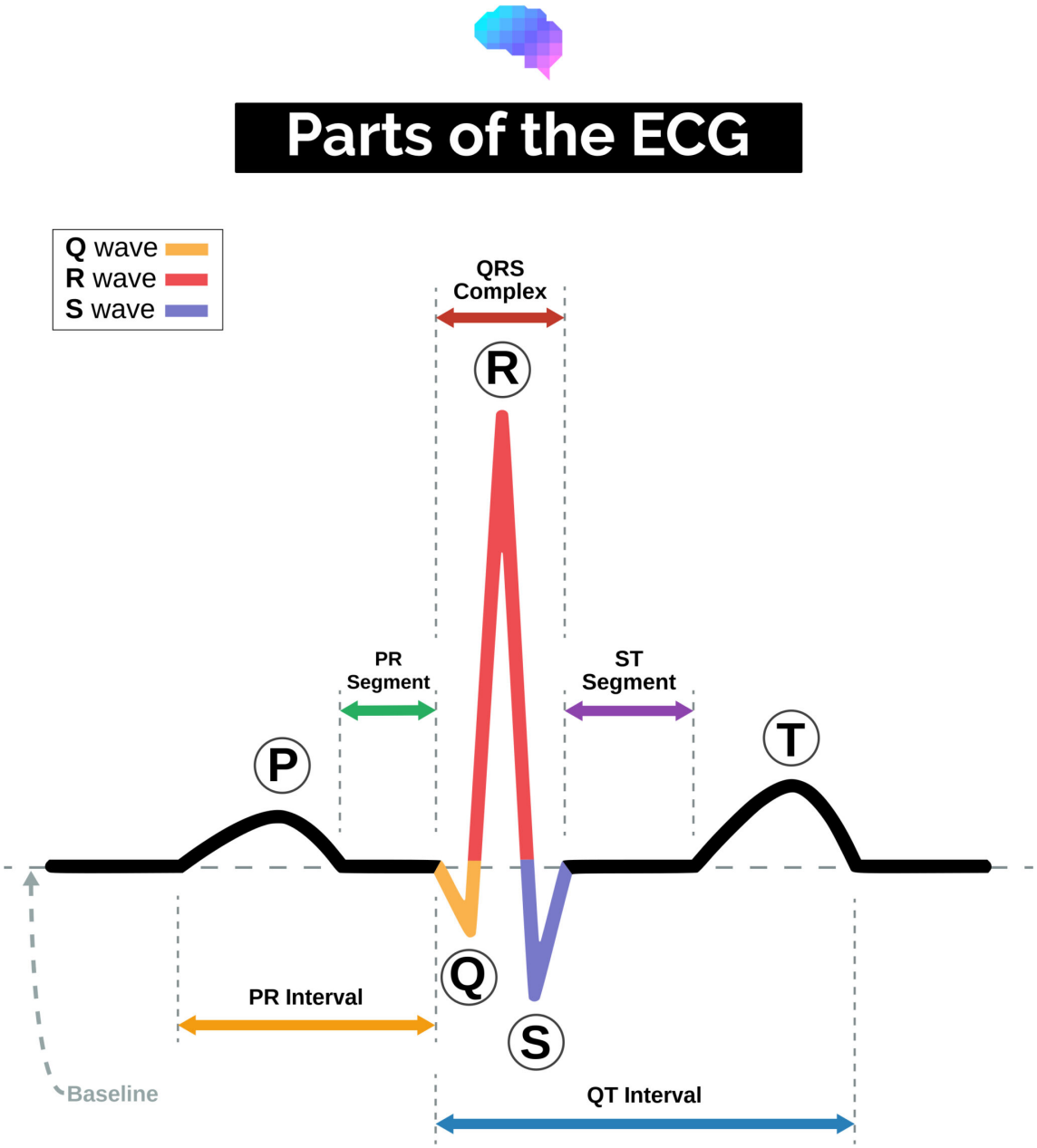

Figura 4.1. ECG parts

Fuente: Geeky Medics, “Understanding an ECG”. Disponible en: https://geekymedics.com/understanding-an-ecg/

La información que proporciona un ECG es útil para analizar el ritmo cardíaco, la frecuencia, la sincronización entre cavidades, así como para detectar posibles anomalías funcionales. Gracias a esta herramienta, es posible diagnosticar enfermedades o alteraciones como:

1. Deterioro del miocardio.
2. Arritmias cardíacas.

3. Desequilibrios en los niveles de calcio, potasio u otras sustancias en la sangre.
4. Cardiopatías congénitas.
5. Agrandamiento del corazón.
6. Secuelas tras un paro cardíaco previo.
7. Paro cardíaco en curso.
8. Reducción del flujo sanguíneo en las arterias.

Como vimos anteriormente, existen numerosas patologías o condiciones que pueden afectar al corazón, por lo que la interpretación adecuada de estas señales es fundamental para una evaluación clínica precisa.

4.1.2. Representación computacional de señales ECG

Una vez comprendidos los fundamentos clínicos del electrocardiograma, es necesario abordar su tratamiento computacional, ya que las señales ECG deben ser adecuadamente transformadas para ser interpretadas por modelos basados en aprendizaje automático.

Digitalización y muestreo

Las señales ECG se capturan en formato analógico y deben ser digitalizadas para su análisis computacional. Este proceso implica la conversión de las señales en una secuencia de valores numéricos discretos mediante técnicas de muestreo. La frecuencia de muestreo más común oscila entre los 250 Hz y los 500 Hz, dependiendo del dispositivo y del tipo de análisis a realizar. Una frecuencia adecuada es fundamental para preservar detalles clínicamente relevantes como la duración de los intervalos o la morfología de los complejos QRS.

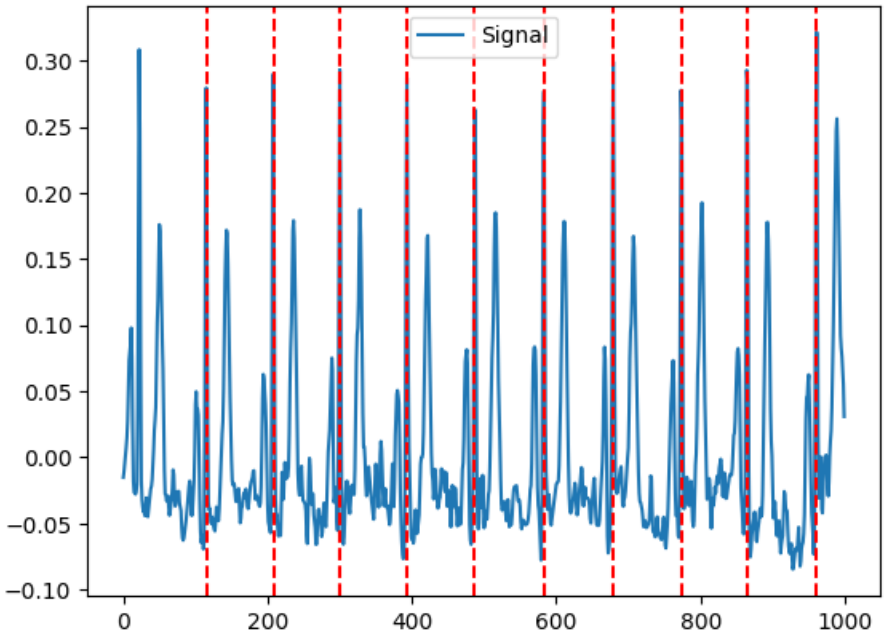

Figura 4.2. R-peaks in ECG signal - records/00001$_lr$.

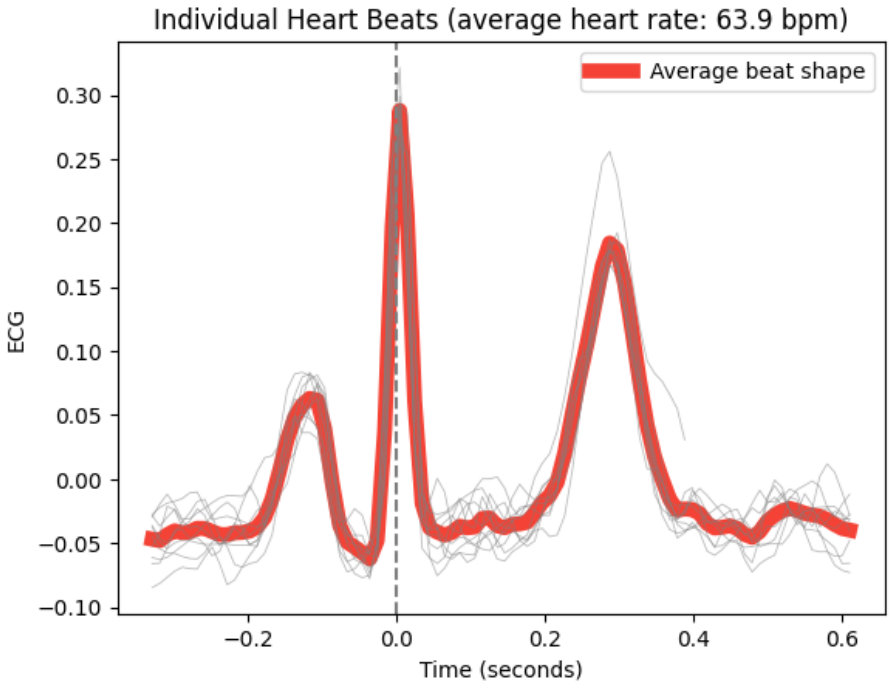

Figura 4.3. The heart beats - records/00001$_lr$.

Ruido y normalización

Las señales crudas suelen contener ruido de diversa naturaleza: interferencias electromagnéticas, movimiento del paciente, artefactos musculares, etc. Para reducir su impacto, se aplican técnicas de filtrado digital (como filtros pasa banda, eliminación de línea base o de la

frecuencia de red). Además, es habitual normalizar las señales para que tengan una media y desviación estándar estables, lo que facilita el entrenamiento de modelos robustos. Segmentación Dependiendo del enfoque del modelo, las señales pueden segmentarse de diferentes formas:

- **Latido a latido**: se extrae cada ciclo cardíaco completo (onda P, QRS, T) de forma individual.
- **Sliding window**: se utiliza una ventana de tiempo fija que se desplaza sobre la señal para capturar contextos más amplios.
- **Eventos clínicos**: en algunos casos, se segmenta según la aparición de eventos significativos (como taquicardias o arritmias).

Representación para modelos de aprendizaje automático

Una vez preprocesadas y segmentadas, las señales pueden representarse de diversas formas:

- **Series temporales**: es la forma más directa, donde se usan las secuencias de voltaje en el tiempo como entrada para redes neuronales (por ejemplo, LSTM o CNN).
- **Espectrogramas o transformadas**: aplicando técnicas como la Transformada Rápida de Fourier (FFT) o de Wavelet, se obtiene una representación visual de la señal en el dominio de la frecuencia. Esto permite utilizar arquitecturas de visión por computador (CNN 2D).
- **Codificación simbólica**: en tareas relacionadas con el procesamiento del lenguaje natural (NLP), se puede transformar la señal en una secuencia simbólica, por ejemplo: "ST↑ Tneg HR: 110 bpm", lo cual permite utilizar modelos de lenguaje como BioGPT o ClinicalBERT. Esta representación computacional es esencial para integrar las señales ECG en un pipeline basado en modelos LLM, como los utilizados en este trabajo, permitiendo la generación automatizada de informes clínicos a partir de entradas biomédicas estructuradas.

4.1.3. Bases de datos públicas relevantes

Para llevar a cabo este trabajo, es fundamental disponer de bases de datos públicas que contengan señales ECG en formato digital y que estén acompañadas de etiquetas clínicas, diagnósticos o anotaciones interpretables. Estas bases de datos no solo permiten entrenar y validar modelos con un volumen considerable de ejemplos, sino que también garantizan la reproducibilidad y comparación con estudios previos. A continuación, se presentan algunas de las bases de datos más utilizadas en la literatura especializada, que serán consideradas en este trabajo:

PTB-XL

- **Fuente**: PhysioNet.
- **Descripción**: Es una de las bases de datos más completas para ECG multicanal. Contiene más de 21.000 registros de 10 segundos a 12 derivaciones estándar.

- **Etiquetas clínicas**: Proporciona múltiples niveles de anotación (diagnóstico cardiológico, morfológico y de ritmo) y etiquetas codificadas en SNOMED CT.
- **Formato**: Las señales están disponibles en formato WFDB, y también se ofrece un CSV estructurado con metadatos y anotaciones.
- **Ventajas**: Alta calidad, buen balance de clases, y permite tareas tanto de clasificación como generación textual a partir de codificaciones clínicas.

MIT-BIH Arrhythmia Database

- **Fuente**: MIT Laboratory for Computational Physiology.
- **Descripción**: Contiene 48 registros de 30 minutos con señales de ECG de 2 derivaciones y anotaciones latido a latido.
- **Etiquetas clínicas**: Clasificación de tipos de arritmias según el estándar AAMI.
- **Formato**: Archivos en formato WFDB con anotaciones de tiempo y tipo de latido.
- **Ventajas**: Muy útil para segmentación de latidos y entrenamiento de modelos centrados en arritmias.

Chapman ECG Dataset

- **Fuente**: Chapman University y Shaoxing People's Hospital.
- **Descripción**: Incluye 10.000 registros de ECG de 12 derivaciones, con una duración de 10 segundos por registro.
- **Etiquetas clínicas**: Clasificación del tipo de ritmo cardíaco, incluyendo fibrilación auricular, taquicardia, bradicardia, entre otros.
- **Formato**: CSV y señal en formato `.mat` (compatible con Python).
- **Ventajas**: Ideal para tareas de clasificación multicategoría y también para generación de etiquetas diagnósticas.

Relevancia para este trabajo

- **PTB-XL** será el dataset principal para este TFM, ya que ofrece registros ECG con etiquetas clínicas estructuradas, ideal para construir pares entrada textual (representación ECG) $\rightarrow$ salida textual (descripción médica).
- **MIT-BIH** y **Chapman** se utilizarán de forma complementaria para análisis de arritmias o validación de modelos.

4.2 Modelos de Lenguaje (LLM) en el ámbito médico

4.2.1. ¿Qué es un modelo de lenguaje?

Un modelo de lenguaje (LLM, por sus siglas en inglés) es un sistema computacional diseñado para comprender, generar y manipular texto de forma similar a como lo hace un ser humano. Estos modelos se entrenan a partir de grandes volúmenes de datos textuales para aprender patrones estadísticos, relaciones semánticas y estructuras gramaticales del lenguaje. En los últimos años, los LLM han evolucionado drásticamente gracias al uso de arquitecturas transformer, que han demostrado un rendimiento excepcional en tareas como generación de texto, traducción automática, respuesta a preguntas o resumen automático.

Arquitectura Transformer: atención y aprendizaje contextual

La arquitectura transformer revolucionó el procesamiento del lenguaje natural al introducir el mecanismo de atención (attention mechanism), que permite al modelo centrarse en partes específicas del texto de entrada según el contexto. A diferencia de modelos anteriores como RNN o LSTM, los transformers no procesan secuencias de forma secuencial, sino que consideran todos los tokens al mismo tiempo (paralelismo), lo que mejora tanto la velocidad de entrenamiento como la capacidad para capturar relaciones de largo alcance.

Embeddings y representación semántica

El primer paso en un transformer consiste en convertir cada palabra o token en un vector numérico denso llamado embedding. Estos vectores capturan propiedades semánticas y sintácticas del lenguaje. Palabras con significados similares tendrán embeddings cercanos en el espacio vectorial. Durante el preentrenamiento, el modelo aprende a ajustar estos vectores y las relaciones entre ellos, lo que le permite generalizar a nuevas frases o contextos.

Preentrenamiento y fine-tuning

Los modelos de lenguaje actuales suelen entrenarse en dos fases:

1. **Preentrenamiento**: el modelo se entrena sobre un corpus masivo (por ejemplo, Wikipedia, PubMed, libros o notas clínicas) con tareas genéricas como predicción de la siguiente palabra (*causal language modeling*) o relleno de huecos (*masked language modeling*).
2. **Fine-tuning**: una vez preentrenado, el modelo se ajusta a una tarea específica con un conjunto de datos más pequeño y especializado. En este trabajo, se aplicará *fine-tuning* para enseñar al modelo a generar informes clínicos a partir de señales ECG representadas simbólicamente.

Modelos generales vs. modelos específicos de dominio

Los modelos generales como *GPT-2* o *BERT* han sido entrenados con datos textuales de propósito general (libros, artículos de noticias, Wikipedia, etc.). Estos modelos tienen una gran capacidad de generación y comprensión del lenguaje común, pero suelen presentar limitaciones cuando se aplican en contextos especializados como el médico, debido a la falta de vocabulario técnico y conocimiento clínico.

Por el contrario, los modelos **específicos de dominio**, como *BioGPT* o *ClinicalBERT*, han sido preentrenados exclusivamente con literatura biomédica o notas clínicas. Esto les permite:

- Comprender terminología médica con mayor precisión.
- Generar descripciones clínicas más realistas.
- Ser más precisos en tareas relacionadas con la salud.

Por ello, en este trabajo se priorizan modelos especializados en medicina, que posteriormente serán ajustados a la tarea concreta de generar informes clínicos basados en señales ECG.

4.2.2. LLMs en salud

El lenguaje clínico plantea una serie de desafíos que lo distinguen del lenguaje general y que deben ser tenidos en cuenta al aplicar modelos de lenguaje en el ámbito médico:

Terminología técnica y especializada

La medicina utiliza un vocabulario altamente técnico, con miles de términos anatómicos, fisiológicos y patológicos, además de acrónimos (como STEMI, LVEF, AF) y codificaciones normalizadas (como ICD-10, SNOMED CT o LOINC). Esta terminología no suele estar presente en modelos de lenguaje generales, lo que dificulta su comprensión y uso correcto en tareas clínicas. Por ello, los modelos deben haber sido expuestos durante el preentrenamiento a textos biomédicos reales, como artículos científicos o notas clínicas, para lograr una correcta internalización del vocabulario.

Ambigüedad semántica

El lenguaje médico es altamente contextual. Un mismo término puede tener significados distintos según el entorno clínico:

- "Bloqueo" puede referirse a un trastorno de conducción cardíaca (bloqueo AV) o a una obstrucción coronaria.
- "Inestable" puede aludir a un paciente, una angina o un marcapasos.

Además, los informes clínicos a menudo presentan abreviaciones, frases fragmentadas y expresiones propias del entorno hospitalario, lo cual aumenta la complejidad del procesamiento automático.

Necesidad de precisión crítica

A diferencia de otras áreas del procesamiento del lenguaje natural (como la generación de texto creativo), la precisión en medicina es esencial. Un error en una palabra puede cambiar completamente el diagnóstico, el tratamiento sugerido o la interpretación del estado del paciente. Por esta razón, la generación de texto clínico requiere modelos altamente fiables, así como mecanismos de validación cualitativa y cuantitativa. No se trata únicamente de generar texto coherente, sino de asegurar que el contenido sea clínicamente válido y seguro.

En este trabajo se emplearán modelos de lenguaje previamente entrenados en dominios biomédicos y clínicos, con el objetivo de adaptarlos a la tarea de generación automática de informes a partir de señales ECG. A continuación, se describen los modelos seleccionados, junto con los corpus de preentrenamiento y las posibilidades que ofrecen para esta investigación.

BioGPT

- **Autor**: Microsoft.
- **Source**: microsoft/BioGPT
- **Arquitectura**: GPT-2 adaptado (autoregresivo, generativo).
- **Corpus de preentrenamiento**: Más de 15 millones de resúmenes biomédicos de *PubMed* (títulos + abstracts).

- **Dominio**: Biomédico, científico, farmacológico.

Relevancia para esta investigación:

- BioGPT es ideal para generar texto técnico biomédico a partir de entradas simbólicas de ECG.
- Al estar entrenado con lenguaje formal y abstractos científicos, su estilo será más académico.

BioMistral-7B

- **Autor**: Equipo BioMistral.
- **Source**: BioMistral/BioMistral-7B
- **Arquitectura**: Basado en *Mistral-7B* (decoder-only, modelo causal).
- **Corpus de preentrenamiento**: Preentrenamiento adicional sobre artículos biomédicos de *PubMed Central* (acceso abierto).
- **Dominio**: Biomédico/clínico; evaluado en un conjunto de 10 tareas de QA médica.

Relevancia para esta investigación:

- Al partir de Mistral-7B y continuar el preentrenamiento en *PubMed Central*, el modelo incorpora terminología y estructuras típicas del discurso clínico; esto favorece la generación de informes a partir de entradas simbólicas de ECG.

BioMedLM

- **Autor**: Stanford CRFM.
- **Source**: stanford-crfm/BioMedLM
- **Arquitectura**: GPT-style (autoregresivo, generativo).
- **Corpus de preentrenamiento**: Mezcla de textos biomédicos: *PubMed abstracts*, *clinical trial reports*, *OMIM*, y otros documentos de salud pública.
- **Dominio**: Medicina general, ensayos clínicos, publicaciones científicas.

Relevancia para esta investigación:

- Muy apropiado para tareas de generación de reportes clínicos sintéticos.

PMC_LLaMA

- **Autor**: Axiong / EPFL.
- **Source**: chaoyi-wu/PMC_LLAMA_7B_10_epoch
- **Arquitectura**: LLaMA adaptado (transformer decoder, generativo).

- **Corpus de preentrenamiento**: PubMed Central (PMC) full text, más de 6 millones de artículos biomédicos completos.
- **Dominio**: Literatura científica biomédica completa (más detallada que los abstracts de PubMed).

Relevancia para esta investigación:

- Modelo con gran capacidad de generación contextual, útil para redactar informes completos y argumentativos.
- Puede ser especialmente interesante en la evaluación cualitativa (ver cómo redacta un informe completo).

4.2.3. Transferencia de aprendizaje y Fine-tuning

En el contexto del aprendizaje automático moderno, la transferencia de aprendizaje (transfer learning) se ha consolidado como una estrategia clave para mejorar el rendimiento en tareas específicas, especialmente cuando no se dispone de grandes volúmenes de datos etiquetados. Consiste en tomar un modelo previamente entrenado sobre una tarea genérica —por ejemplo, predecir la siguiente palabra en textos biomédicos— y reutilizar ese conocimiento para una tarea diferente pero relacionada, como generar informes médicos a partir de entradas simbólicas.

¿Qué es el *fine-tuning* y por qué es útil?

El *fine-tuning* o ajuste fino es una técnica que forma parte de la transferencia de aprendizaje. En lugar de entrenar un modelo desde cero (lo que requiere millones de muestras y recursos computacionales elevados), se parte de un modelo ya preentrenado —por ejemplo, BioGPT o BioMedLM— y se lo ajusta gradualmente con un conjunto de datos específico relacionado con la tarea final.

En este trabajo, se aplicará *fine-tuning* para adaptar modelos de lenguaje biomédico al problema concreto de generar informes clínicos a partir de representaciones computacionales de señales ECG. Esto permite que el modelo aprenda a transformar entradas técnicas (como “ST ↑, T invertida, HR 120 bpm”) en descripciones clínicas coherentes y útiles para el diagnóstico.

El *fine-tuning* es especialmente valioso cuando:

- El dominio es especializado (como el médico).
- La tarea es concreta (generación de texto clínico).
- Los datos disponibles son limitados pero representativos.

Frameworks más utilizados

El desarrollo de modelos LLM y la aplicación de *fine-tuning* se ha visto enormemente facilitado por herramientas de código abierto que abstraen gran parte de la complejidad técnica. Entre las más utilizadas se encuentran:

- **HuggingFace Transformers**: Biblioteca líder para trabajar con modelos de lenguaje preentrenados. Ofrece interfaces fáciles para cargar modelos, realizar *fine-tuning* y compartir resultados. Permite trabajar con modelos como BERT, GPT, BioGPT, entre otros.
- **PyTorch**: Framework de *deep learning* flexible y ampliamente adoptado. Permite implementar *pipelines* de entrenamiento personalizados, incluyendo el control sobre funciones de pérdida, optimizadores y métricas.
- **TensorFlow/Keras**: Alternativa muy popular, especialmente en entornos académicos o con compatibilidad nativa con Google Colab. Aunque se utiliza menos en NLP avanzado en comparación con PyTorch, sigue siendo útil en tareas generales.

En este trabajo se utilizará principalmente **HuggingFace Transformers** combinado con **PyTorch**, debido a su compatibilidad con modelos biomédicos, su integración con *datasets* personalizados y su comunidad activa.

Problemas comunes en el *fine-tuning* clínico

Si bien el *fine-tuning* ofrece grandes ventajas, también presenta desafíos importantes, especialmente en contextos sensibles como el clínico:

- **Overfitting**: Cuando el modelo se ajusta demasiado al conjunto de entrenamiento, pierde capacidad de generalización. Esto puede ser especialmente problemático si el *dataset* es pequeño o poco variado.
- **Sesgos (*bias*)**: Si los datos utilizados para el ajuste fino contienen sesgos (por ejemplo, predominancia de ciertas patologías o grupos demográficos), el modelo los reproducirá en sus predicciones. En medicina, esto puede derivar en errores graves.
- **Calidad de los datos**: La curación del *dataset* es fundamental. El modelo solo aprenderá tan bien como la calidad y coherencia de los ejemplos que se le proporcionen. Es crucial asegurarse de que las entradas estén correctamente formateadas y las salidas sean clínicamente válidas.

Por estas razones, el proceso de *fine-tuning* debe planificarse con cuidado, incluyendo la selección adecuada de ejemplos, el control del entrenamiento (por ejemplo, *early stopping*, regularización) y la validación rigurosa de los resultados.

4.3 Evaluación de modelos generativos en medicina

La evaluación de modelos generativos en el ámbito clínico representa un desafío particular. A diferencia de tareas como clasificación o detección, donde se puede medir el rendimiento con métricas bien definidas (accuracy, F1-score), la generación de texto implica analizar no solo la forma, sino también el contenido, la coherencia semántica y la validez clínica del texto

producido. En este trabajo, se utilizará una combinación de métricas automáticas de evaluación textual y validación cualitativa, a fin de obtener una valoración completa del rendimiento de los modelos.

4.3.1. Métricas de evaluación automática

Las métricas automáticas se basan en la comparación entre los textos generados por el modelo y un conjunto de referencias humanas (informes reales o simulados). Entre las más utilizadas se encuentran:

BLEU (*Bilingual Evaluation Understudy*)

- Originalmente desarrollada para tareas de traducción automática.
- Mide la coincidencia de n-gramas (usualmente hasta $n = 4$) entre el texto generado y el texto de referencia.
- A mayor coincidencia de n-gramas, mayor será la puntuación obtenida.
- **Limitación**: favorece la exactitud literal, penalizando sinónimos o reformulaciones válidas, lo cual es especialmente problemático en contextos clínicos donde existen múltiples formas correctas de expresar una misma observación.

ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*)

- Ampliamente utilizada en tareas de resumen automático.
- Mide la superposición de unidades léxicas (n-gramas, palabras, frases) entre el texto generado y el texto de referencia.
- Incluye múltiples variantes, como ROUGE-N y ROUGE-L (subsecuencia común más larga).
- **Ventaja**: proporciona una evaluación más orientada a la cobertura del contenido relevante.
- **Limitación**: no garantiza la precisión clínica del contenido generado, solo evalúa su similitud superficial con el texto de referencia.

METEOR (*Metric for Evaluation of Translation with Explicit ORdering*)

- Considera coincidencias léxicas, pero también reconoce sinónimos y variantes morfológicas, lo que la hace más flexible que BLEU o ROUGE.
- Utiliza un sistema de alineación entre texto generado y referencia, penaliza repeticiones y tiene en cuenta la fluidez.
- **Ventaja**: mayor sensibilidad a variaciones lingüísticas válidas (por ejemplo, reescrituras clínicas).
- **Limitación**: como otras métricas automáticas, no evalúa directamente la validez médica del contenido generado.

¿Son suficientes estas métricas?

Si bien estas métricas proporcionan una aproximación cuantitativa rápida, no son suficientes por sí solas en contextos médicos. Un texto puede alcanzar una alta puntuación en métricas como ROUGE o BLEU, pero incluir afirmaciones erróneas desde el punto de vista clínico.

Por esta razón, se considera imprescindible complementar las evaluaciones automáticas con validación humana experta, especialmente en tareas sensibles como la generación de informes médicos. Solo mediante la revisión cualitativa por profesionales se puede garantizar la precisión, coherencia y utilidad clínica del contenido generado.

4.3.2. Validación cualitativa

La validación cualitativa consiste en que expertos médicos revisen los textos generados con el objetivo de evaluar su precisión clínica, coherencia, nivel de detalle y potencial riesgo.

Importancia de la revisión por expertos

- Los profesionales sanitarios pueden identificar errores que pasarían desapercibidos para una métrica automática.
- Son capaces de valorar si una descripción es coherente con los signos representados (por ejemplo, detectar si un modelo afirma “sin signos de isquemia” cuando el ECG muestra una elevación del segmento ST).
- También aportan criterios para determinar si el estilo del texto generado se ajusta a los estándares clínicos aceptables, favoreciendo su posible aplicación en entornos reales.

Casos de uso de validación humana en NLP clínico

- Estudios recientes han incorporado revisiones médicas en tareas como la generación de notas de alta hospitalaria, la interpretación automática de ECG o la redacción de informes radiológicos.
- Algunos diseños de evaluación emplean escalas tipo Likert, rúbricas clínicas o validación binaria (aprobado/rechazado) para medir la calidad del contenido generado.
- En este trabajo, si es posible, se incluirá al menos una revisión cualitativa de una muestra de informes generados, con el fin de evaluar su validez desde una perspectiva médica.

4.3.3. Desafíos éticos y clínicos

El uso de modelos generativos en medicina plantea una serie de cuestiones éticas que deben ser consideradas cuidadosamente, tanto durante el desarrollo como en la posible implementación práctica.

Sesgos y errores peligrosos

- Si el modelo se entrena con datos sesgados (por ejemplo, mayor representación de ciertas patologías en determinados grupos poblacionales), es probable que reproduzca dichos sesgos en los informes generados.
- Un error de interpretación o una descripción ambigua puede inducir a diagnósticos erróneos, lo cual resulta especialmente crítico en el contexto clínico.

Explicabilidad y trazabilidad

- Los modelos LLM son generalmente considerados como “cajas negras”, debido a la dificultad para interpretar cómo llegan a una determinada conclusión.
- En entornos clínicos, es imprescindible poder justificar por qué se ha generado un determinado informe, especialmente en caso de auditorías médicas o revisiones legales.
- Idealmente, los modelos deberían incorporar mecanismos de trazabilidad y justificación, por ejemplo: “esta conclusión se basa en la presencia de elevación del segmento ST y onda T invertida”.

Regulación y confianza clínica

- Actualmente, el uso de inteligencia artificial en el ámbito sanitario está regulado por marcos normativos como el Reglamento Europeo de IA o las directrices de la FDA en Estados Unidos.
- La generación automática de informes médicos se considera una aplicación de alto riesgo, por lo que su implementación debe contar con mecanismos de supervisión humana.
- La confianza clínica se construye mediante validación rigurosa, transparencia en el funcionamiento del sistema y la participación activa de profesionales sanitarios en su diseño y evaluación.

Capítulo 5. METODOLOGÍA

El presente trabajo adopta una metodología experimental aplicada, basada en el desarrollo y evaluación de modelos computacionales mediante técnicas de aprendizaje automático supervisado, ajuste fino (*fine-tuning*) de modelos de lenguaje y análisis comparativo de resultados clínicos simulados. El enfoque metodológico integra tres pilares fundamentales:

5.0.1. Procesamiento de señales biomédicas (ECG)

Se utilizarán bases de datos públicas que contienen registros electrocardiográficos multicanal, en formato digital, acompañados de etiquetas clínicas u observaciones médicas. El preprocesamiento incluirá:

- Filtrado de ruido y normalización de la señal.
- Segmentación temporal (por latidos, por ventanas).
- Representación en formatos compatibles con redes neuronales y modelos NLP (por ejemplo, codificación textual de eventos cardíacos, transformaciones tipo espectrograma si fuese necesario).

5.0.2. Aplicación de modelos de lenguaje (LLM) al ámbito clínico

Se seleccionarán cuatro modelos de lenguaje previamente entrenados en dominio médico o general, que serán adaptados a la tarea de generación de informes clínicos a partir de entradas codificadas de señales ECG. La adaptación se realizará mediante:

- Construcción de datasets de entrenamiento con ejemplos *prompt-respuesta* médicos.
- *Fine-tuning* mediante transferencia de aprendizaje, utilizando frameworks como HuggingFace Transformers, PyTorch o TensorFlow.
- Entrenamiento en entornos *cloud* o con uso de GPUs locales si se dispone.

5.0.3. Evaluación del desempeño clínico-sintáctico

La calidad de los informes generados será evaluada mediante:

- Métricas automáticas: BLEU, ROUGE, METEOR (comparando textos generados vs. informes reales).
- Validación cualitativa (cuando sea posible): Análisis por criterios médicos o comparación con informes redactados por especialistas.
- Estudio de casos para analizar fortalezas y limitaciones de cada modelo, identificando errores, ambigüedades o sesgos.

5.0.4. Herramientas y entornos

- Lenguaje de programación: Python
- Librerías: Transformers (HuggingFace), Scikit-learn, PyTorch, TensorFlow, Pandas, NumPy
- Entornos: Google Colab
- Repositorios de datos: PTB-XL

Esta metodología permite una aproximación modular y replicable, con etapas bien definidas, facilitando el análisis objetivo del rendimiento de cada modelo y asegurando la trazabilidad de resultados.

Capítulo 6. DESARROLLO DEL PROYECTO

6.1 Planificación del proyecto

Actividades establecidas

ID	Inicio	Fin	Duración	Actividad
A1	2025-07-01	2025-07-14	2 sem	Revisión bibliográfica y estado del arte (LLM/ECG)
A2	2025-07-15	2025-07-28	2 sem	Selección y preparación de datos (PTB-XL)
A3	2025-07-29	2025-08-04	1 sem	Definición/preparación de modelos (BioGPT, BioMistral-7B, PMC-LLaMA, BioMedLM)
A4	2025-08-05	2025-08-31	4 sem	Fine-tuning (LoRA/QLoRA) de los 4 modelos
A5	2025-09-01	2025-09-07	1 sem	Generación de informes clínicos automáticos
A6	2025-09-08	2025-09-16	1.5 sem	Evaluación automática (BLEU/ROUGE) y cualitativa
A7	2025-09-17	2025-09-23	1 sem	Análisis de resultados y validación de la propuesta
A8	2025-09-24	2025-09-30	1 sem	Redacción final y materiales de defensa

Cronograma (Gantt)

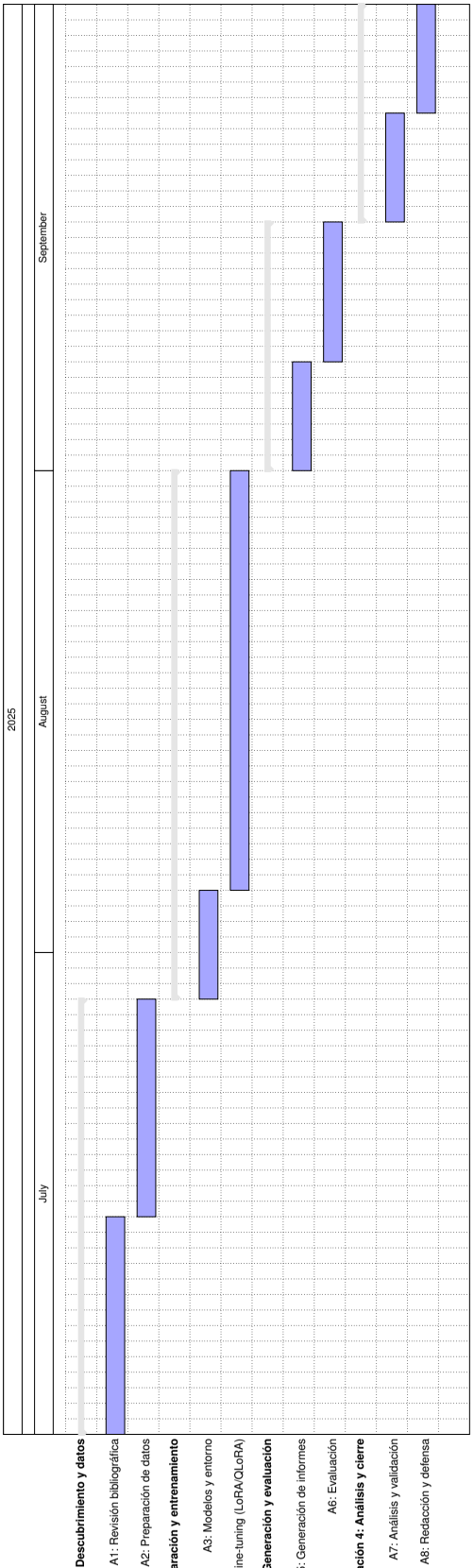

Figura 6.1. Cronograma (Gantt) — julio–septiembre de 2025

6.2 Descripción de la solución, metodologías y herramientas empleadas

6.2.1. Obtención y construcción del conjunto de datos

El objetivo fue construir un único corpus textual para entrenamiento supervisado (*causal LM*) a partir de **PTB-XL**, de modo que cada muestra contuviese una *entrada clínica estructurada* y el *informe* correspondiente, unidos en una sola columna `text` con el separador "↓". Las fuentes empleadas fueron:

- **PTB-XL** (`ptbxl_database_with_hr.csv`): señales y metadatos de ECG, incluyendo edad, sexo, altura, peso, y una frecuencia cardiaca derivada (`heart_rate_neurokit`).
- **SCP statements** (`scp_statements.csv`): diccionario de códigos `scp_codes` → descripciones clínicas. Se tomaron sólo las entradas marcadas con `diagnostic = 1`.

Herramientas

- `pandas`, `ast`, `scikit-learn` (división train/validation).
- `deep-translator` (GoogleTranslator) para traducir al inglés los informes originales (`report`).

Pipeline de construcción

1. **Lectura y saneado**: carga de ambos CSV, renombrado de columnas, y conversión de `scp_codes` a *dict* con `ast.literal_eval`.
2. **Filtrado diagnóstico**: se crea un mapa `SCP_CODE` → "*SCP-ECG Statement Description*" sólo para filas con `diagnostic=1`. Esto asegura que las descripciones usadas representen hallazgos clínicos (no artefactos/ritmos auxiliares).
3. **Enriquecimiento semántico**: cada registro de PTB-XL se anota con `scp_descriptions` (un *join* por "; " de todas las descripciones encontradas para sus `scp_codes`).
4. **Filtro de utilidad**: se retienen únicamente muestras con `scp_descriptions` no vacías, `heart_rate_neurokit` no nulo y `report` disponible.
5. **Estandarización de *metadata***:
   - **Sexo**: mapeo binario a `Male`/`Female`.
   - **Edad, altura, peso, FC**: conversión a enteros y unidades explícitas (cm, kg, bpm).
6. **Traducción del informe**: `GoogleTranslator(source='auto', target='en')` sobre `report` para obtener `report_translated`. Los fallos de traducción (excepciones) se devuelven como cadena vacía y pueden filtrarse en etapas posteriores.
7. **Construcción de la *entrada clínica***: se genera `prompt` concatenando campos clave:

```
Age: <x>; Sex: <y>; Height: <z> cm; Weight: <w> kg; Heart rate: <h>
    bpm; ECG findings: <descripciones>
```

8. **Empaquetado para entrenamiento**: se crea `text` como

   `prompt \n↓\n report_translated`

Esquema resultante

Columna	Descripción
`prompt`	Entrada clínica textual (edad, sexo, antropometría, FC y hallazgos SCP).
`report_translated`	Informe clínico traducido al inglés (`deep-translator`).
`text`	**Campo de entrenamiento**: `prompt` ↓ `report_translated`.

Razones de diseño

- **Unificar idioma a inglés**: los modelos base (BioGPT, BioMistral, PMC-LLAMA) han sido preentrenados mayoritariamente en inglés biomédico; traducir los informes reduce el *mismatch* de dominio y mejora la probabilidad de generalización.
- **Separador explícito** ↓: permite a un modelo causal aprender la transición *entrada*→*informe* sin necesidad de plantillas complejas; el mismo formato sirve para todos los modelos.
- **SCP *diagnostic*=1**: prioriza descriptores clínicos relevantes y disminuye ruido (por ejemplo, etiquetas técnicas o de calidad de señal).
- **Campos clínicos mínimos y normalizados**: la estructura *(edad, sexo, medidas, FC, hallazgos)* es semánticamente suficiente para describir el contexto del ECG sin incluir la señal cruda; esto facilita la reproducibilidad y abarata el cómputo.

Ventajas y limitaciones

Ventajas

- **Simplicidad reproducible**: pipeline puramente tabular y textual; no requiere procesar trazas de 12 derivaciones.
- **Compatibilidad universal**: el mismo `text` funciona con modelos causales diferentes (BioGPT/BioMistral/PMC-LLAMA), tanto para ajuste como para inferencia.
- **Control de estilo**: la construcción del `prompt` fuerza unidades y formato homogéneos (cm, kg, bpm), lo que reduce ambigüedades.

Limitaciones

- **Traducción automática**: puede introducir errores o pérdida de matiz clínico; mitigación: muestreo y verificación manual de una fracción del corpus.
- **Cobertura de SCP**: sólo se retienen códigos con `diagnostic=1`; algunos hallazgos útiles podrían quedar fuera si no están etiquetados como diagnósticos.
- **Campos ausentes**: fallos de traducción o metadatos nulos deben filtrarse/depurarse antes del entrenamiento para evitar ejemplos vacíos.

Ejemplo de muestra (formato final)

```
Age: 54; Sex: Male; Height: 172 cm; Weight: 70 kg; Heart rate: 83 bpm; ECG findings:
inferior myocardial infarction
↓
Inferior infarction with Q waves in II, III, aVF; ST changes consistent with old
MI.
```

6.2.2. Procesamiento y generación de etiquetas a partir de señales ECG

Cabe destacar que el conjunto de datos *PTB-XL* ya contiene el análisis de las señales crudas de ECG de 12 derivaciones. Por este motivo, no es necesario centrarse en el procesamiento inicial de dichas señales. Las etiquetas que proporciona este conjunto de datos son ideales para generar las entradas de los modelos que se emplearán durante el proceso de *fine-tuning*.

Por otro lado, es importante considerar qué ocurriría en caso de no disponer de estos datos ya procesados y etiquetados. Por ello, he investigado brevemente este aspecto y puedo describir el procedimiento que podría aplicarse, el cual se consideraría como una posible línea futura de trabajo.

El proceso básico que he desarrollado consta de dos etapas:

- **Etapa 1: Extracción de *embeddings***. Se realiza utilizando el modelo *HuBERT-ECG-Small*. Este modelo no proporciona diagnósticos directamente, sino que genera una representación abstracta de la forma de la señal, capturando sus características más relevantes.
- **Etapa 2: Aprendizaje de la relación entre los *embeddings* y las etiquetas**. Esta etapa se lleva a cabo mediante un clasificador supervisado que aprende a asociar las representaciones obtenidas con las etiquetas diagnósticas correspondientes.

El flujo general del proceso puede resumirse de la siguiente forma:

$$ECGcrudo \rightarrow HuBERT-ECG(embedding) \rightarrow RandomForest \rightarrow EtiquetasSCP$$

En resumen, este enfoque permite transformar las señales fisiológicas crudas en representaciones numéricas útiles, que posteriormente pueden emplearse para entrenar modelos capaces de predecir etiquetas diagnósticas a partir de los patrones eléctricos del corazón. El código completo se puede consultar en el Anexo del presente documento.

6.2.3. Modelo 1: BioGPT — metodología y configuración

Objetivo
Adaptar **BioGPT** (*decoder-only*, autorregresivo) a la tarea de generar un *informe* de ECG a partir de una *entrada clínica* ya textualizada en una única columna `text` (formato: `prompt \n ↓\n report_translated`). El ajuste fino se realiza como *Language Modeling* causal.

Datos y particionado

Se parte de un CSV con la columna `text` (sin nulos) y se hace una partición reproducible 90/10:

```
df = pd.read_csv("/content/ptbxl_with_translated_reports.csv", dtype=str,
    low_memory=False)
df["text"] = df["text"].fillna("")
train_texts, test_texts = train_test_split(df["text"].tolist(), test_size=0.1,
    random_state=42)

train_ds = Dataset.from_dict({"text": train_texts})
test_ds  = Dataset.from_dict({"text": test_texts})
datasets = DatasetDict({"train": train_ds, "validation": test_ds})
```

Modelo y tokenización

Se usa el tokenizer y el modelo base `microsoft/BioGPT`. Para modelos causales tipo GPT-2 se fija `pad_token = eos_token` y se aplica truncado a `MAX_LEN=256` (compromiso entre coste y cobertura).

```
model_name = "microsoft/BioGPT"
tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=False)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

model = AutoModelForCausalLM.from_pretrained(model_name)
model.config.pad_token_id = tokenizer.pad_token_id

MAX_LEN = 256
def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True, max_length=MAX_LEN)

tokenized = datasets.map(tokenize_function, batched=True, remove_columns=["text"
    ])
```

Formación de batches y etiquetas

Para un LM causal, las `labels` deben ser los propios `input_ids`. Se delega su creación al **collator** (padding dinámico por lote), evitando desajustes de longitudes y errores de tensores.

```
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
```

Ajuste fino (fine-tuning)

Entrenamiento con `Trainer` sobre el objetivo de *next-token prediction*:

```
training_args = TrainingArguments(
    output_dir="./biogpt-finetuned-ecgv02",
    save_strategy="no",
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    num_train_epochs=2,
    learning_rate=2e-5,
    weight_decay=0.01,
    fp16=True,
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized["train"],
    eval_dataset=tokenized["validation"],
    data_collator=data_collator
)
trainer.train()
```

Justificación de hiperparámetros clave

- **Épocas = 2**. En la primera iteración se prioriza rapidez y evitar sobreajuste temprano sobre un modelo ya especializado (BioGPT). Dos épocas suelen ser suficientes para que la pérdida descienda y el modelo capture el *formato* de salida; si las curvas sugieren infraajuste, se amplía en iteraciones posteriores (y se activa validación periódica).
- **Learning rate =** 2×10^{-5}. Tasa conservadora típica para *full fine-tuning* de modelos tipo GPT-2 en dominios cercanos. Es lo bastante baja para evitar *catastrophic forgetting* del conocimiento biomédico previo y lo bastante alta para adaptarse al subdominio ECG con batches pequeños.
- **Weight decay = 0.01**. Regularización L2 *desacoplada* (AdamW) que penaliza pesos grandes y ayuda a controlar el sobreajuste cuando el conjunto de entrenamiento es relativamente específico y el número de pasos no es elevado. El valor 0.01 es un estándar robusto en NLP.

Notas operativas

- **fp16=true**: reduce memoria y acelera en GPU compatibles (Colab T4/L4/A100).
- **Batch = 2 y** `MAX_LEN`**=256**: equilibrio práctico para VRAM de Colab; si la GPU lo permite, aumentar a 512–1024 para cubrir informes más largos.
- **Collator causal**: al generar `labels` tras el padding, evita errores de "sequence length mismatch" en el entrenamiento por lotes.

6.2.4. Modelo 2: BioMedLM — QLoRA y entrenamiento eficiente

Objetivo

Ajustar **BioMedLM** al subdominio ECG empleando **QLoRA (4-bit)** para reducir VRAM y una **máscara de pérdida** que haga que el gradiente sólo afecte a la parte posterior al separador `\n↓\n`. El dataset parte de una columna `text` en formato `prompt \n↓\n reference`.

1) Carga y partición del CSV

```
import os, torch, pandas as pd
from dataclasses import dataclass
from typing import Dict, List, Union
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig,
    TrainingArguments, Trainer
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training

CSV_PATH   = "/content/ptbxl_with_translated_reports.csv"
MODEL_NAME = "stanford-crfm/BioMedLM"
BLOCK_SIZE = 256
SEP = "\n↓\n"

df = pd.read_csv(CSV_PATH)
split = df["text"].astype(str).str.split("↓", n=1, expand=True)
df["prompt_en"] = split[0].fillna("").str.strip()
df["reference"] = split[1].fillna("").str.strip()

dataset = Dataset.from_pandas(df[["prompt_en", "reference"]], preserve_index=
    False)
dataset = dataset.train_test_split(test_size=0.1, seed=42)
train_dataset, eval_dataset = dataset["train"], dataset["test"]
```

2) Tokenizer y carga del modelo en 4-bit (QLoRA)

```
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, use_fast=True)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
PAD_ID, EOS_ID = tokenizer.pad_token_id, tokenizer.eos_token_id

use_bf16 = torch.cuda.is_available() and torch.cuda.get_device_capability(0)[0]
    >= 8
torch.backends.cuda.matmul.allow_tf32 = True
try:
    torch.set_float32_matmul_precision("high")
except:
    pass
os.environ["WANDB_DISABLED"] = "true"
```

```
os.environ["TOKENIZERS_PARALLELISM"] = "false"

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.bfloat16 if use_bf16 else torch.float16,
)

model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_config,
    device_map="auto",
)
model.config.pad_token_id = PAD_ID
```

3) Preparación LoRA con PEFT

```
model = prepare_model_for_kbit_training(model)

possible_targets = ["query_key_value", "c_attn", "q_proj", "k_proj", "v_proj", "
    o_proj"]
present_module_names = [n for n, _ in model.named_modules()]
target_modules = [t for t in possible_targets if any(t in n for n in
    present_module_names)]
if not target_modules:
    target_modules = ["c_attn"]

lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=target_modules,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
)
model = get_peft_model(model, lora_config)
```

4) Tokenización con máscara de pérdida

```
def tokenize_and_mask(example):
    prompt = (example["prompt_en"] or "").strip() + SEP
    target = (example["reference"] or "").strip()
    full = prompt + target
    enc = tokenizer(full, truncation=True, max_length=BLOCK_SIZE, padding=False)
    prompt_ids = tokenizer(prompt, add_special_tokens=False)["input_ids"]
    labels = enc["input_ids"].copy()
```

```
    labels[:len(prompt_ids)] = [-100] * len(prompt_ids)
    enc["labels"] = labels
    return enc

train_dataset = train_dataset.map(tokenize_and_mask, remove_columns=train_dataset
    .column_names)
eval_dataset = eval_dataset.map(tokenize_and_mask, remove_columns=eval_dataset.
    column_names)
```

5) Collator causal

```
@dataclass
class CausalCollator:
    tokenizer: AutoTokenizer
    pad_to_multiple_of: int = 8
    def __call__(self, features: List[Dict[str, Union[List[int], torch.Tensor]]])
     -> Dict[str, torch.Tensor]:
        batch = self.tokenizer.pad(
            features,
            padding=True,
            pad_to_multiple_of=self.pad_to_multiple_of,
            return_tensors="pt"
        )
        max_len = batch["input_ids"].shape[1]
        labels = []
        for f in features:
            lab = f["labels"] + [-100] * (max_len - len(f["labels"]))
            labels.append(lab)
        batch["labels"] = torch.tensor(labels, dtype=torch.long)
        return batch

data_collator = CausalCollator(tokenizer)
```

6) Entrenamiento con `Trainer`

```
training_args = TrainingArguments(
    output_dir="./biomedlm-finetuned",
    eval_strategy="no",
    save_strategy="no",
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    gradient_accumulation_steps=4,
    num_train_epochs=2,
    learning_rate=2e-5,
    weight_decay=0.01,
    report_to="none",
    fp16=False,
```

```
    dataloader_pin_memory=False,
    dataloader_num_workers=0,
    group_by_length=True
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=tokenizer,
    data_collator=data_collator
)

trainer.train()
```

Justificación de hiperparámetros

- **Épocas = 2**: primera iteración rápida para adaptar estilo y formato sin sobreajuste; si la pérdida o las métricas indican infraajuste, se amplía en iteraciones posteriores.
- **Learning rate =** 2×10^{-5}: tasa conservadora y estable en QLoRA; lo bastante baja para no degradar el conocimiento biomédico previo y lo bastante alta para converger con *batch* efectivo pequeño.
- **Weight decay = 0.01**: regularización L2 (AdamW) que estabiliza el entrenamiento y mitiga sobreajuste con un número moderado de pasos.
- **Gradient accumulation = 4**: con *batch* 1 en T4, logra un *batch* efectivo ≈ 4 sin agotar VRAM.
- **group_by_length = true**: agrupa por longitud para reducir padding medio y acelerar el entrenamiento.

6.2.5. Modelo 3: PMC-LLaMA — QLoRA/LoRA y plantilla instruccional

Objetivo

Adaptar **PMC-LLaMA** al subdominio ECG empleando **QLoRA (4-bit)** y una **plantilla instruccional** que separa claramente entrada y salida: *### Input: ...* y *### Report: ...*. Se utiliza la variante `chaoyi-wu/PMC_LLAMA_7B_10_epoch` y se fuerza el uso de `LlamaTokenizer` para garantizar compatibilidad del tokenizador.

1) Modelo y tokenizador en 4-bit

```
from transformers import AutoModelForCausalLM, BitsAndBytesConfig, LlamaTokenizer
import torch

MODEL_NAME = "chaoyi-wu/PMC_LLAMA_7B_10_epoch"
MAX_LEN = 256
```

```

bnb_cfg = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.float16
)

tokenizer = LlamaTokenizer.from_pretrained(MODEL_NAME, use_fast=False)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_cfg,
    device_map="auto",
    trust_remote_code=True
)
model.config.use_cache = False
model.gradient_checkpointing_enable()
model.config.pad_token_id = tokenizer.pad_token_id
```

2) Construcción del dataset con plantilla

```
from datasets import load_dataset

CSV_PATH = "/content/ptbxl_with_translated_reports.csv"
SEED = 42

raw = load_dataset("csv", data_files=CSV_PATH)["train"].train_test_split(
    test_size=0.1, seed=SEED)

def format_examples(batch):
    out = []
    for t in batch["text"]:
        if "↓" in t:
            src, tgt = t.split("↓", 1)
        else:
            src, tgt = t, ""
        out.append(f"### Input:\n{src.strip()}\n\n### Report:\n{tgt.strip()}")
    return {"formatted": out}

raw = raw.map(format_examples, batched=True, remove_columns=raw["train"].
    column_names)
```

3) Activación de LoRA con PEFT

```python
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training

model = prepare_model_for_kbit_training(model)

peft_cfg = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","down_proj","up_proj"],
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, peft_cfg)
```

4) Tokenización y *packing* por bloques

```python
block_size = MAX_LEN

def tokenize_function(batch):
    return tokenizer(batch["formatted"])

def group_texts(examples):
    concatenated = {k: sum(examples[k], []) for k in examples.keys()}
    total_length = (len(concatenated["input_ids"]) // block_size) * block_size
    result = {k: [t[i:i+block_size] for i in range(0, total_length, block_size)]
    for k, t in concatenated.items()}
    result["labels"] = result["input_ids"].copy()
    return result

tok_train = raw["train"].map(tokenize_function, batched=True, remove_columns=raw["train"].column_names)
lm_train  = tok_train.map(group_texts, batched=True)

tok_eval = raw["test"].map(tokenize_function, batched=True, remove_columns=raw["test"].column_names)
lm_eval  = tok_eval.map(group_texts, batched=True)
```

5) Entrenamiento con `Trainer`

```python
from transformers import TrainingArguments, DataCollatorForLanguageModeling, Trainer

collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)

```

```
args = TrainingArguments(
    output_dir="/content/pmc-llama7b-ptbxl-lora",
    num_train_epochs=2,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    learning_rate=2e-5,
    bf16=False,
    fp16=True,
    save_strategy="no",
    evaluation_strategy="no",
    weight_decay=0.01,
    lr_scheduler_type="cosine",
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=lm_train,
    eval_dataset=lm_eval,
    data_collator=collator
)

trainer.train()
```

Justificación de hiperparámetros

- **Plantilla instruccional** (*### Input* y *### Report*): refuerza el formato de salida deseado y mejora la estabilidad del ajuste en modelos instruccionales.
- **QLoRA 4-bit** (nf4): reduce VRAM manteniendo calidad suficiente para el dominio; apropiado para T4/L4 y escalable a A100.
- **LoRA** ($r = 16$, $\alpha = 32$, target_modules en proyecciones de atención/MLP): actualiza pocos parámetros, acelera y mitiga sobreajuste en comparación con *full FT*.
- **Épocas = 2**: primera iteración rápida para adaptar formato y estilo; si hay infraajuste, se amplía con validación por época y *early stopping* en la siguiente iteración.
- **Learning rate =** 2×10^{-5}: conservador para QLoRA en modelos ya especializados; suficiente para converger con *batch efectivo* pequeño sin borrar conocimiento previo.
- **Weight decay = 0.01**: regularización L2 (AdamW) estable para controlar sobreajuste con número moderado de pasos.
- **MAX_LEN = 256** y **gradient_accumulation = 4**: equilibrio entre cobertura y VRAM en Colab; permite secuencias con coste razonable.

6.2.6. Modelo 4: BioMistral-7B — QLoRA/LoRA con plantilla instruccional

Objetivo

Ajustar **BioMistral-7B** al subdominio ECG con **QLoRA (4-bit)** y una **plantilla instruccional**

que separa entrada y salida: *### Input: ...* y *### Report: ...*. Se emplea `BioMistral/BioMistral-7B` y se activan adaptadores LoRA para un entrenamiento eficiente.

1) Modelo y tokenizador en 4-bit

```python
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig
import torch

MODEL_NAME = "BioMistral/BioMistral-7B"
MAX_LEN = 256

bnb_cfg = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.float16
)

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, use_fast=True,
    trust_remote_code=True)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_cfg,
    device_map="auto",
    trust_remote_code=True
)
model.config.use_cache = False
model.gradient_checkpointing_enable()
model.config.pad_token_id = tokenizer.pad_token_id
```

2) Construcción del dataset con plantilla

```python
from datasets import load_dataset

CSV_PATH = "/content/ptbxl_with_translated_reports.csv"
SEED = 42

raw = load_dataset("csv", data_files=CSV_PATH)["train"].train_test_split(
    test_size=0.1, seed=SEED)

def to_formatted(batch):
    out = []
    for t in batch["text"]:
        src, tgt = (t.split("↓", 1) + [""])[:2] if "↓" in t else (t, "")
        out.append(f"### Input:\n{src.strip()}\n\n### Report:\n{tgt.strip()}")
```

```
    return {"formatted": out}

raw = raw.map(to_formatted, batched=True, remove_columns=raw["train"].
    column_names)
```

3) Activación de LoRA con PEFT

```
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training

model = prepare_model_for_kbit_training(model)

peft_cfg = LoraConfig(
    r=16, lora_alpha=32, lora_dropout=0.05, bias="none",
    target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","
    down_proj"],
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, peft_cfg)
```

4) Tokenización y *packing* por bloques

```
block_size = MAX_LEN

def tok_fn(batch):
    return tokenizer(batch["formatted"])

def group_texts(examples):
    concatenated = {k: sum(examples[k], []) for k in examples.keys()}
    total_length = (len(concatenated["input_ids"]) // block_size) * block_size
    result = {k: [t[i:i+block_size] for i in range(0, total_length, block_size)]
    for k, t in concatenated.items()}
    result["labels"] = result["input_ids"].copy()
    return result

tok_train = raw["train"].map(tok_fn, batched=True, remove_columns=raw["train"].
    column_names)
tok_eval  = raw["test"].map(tok_fn,  batched=True, remove_columns=raw["test"].
    column_names)

lm_train = tok_train.map(group_texts, batched=True)
lm_eval  = tok_eval.map(group_texts,  batched=True)
```

5) Entrenamiento con `Trainer`

```python
from transformers import TrainingArguments, DataCollatorForLanguageModeling,
    Trainer

collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)

args = TrainingArguments(
    output_dir="/content/biomistral7b-ptbxl-lora",
    num_train_epochs=2,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    learning_rate=2e-5,
    fp16=True,
    logging_steps=50,
    save_strategy="no",
    evaluation_strategy="no",
    weight_decay=0.01,
    lr_scheduler_type="cosine",
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=lm_train,
    eval_dataset=lm_eval,
    data_collator=collator
)

trainer.train()
```

Justificación de hiperparámetros

- **Plantilla instruccional** (*### Input* y *### Report*): refuerza el formato de salida deseado y estabiliza el ajuste.
- **QLoRA 4-bit** (nf4): reduce VRAM manteniendo calidad; adecuado para T4/L4 y escalable a A100.
- **LoRA** ($r = 16$, $\alpha = 32$, proyecciones en atención/MLP): menos parámetros actualizados, entrenamiento más rápido y menor riesgo de sobreajuste que *full fine-tuning*.
- **Épocas = 2**: primera iteración rápida para adaptar estilo/formato; si hay infraajuste, se amplía con validación por época y *early stopping*.
- **Learning rate =** 2×10^{-5}: conservador para QLoRA en un modelo ya especializado; suficiente para converger con *batch efectivo* pequeño.
- **Weight decay = 0.01**: regularización L2 (AdamW) estable para controlar sobreajuste con número moderado de pasos.
- **MAX_LEN = 256** y **gradient_accumulation = 4**: equilibrio entre cobertura y VRAM en Colab; ampliable a 512–1024 en GPUs con más memoria.

6.2.7. Resultados y análisis de los modelos

Tabla 6.1. Resultados en *test* (n=150). ↑ indica "cuanto mayor, mejor".

Modelo	BLEU ↑	ROUGE-1 ↑	ROUGE-2 ↑	ROUGE-L ↑	METEOR ↑	n	Avg len (tok)
PMC-LLaMA	4.265	0.161	0.109	0.154	0.301	150	89.96
BioMistral-7B	2.954	0.127	0.082	0.122	0.241	150	116.35
BioGPT	2.453	0.148	0.096	0.143	0.233	150	95.48
BioMedLM	0.741	0.051	0.026	0.050	0.131	150	166.75

Qué mide cada métrica

- **BLEU**: precisión de n–gramas con penalización por brevedad; muy sensible a coincidencia literal. En texto clínico libre valores bajos son habituales.
- **ROUGE-1/2/L**: recuperación de n–gramas (1 y 2) y subsecuencia común más larga (L); mide cobertura del contenido de la referencia.
- **METEOR**: alineación con raíces y sinónimos, penalizando fragmentación; suele correlacionar mejor con juicio humano que BLEU.

Lectura de los resultados

- **PMC-LLaMA** es el mejor global: lidera **BLEU**, **ROUGE** y **METEOR**, con salidas relativamente *concisas* (90 tokens). Indica buena alineación léxico–semántica y buen ajuste al formato instruccional.
- **BioMistral-7B** ocupa el segundo lugar en todas las métricas. Sus hipótesis son más *largas* (116 tokens), lo que puede introducir verbosidad que reduce BLEU/ROUGE aun manteniendo buen METEOR.
- **BioGPT** queda cerca de BioMistral: destaca en **ROUGE-1** frente a éste (más unidades léxicas aisladas coinciden), pero algo peor en **BLEU**/**METEOR**, lo que sugiere menor fluidez/estructura global.
- **BioMedLM** es claramente inferior en todas las métricas y produce *salidas muy largas* (167 tokens), lo que penaliza BLEU/ROUGE y tampoco mejora METEOR; probable desajuste de plantilla y/o necesidad de más regularización.

Discusión

- **Efecto de la longitud**: las predicciones más extensas (especialmente BioMedLM) tienden a añadir información no presente en la referencia, lo que reduce BLEU/ROUGE. Controlar la longitud (p. ej., `max_new_tokens`, *length penalty*) suele mejorar estas métricas.
- **Formato instruccional**: los modelos instruccionales con plantilla (`### Input / ### Report`) rinden mejor; PMC-LLaMA aprovecha mejor dicha plantilla que BioMistral.
- **Métricas léxicas vs. semánticas**: BLEU/ROUGE favorecen coincidencia literal; METEOR (mayor en PMC-LLaMA) apunta a mejor proximidad semántica. Para uso clínico

conviene complementar con evaluación por hallazgos (F1 por entidad: bradicardia, QT, IAM, etc.).

Conclusión
PMC-LLaMA es la opción más robusta en este primer corte, seguida de **BioMistral-7B** y **BioGPT**. **BioMedLM** requiere revisión (control de longitud, plantilla más estricta y/o más pasos con validación).

posibles mejora

- Ajustar longitud: `max_new_tokens`, *length penalty*, *repetition penalty*.
- Validación por *epoch* y *early stopping*; posible aumento de épocas si no hay sobreajuste.
- Evaluación clínica adicional: F1 por diagnóstico/etiqueta del PTB-XL y revisión de expertos.

6.3 Recursos requeridos

Recursos de cómputo

- Google Colab (suscripción Pro/Pro+), GPU NVIDIA T4 / L4 / A100 (según disponibilidad).
- CPU x86_64 y RAM del entorno Colab; almacenamiento en Google Drive.

Datos

- **PTB-XL** (PhysioNet): metadatos y señales ECG de 12 derivaciones.
- `scp_statements.csv` (mapeo código → descripción).
- Ficheros derivados de trabajo: `ptbxl_database_with_hr.csv`, `ptbxl_with_translated_reports.csv` (columna `text` con `prompt \n↓\n report`).

Modelos preentrenados

- `microsoft/BioGPT`.
- `stanford-crfm/BioMedLM`.
- `chaoyi-wu/PMC_LLAMA_7B_10_epoch`.
- `BioMistral/BioMistral-7B`.

Software y librerías

- Python (entorno Colab) y PyTorch.
- Hugging Face: `transformers`, `peft`, `bitsandbytes`, `accelerate`, `datasets`.
- Evaluación: `evaluate`, `sacrebleu`, `rouge_score`, `meteor` (requiere OpenJDK 11).

- Procesamiento: `pandas`, `numpy`, `scikit-learn`, `tqdm`.
- Tokenización y dependencias: `sentencepiece`, `sacremoses`.
- Traducción (preparación de datos): `deep-translator`.

Gestión y almacenamiento

- Google Drive (almacenamiento de artefactos y resultados).

6.4 Presupuesto

- Periodo del proyecto: julio–septiembre de 2025 (3 meses).
- Valoración del tiempo propio: **180 h** a **20 €/h** ⇒ **3 600 €**.
- Portátil personal: valor de reposición **1 200 €**; vida útil **36 meses**; prorrateo **3/36** ⇒ **100 €**.
- GPU en la nube (Colab): **3 meses** × **11,19 €/mes** ⇒ **33,57 €**.
- Software open source (Transformers, PyTorch, etc.): **0 €**.
- Datos PTB-XL (PhysioNet): **0 €** (acceso abierto).

Detalle de costes

Tipo de coste	Valor (€)	Comentarios
Horas de trabajo del autor	3 600	180 h × 20 €/h.
Portátil (prorrateo)	100	Valor 1 200 €, vida 36 meses; prorrateo 3 meses.
GPU en la nube (Colab)	33,57	3 meses × 11,19 €/mes.
Software	0	PyTorch, Transformers, PEFT, bitsandbytes, Accelerate, Datasets, Evaluate, etc.
Datos	0	PTB-XL (acceso abierto).
Almacenamiento	0	Google Drive (incluido).
Asistencia/tutoría	0	Tutoría académica.
Material de laboratorio	0	No aplica.

Tabla 6.2. Detalle de costes del proyecto.

Resumen por categorías

Categoría	Total (€)
Coste de personal	3 600,00
Infraestructura y computación (HW + nube)	133,57
Software y licencias	0,00
Datos y material	0,00
Total proyecto	**3 733,57**

Tabla 6.3. Resumen por categoría.

6.5 Resultados del proyecto

A lo largo del periodo julio–septiembre de 2025 he construido y evaluado una metodología de generación de informes de ECG basada en LLMs especializados y datos públicos, siguiendo los objetivos propuestos.

Resumen cuantitativo

Entrené y evalué cuatro modelos: **BioGPT** (ajuste completo), **BioMedLM** (QLoRA + máscara de pérdida), **PMC-LLaMA** (QLoRA/LoRA con plantilla instruccional) y **BioMistral-7B** (QLoRA/LoRA con plantilla). Para todos utilicé el mismo *test set* (n=**150** pares *input*→*report*) y la misma configuración de inferencia (greedy, `max_new_tokens`=128). Reporto **BLEU**, **ROUGE-1**/**2**/**L** y **METEOR** en la Tabla 6.1.

El mejor desempeño global lo obtuvo **PMC-LLaMA**, liderando simultáneamente BLEU, ROUGE y METEOR en el *test*. Además, sus salidas muestran una longitud media contenida (en torno a 90 palabras), lo que sugiere buena adecuación al formato `### Input / ### Report`. BioMistral-7B y BioGPT quedaron a distancias moderadas y BioMedLM resultó claramente por debajo en todas las métricas.

Capítulo 7. DISCUSIÓN

Resultados por objetivo

Obj. 1: Explorar LLMs para interpretar ECG. Probé cuatro LLMs biomédicos con enfoques de ajuste distintos (full fine-tuning vs. QLoRA/LoRA) y dos estilos de *prompting*: separador `\n` ↓ `\n` (BioGPT/BioMedLM) y plantilla instruccional `Input` / `Report` (PMC-LLaMA/BioMistral). Observo que la **plantilla instruccional estabiliza el formato** y mejora la recuperación léxica (ROUGE) frente al separador simple.

Obj. 2: Identificar y recopilar bases de datos públicas. Aunque se nombraron diferentes conjuntos de datos, prioricé PTB-XL por su tamaño, calidad de anotaciones (SCP) y compatibilidad con el objetivo de generación de informes. Preparé un CSV con columna `text` (*prompt* + "↓" + *report*) que alimenta directamente el entrenamiento de LLMs.

Obj. 3: Diseñar la metodología de integración LLM + redes de señal. Implementé la **fase generativa** (LLM) condicionada por un *input* clínico textualizado (edad, sexo, frecuencia cardiaca cuando disponible, y hallazgos SCP). El componente de **etiquetado automático desde señal cruda** ("ECG-BERT"/CNN multietiqua) se ha documentado y queda como *línea futura* para cerrar el circuito señal→etiquetas→informe.

Obj. 4: Fine-tuning de cuatro LLMs. Ajusté: BioGPT con *full FT* (LM causal + collator), BioMedLM con **QLoRA** y máscara de pérdida (el gradiente sólo afecta al texto tras "↓"), PMC-LLaMA y BioMistral-7B con **QLoRA**/**LoRA** y **plantilla instruccional**. En todos los casos utilicé 2 épocas, (learning$_rate$)$\eta = 2 \times 10^{-5}$ y *weight decay* $= 0.01$, buscando un equilibrio entre adaptación rápida y control del sobreajuste en GPU de Colab.

Obj. 5: Evaluar la generación de informes. Completado con métricas automáticas (**BLEU**, **ROUGE**, **METEOR**). **PMC-LLaMA** lidera: BLEU=4.265, ROUGE-1/2/L=0.161/0.109/0.154, METEOR=0.301 (n=150). BioMistral-7B y BioGPT ofrecen resultados intermedios; BioMedLM tiende a ser más verboso, lo que perjudica la coincidencia léxica. En general, las diferencias son consistentes entre métricas, y METEOR apoya la mayor proximidad semántica de PMC-LLaMA.

Obj. 6: Comparar con informes de especialistas. La comparación se realizó contra informes de referencia del dataset (traducciones y anotaciones), sin evaluación humana directa. Lo recojo como *limitación* y trabajo futuro: incorporar lectura cualitativa por cardiólogos y F1 por hallazgo clínico.

Obj. 7: Recomendación fundamentada. Recomiendo **PMC-LLaMA (7B) con QLoRA/LoRA y plantilla instruccional** como opción preferente para este caso de uso: mejor equilibrio cuantitativo (BLEU/ROUGE/METEOR), coste computacional contenido (4-bit), y salidas más concisas y bien formateadas. Como siguiente paso, acoplaría un *etiquetador de hallazgos* (multietiqua SCP) para cerrar el ciclo señal→informe y evaluar con expertos.

Plan de pruebas aplicado

Para garantizar comparabilidad, fijé una **batería común**: mismo *test set* (n=150), misma configuración de decodificación (greedy, `max_new_tokens`=128), mismas métricas (BLEU, ROUGE-1/2/L, METEOR) y librerías, registro de salidas por modelo (`preds_*.csv`). Además, controlé la longitud promedio de las predicciones (palabras) para interpretar el efecto de verbosidad en las métricas léxicas.

Capítulo 8. CONCLUSIONES

8.1 Conclusiones del trabajo

He demostrado que es **viable** generar informes de ECG basados en LLM con datos públicos y recursos modestos. En mi configuración, **PMC-LLaMA** ofrece el mejor compromiso entre calidad, coste y estabilidad de formato. La integración de un etiquetador desde señal cruda y la evaluación clínica humana constituirán los siguientes pasos para llevar esta metodología a un entorno real.

8.2 Conclusiones personales

Este proyecto ha sido, ante todo, un recorrido de aprendizaje muy práctico en la intersección entre el procesamiento de señal biomédica y los modelos de lenguaje. Llegué con la intuición de que los LLM podían ayudar a estandarizar y acelerar la redacción de informes de ECG; termino con una metodología reproducible, comparativas objetivas y una visión clara de qué funciona, qué no, y por qué.

En lo técnico, he afianzado competencias que antes veía como “detalles” y que resultaron decisivas: el papel de la tokenización y el *collation* (padding dinámico) para evitar errores y sesgos de longitud; el valor real de **QLoRA**/**LoRA** para entrenar de forma eficiente en recursos limitados; y la importancia de las **plantillas instruccionales** (`### Input / ### Report`) para estabilizar el formato de salida. También aprendí a leer las métricas con espíritu crítico: **BLEU** y **ROUGE** penalizan verbosidad y diferencias léxicas, mientras **METEOR** ofrece una perspectiva más semántica. Esa combinación me permitió concluir, con evidencia, que **PMC-LLaMA** era la mejor opción en mi configuración.

Capítulo 9. FUTURAS LÍNEAS DE TRABAJO

Etiquetador automático de hallazgos (tipo ECG-BERT). Como siguiente paso, se propone entrenar un codificador de señal (p. ej., "ECG-BERT" o una CNN 1D) para predecir *SCP-statements* de PTB-XL en régimen multietiqua. La tubería sería:

1. **Preprocesado**: lectura de las 12 derivaciones, normalización por canal y recorte/padding a una ventana fija.
2. **Encoder**: modelo de señales (ECG-BERT o CNN) que produce representaciones latentes.
3. **Cabeza multietiqua**: capa lineal con dimensión igual al número de códigos SCP; entrenamiento con **BCEWithLogitsLoss**.
4. **Calibración**: conversión a probabilidades via *sigmoid* y umbral por etiqueta (o *top-k*) ajustado en validación (p. ej., criterio de Youden).
5. **Mapeo a texto**: transformar las etiquetas predichas en descripciones clínicas (según `scp_statements.csv`) y usarlas como *ECG findings* en el *prompt* del LLM.

Métricas sugeridas: AUROC y F1 (micro/macro) por etiqueta en los pliegues estratificados de PTB-XL; análisis de *precision-recall* por hallazgo (bradicardia, QT, IAM, etc.).

Acoplamiento etiquetador $\rightarrow$ LLM. Con las etiquetas predichas, el LLM (PMC-LLaMA) generaría el informe usando una plantilla instruccional:

`‘‘ Input: Age: ...; Sex: ...; ECG findings: <labels>’’` $\Rightarrow$ `‘‘ Report: <informe generado>’’.`

Se explorará *constrained decoding* (palabras clave obligatorias) para garantizar que los hallazgos del clasificador aparezcan en el texto.

Capítulo 10. ANEXOS

10.0.1. ANEXO: Código HuBERT-ECG + Clasificador

Listing 10.1: Pipeline HuBERT-ECG-Small + Random Forest multi-etiqueta

```
from transformers import AutoModel
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
hubert_ecg = AutoModel.from_pretrained("Edoardo-BS/hubert-ecg-small",
    trust_remote_code=True)
hubert_ecg.to(device).eval()

import wfdb
record = wfdb.rdrecord('/content/00001_hr')  # ruta a .hea y .dat juntos
signal = record.p_signal.T  # formato (12 derivaciones, muestras)

import numpy as np
import torch

# Asegura que la señal esté en formato (12, 5000)
if signal.shape[1] != 5000:
    signal = signal[:, :5000] if signal.shape[1] > 5000 else np.pad(signal, ((0,
    0), (0, 5000 - signal.shape[1])), mode='constant')

# Inicializamos lista de embeddings
embeddings = []

# Iteramos por cada derivación
for i in range(12):
    lead = signal[i, :]  # derivación individual [5000]

    # Convertimos a tensor con forma [1, 1, 5000]
    lead_tensor = torch.tensor(lead, dtype=torch.float32).unsqueeze(0).to(device)

    # Paso por el modelo sin gradientes
    with torch.no_grad():
        output = hubert_ecg(lead_tensor)
        emb = output.last_hidden_state.mean(dim=1).cpu().numpy()[0]
        embeddings.append(emb)

# Combinamos los embeddings (por promedio, aunque podrías usar concatenación)
embedding_final = np.mean(embeddings, axis=0)
print("Embedding final de las 12 derivaciones:", embedding_final.shape)

import pandas as pd
from ast import literal_eval
from sklearn.preprocessing import MultiLabelBinarizer
```

```
42
43  # Cargar archivo de PTB-XL
44  df = pd.read_csv("ptbxl_database.csv")
45  df['scp_codes'] = df['scp_codes'].apply(literal_eval)
46
47  # Cargar lista de SCP diagnósticos
48  scp_stat = pd.read_csv("scp_statements.csv", index_col=0)
49  diagnostic_scp = scp_stat[scp_stat.diagnostic == 1].index
50  df['diagnostic_codes'] = df['scp_codes'].apply(lambda d: [code for code in d if
        code in diagnostic_scp])
51
52  # Filtrar solo registros con al menos una etiqueta diagnóstica
53  subset_df = df[df['diagnostic_codes'].map(len) > 0].head(500)
54
55  # Paso 4: Generar embeddings
56  X = []
57  Y = []
58  errores = []
59
60  import os
61
62  for _, row in subset_df.iterrows():
63      try:
64          # Obtener nombre del archivo desde filename_hr
65          file_id = os.path.basename(row['filename_hr'])  # '00001_hr'
66          record = wfdb.rdrecord(f"/content/{file_id}")
67          signal = record.p_signal.T  # (12, muestras)
68
69          # Padding / truncamiento a 5000
70          if signal.shape[1] != 5000:
71              signal = signal[:, :5000] if signal.shape[1] > 5000 else np.pad(
        signal, ((0,0),(0, 5000 - signal.shape[1])), mode='constant')
72
73          # Obtener embeddings por derivación y promediar
74          embeddings = []
75          for i in range(12):
76              lead_tensor = torch.tensor(signal[i], dtype=torch.float32).unsqueeze
        (0).to(device)  # [1, 5000]
77              with torch.no_grad():
78                  out = hubert_ecg(lead_tensor)
79                  emb = out.last_hidden_state.mean(dim=1).cpu().numpy()[0]
80                  embeddings.append(emb)
81
82          embedding_final = np.mean(embeddings, axis=0)
83          X.append(embedding_final)
84          Y.append(row['diagnostic_codes'])
85
86      except Exception as e:
87          errores.append((file_id, str(e)))
88          print(f"Error con {file_id}: {e}")
```

```
89
90  print(f"\nProcesados correctamente: {len(X)} ECGs")
91  print(f"Con errores: {len(errores)}")
92
93  import numpy as np
94  import pandas as pd
95  from sklearn.preprocessing import MultiLabelBinarizer
96  from sklearn.model_selection import train_test_split
97  from sklearn.ensemble import RandomForestClassifier
98  from sklearn.multioutput import MultiOutputClassifier
99  from sklearn.metrics import f1_score, hamming_loss, classification_report
100
101 # Binarizar etiquetas
102 mlb = MultiLabelBinarizer()
103 Y_bin = mlb.fit_transform(Y)
104
105 # Dividir en train/test
106 X_train, X_test, y_train, y_test = train_test_split(X, Y_bin, test_size=0.2,
        random_state=42)
107
108 # Entrenamiento
109 clf = MultiOutputClassifier(RandomForestClassifier(n_estimators=100, random_state
        =42))
110 clf.fit(X_train, y_train)
111
112 # Evaluación
113 y_pred = clf.predict(X_test)
114
115 print("\nResultados del clasificador:")
116 print("Micro F1-score:", f1_score(y_test, y_pred, average='micro'))
117 print("Macro F1-score:", f1_score(y_test, y_pred, average='macro'))
118 print("Hamming loss:", hamming_loss(y_test, y_pred))
119 print("Clases SCP:", list(mlb.classes_))
120
121 print(classification_report(y_test, y_pred, target_names=mlb.classes_))
```

Bibliografía

[1] Wagner, P., Strodthoff, N., Bousseljot, R.-D., Samek, W., & Schaeffter, T. (2022). *PTB-XL, a large publicly available electrocardiography dataset* (version 1.0.3). PhysioNet. RRID:SCR_007345. https://doi.org/10.13026/kfzx-aw45

[2] Wagner, P., Strodthoff, N., Bousseljot, R., Kreiseler, D., Lunze, F. I., Samek, W., & Schaeffter, T. (2020). PTB-XL, a large publicly available electrocardiography dataset. *Scientific Data*, 7(1). https://doi.org/10.1038/s41597-020-0495-6.

[3] Zheng, J., Guo, H., & Chu, H. (2022). A large scale 12-lead electrocardiogram database for arrhythmia study (version 1.0.0) [Dataset]. PhysioNet. RRID:SCR_007345. https://doi.org/10.13026/wgex-er52.

[4] Zheng, J., Chu, H., Struppa, D., Zhang, J., Yacoub, M., El-Askary, H., Chang, A., Ehwerhemuepha, L., Abudayyeh, I., Barrett, A., Fu, G., Yao, H., Li, D., Guo, H., & Rakovski, C. (2020). Optimal Multi-Stage Arrhythmia Classification Approach. *Scientific Reports*, 10(1). https://doi.org/10.1038/s41598-020-59821-7.

[5] Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation* [Online], 101(23), e215–e220. RRID:SCR_007345. https://physionet.org.

[6] Ashley, E. A., & Niebauer, J. (2004). Conquering the ECG. *Cardiology Explained* — NCBI Bookshelf. https://www.ncbi.nlm.nih.gov/books/NBK2214/.

[7] Bergmann, D. (2025, July 22). Fine Tuning. What is fine-tuning? *IBM Think*. Retrieved October 13, 2025, from https://www.ibm.com/think/topics/fine-tuning.

[8] Sclar, M., Choi, Y., Tsvetkov, Y., & Suhr, A. (2023). Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design *or: How I Learned to Start Worrying About Prompt Formatting*. *arXiv preprint* arXiv:2310.11324. https://arxiv.org/abs/2310.11324.

[9] He, J., Rungta, M., Koleczek, D., Sekhon, A., Wang, F. X., & Hasan, S. A. (2024). Does Prompt Formatting Have Any Impact on LLM Performance? *arXiv preprint* arXiv:2411.10541. https://arxiv.org/abs/2411.10541.

[10] Avinash. (2024, August 7). LLM evaluation metrics — BLEU, ROUGE and METEOR explained. *Medium*. https://avinashselvam.medium.com/llm-evaluation-metrics-bleu-rogue-and-meteor-explained-a5d2b129e87f.

[11] BioMistral/BioMistral-7B · Hugging Face. (2024, February 19). Retrieved October 13, 2025, from https://huggingface.co/BioMistral/BioMistral-7B.

[12] chaoyi-wu/PMC_LLAMA_7B_10_epoch · Hugging Face. (n.d.). Retrieved October 13, 2025, from https://huggingface.co/chaoyi-wu/PMC_LLAMA_7B_10_epoch.

[13] Edoardo-BS/hubert-ecg-small · Hugging Face. (n.d.). Retrieved October 13, 2025, from https://huggingface.co/Edoardo-BS/hubert-ecg-small.

[14] microsoft/biogpt · Hugging Face. (n.d.). Retrieved October 13, 2025, from https://huggingface.co/microsoft/biogpt.

[15] stanford-crfm/BioMedLM · Hugging Face. (2024, January 26). Retrieved October 13, 2025, from https://huggingface.co/stanford-crfm/BioMedLM.