Tech Sphere 2025

Clasificación Multi-Label de Literatura Biomédica

Solución con Transformers : Modelo BioBERT

Challenge de Clasificación Biomédica con IA

Presentado por:

Tomás Rodríguez Taborda

Estudiante de Ingeniería de Sistemas e Informática y Estadística

Área de Concentración: Inteligencia Artificial y Machine Learning

Agosto, 2025

Índice

1. Introducción al Problema

La clasificación automática de texto en distintas categorías es una de las aplicaciones más extendidas de la inteligencia artificial en la actualidad. Estos modelos permiten construir sistemas de recomendación más precisos y coherentes con el contenido real y, en contextos académicos, facilitan los procesos de revisión y organización de literatura científica de manera más eficiente, algo especialmente útil en áreas como la medicina, donde la producción de artículos crece de manera exponencial.

Sin embargo, trabajar con texto presenta diversos retos. Uno de ellos es la polisemia, en la que una misma palabra puede tener múltiples significados dependiendo del contexto. Otro es la sinonimia, ya que distintas palabras pueden expresar la misma idea. Estas características del lenguaje natural dificultan las tareas de clasificación automática. En el contexto biomédico, esta complejidad se acentúa: por ejemplo, el término falla puede aparecer en expresiones como falla renal o falla cardíaca, y el modelo debe ser capaz de diferenciar con precisión los significados semánticos asociados a cada dominio [1].

En este escenario se han desarrollado diferentes enfoques. Por un lado, existen los modelos de propósito general, entrenados con cantidades masivas de datos (como GPT o Claude), capaces de abarcar múltiples dominios. Por otro, se encuentran los modelos especializados en un área concreta del conocimiento, que suelen ofrecer mejores resultados cuando se trabaja con datos altamente técnicos o específicos, como en el caso de la literatura biomédica [3].

Un desafío adicional surge con la longitud de los documentos: a medida que los textos se hacen más extensos, resulta difícil para los sistemas relacionar términos que aparecen en distintas secciones (inicio, medio y final) y mantener la coherencia semántica. Este problema fue mitigado con la aparición de los Transformers y su mecanismo de atención [4], que permite modelar dependencias a largo plazo dentro del texto. Estos avances dieron origen a arquitecturas de gran impacto como GPT (Generative Pretrained Transformer) y, además, impulsaron el desarrollo de múltiples variantes entrenadas específicamente en distintos dominios del conocimiento.

En este contexto, nuestra propuesta se fundamenta en el uso de modelos basados en Transformers y técnicas de Deep Learning, aprovechando su capacidad para capturar la semántica del texto y adaptarla a la clasificación biomédica. Dentro de la solución, hay que tener en cuenta que este es un escenario de clasificación multi-etiqueta, en el cual un mismo artículo puede pertenecer a más de una categoría de manera simultánea. Esto exige que el modelo no solo capture el significado dentro de un dominio específico, sino también las interrelaciones semánticas entre múltiples dominios biomédicos.

2. Análisis Descriptivo

Verificamos inicialmente la integridad del dataset, asegurándonos de que no existan duplicados ni textos excesivamente similares que pudieran generar alertas. Esto se realiza con el fin de evitar posibles problemas en la etapa de entrenamiento, como data leakage o la exclusión de clases debido a errores tipográficos en la variable de respuesta. Los resultados muestran que no hay textos repetidos ni vacíos, y que todas las etiquetas son válidas. Únicamente se detectaron un par de textos con una similitud superior al 80 %; sin embargo, tras revisarlos se concluyó que, aunque tratan un tema parecido, corresponden a artículos distintos, por lo que se decidió conservar ambos

para no perder información útil.
Analizarems, mediante medidas de tendencia central, la longitud de los títulos y abstracts en su forma cruda, es decir, antes de aplicar cualquier tipo de preprocesamiento. En los títulos se observa una distribución bastante uniforme, mientras que en los abstracts se presenta una mayor variabilidad en el número de palabras. Esta dispersión se evidencia en los saltos pronunciados entre los percentiles 50 % y 75 %, y entre el 75 % y el valor máximo.

Estadística	**Títulos**	**Abstracts**	**Texto Completo**
Count	3565.00	3565.00	3565.00
Mean	8.73	100.06	108.79
Std	4.88	93.07	96.62
Min	2.00	22.00	28.00
25 %	5.00	31.00	37.00
50 %	7.00	37.00	43.00
75 %	11.00	172.00	185.00
Max	38.00	525.00	535.00

Tabla 1: Estadísticas descriptivas de longitud de los textos

En el contexto del problema de clasificación que abordamos, este análisis permite identificar que en los abstracts se concentra la mayor parte de la información semántica útil para la tarea, mientras que los títulos resultan más homogéneos y menos informativos. Además, si se adopta un enfoque basado en modelos de tipo Transformer, es necesario considerar un preprocesamiento adecuado, ya que muchos abstracts superan el límite de 512 tokens. Esto obliga a aplicar técnicas de truncamiento, segmentación o selección de fragmentos relevantes. Finalmente, examinar la variabilidad en la longitud también posibilita detectar outliers y fundamentar la decisión metodológica sobre si utilizar títulos, abstracts o el texto completo como insumo del modelo.

También miraremos que tan balanceadas estan las clases, inicialmente de manera individual para tener una imagen más transparente de la presencia individual de cada característica. Vemos como el dataset presenta un desbalanceo, pues una de las clases representa ya la mitad del dataset, neurological, mientras que oncologial representan tan solo el 16.9 % de todo el dataset, mientras cardiovascual y hepatorenal estan relativamente similares entre si.

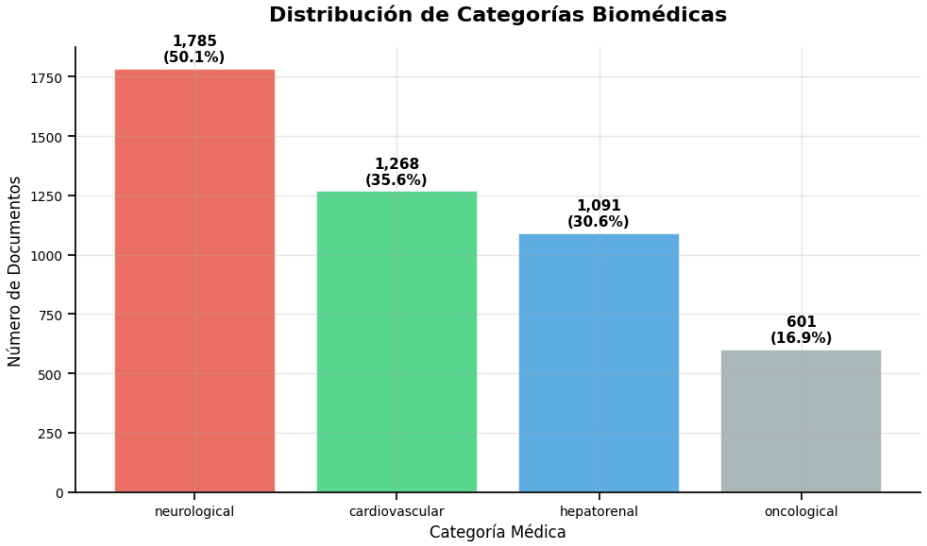

Figura 1: Gráfico de barras que muestra la frecuencia de aparición de cada etiqueta individual en el dataset.

En el caso multietiqueta, donde cada documento puede pertenecer a más de una categoría, se observa una marcada diferencia en la cantidad de datos disponibles para las combinaciones de 2, 3 y 4 etiquetas. En particular, la clase correspondiente a la combinación de las cuatro etiquetas está subrepresentada, con tan solo 7 observaciones. No obstante, este patrón es relevante, ya que indica al modelo que la coocurrencia de las cuatro etiquetas es posible.
Asimismo, entre las combinaciones de dos etiquetas también existe una gran disparidad, explicada principalmente por la alta frecuencia de la categoría neurológica, que domina las asociaciones.
En conjunto, tanto la Figura 1 como la Figura 2 evidencian la necesidad de considerar el desbalance de clases durante el entrenamiento del modelo, dado que varias categorías y combinaciones se encuentran subrepresentadas.

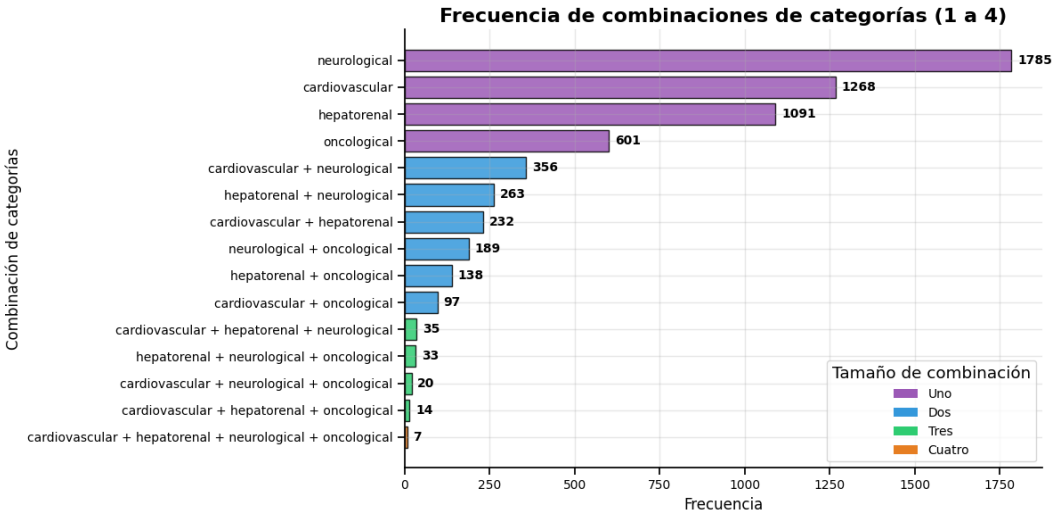

Figura 2: Gráfico de barras que muestra la frecuencia de aparición de las combinaciones de etiquetas en el dataset.

Se observa que la distribución de la longitud de los títulos está sesgada a la derecha, con la mayoría de los valores concentrados en un rango de aproximadamente 5 a 10 palabras.

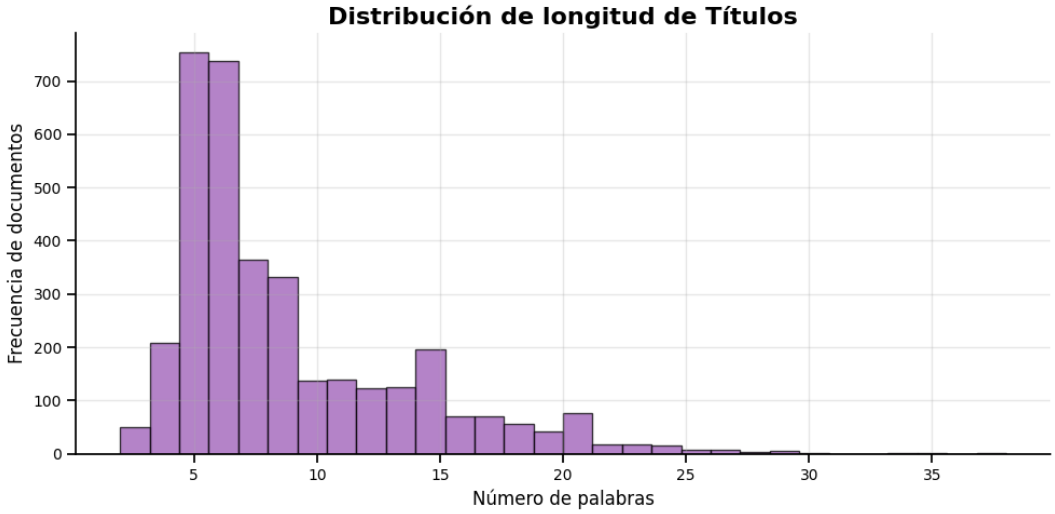

Figura 3: Histograma con la distribución de longitud de los títulos.

En el caso de la longitud de los abstracts ocurre un comportamiento similar: la distribución está fuertemente sesgada a la derecha, concentrándose la mayoría de los datos en abstracts con menos de 50 palabras. Esto coincide con las medidas de tendencia central obtenidas anteriormente, las cuales evidencian que en los valores extremos de la distribución hacia la derecha se presentan muy pocos casos, lo que explica los saltos pronunciados observados en los percentiles superiores.

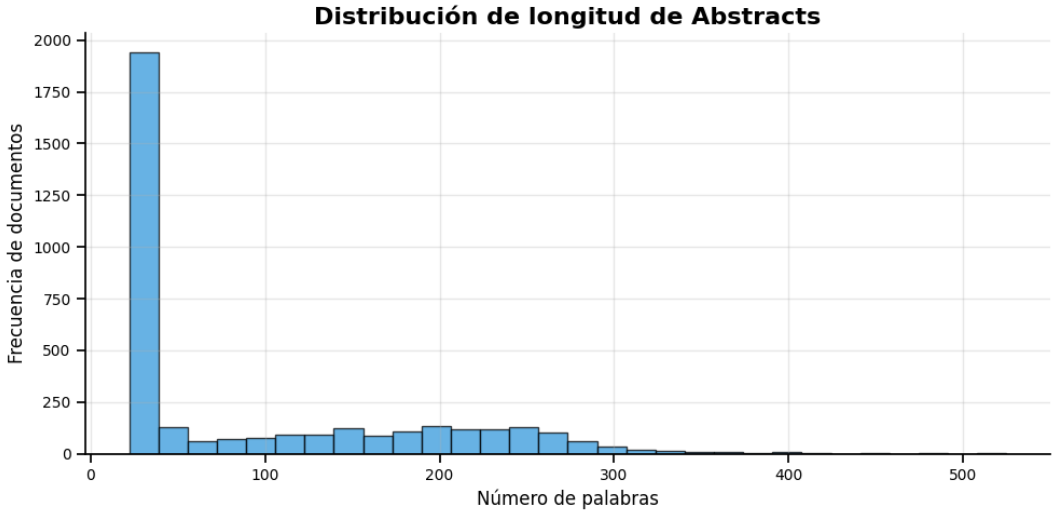

Figura 4: Histograma con la distribución de longitud de los abstracts.

Como complemente a los histogramas, presentamos los boxplots que permiten visualizar de manera más clara la presencia de datos atípicos tanto en los títulos como en los abstracts. Si bien en ambos casos se observan datos atípicos, textos con una mayor cantidad de palabras respecto al promedio,

no se identifican valores extremos que puedan comprometer el aprendizaje del modelo debido a longitudes excesivamente altas o inusualmente bajas.

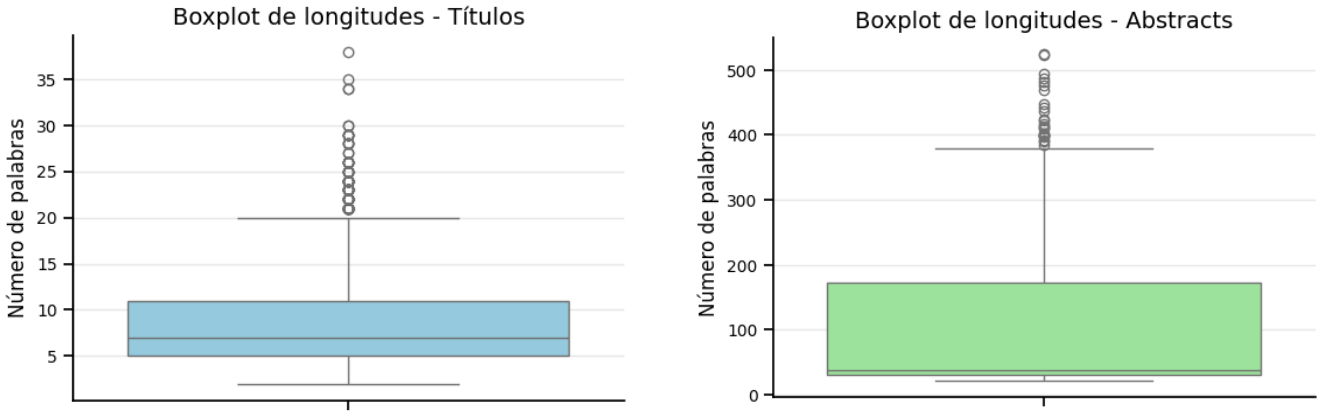

Figura 5: Boxplot con la longitud de títulos y abstracts.

En las Figuras 6 y 20 se presentan las palabras más frecuentes (quitando stopwords) según el tipo de etiquetas asociadas al título y al abstract. Este análisis revela dos aspectos relevantes: por un lado, la variedad semántica presente en cada una de las categorías y sus combinaciones; y por otro, que se trata de un problema enmarcado dentro de un vocabulario altamente especializado. En consecuencia, cualquier solución propuesta debe considerar esta característica, ya sea incorporando dicho vocabulario en la base de datos o utilizando modelos entrenados específicamente en este dominio, de modo que puedan capturar adecuadamente el significado semántico.

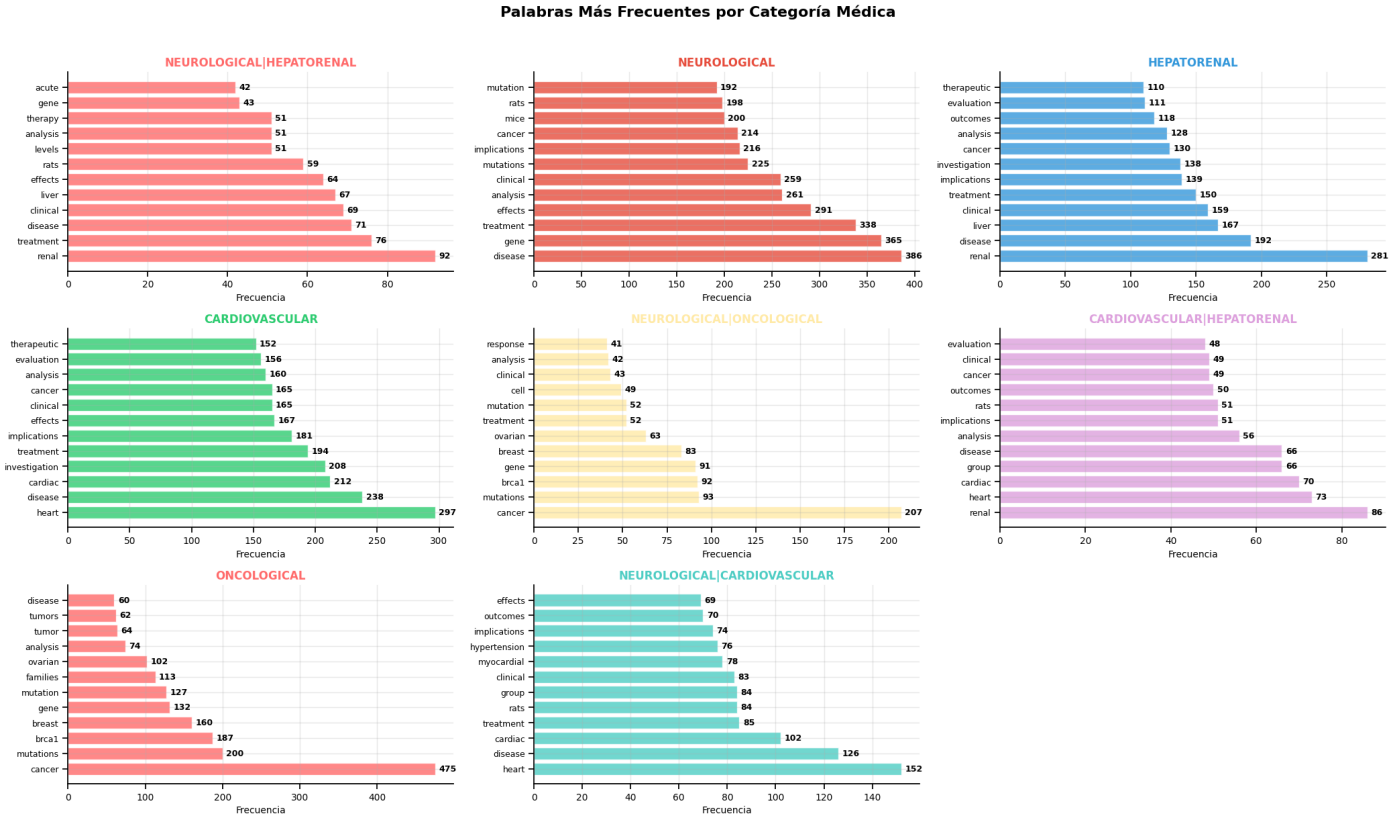

Figura 6: Palabras más frecuentes según la etiqueta a la que pertenece el artículo.

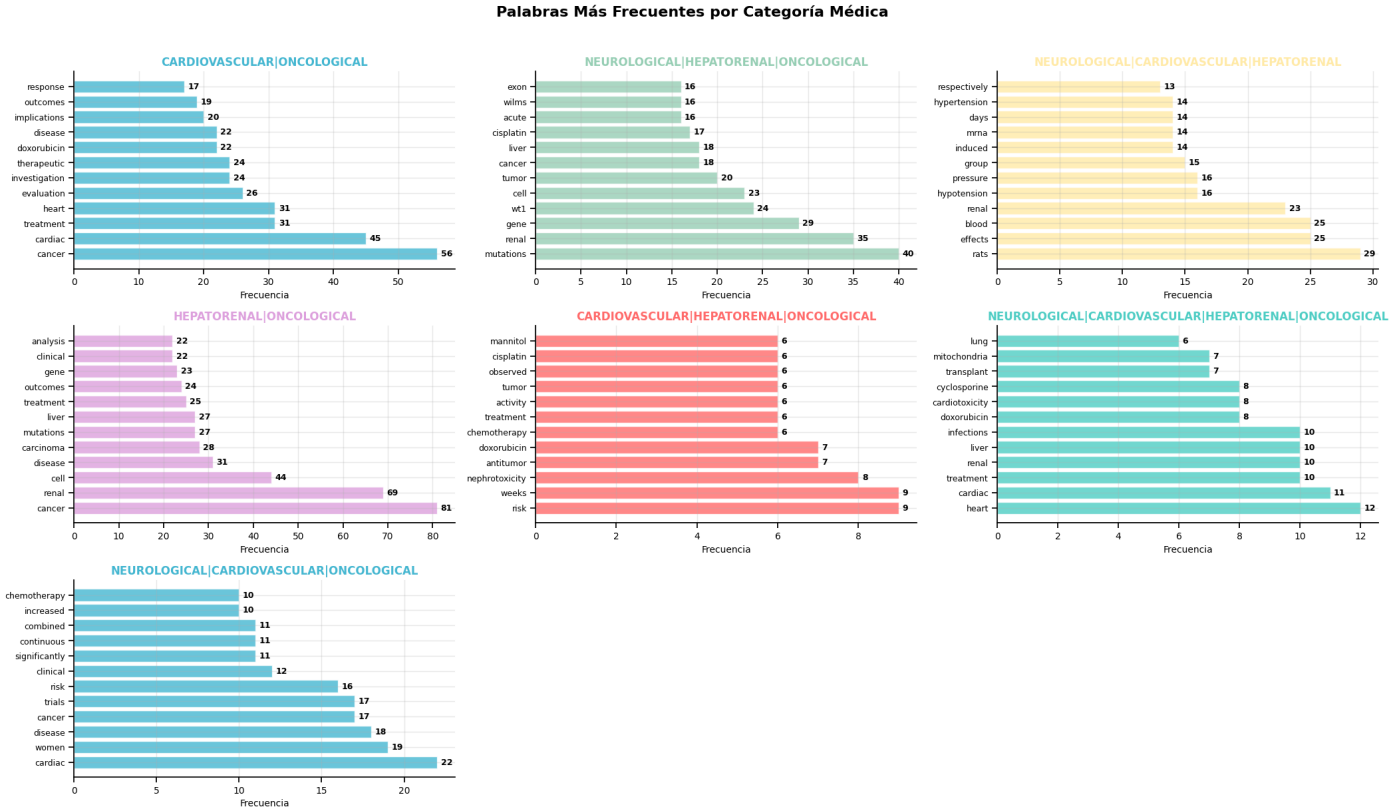

Figura 7: Palabras más frecuentes según la etiqueta a la que pertenece el artículo.

Teniendo en cuenta lo anterior, los principales desafíos para realizar la clasificación correctamente son, por un lado, el desbalance de clases, que sea capaz de realizar la clasificación sin importar la longitud variada de los textos, y por otro, la necesidad de diseñar una solución que sea capaz de incorporar el significado semántico dentro de un contexto de vocabulario especializado.

3. Propuesta de Solución

Teniendo en cuenta lo expuesto en la introducción al problema y apoyándonos en el análisis descriptivo realizado, se refuerza la pertinencia de plantear una solución basada en modelos Transformers, en particular en uno entrenado específicamente sobre literatura médica: BioMedBERT. Este modelo fue entrenado utilizando abstracts de PubMed, lo que le otorga una alta capacidad para capturar el significado semántico de textos biomédicos.

Como se mencionó previamente, el uso del mecanismo de atención resulta especialmente valioso para preservar dicha captura semántica en textos extensos, como ocurre con frecuencia en los abstracts científicos. Además, al estar fundamentado en la arquitectura BERT, el modelo aprovecha el análisis de contexto bidireccional, es decir, la capacidad de interpretar el sentido de cada palabra considerando aquellas que la rodean [2].

En conjunto, estas características hacen de BioMedBERT una alternativa altamente adecuada y eficaz para abordar el problema de clasificación planteado.

3.1. Implementación de la Solución

3.1.1. Preprocesamiento

Trabajaremos tanto con el título como con el abstract como un único texto, ya que esto permite tener una base semántica mucho más rica para clasificar los artículos.

- **Tokenización**: Dividimos el texto en palabras y signos de puntuación, pero teniendo en cuenta números y unidades médicas que aportan al significado semántico (mg, ml, mmHg, etc.).
- **Protección de tokens críticos**: Definimos un conjunto de palabras médicas y unidades que no deben ser modificadas ni eliminadas, como *patient*, *treatment*, *clinical*, *bp*, entre otras.
- **EDA (Easy Data Augmentation)**:
  - **Synonym Replacement**: Reemplazo de hasta dos palabras por sinónimos médicos conservadores, respetando la capitalización y sin alterar términos críticos.
  - **Random Deletion**: Eliminación aleatoria de palabras con baja probabilidad ($\approx 8\,\%$), sin afectar términos clínicos relevantes.
  - **Identity Transformation**: En algunos casos se mantiene el texto sin cambios, lo que permite conservar ejemplos originales en el dataset.

Respecto al preprocesamiento, como el manejo de stopwords o la aplicación de stemming, la ventaja de trabajar con modelos Transformers es que no es necesario eliminar las stopwords, ya que contribuyen al contexto semántico y el mecanismo de atención se encarga de asignarles el peso adecuado. Además, tampoco es necesario realizar stemming ni lematización manual, pues el tokenizador de subpalabras de estos modelos ya captura la raíz y las variaciones morfológicas de las palabras de manera eficiente.
Como observamos en el análisis descriptivos, las clases estan desbalanceadas, por lo cual asignaremos pesos diferenciados a las categorías, de modo que los errores en clases minoritarias resulten más costosos que en las mayoritarias. Adicionalmente, emplearemos técnicas de aumento de datos, tales como *Synonym Replacement*, *Random Deletion* e *Identity*, estrategias comúnmente agrupadas bajo el enfoque de Easy Data Augmentation for Text (EDA) que permiten auemtnar la riqueza semántica del dataset.
Para mitigar el riesgo de sobreajuste, se implementará una estrategia comparativa. Primero, entrenaremos un clasificador base (dummy) que actuará como referencia mínima, equivalente a un clasificador aleatorio. Posteriormente, ajustaremos múltiples variantes de BiomedBERT, explorando además mostraremos más adelantes técncas anti overfitting para que el model sea capaz de generalizar ante datos no vistos.
El modelo utilizado se encuentra disponible en Hugging Face en el siguiente enlace: `BiomedBERT`.

Manejo del Multi-Label La codificación del problema *multi-label* se realizará utilizando la clase de Scikit-Learn `MultiLabelBinarizer()`. Esta herramienta transforma las etiquetas en una representación de tipo *one-hot encoding* extendida, lo que permite representar de forma numérica la pertenencia de un mismo artículo a una o varias clases simultáneamente.

3.1.2. Modelo Dummy

Implementamos un modelo dummy que funciona como referencia mínima a superar, equivalente al desempeño esperado de un clasificador aleatorio. Para ello utilizamos la clase `DummyClassifier` de Scikit-Learn, evaluando distintas estrategias de predicción. Posteriormente, seleccionamos la que obtuvo el mejor desempeño según las métricas definidas. Los tres escenarios considerados y sus resultados se presentan en la Tabla 2.

Estrategia	F1-micro	F1-macro	F1-weighted	Hamming Loss	Exact Match
Most Frequent	0.4050	0.1608	0.2274	0.3482	0.2707
Stratified	0.3650	0.3279	0.3621	0.4197	0.1417
Uniform	**0.4086**	**0.3986**	**0.4244**	0.5084	0.0014
Constant	0.0000	0.0000	0.0000	**0.3352**	0.0000

Tabla 2: Resultados de `DummyClassifier` con diferentes estrategias

Vemos entonces como la estrategia uniforme es aquella que maximiza el F1 weighted bajo un clasificador Dummy, fijamos entonces 0.4244 como el valor mínimo que debemos superar para tener un mejor rendimiento que una clasificación aleatoria.

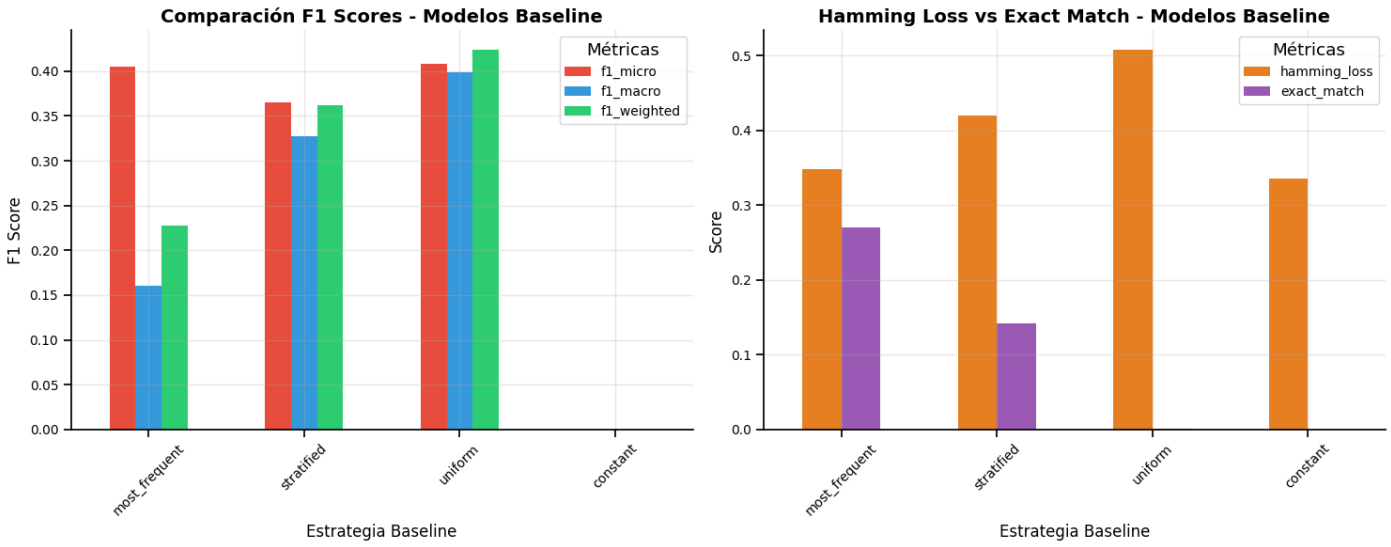

Figura 8: Gráfico de barras con las métricas del modelo Dummy.

3.1.3. Modelo Transformer

Pasaremos ya a entrenar el modelo Transformer, este entrenamiento, teniendo en cuenta el *multi-label* en el dataset se realizará mediante un proceso de clasificación binaria independiente para cada clase. En este enfoque, el modelo formula una tarea binaria del tipo:

Pertenece a la clase c **vs No pertenece a la clase** c

Este procedimiento se repite para cada una de las clases del conjunto de etiquetas. Posteriormente, se aplica la función sigmoide a la salida (*logits*) de cada clasificador, lo que permite obtener una probabilidad por clase. Finalmente, aquellas probabilidades que superen un umbral definido se consideran como clases activas, asignando así al artículo todas las etiquetas correspondientes.
El siguiente diagrama ayuda a la comprensión de como se realizarán las predicciones con un umbral de ejemplo de 0.5:

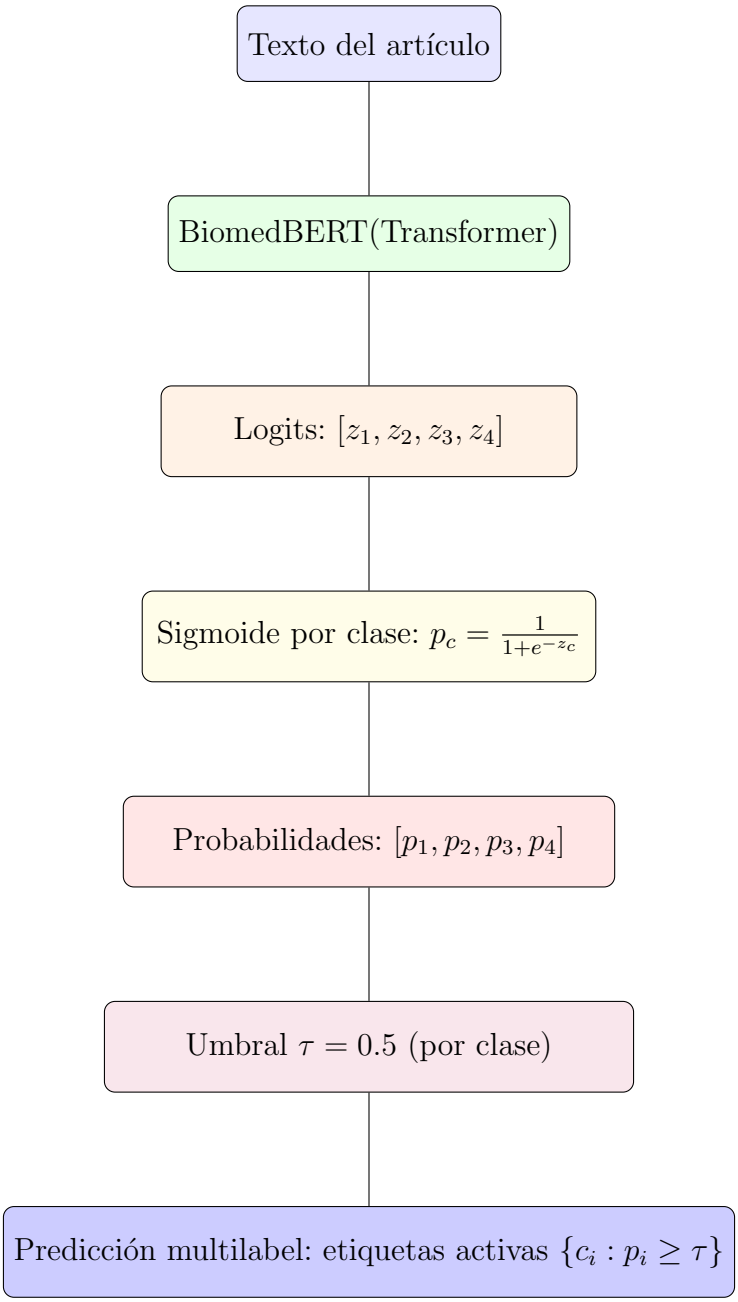

Figura 9: Esquema del proceso de predicción multilabel con BiomedBERT.

3.1.4. Preparación Dataset

usando PyTorch definimos nuestra propia clase para el dataset médico que hereda de torch.utils.data.Dataset, lo que permite trabajar con múltiples etiquetas por muestra, hacer data augmentation y tokenización con Hugging Face al usar el DataLoader (parte de torch.utils.data.Dataset). Tenemos entonces lo siguiente

- Se toma el **texto original** en la posición `idx`.
- Si `augment=True`, con una **probabilidad del 30 %** se aplica la función `augment_text`, la cual puede realizar transformaciones como reemplazo de sinónimos o eliminación de palabras.
- Luego, el texto se **tokeniza** con el tokenizador de Hugging Face:
  - `input_ids`: secuencia de tokens.
  - `attention_mask`: máscara que distingue entre tokens reales (1) y relleno (`padding`, valor 0).
- Finalmente, el método devuelve un **diccionario listo para entrenamiento** en PyTorch, compuesto por:
  - `input_ids` → secuencia tokenizada.
  - `attention_mask` → máscara de atención.
  - `labels` → vector multilabel (ejemplo: `[0,1,0,1]`).

3.1.5. Split Train & Test y Pesos a las clases

Utilizamos el objeto `iterative_train_test_split` de *scikit-multilearn* para realizar la partición entre *train* y *test*, garantizando que las proporciones de cada clase se mantengan lo más similares posible en ambos subconjuntos. El conjunto de entrenamiento quedó conformado por 2,852 muestras y el de validación por 713 muestras, con las siguientes proporciones por clase:

Clase	Train	Val	Diff
cardiovascular	0.356	0.356	0.001
hepatorenal	0.306	0.306	0.000
neurological	0.501	0.501	0.000
oncological	0.169	0.168	0.000

Tabla 3: Distribución de clases en train y validación

Recordemos que en la exploración del dataset verificamos que no hubieran datos repetidos dentro del dataset para evitar redundancias o que un ejemplo quede en train y otro en test, provocando un pequeño data leakage.
Después de esto, realizamos nivel individual para el balanceo según el porcentaje de aparición de una clase le asignamos un peso, de tal manera que las clases minoritarias pasan a ser representativas y un error en el modelo en estas clases se castiga severamente, esto lo hacemos después de haber dividido el conjunto de datos en entrenamiento y test ya que se podría considerar un pequeño

data leakage el calcular el peso según las proporciones completas del dataset, debe ser con solo el entrenmaineot ya que esa es la única información con la que debe de contar.

Clase	Documentos	Porcentaje (%)	Peso asignado
Cardiovascular	1014	35.6	1.813
Hepatorenal	873	30.6	2.267
Neurological	1428	50.1	0.997
Oncological	481	16.9	4.932

Tabla 4: Pesos asignados a cada clase para el manejo del desbalance

3.1.6. Modelos Entrenados

Tomaremos el Transformer BiomedBERT y lo haremos algo más robusto. En nuestro código esto será la clase con el nombre 'Improved Medical BERT', quedando entonces con las siguientes características

- **Base biomédica especializada:** parte de un modelo preentrenado (ej. BioBERT, PubMedBERT) que ya entiende el lenguaje médico mejor que un BERT genérico.
- **Atención ligera adicional:** una capa de `MultiHeadAttention` que permite refinar las dependencias entre tokens incluso después del encoder.
- **Clasificador más profundo:** en lugar de una sola capa lineal, incluye un MLP progresivo (`hidden → hidden/2 → hidden/4 → num_labels`) con activaciones ReLU, lo que añade mayor capacidad de representación.
- **Regularización robusta:** uso de `Dropout` configurable y `LayerNorm` en varias partes de la arquitectura, reduciendo riesgo de sobreajuste.
- **Pérdida ponderada:** utiliza `BCEWithLogitsLoss` con `pos_weight`, lo que permite compensar el desbalance entre clases poco frecuentes y frecuentes.
- **Pooling mediante [CLS]:** se aprovecha la representación global del token especial `[CLS]`, enriquecido tras la capa de atención ligera.
- **Inicialización estable:** las capas lineales nuevas (`fc1, fc2, classifier`) se inicializan con el esquema Xavier, lo que mejora la estabilidad del entrenamiento.

Teniendo entonces así un modelo BiomedBert con unas mejoras para estabilizar el entrenamiento y capturar las relaciones semánticas con mayor capacidad.

Métricas a Monitorear Para medicar la capacidad de clasificación del modelo utilizaremos las siguientes métricas para poder medir sus acietos y sus errores, siendo la métricas que vamos a priorizar el F1-Weighted

$$AP = \frac{1}{C} \sum_{i=1}^{C} \frac{TP_i}{TP_i + FP_i}$$

Average Precision (AP): Promedio de las precisiones por clase, útil en problemas multilabel.

$$F1 = 2 \cdot \frac{\text{AP} \cdot \text{Recall}}{\text{AP} + \text{Recall}}$$

F1: Media armónica entre AP y Recall, balancea exhaustividad y precisión.

$$F1_{\text{Macro}} = \frac{1}{C} \sum_{i=1}^{C} F1_i$$

F1-Macro: Promedio no ponderado de los F1 por clase, trata todas las clases por igual.

$$F1_{\text{Weighted}} = \frac{1}{N} \sum_{i=1}^{C} n_i \cdot F1_i$$

F1-Weighted: Promedio ponderado de los F1, da más peso a clases frecuentes.

$$HL = \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{j=1}^{L} \mathbf{1}[y_{ij} \neq \hat{y}_{ij}]$$

Hamming Loss: Proporción de etiquetas mal clasificadas en problemas multilabel.

Proponemos además un *score* que permita cuantificar los indicios de *overfitting* del modelo, con el objetivo de identificar la configuración con mejor capacidad de generalización. Dicho score se define como:

$$\text{OverfittingScore} = \sum_{m \in \{F1_{\text{weighted}}, F1_{\text{macro}}, AP\}} \mathbf{1}(\Delta_m > \delta) \cdot \Delta_m + \sum_{m \in \{\text{HL}\}} \mathbf{1}(\Delta_m < -\gamma) \cdot \frac{|\Delta_m|}{2}$$

Cabe resaltar que esta es una *métrica ad hoc*, diseñada únicamente para reflejar la magnitud de las diferencias entre *train* y *validation* en diversas métricas, con el fin de detectar posibles señales de *overfitting*. No debe interpretarse como un indicador sustituto de los valores de F1 ni como una medida directa de desempeño del modelo.

Modelo calibración F1-Score weighted Como nuestro objetivo es maximizar el F1 Score Weighted del modelo, probaremos diferentes umbrales en la función sigmoide para determinar qué valor debe alcanzar cada clase a fin de clasificar un artículo como perteneciente a ella. De esta forma, evaluaremos qué tan sensible es la métrica a la elección de dicho umbral.

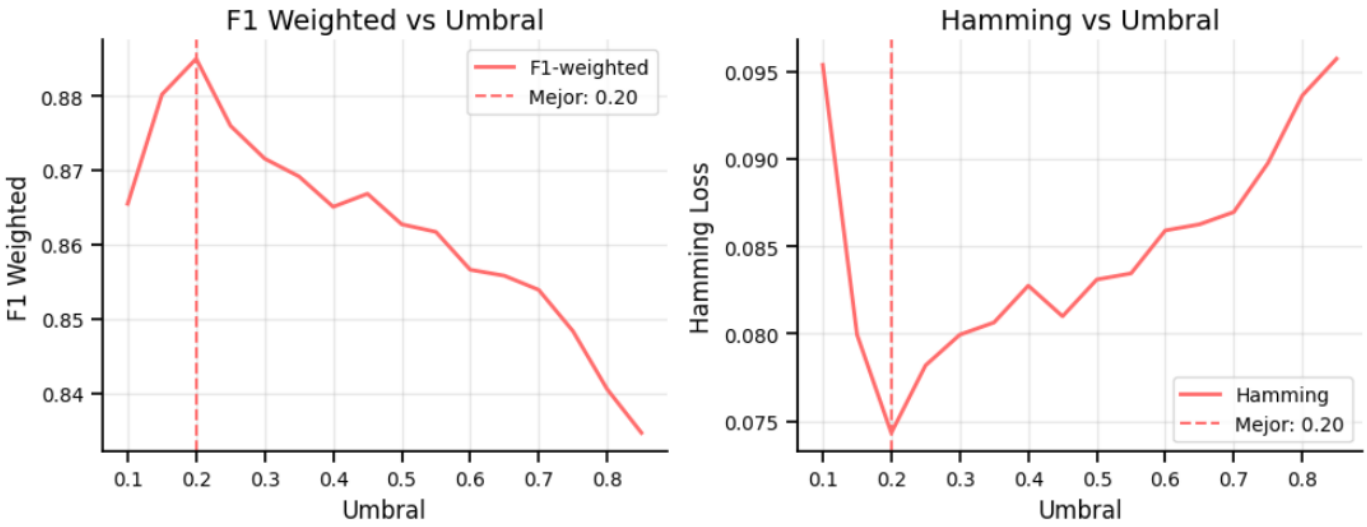

Figura 10: Variación en el F1 Weighted y Hamming Loss según el umbral utilizado en la función sigmoide.

Podemos ver como el F1 Score weighted varia según movemos el umbral, por lo que buscando maximizarlo vamos a variar también estos umbrales durante el entrenamiento del modelo para otbener aquel que maximize el F1 Score durante la validación, es decir, frente a datos que el modelo no utilizo para su entrenamiento. El modelo utilizado era el Improved Medical Bert con un dropout de 0.3 y se ejecuto durante 1 epoch.

Sin técnicas anti overfitting Ajustamos 3 modelos donde no usamos técnicas para atacar directamente al overfitting, únicamente variamos el dropout que es una técnica de regularización simple. En la siguiente tabla presentamos el resumen de los modelos:

Modelo	Descripción	Epochs	Dropout	Atención	Observaciones
modelo 1	Conservador	3	0.2	Sí	Menor regularización, puede sobreajustar un poco más rápido.
modelo 2	Más regularización	4	0.4	Sí	Más robusto contra overfitting, pero requiere más épocas.
modelo 3	Sin atención (ablation)	3	0.3	No	Sirve como baseline para analizar el impacto de la capa de atención.

Tabla 5: Configuraciones de los tres modelos entrenados

Los tres modelos exploran diferentes configuraciones: el primero es más conservador, con bajo dropout y riesgo de sobreajuste; el segundo incorpora mayor regularización y más épocas para mejorar la generalización; y el tercero elimina la atención como ablación, sirviendo de referencia para evaluar el impacto de esta capa en el rendimiento.

Tabla 6: Comparación de modelos entrenados

Modelo	F1 Weighted	Runtime (min: seg)	Epochs
Conservador (dropout=0.2, attn)	0.927	4:23	3
Más regularización (dropout=0.4, attn)	0.923	6:09	4
Sin atención (dropout=0.3)	0.925	4:17	3

Utilizamos los siguientes diagnósticos para detectar el overfitting:

- **Brechas significativas entre métricas de entrenamiento y validación:** El modelo obtiene mejores resultados en *train* (ej. F1 Weighted = 0.959) que en *val* (0.939), con diferencias mayores al 2 %.
- **Pérdida relativa en error:** El Hamming Loss pasa de 0.027 en *train* a 0.041 en *val*, lo que representa un deterioro cercano al 50 %.
- **Overfitting Score elevado:** La métrica global de comparación entre *train* y *val* alcanza un valor de 24.8, lo que confirma el sobreajuste.
- **Confianza desigual en predicciones:** Para ciertas clases (ej. *neurological*), los positivos en validación presentan menor proporción de alta confianza (¿0.8), comparado con entrenamiento.
- **Ejemplos mal clasificados con baja confianza:** Se identifican casos positivos en validación con predicciones muy bajas, lo que refleja falta de generalización.
- **Mayor sensibilidad en clases minoritarias:** Aunque algunas clases como *oncological* parecen estables, otras muestran caída notable de rendimiento y confianza en validación.

Métrica	Train	Validation	Diff (%)
F1 Weighted	0.9591	0.9388	+2.2 %
F1 Macro	0.9623	0.9428	+2.1 %
Hamming Loss	0.0272	0.0407	-49.7 %
Avg. Precision	0.9911	0.9713	+2.0 %

Tabla 7: Comparación Train vs Validation (Overfitting)

Observamos en la Tabla 7 que, si bien los valores de F1 Weighted, F1 Macro y Average Precision se mantienen relativamente estables entre entrenamiento y validación, la pérdida de Hamming prácticamente se duplica. Esto indica que, para garantizar que el modelo generalice de manera adecuada, es necesario incorporar técnicas anti-overfitting más especializadas durante el entrenamiento.
Es precisamente en este tipo de escenarios donde resulta útil la métrica propuesta de Overfitting Score, ya que, aunque en las métricas de F1 no se evidencian grandes diferencias, el incremento en el Hamming Loss hace que el score alcance valores entre 50 y 90 puntos, dependiendo de la

ejecución, lo cual pone de manifiesto señales claras de overfitting.

También vemos en la Tabla 8 como algunas de las clasificaciones el modelo parece dudar de ellas, es decir, pasan el umbral fijado pero muy en el borde, lo cual también es una señal de que debemos incluir más regularización durante el entrenamiento para que el modelo generalice mejor.

Dominio	**Positivos Alta Conf**	**Negativos Baja Conf.**	**Positivos Muy Baja Conf.**
Cardiovascular	92.9 %	97.4 %	12
Hepatorenal	91.7 %	98.4 %	12
Neurological	81.6 %	83.9 %	21
Oncological	84.2 %	99.7 %	0

Tabla 8: Análisis de confianza por dominio

Con técnicas anti overfitting Como técnicas anti overfitting utilizaremos

- **Regularización con Dropout**: se desactivan aleatoriamente neuronas durante el entrenamiento para evitar co-adaptación de los pesos.
- **Regularización con distintas configuraciones de modelo**: entrenamiento de variantes (ej. dropout alto/bajo, ablation) para controlar sensibilidad a hiperparámetros.
- **Ensembling**: combinación de predicciones de múltiples modelos para mejorar la generalización.

Siendo los modelos que entrenamos los siguientes, donde: **Ep.** = Número de *epochs*, **Drop** = Tasa de *dropout*, **Attn** = Uso de capa de atención, **LR** = *Learning rate*, **WD** = *Weight decay*, **GradAcc** = Pasos de *gradient accumulation*, **ES** = Paciencia del *early stopping*.

Modelo	**Ep.**	**Drop**	**Attn**	**LR**	**WD**	**GradAcc**	**ES**
Reg. Conservadora	3	0.20	Sí	2e−5	0.01	2	3
Reg. Moderada	4	0.30	Sí	1.5e−5	0.02	3	2
Reg. Agresiva	3	0.40	No	1e−5	0.03	4	1
Var. Ensemble	4	0.25	Sí	2.5e−5	0.015	2	2

Tabla 9: Configuraciones de modelos con distintas regularizaciones

Las configuraciones presentadas muestran distintos niveles de regularización. La variante conservadora prioriza estabilidad con bajo dropout y early stopping más laxo, la moderada equilibra regularización y número de épocas, la agresiva intensifica las restricciones (alto dropout, mayor weight decay, sin atención), y la variante ensemble adopta parámetros intermedios pensados para la combinación de modelos. Este diseño permite comparar cómo los diferentes grados de regularización afectan el entrenamiento y la generalización.

Modelo	F1-Weighted	Runtime (s)
Regularización Conservadora	0.9308	38.63
Regularización Moderada	0.9241	38.71
Regularización Agresiva	0.7763	32.29
Variante para Ensemble	**0.9437**	39.05

Tabla 10: Resumen de desempeño de los modelos (F1-weighted y Runtime)

Los valores de Hamming Loss también se reducen con este modelo, alcanzando aproximadamente 0.03, lo cual resulta mucho más consistente con lo observado en el conjunto de entrenamiento. Esto confirma que las técnicas anti-overfitting contribuyen de manera efectiva a disminuir dicho error, evidenciando una capacidad de generalización sólida. Asimismo, el Overfitting Score muestra una reducción significativa, situándose entre 2.2 y 3 como máximo según la ejecución, lo que refuerza la estabilidad del modelo frente a datos no vistos.
Por esta razón, decidimos quedarnos con el modelo basado en Ensemble, ya que es el que alcanza el mayor F1-Score Weighted sin presentar indicios adicionales de overfitting.

3.2. Resultados

A continuación encontramos la matriz de confusión y el reporte de clasificación para el modelo seleccionado

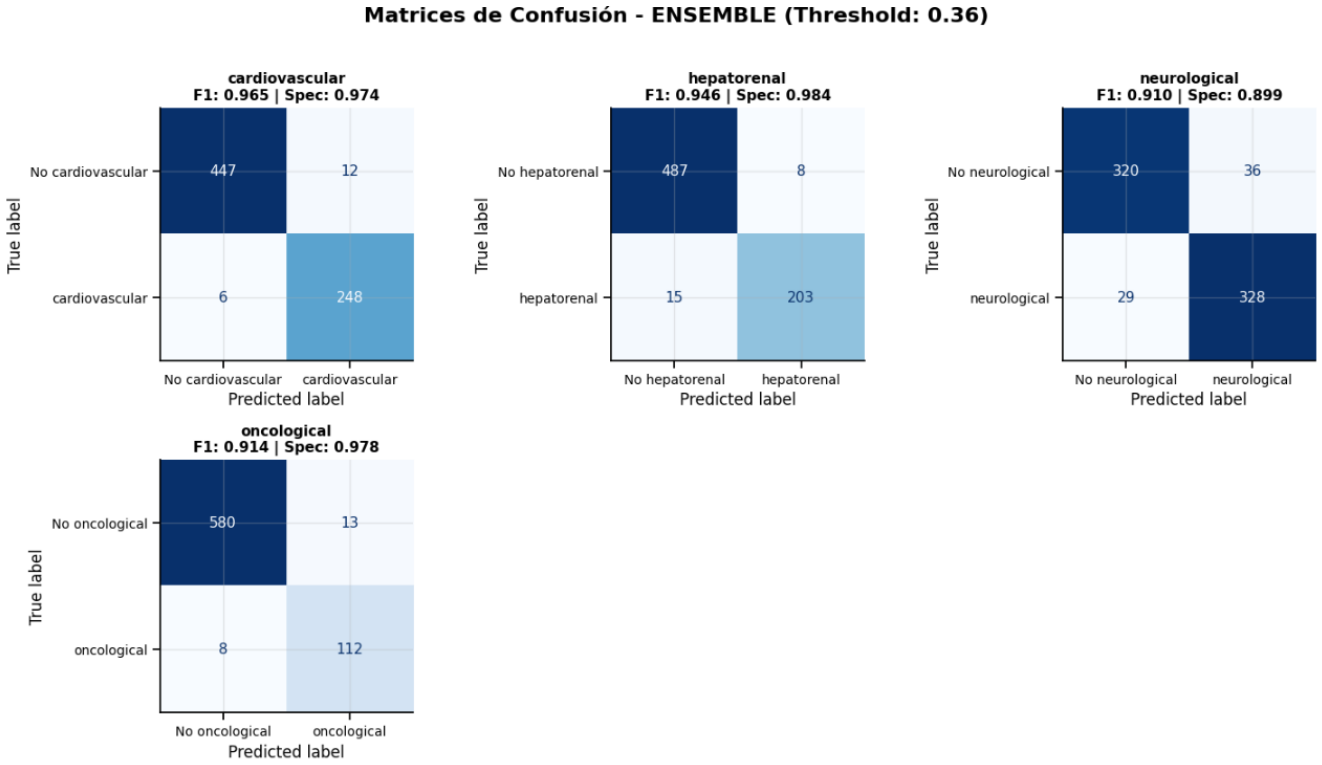

Figura 11: Matriz de confusión validación.

Vemos que el modelo comete en general pocos errores. En la mayoría de las clases (cardiovascular, hepatorenal y oncológica) los falsos negativos superan a los falsos positivos, lo que significa que el modelo tiende a ser más conservador y prefiere no asignar una etiqueta antes que arriesgarse a equivocarse. Esta es una estrategia razonable en un escenario donde es más costoso, en términos de tiempo, recomendar artículos irrelevantes que dejar pasar algunos relevantes.
La excepción es la clase neurológica, donde se observa el patrón contrario: el número de falsos positivos supera al de falsos negativos. Esto sugiere que el modelo tiende a sobrepredecir esta categoría, confundiendo con cierta frecuencia artículos que no pertenecen al dominio neurológico como si lo fueran. Este comportamiento era en parte esperado, ya que, a pesar de haber asignado pesos para equilibrar las clases, la configuración natural del conjunto de entrenamiento incluye una alta proporción de ejemplos neurológicos, lo que favorece este sesgo.

Clase	**Precisión**	**Recall**	**F1-Score**	**Especificidad**	**Soporte**
Cardiovascular	0.972	0.965	0.968	0.985	254
Hepatorenal	0.980	0.926	0.953	0.992	217
Neurological	0.914	0.919	0.916	0.913	358
Oncological	0.991	0.925	0.957	0.998	120

Tabla 11: Métricas detalladas por clase

Métrica	**F1-Score**
Macro	0.949
Micro	0.943
Weighted	0.939

Tabla 12: Promedios globales de desempeño

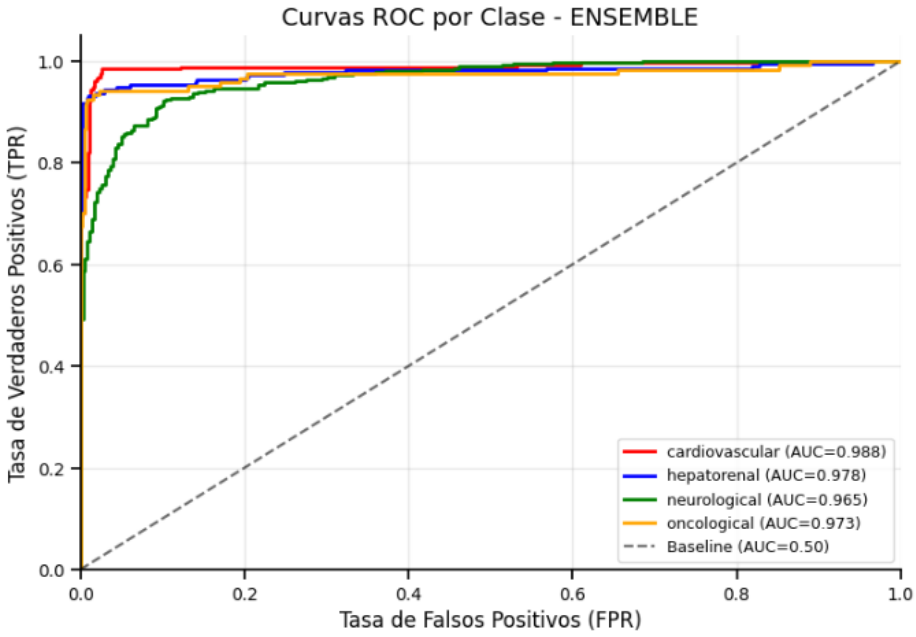

Figura 12: Curva ROC modelo seleccionado.

3.2.1. Prueba con datos nuevos

Ahora utilizaremos artículos médicos publicados en diferentes portales que tratan sobre una o varias de las 4 etiquetas posibles para ver el resultado ante datos nuevos que no hacían parte del entrenamiento ni de la validación. Todos los arttículos que usaremos serán artículos publicados en diferentes revistas académicas médicas

Título	Abstract (resumido)	Journal	Real	Predicción
Treatment of Small-Cell Lung Cancer: American Society of Clinical Oncology Endorsement of the American College of Chest Physicians Guideline	The American College of Chest Physicians (ACCP) produced an evidence-based guideline on treatment of patients with small-cell lung cancer (SCLC). Because of the relevance of this guideline to American Society of Clinical Oncology (ASCO) membership, ASCO reviewed the guideline, applying a set of procedures and policies used to critically examine guidelines developed by other organizations.	Journal of Clinical Oncology	Oncology	Oncology
Integrative Assessment of Congestion in Heart Failure Throughout the Patient Journey	Congestion is one of the main predictors of poor patient outcome in patients with heart failure. However, congestion is difficult to assess, especially when symptoms are mild.	JACC: Heart Failure	Cardiovascular	Cardiovascular y Hepatorenal
On Neurocardiology Updates: an Interdisciplinary Field at the Intersection of Neurology and Cardiology	This review examines the evolving field of neurocardiology, tracing its development from early homeostatic theories to modern understandings of the bidirectional neural networks that link the heart and brain. We propose an integrative framework to explain neuro-cardiac interactions in health and disease, with a focus on clinical applications and emerging therapies.	Current Neurology and Neuroscience Reports	Neurologia y Cardiología	Neurología y Cardiología
Comparative efficacy of pharmacological strategies for management of type 1 hepatorenal syndrome: a systematic review and network meta-analysis	Several drugs have been studied to improve outcomes for patients with hepatorenal syndrome, but trials have reported variable efficacy. We aimed to compare the efficacy of different management strategies for type 1 hepatorenal syndrome.	The lancet Gastroenterology	Hepatorenal	Hepatorenal

Tabla 13: Ejemplos de títulos y abstracts con su journal, clasificación real y predicción esperada del modelo.

Podemos observar que el modelo, al enfrentarse a datos nuevos, mantiene un rendimiento muy bueno. Solo cometió un error en 4 ejemplos, clasificando un artículo como perteneciente tanto a la categoría cardiovascular como a la hepatorenal, cuando en realidad correspondía únicamente a

la primera. Al revisar la literatura, se encuentra que esta confusión puede deberse a la relación entre ambas áreas, explicada por síndromes como el cardiorrenal o la congestión hepática en la insuficiencia cardíaca, entre otros. Esto sugiere un posible sesgo a corregir en futuras versiones del modelo. No obstante, el desempeño general sigue siendo muy sólido.

4. Aplicación V0

4.1. Arquitectura de la Aplicación

Se desarrolló el **frontend** de una aplicación web utilizando la herramienta *V0*, lo que permitió generar diseños estéticos a partir de mensajes en lenguaje natural y facilitó la elaboración de la demo. El **backend** se implementó con *FastAPI*, y el despliegue se realizó en *DigitalOcean* mediante *Docker*.

Frontend: Next.js 14 (App Router), TypeScript, Tailwind CSS, Shadcn/ui, Radix UI, Recharts, Lucide React, V0 (Vercel).

Backend: FastAPI (Python), Uvicorn, Pydantic, FastAPI-CORS, Pandas, NumPy, Scikit-learn.

Machine Learning: Transformers (Hugging Face), PyTorch/TensorFlow, AutoTokenizer, Tokenizers, clasificación multi-label con umbral optimizado.

Deployment Docker + Docker Compose, DigitalOcean Droplet

Desarrollo: Git, Visual Studio Code.

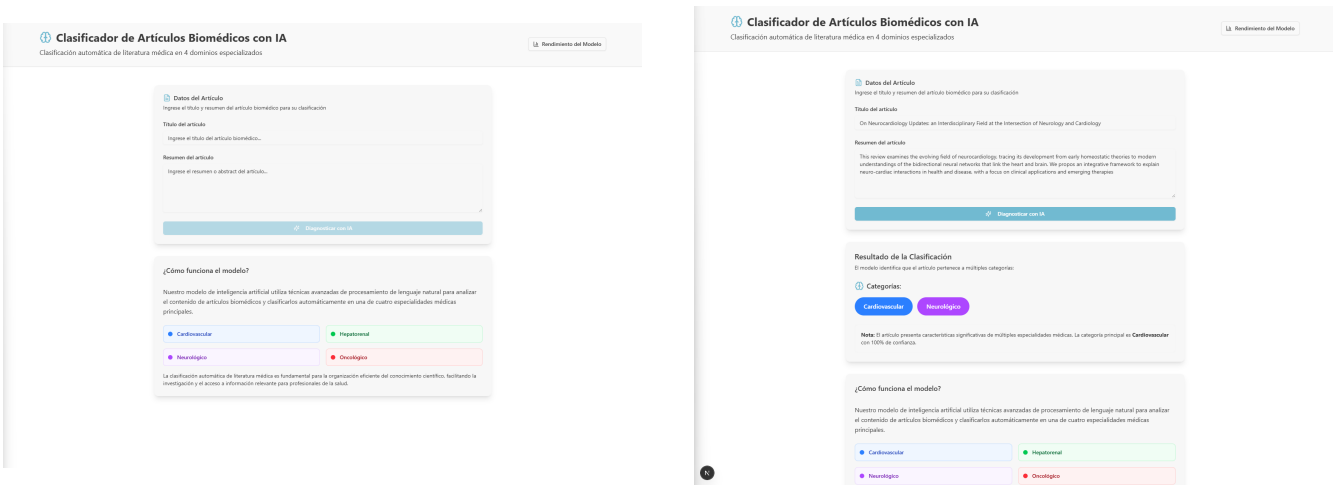

Figura 13: Demo creada utilizando V0.

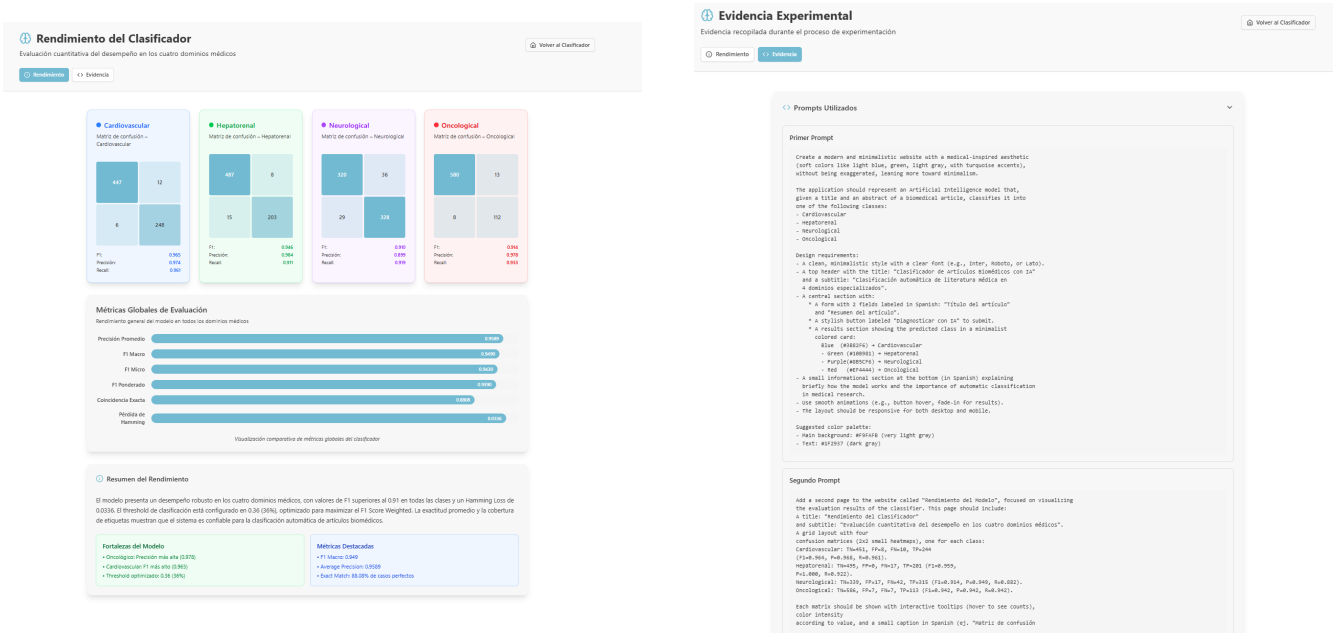

Figura 14: Demo creada utilizando V0.

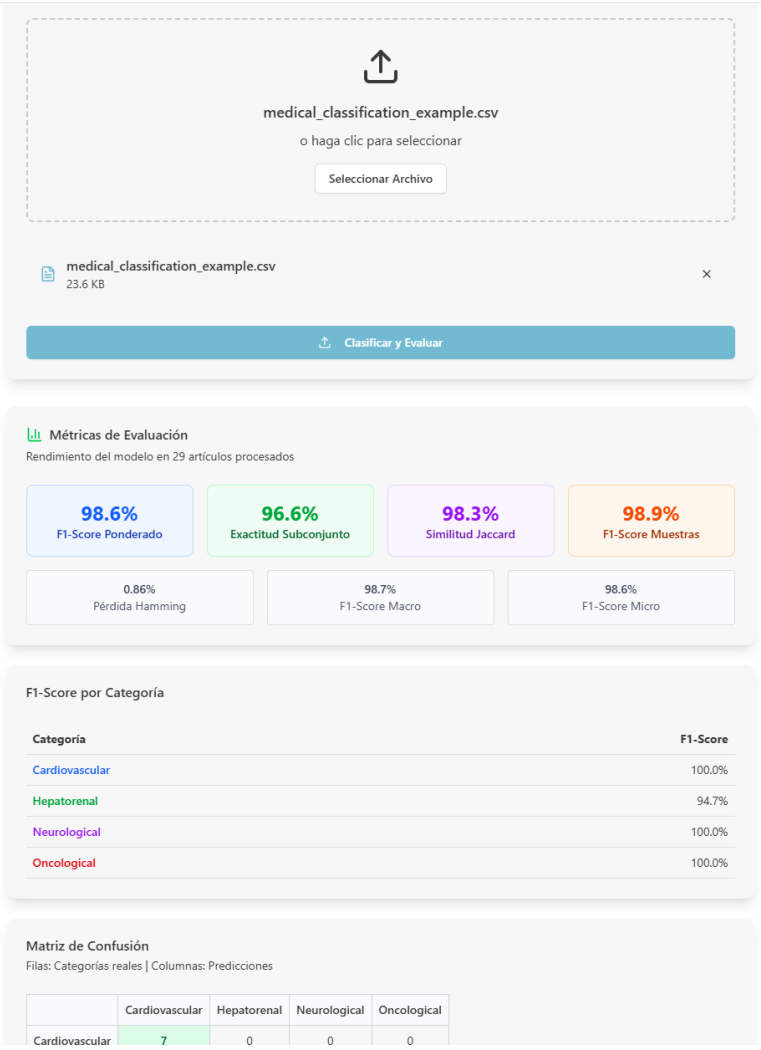

Figura 15: Demo creada utilizando V0.

5. Conclusión

El modelo propuesto, basado en un enfoque de ensambles, alcanza un F1-score ponderado (weighted) muy alto, lo que evidencia un excelente rendimiento tanto en la clasificación individual de las clases como en el escenario de clasificación multi-etiqueta. Además, supera ampliamente al modelo Dummy utilizado como referencia inicial, así como al clasificador aleatorio mostrado en la curva ROC.

Uno de los principales problemas identificados en el modelo es la tendencia a cometer falsos negativos, siendo este el tipo de error más frecuente. Esto refleja un comportamiento más conservador en la clasificación: el modelo tiende a evitar asignar etiquetas cuando no tiene suficiente confianza. En este caso, esta característica resulta deseable, ya que en un entorno en el que un usuario busca artículos de un campo específico, es preferible no recomendar documentos irrelevantes (evitar falsos positivos), aunque ello implique omitir algunos que sí son relevantes. Este criterio se justifica aún más en el escenario multi-etiqueta, pues incluso si una de las categorías correctas no es detectada,

otras etiquetas asociadas al mismo artículo pueden seguir representando los intereses del usuario. Asimismo, se evidenció que el manejo de la clasificación multi-etiqueta requiere un cuidado adicional en la conformación de los conjuntos de entrenamiento y validación. Es necesario garantizar una adecuada representatividad de todas las etiquetas, ya que las proporciones entre clases afectan directamente el desempeño. Tratar cada combinación posible de etiquetas como una clase independiente (16 en este caso) incrementaría significativamente la complejidad y la carga computacional. En contraste, la estrategia adoptada de manejar cada etiqueta de forma individual y luego permitir predicciones múltiples mostró ser más eficiente, alcanzando resultados competitivos sin disparar los requerimientos de cómputo ni los tiempos de entrenamiento.

A un nivel aplicado, la aplicación desarrollada tiene potenciales usos en diferentes ámbitos. Esta versión inicial propuesta en el campo educativo, por ejemplo, puede resultar especialmente útil en contextos de investigación, donde es común trabajar con grandes volúmenes de artículos científicos. El sistema permite agilizar la búsqueda y clasificación de papers, facilitando el acceso a los temas relevantes de manera más eficiente y reduciendo el tiempo invertido en la revisión manual de literatura.

En cuanto a limitaciones y trabajos futuros, se identificó que el modelo aún presenta margen de mejora en la clase neurológica, debido a que tiende a sobrepredecirla en ciertos casos. Aunque este comportamiento no representa un obstáculo crítico para el uso de la aplicación, sí señala la necesidad de contar con un conjunto de datos de entrenamiento más balanceado, lo cual podría mitigar este sesgo. Asimismo, una línea de evolución interesante sería la integración de un Large Language Model (LLM) que no solo clasifique los artículos, sino que también proporcione resúmenes automáticos e información clave, ampliando el valor práctico de la herramienta para los usuarios.

Referencias

[1] Miuru Abeysiriwardana and Deshan Sumanathilaka. A survey on lexical ambiguity detection and word sense disambiguation, 2024.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, Tianjiao Zhao, Amit Panalkar, Dhagash Mehta, Stefano Pasquali, Wei Cheng, Haoyu Wang, Yanchi Liu, Zhengzhang Chen, Haifeng Chen, Chris White, Quanquan Gu, Jian Pei, Carl Yang, and Liang Zhao. Domain specialization as the key to make large language models disruptive: A comprehensive survey, 2024.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Anexos y Evidencias

GitHub

Se adjunta el enlace al repositorio de GitHub:
GitHub Enlace

Diagramas

Adjuntamos dos diagramas, en la Figura 15 pipeline que muestra el entrenamiento del modelo y en la Figura 16 que muestra la arquitectura de la aplicación.

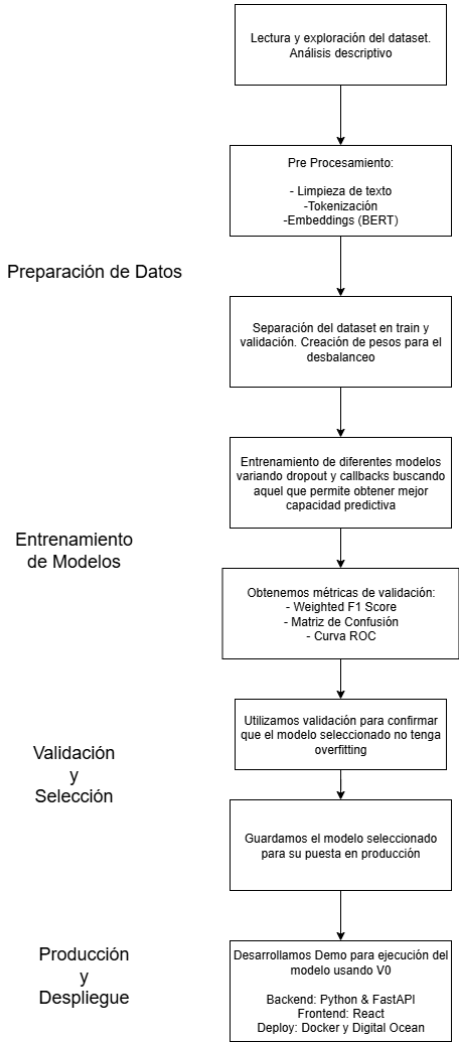

Figura 16: Diagrama pipeline entrenamiento del modelo.

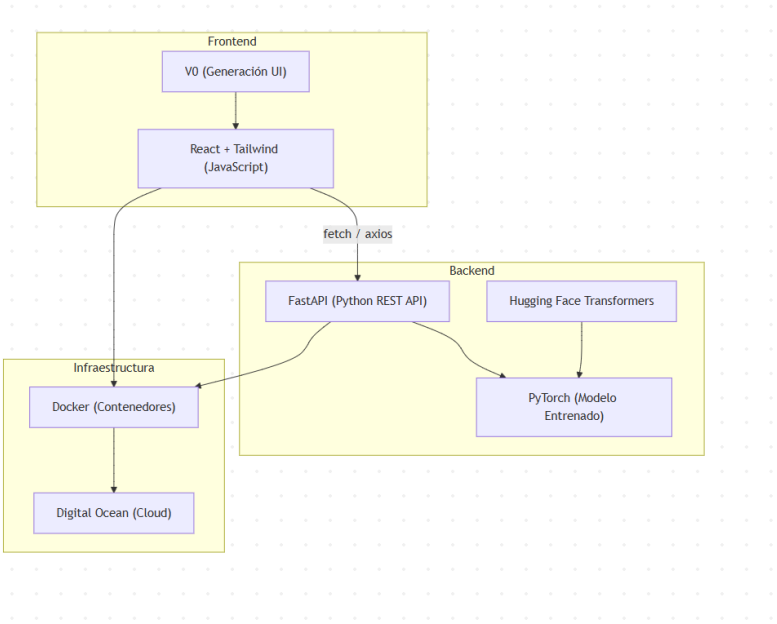

Figura 17: Diagrama arquitectura de la aplicación.

Prompts Utilizados

Los prompts utilizados en V0 fueron los siguientes:

Primer Prompt

```
Create a modern and minimalistic website with a medical-inspired aesthetic
(soft colors like light blue, green, light gray, with turquoise accents),
without being exaggerated, leaning more toward minimalism.

The application should represent an Artificial Intelligence model that,
given a title and an abstract of a biomedical article, classifies it into
one of the following classes:
- Cardiovascular
- Hepatorenal
- Neurological
- Oncological

Design requirements:
- A clean, minimalistic style with a clear font (e.g., Inter, Roboto, or Lato).
- A top header with the title: \Clasificador de Artículos Biomédicos con IA"
  and a subtitle: \Clasificación automática de literatura médica en
  4 dominios especializados".
- A central section with:
    * A form with 2 fields labeled in Spanish: \Título del artículo"
```

```
      and \Resumen del artículo".
    * A stylish button labeled \Diagnosticar con IA" to submit.
    * A results section showing the predicted class in a minimalist
      colored card:
        Blue  (#3B82F6) → Cardiovascular
        · Green (#10B981) → Hepatorenal
        · Purple(#8B5CF6) → Neurological
        · Red   (#EF4444) → Oncological
- A small informational section at the bottom (in Spanish) explaining
  briefly how the model works and the importance of automatic classification
  in medical research.
- Use smooth animations (e.g., button hover, fade-in for results).
- The layout should be responsive for both desktop and mobile.

Suggested color palette:
- Main background: #F9FAFB (very light gray)
- Text: #1F2937 (dark gray)
```

Segundo Prompt

```
Add a second page to the website called \Rendimiento del Modelo", focused on visualizing
the evaluation results of the classifier. This page should include:
A title: \Rendimiento del Clasificador"
and subtitle: \Evaluación cuantitativa del desempeño en los cuatro dominios médicos".
A grid layout with four
confusion matrices (2x2 small heatmaps), one for each class:
Cardiovascular: TN=451, FP=8, FN=10, TP=244
(F1=0.964, P=0.968, R=0.961).
Hepatorenal: TN=495, FP=0, FN=17, TP=201 (F1=0.959,
P=1.000, R=0.922).
Neurological: TN=339, FP=17, FN=42, TP=315 (F1=0.914, P=0.949, R=0.882).
Oncological: TN=586, FP=7, FN=7, TP=113 (F1=0.942, P=0.942, R=0.942).

Each matrix should be shown with interactive tooltips (hover to see counts),
color intensity
according to value, and a small caption in Spanish (ej. \Matriz de confusión

Cardiovascular").
A bar chart comparing global performance metrics:F1 Micro = 0.9430F1 Macro = 0.9463
F1 Weighted = 0.9428
Hamming Loss = 0.0372Average Precision = 0.9753Exact Match Ratio = 0.8668Each bar labeled c
At the bottom, add a small explanatory text box in Spanish summarizing:
\El modelo presenta un desempeño robusto en los cuatro dominios médicos, con valores
de F1 superiores al 0.91 en todas las clases y un Hamming Loss de 0.0372.
La exactitud promedio y la cobertura de etiquetas muestran que el sistema es confiable
```

para la clasificación automática de artículos biomédicos.
"Maintain the same minimalistic, medical-inspired style as the homepage (light backgrounds, turquoise accents, clean typography, smooth hover effects). Ensure the layout is responsive for desktop and mobile.

Tercer Prompt

In the section \Métricas Globales de Evaluación", instead of showing the values as progress bars or text, create a horizontal bar plot where each bar represents a global metric of the model.Metrics to include:F1 Micro = 0.9430F1 Macro = 0.9463F1 Weighted = 0.9428Hamming Loss = 0.0372Average
Precision = 0.9753Exact Match Ratio = 0.8668
Use one bar per metric, with the value shown at the end of the bar.
Order the bars from highest to lowest.
Add labels in Spanish on the left (ej. \F1 Micro", \F1 Macro", \Hamming Loss").
Style: minimalistic, clean, with turquoise accent color for the bars.
Add a short caption in Spanish below:
\Visualización comparativa de métricas globales del clasificador".
Ensure the bar plot is interactive (hover shows exact value) and responsive for desktop and mobile.

Capturas

Se adjuntan capturas de algunas fragmentos de código y su salida como evidencia del proceso realizado.

En la Figura 15 encontramos el código usado para verificar la integridad del dataset, es decir, que no haya datos repetidos ni vacíos, y uno de los gráficos descriptivos realizados.

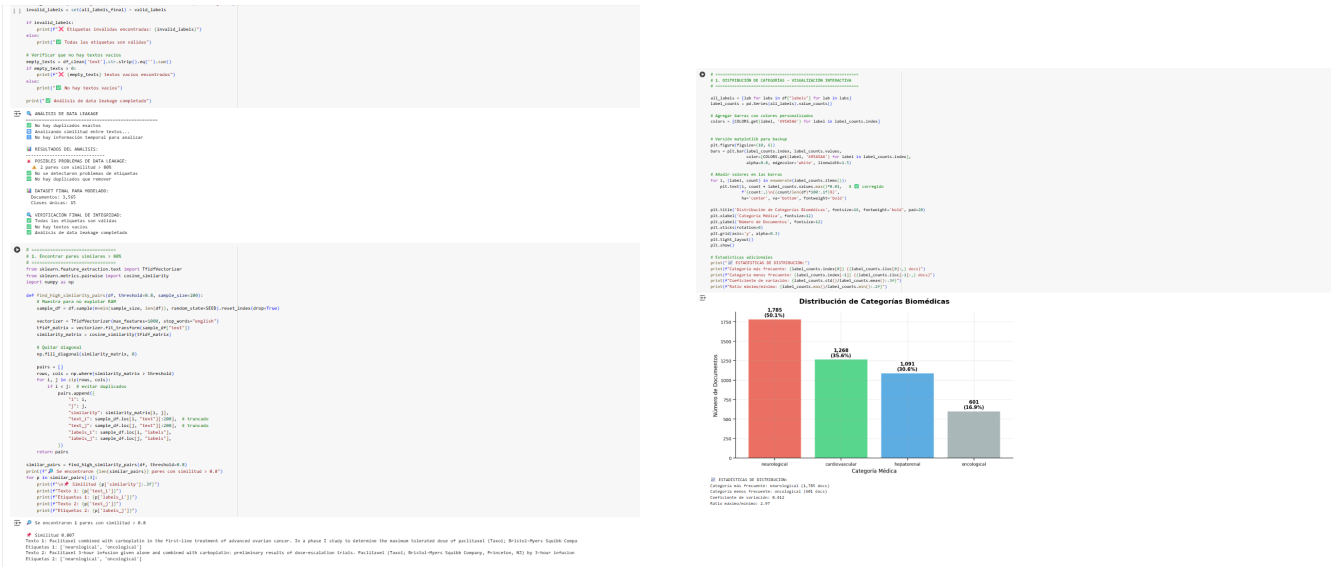

Figura 18: Verificación dataset sin repetidos ni vacíos y gráfico descriptivo.

En la Figura 15 encontramos los resultados del modelo Dummy creado y los resultados de la calibración del F1 Score Weighted según el umbral.

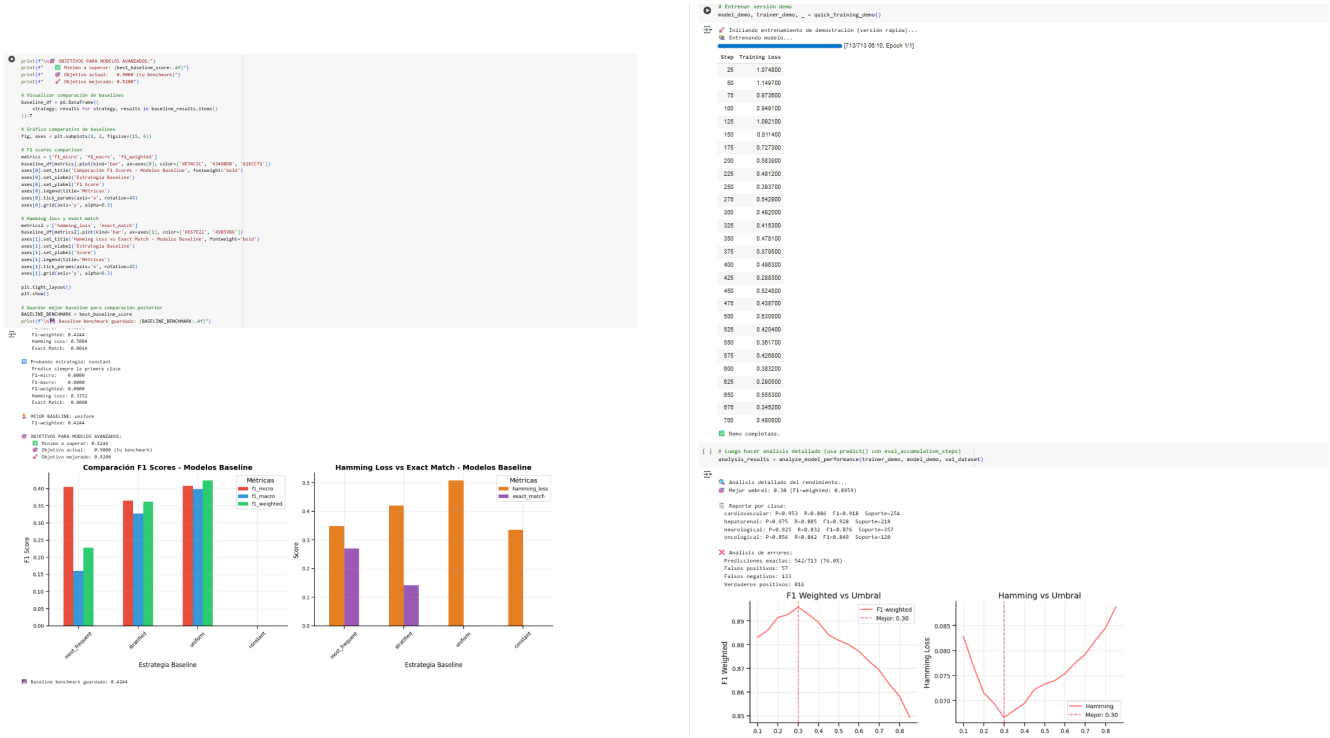

Figura 19: Creación modelo Dummy y Calibración del F1 Score Weighted.

En la Figura 16 encontramos la matriz de confusión del modelo sin técnicas anti overfitting y el entrenamiento de uno de los modelos que si incorporan dichas técnicas, específicamente el modelo con una regularización conservadora

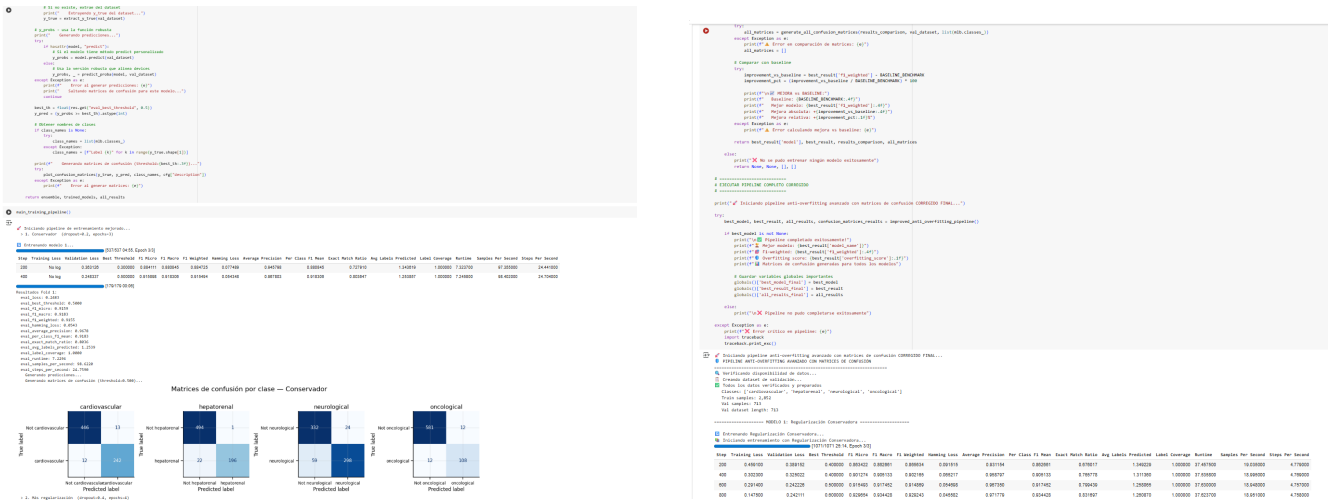

Figura 20: Matriz de confusión y entrenamiento de modelos diferentes.

Código

A continuación se encuentran algunos códigos utilizados para la limpieza de datos, creación del modelo, métricas, etc.

Pipeline: Parte del pipeline encargado de la limpieza de datos

```
"""
Data processing utilities for medical text classification.

This module contains functions for loading, cleaning, and preprocessing
medical text data for machine learning tasks.
"""

import pandas as pd
import numpy as np
import re
from typing import List, Dict, Tuple, Optional, Any
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
import logging

logger = logging.getLogger(__name__)


def load_medical_data(file_path: str, encoding: str = "utf-8") -> pd.DataFrame:
    """
    Load medical dataset from CSV file.

    Args:
        file_path: Path to the CSV file
        encoding: File encoding

    Returns:
        Loaded DataFrame
    """
    try:
        df = pd.read_csv(file_path, encoding=encoding)
        logger.info(f"Successfully loaded {len(df)} records from {file_path}")
        return df
    except UnicodeDecodeError:
        logger.warning("UTF-8 decoding failed, trying latin-1")
        df = pd.read_csv(file_path, encoding="latin-1")
        logger.info(f"Successfully loaded {len(df)} records with latin-1 encoding")
        return df
    except Exception as e:
```

```
        logger.error(f"Error loading file {file_path}: {e}")
        raise


def clean_medical_text(
    df: pd.DataFrame,
    text_columns: List[str] = ["title", "abstract"],
    combine_columns: bool = True,
    combined_column: str = "text",
) -> pd.DataFrame:
    """
    Clean and preprocess medical text data.

    Args:
        df: Input DataFrame
        text_columns: Columns containing text to clean
        combine_columns: Whether to combine text columns
        combined_column: Name of combined text column

    Returns:
        DataFrame with cleaned text
    """
    df_clean = df.copy()

    for col in text_columns:
        if col in df_clean.columns:
            # Fill missing values
            df_clean[col] = df_clean[col].fillna("")

            # Convert to string
            df_clean[col] = df_clean[col].astype(str)

            # Clean text
            df_clean[col] = df_clean[col].apply(clean_text)

    # Combine text columns if requested
    if combine_columns and len(text_columns) > 1:
        existing_cols = [col for col in text_columns if col in df_clean.columns]
        if existing_cols:
            df_clean[combined_column] = df_clean[existing_cols].apply(
                lambda x: ". ".join(filter(None, x.astype(str))), axis=1
            )

    logger.info(f"Cleaned text in columns: {text_columns}")
    return df_clean
```

```
def clean_text(text: str) -> str:
    """
    Clean individual text string.

    Args:
        text: Input text

    Returns:
        Cleaned text
    """
    if pd.isna(text) or text == "":
        return ""

    # Convert to string
    text = str(text)

    # Remove extra whitespace
    text = re.sub(r"\s+", " ", text)

    # Remove special characters but keep medical abbreviations
    text = re.sub(r"[^\w\s\-\.\,\:\;\(\)\[\]]", " ", text)

    # Fix common OCR errors
    text = text.replace("~", "-")
    text = text.replace("|", "I")

    # Normalize whitespace
    text = text.strip()

    return text


def extract_medical_features(
    df: pd.DataFrame, text_column: str = "text"
) -> pd.DataFrame:
    """
    Extract medical-specific features from text.

    Args:
        df: Input DataFrame
        text_column: Column containing text

    Returns:
```

```
    DataFrame with additional feature columns
"""
df_features = df.copy()

# Text length features
df_features["text_length"] = df_features[text_column].str.len()
df_features["word_count"] = df_features[text_column].str.split().str.len()

# Medical term counts
medical_patterns = {
    "medication_count": r"\b(?:drug|medication|treatment|therapy|medicine)\b",
    "symptom_count": r"\b(?:pain|symptom|discomfort|ache|fever|nausea)\b",
    "anatomy_count": r"\b(?:heart|brain|liver|kidney|lung|blood|cell)\b",
    "procedure_count": r"\b(?:surgery|operation|procedure|examination|test)\b",
    "diagnosis_count": r"\b(?:diagnosis|condition|disease|disorder|syndrome)\b",
}

for feature_name, pattern in medical_patterns.items():
    df_features[feature_name] = df_features[text_column].str.count(
        pattern, flags=re.IGNORECASE
    )

# Numerical values (dosages, measurements)
df_features["numeric_count"] = df_features[text_column].str.count(r"\b\d+\.?\d*\b")

# Abbreviation count
df_features["abbreviation_count"] = df_features[text_column].str.count(
    r"\b[A-Z]{2,}\b"
)

logger.info(f"Extracted medical features for {len(df_features)} records")
return df_features
```

Clase ImprovedMedicalBERT: Clase que toma el Tarsnformer BiomedBERT y mejora la capacidad de captura semántica.

```
class ImprovedMedicalBERT(nn.Module):
"""
Improved BERT model for medical text classification.

Features:
- Attention-based pooling
- Multiple dropout layers
- Residual connections
- Positive class weighting
```

```
"""

def __init__(
    self,
    model_name: str = "microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract",
    num_labels: int = 4,
    pos_weight: Optional[List[float]] = None,
    dropout_rate: float = 0.3,
    use_attn: bool = True,
):
    """
    Initialize the model.

    Args:
        model_name: Name of the pre-trained model
        num_labels: Number of classification labels
        pos_weight: Positive class weights for imbalanced data
        dropout_rate: Dropout probability
        use_attn: Whether to use attention pooling
    """
    super(ImprovedMedicalBERT, self).__init__()

    self.num_labels = num_labels
    self.use_attn = use_attn

    # Load pre-trained model
    self.bert = AutoModel.from_pretrained(model_name)
    self.config = self.bert.config
    hidden_size = self.config.hidden_size

    # Pooling layer
    if use_attn:
        self.pooler = AttentionPooling(hidden_size)
    else:
        self.pooler = None

    # Classification layers
    self.dropout1 = nn.Dropout(dropout_rate)
    self.classifier = nn.Linear(hidden_size, num_labels)
    self.dropout2 = nn.Dropout(dropout_rate * 0.5)

    # Loss function with class weighting
    if pos_weight is not None:
        pos_weight_tensor = torch.tensor(pos_weight, dtype=torch.float)
        self.loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight_tensor)
```

```
    else:
        self.loss_fn = nn.BCEWithLogitsLoss()

    # Initialize classifier weights
    self._init_weights()

def _init_weights(self):
    """Initialize classifier weights."""
    nn.init.xavier_uniform_(self.classifier.weight)
    nn.init.zeros_(self.classifier.bias)

def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    labels: Optional[torch.Tensor] = None,
) -> Dict[str, torch.Tensor]:
    """
    Forward pass.

    Args:
        input_ids: Input token IDs
        attention_mask: Attention mask
        labels: Target labels (optional)

    Returns:
        Dictionary containing logits and loss (if labels provided)
    """
    # Get BERT outputs
    outputs = self.bert(
        input_ids=input_ids, attention_mask=attention_mask, return_dict=True
    )

    # Pool representations
    if self.use_attn and self.pooler is not None:
        pooled_output = self.pooler(outputs.last_hidden_state, attention_mask)
    else:
        pooled_output = outputs.pooler_output

    # Apply dropout and classification
    pooled_output = self.dropout1(pooled_output)
    logits = self.classifier(pooled_output)
    logits = self.dropout2(logits)

    result = {"logits": logits}
```

```
        # Calculate loss if labels provided
        if labels is not None:
            loss = self.loss_fn(logits, labels)
            result["loss"] = loss

        return result
```

Métricas y Threshold: Código donde se calculan las métricas y el threshold óptima para el problema multi-label.

```
    """
Evaluation utilities for biomedical text classification.

This module contains comprehensive evaluation functions and metrics
for multilabel medical text classification.
"""

import numpy as np
import matplotlib.pyplot as plt
from typing import Dict, List, Tuple, Optional, Any
from sklearn.metrics import (
    f1_score,
    precision_score,
    recall_score,
    hamming_loss,
    average_precision_score,
    roc_auc_score,
    multilabel_confusion_matrix,
    ConfusionMatrixDisplay,
)


def compute_multilabel_metrics(
    y_true: np.ndarray,
    y_pred: np.ndarray,
    y_probs: Optional[np.ndarray] = None,
    class_names: Optional[List[str]] = None,
    average_types: List[str] = ["micro", "macro", "weighted"],
) -> Dict[str, Any]:
    """
    Compute comprehensive multilabel classification metrics.

    Args:
        y_true: Ground truth binary labels
```

```
    y_pred: Predicted binary labels
    y_probs: Predicted probabilities (optional)
    class_names: Names of classes
    average_types: Types of averaging for metrics

Returns:
    Dictionary containing all computed metrics
"""
if class_names is None:
    class_names = [f"Class_{i}" for i in range(y_true.shape[1])]

metrics = {}

# Basic multilabel metrics
for avg in average_types:
    metrics[f"f1_{avg}"] = f1_score(y_true, y_pred, average=avg, zero_division=0)
    metrics[f"precision_{avg}"] = precision_score(
        y_true, y_pred, average=avg, zero_division=0
    )
    metrics[f"recall_{avg}"] = recall_score(
        y_true, y_pred, average=avg, zero_division=0
    )

# Hamming loss (multilabel-specific)
metrics["hamming_loss"] = hamming_loss(y_true, y_pred)

# Exact match ratio (all labels correct)
metrics["exact_match_ratio"] = (y_true == y_pred).all(axis=1).mean()

# Label-based metrics
metrics["avg_labels_true"] = y_true.sum(axis=1).mean()
metrics["avg_labels_pred"] = y_pred.sum(axis=1).mean()
metrics["label_coverage"] = (y_pred.sum(axis=0) > 0).mean()

# Per-class metrics
per_class_f1 = f1_score(y_true, y_pred, average=None, zero_division=0)
per_class_precision = precision_score(y_true, y_pred, average=None, zero_division=0)
per_class_recall = recall_score(y_true, y_pred, average=None, zero_division=0)

metrics["per_class_metrics"] = {}
for i, class_name in enumerate(class_names):
    metrics["per_class_metrics"][class_name] = {
        "f1": per_class_f1[i],
        "precision": per_class_precision[i],
        "recall": per_class_recall[i],
```

```
            "support": y_true[:, i].sum(),
        }

    # Probability-based metrics (if probabilities provided)
    if y_probs is not None:
        try:
            metrics["average_precision_macro"] = average_precision_score(
                y_true, y_probs, average="macro"
            )
            metrics["average_precision_micro"] = average_precision_score(
                y_true, y_probs, average="micro"
            )
        except Exception:
            metrics["average_precision_macro"] = 0.0
            metrics["average_precision_micro"] = 0.0

        try:
            metrics["roc_auc_macro"] = roc_auc_score(y_true, y_probs, average="macro")
            metrics["roc_auc_micro"] = roc_auc_score(y_true, y_probs, average="micro")
        except Exception:
            metrics["roc_auc_macro"] = 0.0
            metrics["roc_auc_micro"] = 0.0

    return metrics


def find_optimal_thresholds(
    y_true: np.ndarray,
    y_probs: np.ndarray,
    metric: str = "f1_macro",
    threshold_range: Tuple[float, float] = (0.1, 0.9),
    step: float = 0.05,
) -> Dict[str, Any]:
    """
    Find optimal classification thresholds for multilabel classification.

    Args:
        y_true: Ground truth binary labels
        y_probs: Predicted probabilities
        metric: Metric to optimize
        threshold_range: Range of thresholds to test
        step: Step size for threshold search

    Returns:
        Dictionary with optimal thresholds and metrics
```

```
"""
thresholds = np.arange(threshold_range[0], threshold_range[1] + step, step)

best_threshold = 0.5
best_score = -1.0
threshold_results = []

for threshold in thresholds:
    y_pred = (y_probs >= threshold).astype(int)

    # Calculate target metric
    if metric == "f1_macro":
        score = f1_score(y_true, y_pred, average="macro", zero_division=0)
    elif metric == "f1_weighted":
        score = f1_score(y_true, y_pred, average="weighted", zero_division=0)
    elif metric == "f1_micro":
        score = f1_score(y_true, y_pred, average="micro", zero_division=0)
    elif metric == "hamming_loss":
        score = -hamming_loss(y_true, y_pred)  # Negative because lower is better
    else:
        raise ValueError(f"Unsupported metric: {metric}")

    threshold_results.append(
        {
            "threshold": threshold,
            "score": score,
            "f1_macro": f1_score(y_true, y_pred, average="macro", zero_division=0),
            "f1_weighted": f1_score(
                y_true, y_pred, average="weighted", zero_division=0
            ),
            "hamming_loss": hamming_loss(y_true, y_pred),
        }
    )

    if score > best_score:
        best_score = score
        best_threshold = threshold

return {
    "best_threshold": best_threshold,
    "best_score": best_score,
    "all_results": threshold_results,
}
```