UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Estudio de tecnologias de IA para extracción de información de documentos textuales

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Ferrandis Morant, Pascual

Tutor/a: Albiol Colomer, Alberto

CURSO ACADÉMICO: 2022/2023

Resumen

Este trabajo de fin de grado busca realizar un estudio de las diferentes tecnologías que existen actualmente para la creación de un modelo de *machine learning* para la extracción de información de documentos de texto.

Para ello se analizarán las diferentes formas de implementar estos modelos de preguntas y respuestas, que pueden ser extractivos o abstractivos, para definir cuales son los más óptimos para la tarea. De esta manera, se tratarán cuáles son los recursos necesarios para la creación del modelo, incluyendo la importancia de disponer de un ordenador con un GPU potente o la necesidad de contar con una buena base de datos, así como la manera de crearla.

También se desarrollará el proceso a seguir para la creación del modelo, estudiando los grados de libertad disponibles y los factores que más influyen en su rendimiento óptimo, así como las métricas que se pueden utilizar para evaluarlo.

Para concretar el análisis, y sirviendo como caso práctico, se aplicarán las tecnologías desarrolladas para la extracción de información en una base de datos de reportes radiológicos en inglés.

Resum

Aquest treball de final de grau busca dur a terme un estudi de les diferents tecnologies que existeixen actualment per a la creació d'un model de aprenentatge automàtic per a l'extracció d'informació de documents de text.

Per això s'analitzaran les diferents formes d'implementar aquests models de preguntes i respostes, que poden ser extractius o abstractius, per definir quins són els òptims per a la tasca. D'aquesta manera, es tractaran quins són els recursos necessaris per a la creació del model, incloent-hi la importància de disposar d'un ordinador amb una GPU potent o la necessitat de comptar amb una bona base de dades, així com la manera de crear-la.

També es desenvoluparà el procés a seguir per a la creació del model, estudiant els graus de llibertat disponibles i els factors que més influeixen en el seu rendiment òptim, així com les mètriques que es poden utilitzar per avaluar-lo.

Per concretar l'anàlisi, i servint com a cas pràctic, s'aplicaran les tecnologies desenvolupades per a l'extracció d'informació en una base de dades d'informes radiològics en anglés.

Abstract

This final degree work aims to study the different technologies that currently exist for creating a machine-learning model for extracting information from text documents.

For this purpose, the different ways of implementing these models of questions and answers, which can be extractive or abstractive, will be analyzed to define the most optimal for the task. In this way, the necessary resources for creating the model will be discussed, including the importance of having a computer with a powerful GPU or the need to have a good database and the way to create it.

The process to be followed for creating the model will also be developed, studying the available degrees of freedom, the factors that most influence its optimal performance, and the metrics that can be used to evaluate it.

The technologies developed for extracting information will be applied to a database of radiological reports in English to make the analysis more concrete and act as a practical case.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

1.1 Motivación

En los últimos años, la inteligencia artificial ha dejado de ser un concepto exclusivo para personas con habilidades técnicas y se ha convertido en un tema de conversación recurrente entre toda la población. Esta explosión de la inteligencia artificial, en gran parte detonada por la irrupción de *Chat GPT* en el mercado, está definiendo numerosos proyectos dentro de las grandes empresas. Todas quieren ser las primeras en implementarla para optimizar sus servicios, reducir costes y adquirir una ventaja competitiva.

Durante el transcurso de la carrera se han tratado en diferentes asignaturas conceptos relacionado con la inteligencia artificial, como la identificación de imágenes y audios para su clasificación mediante el uso de redes neuronales. Una rama de la inteligencia que no se trató, y por tanto desconocía, fue la encargada de trabajar con texto natural, es decir, con las propias palabras. Este Trabajo de Fin de Grado nace de ahí, de la curiosidad de querer aprender cómo funcionan modelos como Chat GPT y que otros tipos de usos de les pueden dar.

Paralelamente a la concepción del trabajo surgió una oportunidad de trabajar con una empresa que quería implementar una inteligencia artificial que, dado un informe, se le pudiera realizar diferentes preguntas y esta indicara en que parte del texto se encontraba la respuesta agilizando de esta forma el trabajo de los empleados. Esta oportunidad al final no fue posible por parte de la empresa, pero se decidió utilizar toda la investigación realizada alrededor de la extracción de información con el uso de la inteligencia artificial para realizar un estudio genérico de estas técnicas y después aplicarlas a un caso concreto.

Para esta implementación concreta se va a usar el *dataset* RadQA. Este contiene diferentes reportes radiológicos a los que se le realizarán preguntas sobre el paciente y se indicará en que parte exacta del reporte se encuentra la respuesta. Una implementación real de este tipo agilizaría el trabajo de todo el personal sanitario que trabaje con este tipo de informes.

1.2 Objetivos

El objetivo principal de este trabajo es llevar a cabo un estudio de las diferentes opciones y algoritmos disponibles para la extracción de información de documentos de texto. Esta recogida de la información, cómo se explicará más adelante, siempre será extractiva; es decir, la inteligencia artificial devolverá el fragmento del documento introducido dónde se encuentre la respuesta y no parafraseará ni generará otro tipo de respuestas.

Para poder caracterizar la investigación, se utilizará un data set compuesto por informes radiológicos con preguntas asociados a estos. Para ello se examinarán y compararán diferentes técnicas de procesamiento del lenguaje natural, reconocimiento de patrones y aprendizaje automático, con el fin de determinar cuáles son las más efectivas para extraer datos relevantes de estos informes radiológicos.

Los resultados obtenidos en este estudio servirán como guía para la implantación y desarrollo de herramientas que contribuyan a automatizar y mejorar la eficiencia de cualquier empresa que pueda necesitar de esto servicios: un hospital, un bufete de abogados, una biblioteca, etc.

Capítulo 2. Marco teórico

2.1 Inteligencia artificial y *machine learning*

El término de inteligencia artificial, IA en adelante, fue acuñado por John McCarthy [1]. Este la define como la ciencia e ingeniería encargada de hacer máquinas inteligentes, especialmente programas inteligentes. Lo que se pretende con esto es emular la inteligencia humana buscando la capacidad de crear programas que además de emular comportamientos adquiridos sean capaces de aprender, razonar, percibir, planificar y tomar decisiones de manera autónoma igual, o mejor de lo que podría hacerlo un humano.

La IA ha sufrido una evolución exponencial en los últimos años, aumentando considerablemente su capacidad y sofisticación. Muchos son los factores que se pueden considerar claves en este abrupto avance, como el aumento de la capacidad hardware gracias a los GPU, el aumento masivo del tamaño de las bases de datos utilizadas y la investigación colaborativa entre entidades públicas y privadas llevada a cabo en el sector.

El aprendizaje automático, o *machine learning* en inglés, es la rama dentro de la IA encargada de encontrar la manera de dotar a las máquinas de esta capacidad de aprendizaje, para que vayan más allá de la simple imitación de comportamientos inteligentes [2]. Este aprendizaje puede dividirse en tres grupos diferentes: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo [3].

En el **supervisado**, los algoritmos se entrenan utilizando conjuntos de datos etiquetados, donde se proporcionan ejemplos de entrada junto con las respuestas deseadas. Este enfoque permite que el modelo comprenda las relaciones entre los datos y las etiquetas, lo que le permite hacer predicciones precisas en nuevos datos no etiquetados.

Por otro lado, el aprendizaje **no supervisado** implica explorar patrones en conjuntos de datos no etiquetados, sin la guía de respuestas predefinidas. Los algoritmos en este grupo buscan encontrar estructuras y relaciones ocultas en los datos, como agrupaciones y dimensiones latentes.

Por último, el aprendizaje **por refuerzo** se basa en la interacción del modelo con un entorno dinámico. El modelo toma decisiones y recibe retroalimentación en forma de recompensas o castigos, lo que le permite aprender a tomar acciones que maximicen una recompensa acumulativa a lo largo del tiempo.

El *machine learning* representa por tanto un pilar esencial dentro de la IA. Existen diferentes técnicas a utilizar para crear los algoritmos, tales como los árboles de decisión, modelos de regresión, modelos de clasificación y técnicas de clusterización, entre otras. Sin duda, las redes neuronales son una de las técnicas que más destacan debido a la popularidad alcanzada por su

capacidad para aprender y adaptarse a patrones complejos en los datos, inspirándose en la estructura y funcionamiento del cerebro humano [4].

Es por esto por lo que las redes neuronales aprenden de una forma jerarquizada, es decir, la información se aprende por niveles. En los primeros niveles, o capas, se aprenden conceptos más concretos mientras que en las capas posteriores se utiliza esta información para sintetizar conceptos más abstractos. En general, una mayor cantidad de capas se traduce en conclusiones más abstractas y elaboradas. Por esta razón la tendencia actual es la de que los algoritmos cada vez tengan más capas.

Estos algoritmos más complejos reciben el nombre de *deep learning*, o aprendizaje profundo en español. La complejidad y potencia de estos algoritmos los convierten en un elemento clave para realizar análisis de conjuntos de datos masivos. Dentro del aprendizaje profundo se engloban diferentes familias de algoritmos adaptadas a los distintos tipos de datos existentes.

Entre estas familias se encuentran las redes neuronales convolucionales (CNN, por sus siglas en inglés), ideadas para entender datos espaciales como imágenes; las redes neuronales recurrentes (RNN), creadas para analizar datos secuenciales como textos o audios; o la arquitectura de los *Transformers*, la cual se ha utilizado para la creación de este trabajo.

2.2 Transformers

Los *Transformers* son una arquitectura para modelos de deep learning, que se ha popularizado gracias a su alto rendimiento con tareas de procesamiento de lenguaje natural. Los diferentes modelos utilizados en este trabajo tienen como base esta arquitectura, o parte de ella. Este tipo especifico de red neuronal fue introducido en 2017 por Google Brain, como mejora a las redes neuronales recurrentes [5].

Inicialmente se ideó para ser utilizada en la traducción de textos, pero acabó siendo aplicada en otros campos del procesamiento de secuencias textuales, debido a que se comprobó que realizaba traducciones de mayor calidad necesitando un menor tiempo de entrenamiento. En la actualidad, los *Transformers* son utilizados también para otros tipos de tareas: OpenAI los utiliza a través de GPT3 para el funcionamiento de su chatbot, Tesla lo usa en su sistema de conducción automático mientras que AlphaFold 2 lo utiliza para analizar secuencias de genomas humanos.

Los *Transformers* fueron creados para solventar los problemas que existían con la transducción de secuencias. Esto hace referencia a todas las tareas que transforman una secuencia de entrada (input) en una secuencia de salida (output). Esto incluye el reconocimiento de voz para

generar subtítulos o transcripciones, la transformación de texto a voz o la capacidad de resumir textos automáticamente.

Para realizar esta transducción de secuencias, que en nuestro caso serán textos, los modelos necesitan hacer uso de algún tipo de memoria. Un ejemplo dónde se puede observar esto, sería en la traducción del siguiente texto a otro idioma: “El Padrino es una película de 1972. Esta película gano tres óscares de los diez a los que estaba nominada.”

En este caso concreto, la secuencia “Esta película” de la segunda frase hace referencia a “El Padrino” mencionado en la primera oración. Cuando las personas leemos la segunda frase, sabemos a qué película se está refiriendo a la perfección. Esta información podría ser importante en la traducción por lo que el modelo de *deep learning* a utilizar también debería tener en cuenta estas dependencias y conexiones dentro de la propia secuencia.

Lo que ha popularizado tanto a los *Transformers* es que combina las características claves de las redes neuronales recurrentes y convolucionales y suple sus limitaciones de forma novedosa. Es por esto, que, para entender el funcionamiento de esta arquitectura, hay que comprender antes algunos conceptos claves de las RNN y CNN.

Las redes neuronales recurrentes están diseñadas para trabajar con datos secuenciales en los que el orden y la relación entre elementos son importantes. La principal característica de estas es, que como indica su nombre, tiene conexiones recurrentes. Esto se traduce en que las salidas de las neuronas en una capa sirven como entradas para otras neuronas en la siguiente iteración, almacenando de esta forma información sobre los estados anteriores [6].

La Figura 1 muestra una representación de cómo sería este proceso para el caso de una secuencia textual. Inicialmente se toma la primera palabra de la secuencia como input para la red neuronal. Esta palabra será sometida al proceso de multiplicación capa tras capa utilizando los parámetros aprendidos por la red. El output de la red neuronal (la palabra procesada) se fusionará con la nueva información a introducir en la red neuronal (la siguiente palabra de la secuencia). Este proceso de introducir el output anterior como parte del nuevo input se repetirá hasta haber analizado toda la secuencia.

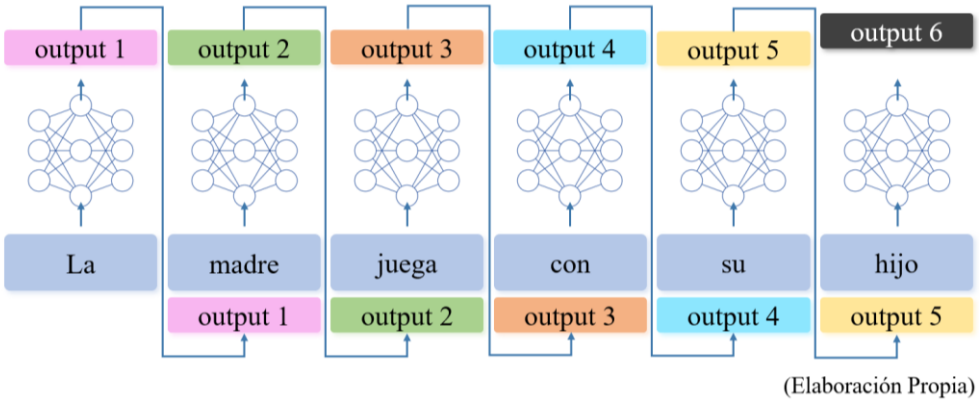

Figura 1. Esquema del funcionamiento de una red neuronal recurrente

La gran limitación de este tipo de redes es la "falta de memoria", o desvanecimiento de gradientes. Esto ocurre cuando los gradientes, los cuales indican cómo han de actualizarse los pesos en función del error, de las primeras palabras procesadas se diluyen respecto a las últimas que han entrado en la red neuronal. El resultado de esto es que el algoritmo acaba teniendo más en cuenta a aquellas palabras ha procesado recientemente, o lo que es lo mismo, le "falta de memoria".

Para solventar este problema, se desarrollaron los mecanismos de atención, componente fundamental de los *Transformers*. Estos mecanismos permiten al modelo centrar su atención en aquellas partes específicas de la secuencia de entrada que son relevantes para la tarea deseada. La idea detrás de esto es imitar la forma en que las personas son capaces de prestar atención a diferentes aspectos a la hora de realizar una tarea.

Los mecanismos de atención, en su forma más básica, están compuestos por tres vectores [7]. El primero de estos es el vector Consulta o *Query* en inglés, y representa la información actual que el modelo está procesando. El siguiente es el vector Clave o *Key*, y sirve para medir la similitud entre la Consulta y cada elemento en la entrada. Finalmente, el vector Valor o *Value*, es el que representa numéricamente cada uno de los valores de entrada. Estos tres vectores se utilizan para calcular los pesos de atención tal y como muestra la Figura 2.

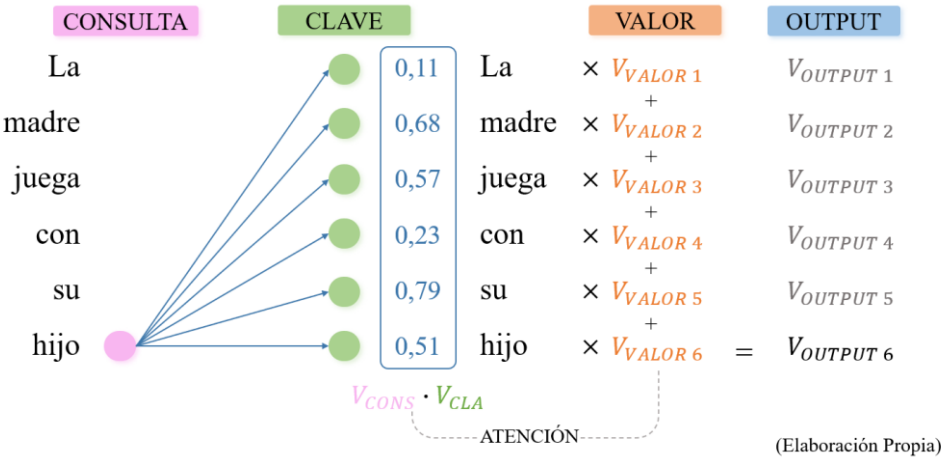

Figura 2. Representación del funcionamiento del mecanismo de atención

El ejemplo muestra la creación de los pesos de atención para la frase: "La madre juega con su hijo". En concreto muestra el proceso para la última palabra de la secuencia, "hijo". A la izquierda del todo podemos ver la secuencia completa y cómo el vector Consulta se sitúa en la palabra a analizar. A continuación, se observa el resultado del producto escalar entre el vector Consulta y el vector Clave, resultando en el vector Atención.

Con este vector se puede ver cómo de relacionadas están la palabra que hace la consulta y el resto de palabras de la secuencia; cómo de similares son. En el caso de ejemplo, se observa como "hijo" tiene una mayor relación con las palabras "su" y "madre". A continuación, se multiplica el vector Atención con el vector Valor, que está asociado a cada palabra; obteniendo de esta forma el valor final de la atención, u output de la red neuronal [8].

Este mecanismo de atención ha permitido que sea posible contextualizar cada una de las palabras de la secuencia con cualquier otra palabra, sin importar la distancia a la que se encuentren; solventando el problema de la "falta de memoria" de las RNN. Los Transformers cómo indica el nombre del *paper* donde fueron presentados: *Attention Is All You Need* [5], parten de la premisa de que las tareas de *deep learning* se pueden realizar únicamente con los mecanismos de atención; sin ser necesaria la combinación con las RNN.

Como alternativa a las redes recurrentes, los *Transformers* procesan en paralelo las diferentes partes de las secuencias. En el caso de las secuencias textuales, todas las palabras son procesadas simultáneamente tal y como se muestra en la Figura 3.

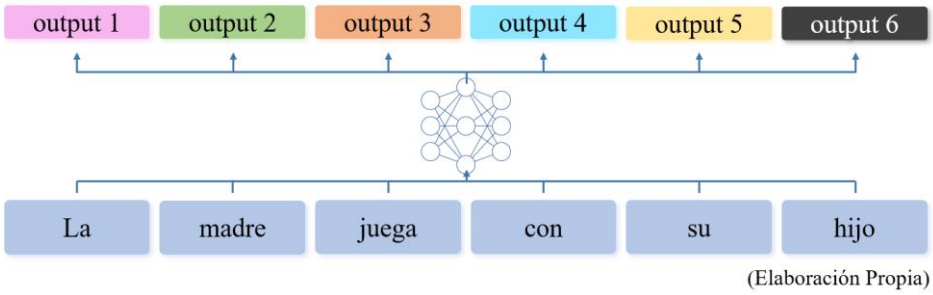

Figura 3. Esquema del funcionamiento de un *Transformer*

Para procesar las secuencias en paralelo los *Transformers* utilizan una técnica llamada Codificación Posicional [9]. Gracias a esta son capaces de entender y trabajar con el orden de las palabras en una secuencia, ya que a pesar de que procesan los datos en paralelo, los inputs con los que trabajan siguen teniendo una estructura secuencial, por lo que el orden de las palabras importa. Este proceso de Codificación Posicional consiste en agregar información, sobre la posición relativa de las palabras en la secuencia, a la representación de cada palabra con su vector Valor. Esto se logra añadiendo vectores de posición específicos a los vectores de entrada antes de que sean procesados en las capas de atención y transformación.

En conclusión, los *Transformers* han resultado ser muy útiles desde su presentación gracias a las dos características claves comentadas. El mecanismo de atención les permite la captura de relaciones a largo plazo en las secuencias, lo que ha facilitado el uso secuencias textuales mucho mayores; mientras que la Codificación Posicional les permite procesar en paralelo estas grandes secuencias sin perder la noción del orden de las palabras y de una manera mucho más rápida y eficaz que con las redes neuronales recurrentes. Estas características han sido cruciales para el éxito de los *Transformers* en tareas de procesamiento de lenguaje natural y otras para los que no fueron creados originalmente.

2.3 Procesamiento del Lenguaje Natural

Este trabajo, como se comentó en el primer capítulo, se centra en las inteligencias artificiales enfocadas al Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés). Este campo aborda el entendimiento del lenguaje por parte de los ordenadores de una forma “natural” o análoga a la compresión humana [10]. Esta habilidad ha permitido para desarrollar numerosas aplicaciones de gran utilidad, como evaluar reseñas *online* para determinar su tono positivo o negativo, diferenciar los sentimientos de mensajes de redes sociales como los *tweets*, sintetizar textos extensos de forma automática o generar respuestas coherentes dado un contexto específico.

Uno de los usos más destacados es el de predecir la siguiente palabra dada una secuencia de texto, en la que las inteligencias artificiales haciendo uso del contexto anticipan la próxima palabra más congruente para mantener la coherencia del texto. Esta funcionalidad a su vez ha facilitado la posibilidad de generar texto de manera automática, ya que, al ser conocidos los diferentes patrones existentes en las secuencias textuales, la IA puede recrear estos en nuevos textos dada una oración, un tema o un contexto. Finalmente, la aplicación que se va a tratar en este trabajo es la de contestar a una pregunta siendo conocido el contexto de la respuesta. Existen dos formas de afrontar esta tarea tal y como se explicará en la sección 2.4 de este capítulo.

Para que un algoritmo de *machine learning* sea capaz de interpretar datos en formato de texto, hacen falta varias técnicas de preprocesamiento siendo la más importante de ellas la tokenización. Este proceso consiste en dividir la secuencia de entrada, que el ordenador interpreta como una cadena de caracteres, en subunidades llamadas tokens [11].

Existen diferentes técnicas o formas de realizar la tokenización en función de cuanto abarca un token. Como máximo un token se corresponde con una palabra entera. En el proceso de tokenización también se incluye la creación (o no) de tokens para los números, signos de puntuación o hasta emojis. Para comprender cual es el tokenizador más utilizado en la actualidad, se va a explicar cómo ha ido evolucionando el enfoque que se le ha dado a este [12].

Inicialmente se partió de un tokenizador basado en palabras o **Word-based tokenizer**. En este enfoque, cada token se corresponde con una palabra y a estas se le asocio un *ID* numérico único, el token. De esta forma cada número o *ID* contiene mucha información semántica y contextual. Este enfoque es muy útil, pero contiene una gran limitación, el tokenizador otorgará dos *ID* diferentes a palabras muy parecidas, como puede ser una palabra y su plural o su femenino. En la Figura 4, se ve como las palabras *gato* y *gata* tienen un *ID* muy distinto, a pesar de que ambos hacen referencia al mismo animal únicamente variando el género.

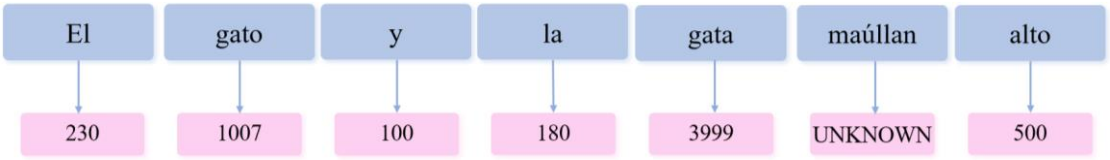

Figura 4. Frase tokenizada con un enfoque Word-based

El hecho de usar dos *IDs* significa que el modelo realizará dos procesos de embedding diferentes. El algoritmo de embedding es el encargado de representar los tokens en un espacio vectorial continuo dónde las palabras con significados y contextos similares se encuentran cercanas entre ellas [13], tal y como se muestra en la Figura 5. En esta se puede observar cómo los vectores de los tokens que hacen referencia a frutas están representados en la misma zona

mientras que el token de *silla*, no lo está. Es esperable que otros tokens como *mesa* o *puerta*, sí que se encuentren cerca de este.

Al crearse dos *IDs* diferentes para palabras muy similares, es más fácil que la representación vectorial de estas palabras no sea tan exacta, y no se encuentren tan cerca como deberían estar por su significado.

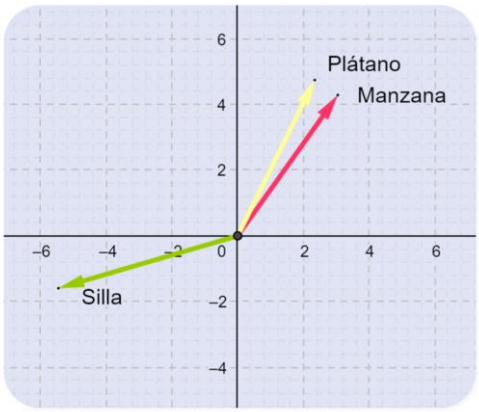

Figura 5. Representación de tres tokens en un espacio vectorial

Otra limitación de este modelo es que al crear una *ID* para cada palabra, si se trabaja con grandes textos con un vocabulario rico y variado se necesitará hacer uso de una gran cantidad de *IDs*. Esto será problemático a la hora de crear el vector que los representa en el espacio vectorial porque se necesitará de un gran proceso de ponderaciones.

Una solución a esto es optar por limitar el máximo de *IDS* a asignar. Por ejemplo, se pueden numerar los 10.000 tokens más comunes de una base de datos y asignar al resto de palabras más raras el token de *UNKNOWN*, que significa desconocido en inglés. Esto es lo que ocurre en el ejemplo de la Figura 4 con la palabra *maúllan*. Esta solución supone cierta perdida de información debido a que dos palabras que no sean muy comunes como podrían ser *natillas* y *condensador* tendrán la misma representación en el modelo.

Para hacer frente a las limitaciones anteriores, se desarrolló el tokenizador basado en caracteres o **Character-based tokenizer**. En él cada token se corresponde con un carácter individual. Esto reduce drásticamente la cantidad de tokens necesarios. Por ejemplo, el idioma inglés cuenta con más de 170.000 palabras diferentes, sin tener en cuenta nombre propios; mientras que todas estas palabras podrían ser escritas usando tan solo 256 caracteres. Dentro de estos caracteres a parte del alfabeto latino (sin acentuar) se incluyen además los números, signos de exclamación y caracteres espaciales como el símbolo del euro o la arroba.

Este enfoque tiene un vocabulario más completo que el tokenizador basado en palabras gracias a que al contener todos los caracteres usados en un idioma puede tokenizar cualquier texto dado

en ese idioma, incluso aquellos que contengan palabras que no hayan sido visto durante el proceso de entrenamiento del tokenizador. De esta forma se consigue reducir los tokens desconocidos, como ocurre en la Figura 6 con la palabra *epiglotis*. Esto es muy útil a la hora de tokenizar textos que contengan errores ortográficos, dado que en lugar de descartar esas palabras clasificándolas como *UNKNOWN* podrá tokenizarlas.

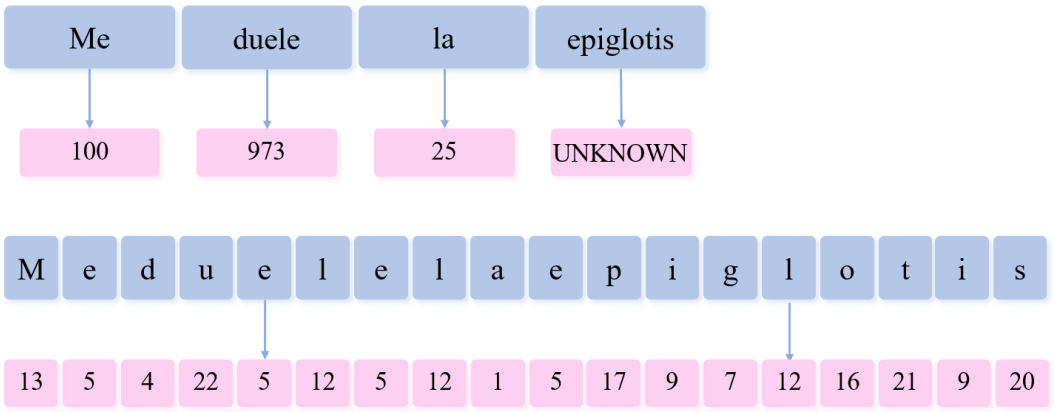

Figura 6. Comparación del enfoque Word-based y Character-based al tokenizar una frase

La gran limitación de este enfoque es que los caracteres no contienen por sí mismos tanta información como si lo hacen las palabras. Por lo que para entender una palabra se necesitará hacer uso de varios tokens simultáneamente en lugar de solo uno como ocurría en el anterior enfoque. Esto además afecta al tamaño de las secuencias a procesar por el modelo. Un ejemplo de esto se da en la Figura 6superior, dónde una frase que con el anterior tokenizador tenía cuatro tokens pasa a tener dieciocho. Esto limita el tamaño de los textos que se pueden utilizar como entrada en el modelo, reduciéndolos considerablemente.

Finalmente, se ha impuesto un modelo que se encuentra a medio camino entre los dos mencionados. Este es el tokenizador basado en subpalabras o **Subword-based tokenizer**, que hereda las mejores cualidades de los modelos anterior y solventa las limitaciones. La filosofía detrás de este tokenizador es que las palabras más frecuentes no deberían ser divididas en subpalabras más pequeñas mientras que las palabras más raras deberían ser descompuestas en subpalabras con significado como se puede ver en la Figura 7, donde *salpican* se descompone en dos palabras conocidas *sal* y *pican*.

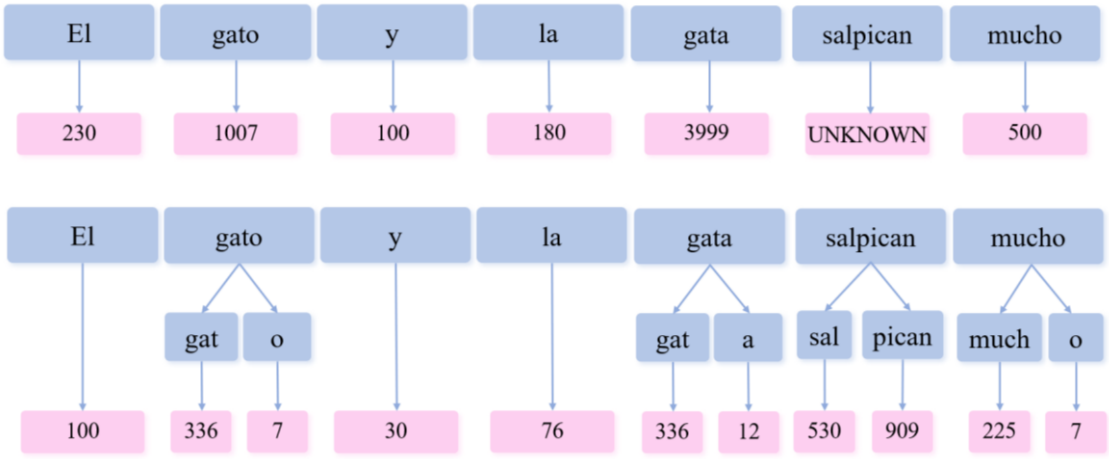

Figura 7. Comparación del enfoque Word-based y Subword-based al tokenizar una frase

Las palabras en plural son otro ejemplo de esto. Según este enfoque la palabra *playas* constaría de dos tokens: [“playa”, “s”]. Lo que se pretende con esto es recalcar al algoritmo que tanto la palabra “playa” como “playas” hacen referencia a un mismo contexto, solo que a la segunda se le ha añadido un sufijo que no varía su significado ni contexto, solo lo detalla más.

Por otra parte, otro ejemplo de palabra rara, que de hecho no está aceptada por la RAE al tratarse un anglicismo, sería *tokenización*. Esta se descompondría de la siguiente forma: [“token”,” ización”], siendo *token* la raíz de la palabra e *ización* información adicional a esta que la complementa. De esta forma el modelo puede entender que todas las palabras cuyo token inicial sea “token” tendrán un significado parecido, mientras que todas las palabras que finalicen con un token como “ización” podrán ser usadas en situaciones sintácticas parecidas.

Los modelos usan diferentes formas de tokenizar estas raíces y sufijos. Por ejemplo, el modelo BERT, que se tratará con mayor profundidad más adelante, lo hace añadiendo dos hashtags de la siguiente forma: [“token”, ”##ización”] [12]. En la actualidad, la mayoría de modelos que obtienen resultados del estado del arte en el ámbito del NLP usan un algoritmo de tokenización de este tipo, es decir, basado en subpalabras. Los tres más usados son Unigram, Byte-Pair Encoding y WordPiece, que es el que usa este trabajo.

2.4 *Question Answering*: preguntas y respuestas

Desde la aparición de los Transformers, han surgido diferentes arquitecturas y modelos que implementan o toman como base esta tecnología, y que han sido entrenados con grandes volúmenes de secuencias de texto. De esta forma se ha podido producir resultados del estado del arte en diversas tareas relacionadas con el NLP, como se ha comentado en las anteriores secciones de este capítulo.

Una de estas tareas, y la que se va a estudiar en este trabajo, es el *Question Answering* (*QA*). Estos modelos de preguntas y respuestas tienen la finalidad de contestar lo más preciso posible a las preguntas efectuadas por los usuarios. En los próximos años, se estima que una gran variedad de empresas pertenecientes a diversos sectores, las cuales dispongan y puedan hacer uso de grandes cantidades de datos, implementen herramientas de este tipo.

La empresa que se puso en contacto con la Universitat Politècnica de València para implementar una herramienta de este tipo se trataba de un bufete de abogados. Este buscaba automatizar y optimizar la contestación rutinaria de ciertas preguntas sobre documentos legales, cómo cual era la entidad demandada, el tipo de contrato, el suplico de la demanda o si existía reclamación previa.

Hoy en día, estas tareas de entendimiento las llevan a cabo abogados cualificados que cobran una gran minuta por las labores que realizan, de modo que implantar un modelo de QA que realice estas funciones, podría suponer un gran ahorro económico y un mejor aprovechamiento de los recursos humanos de la compañía.

Para hacer frente este tipo de tareas de preguntas y respuesta, existen dos enfoques de modelos de inteligencia artificial que las implementan, tal y como se muestra en la Figura 8: los modelos QA Extractivos, que devuelven la respuesta tal cual se encuentra en la secuencia original de donde se extrae; y los modelos QA Abstractivos, que dan formato a la respuesta pudiendo añadirle contexto [14].

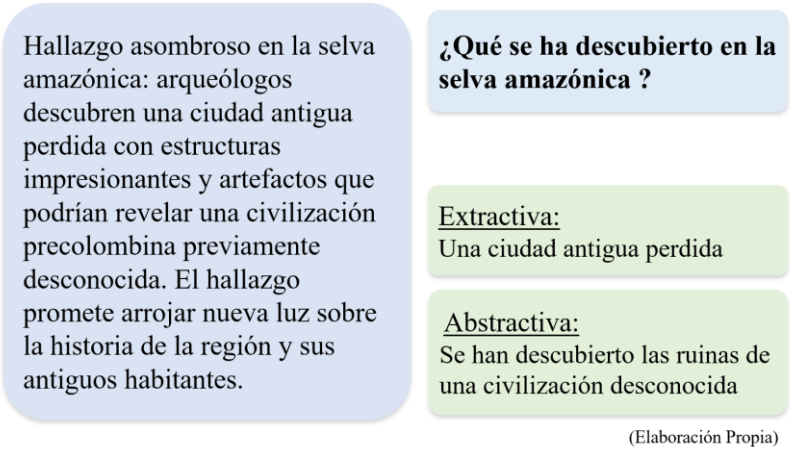

Figura 8. Respuesta Extractiva y Abstractiva a una pregunta

Los modelos **QA Extractivos** buscan responder preguntas formuladas por los usuarios sobre un determinado texto mediante la selección y extracción de la respuesta directa más relevante de dicho texto. Estos modelos no generan ninguna información nueva ni extra en sus contestaciones;

al formular preguntas sobre un contexto dado, las respuestas están basadas plenamente en la información proporcionada.

Entre los modelos QA Extractivos, destaca la familia BERT cuyo nombre es la abreviatura de: *Bidirectional Encoder Representations from Transformers*. El BERT original fue publicado en 2019 por el equipo de Google Language y fue entrenado usando secuencias de texto sin etiquetar, en las que se enmascaraban palabras para enseñar al modelo a predecir que palabras ocupaban ese lugar, basándose en el contexto [15].

A diferencia de otros modelos, la inclusión de los Transformers permite al BERT capturar mejor el contexto y ofrecer por tanto mejores rendimientos en este tipo de entrenamientos. En concreto, los BERT solo utilizan la mitad de la arquitectura original de los Transformers, es decir, la parte del codificador.

A parte del entrenamiento inicial comentado, BERT fue afinado para diferentes tipos de tareas. Una de las más destacadas de estas fue el *fine-tuning* al que lo sometieron para tareas de QA, dado que lo entrenaron con cientos de miles de preguntas y respuestas del conjunto de datos SQuAD [16]. Este *dataset* está compuesto por preguntas sacadas de artículos de la Wikipedia, donde cada respuesta corresponde a un fragmento específico del texto. De esta forma, los modelos BERT son capaces de devolver a su salida, donde se encuentra representado el inicio y final de las respuestas del contexto dado.

El otro enfoque existente para este tipo de tareas es el de los **modelos QA Abstractivos**. Estos modelos, al contrario que los extractivos, son capaces de generar respuestas que pueden no estar presentes en el texto original. Esto dota a las contestaciones de una mayor creatividad y flexibilidad, disminuyendo a la vez su exactitud.

El ejemplo más conocido de los usos de este tipo modelos es *ChatGPT*. Este chatbot, está basado en un modelo GPT que utiliza la capa de decodificación de los Transformers originales. Los GPT están preentrenados para predecir la siguiente palabra en una secuencia de manera no supervisada, y luego pueden ser afinados para distintos tipos de tareas mediante el uso del *fine-tuning* de forma supervisada por personas.

En el caso de tareas de preguntas y respuestas, durante el proceso de entrenamiento a los modelos GPT se les muestran múltiples propuestas de respuesta a través de numerosos ejemplos, entrenándolos a elegir la respuesta correcta. Es por esto, que sus respuestas son generadas a partir de la información contextual proporcionada, pero también por conocimientos previos.

En el caso de querer implementar un modelo de este tipo se podría estudiar la viabilidad de hacerlo usando una API de OpenAI. En el momento de la realización de este trabajo, existen diferentes modelos y precios disponibles para esta implementación. El precio para realizarlo con

el modelo GPT-4, es de 0,03 $/1K tokens para las secuencias de entrada y 0,06 $/1K tokens para las secuencias de salida. Para hacerse una idea, 1000 tokens son de media unas 750 palabras [17].

Este es el modelo más potente generado hasta la fecha, pero también existiría la posibilidad de customizar uno de los modelos de la familia GPT mediante el fine-tuning con nuestros propios datos. De esta forma se podría entrenar el modelo para que conociese mejor el trabajo a realizar y potenciar sus resultados. En este formato de API, el precio varía en función del modelo base y se cobra primero por el entrenamiento y luego por las secuencias de entrada. Las de salida no se cobran.

En este trabajo nos vamos a centrar los modelos QA Extractivos, en concreto en la familia BERT. Esto se debe a que, en el caso del bufete de abogados, o los reportes médicos que se comentarán en el próximo capítulo; lo que se busca es la mayor fidelidad de las respuestas respecto al texto al cual se le hace la pregunta. Es por esto por lo que son mejor opción los modelos extractivos que los abstractivos, dado que estos últimos pueden proporcionar respuestas que suenen muy plausibles sin usar nada de la información del contexto introducido.

Capítulo 3. Metodología

3.1 *Dataset* de reportes radiológicos

Este trabajo pretende ser un estudio general de los modelos de preguntas y respuestas extractivos, con la finalidad de definir las guías para una posible implementación por parte de cualquier entidad que esté interesada en este tipo de herramientas. Aun así, para poder detallar mejor el estudio se va a caracterizar para un caso concreto indicando como se podría extrapolar para otras situaciones.

En este caso la implementación del modelo QA se va a hacer sobre reportes radiológicos. La adopción de una medida de este tipo podría aportar grandes beneficios en el ámbito sanitario, optimizando el tiempo del personal implicado. Por ejemplo, si un médico quisiera una segunda opinión sobre un paciente que lleva tratando varios años, y por tanto cuenta con un largo historial; el médico podría preguntar por algún aspecto concreto, cómo el estado de salud de los pulmones del paciente, a la inteligencia artificial.

El modelo podría devolverle al médico toda la información disponible en el historial del paciente, así como de que reportes se ha extraído toda esa información. Ahora solo quedaría que el médico le enseñara esto al otro especialista, habiéndose ahorrado de esta forma el tiempo que tendría que dedicar algún miembro de la plantilla para recopilar la información.

Para poder conseguir respuestas certeras es mejor usar un modelo específico para el tema a tratar, dado que conocerá mejor el vocabulario, por ejemplo. Esta idea se desarrollará mejor en el siguiente capítulo. A parte de un modelo base, es importante disponer de una buena base de datos representativa de las tareas a realizar.

Para este trabajo se va a usar el *dataset* público de RadQA, creado para la mejora del tratamiento de datos médicos relacionados con la radiología por la Escuela de Informática Biomédica de la Universidad de Texas [18]. Este está compuesto por 3074 pares únicos de preguntas y reportes, los cuales abarcan 1009 informes de radiología de 100 pacientes.

Los reportes radiológicos cuentan con diferentes secciones, de entre las que vamos a destacar tres por la importancia que tienen para la base de datos. La primera se llama *Indication* y contiene la información que el paciente proporcionó previamente a su médico y terminó con la realización de la prueba radiológica. La segunda sección tiene por nombre *Findings* e incluye la información de las observaciones para cada área del cuerpo realizadas por el radiólogo. Finalmente, en la sección *Impressions*, el radiólogo indica las observaciones más importantes y posibles causas por las que se pueden haber originado [19].

Cada pregunta, tiene dos respuestas en el reporte, una en la sección de *Findings* y otra en la de *Impressions*, por lo que en total el *dataset* cuenta con 6148 pares de preguntas-respuestas. Estas respuestas pueden darse en forma de frases largas que abarquen varias líneas. Además, para contestar correctamente a las preguntas es necesario un amplio conocimiento del ámbito médico, por lo que es un conjunto de datos desafiante pero muy útil [18].

Para la creación de RadQA, o cualquier otro *dataset* destinado a ajustar finamente un modelo, se ha de partir de un buen conjunto de datos representativos para la tarea a realizar o del sector específico y llevar a cabo un proceso de transformación de los datos para que tengan una estructura adecuada. En el caso de concreto de los modelos QA, esta transformación consiste en asegurarse que la base de datos contiene tanto contextos y preguntas como las respuestas asociadas a estas.

En este caso, RadQA se creó partiendo de informes radiológicos comprendidos en MIMIC-III, una de las mayores bases de datos públicas existente para el área de la investigación médica y que recopila diferentes tipos de informes de pacientes de la unidad de cuidados intensivos [20]. Para seleccionar que informes de MIMIC-III se incorporaban en RadQA, se seleccionaron primero a 100 pacientes que fueran representativos para la muestra y después se cogieron los distintos informes asociados. De media cada paciente cuenta con 10,09 reportes, siendo más de la mitad de estos de la modalidad de rayos X, tal y como ocurre en el conjunto del *dataset* también.

Después se utilizó la proporción de 8:1:1 para dividir a los pacientes, y por tanto los reportes que conforman la base de datos en tres conjuntos diferenciados y representativos de manera equitativa para el entrenamiento, validación y prueba del modelo.

Con los reportes ya seleccionados, el siguiente paso es la inclusión en el *dataset* de las preguntas y respuestas asociadas. La creación de estas preguntas fue encomendada a radiólogos, a los cuales se les enseñaron dos secciones del reporte diferentes a las comentadas anteriormente. Estas incluían una breve descripción sobre el estado del paciente y los motivos para solicitar el estudio de radiología. Al elaborar las preguntas únicamente con esta información se evita que contengan algún tipo de sesgo relacionado con lo anotado por los médicos en las secciones de *Findings* e *Impressions*.

Además, se instruyó a los radiólogos para que crearan preguntas que utilizaran tanto información explicita como implícita de la sección o que pudieran necesitar altos conocimientos técnicos para ser respondidas. También se les advirtió de que variasen la sintaxis a la hora de formular las preguntas, diversificando en forma y contenido. Al final, dos radiólogos crearon de manera independiente entre ellos preguntas para todos los reportes, las cuales fueron posteriormente agregadas y se descartaron aquellos casos dónde hubiera algún tipo de duplicidad.

Para la parte de apuntar las respuestas, se enseñó a los anotadores el reporte completo y se les solicitó que indicaran las respuestas más cortas pero completas a las preguntas realizadas. Cada pregunta debería tener una respuesta tanto en la sección de *Findings* como en la de *Impressions*. Al estar las preguntas generadas sin tener en cuenta los informes resultantes a las pruebas radiológicas, habrá preguntas que no cuenten con respuesta, y así deberán apuntarlo los radiólogos. Las anotaciones de las respuestas se han realizado usando la herramienta *Annotation Tool* de Haystack.

Haystack es un marco de código abierto desarrollado por Deepset utilizado para la recuperación y el análisis de información de grandes conjuntos de datos no estructurados. Esta biblioteca de código abierto basada en Python está diseñada específicamente para trabajar con datos de texto y realizar tareas de NLP a gran escala [21].

La *Annotation Tool* permite dada una secuencia de texto, añadir preguntas y anotar las respuestas sobre el mismo texto, tal y como se muestra en el ejemplo de la Figura 9. Cualquier empresa o entidad que quiera crear un modelo de QA para sus documentos tendrá que hacer uso de esta herramienta, o una similar, tras haber recopilado una muestra representativa de estos. El uso de esta herramienta tiene como finalidad pasar a tener una base de datos etiquetada, la cual se usará para poder mejorar la precisión del modelo concreto a generar por la empresa.

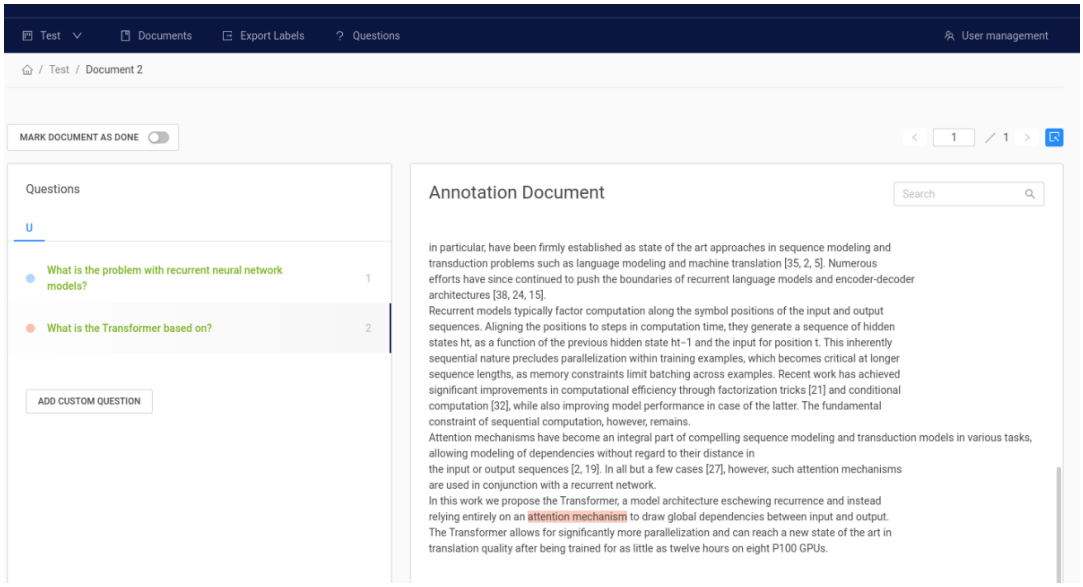

Figura 9. Annotation Tool Haystack

Esta herramienta facilita considerablemente un proceso bastante laborioso y lioso. Su uso es bastante intuitivo, primero hay que crear un proyecto dentro de su interfaz en el que cargar los documentos para luego proceder a anotar las respuestas a las preguntas como si se estuviera subrayando un documento. Para obtener buenos resultados en un modelo se necesitan mínimo unos 2000 pares de preguntas-respuestas [22].

En el caso del bufete de abogados por el que nace este trabajo, se tenía pensado preguntar las mismas diez o doce preguntas a todos los documentos, por lo que con subir 200 informes sería suficiente para entrenar un modelo correctamente. Se pueden usar preguntas estándar para cada documento, como en el caso del bufete, o se pueden crear preguntas específicas para cada documento, como es el caso de la creación de la base de datos RadQA.

Tras el proceso de anotación de las respuestas, que tiene que ser realizado por personal experto en la materia que se pretende analizar como los radiólogos o abogados, Haystack permite exportar los documentos etiquetados en un formato SQuAD, lo que facilitará mucho su uso para entrenar el modelo.

Con esto se ha visto la base de datos que se va a utilizar en el proyecto y como fue creada pero también como una empresa podría replicarlo para crear ellos su propia base de datos para entrenar su modelo preguntas y respuestas.

3.2 Aceleradores de GPU

Para poder entrenar cualquier tipo de modelo se necesita un ordenador que disponga de un buen GPU, o *Unidad de Procesamiento Gráfico* por sus siglas en inglés. Este procesador está especializado en acelerar el procesamiento de gráficos y tareas visuales. Gracias a su gran potencia en cálculos parametrizables y procesamiento de datos en paralelo, además de para tareas gráficas, también se suele utilizar en el ámbito de la inteligencia artificial y el *machine learning*.

El entrenamiento de la mayoría de los modelos de redes neuronales, sin importar su tamaño, se beneficia significativamente del uso de GPUs potentes, ya que aceleran considerablemente el proceso de entrenamiento y aprovechando mejor los recursos disponibles. Es por esto por lo que disponer de un ordenador con las capacidades adecuadas es importante para realizar este tipo de tareas.

Para los casos donde los recursos de la GPU del ordenador son insuficientes o inexistentes, y gracias al aumento de la popularidad e inversión en este tipo de modelos; han surgido diferentes alternativas tanto de pago como gratuitas. En mi caso, al trabajar desde un portátil con poca potencia, se he necesitado hacer uso de dos de estos aceleradores de GPU online: Google Colab y Kaggle.

Google Colab, cuyo nombre completo es Google Colaboratory, es un entorno de desarrollo basado en la nube que permite a cualquier usuario que disponga de una cuenta de Gmail escribir, ejecutar y compartir código de Python. Además, esta herramienta gratuita permite el acceso a GPUs virtuales de Google para acelerar el entrenamiento de los modelos. Los GPUs de Colab son

compartidos por los usuarios, por lo que su disponibilidad y rendimiento no está garantizado y fluctúan en función de la demanda en el momento de trabajo [23].

Si los usuarios desean asegurarse el poder hacer uso de los aceleradores de GPU, Colab dispone de tres planas de pago diferentes: *Pay As You Go*, dónde se paga únicamente por las unidades informáticas que se usa, *Colab Pro*, que incluye 100 unidades informáticas; y *Colab Pro +* con 500 unidades informáticas al mes, GPUs más rápidos, más memoria y otros beneficios [24]. Para la realización de este trabajo se ha optado por la versión gratuita, complementándola con otro acelerador de GPU para en los momentos dónde Colab estaba saturado.

Kaggle es una plataforma en línea enfocada a la comunidad de la ciencia de datos, la inteligencia artificial y el *machine learning*. En ella, se pueden participar en diversas competiciones relacionadas con estos ámbitos, donde los participantes compiten, en muchas ocasiones, por premios económicos en diferentes desafíos de análisis y modelización de datos.

Estas competencias suelen involucrar también conjuntos de datos muy interesantes proporcionados por empresas y organizaciones, que los ponen a la disposición de la comunidad de Kaggle para que los usuarios pueden realizar sus propios proyectos. Para que cualquier usuario pueda participar en las competiciones, o simplemente aprender sobre ciencia de datos o aprendizaje automático, Kaggle ofrece a sus usuarios un mínimo de 30 horas semanales de uso de sus distintos aceleradores de GPU, siendo la máxima duración de una de estas sesiones de 9 horas ininterrumpidas [25].

El desarrollo de este trabajo se ha basado en la alternancia entre estas dos herramientas con aceleradores de GPU gratuitos, debido a la indisponibilidad por demanda de Colab o al exceso de cuota semanal de Kaggle.

3.3 Proceso de creación del modelo

Una vez explicado el *dataset* que se va a utilizar para caracterizar un modelo de preguntas y respuestas para el ámbito sanitario de los reportes radiológicos, y los diferentes entornos de programación con sus aceleradores de GPU; se va a detallar el proceso seguido para la creación del modelo. Este proceso se ha realizado en una *Jupyter Notebook* , la cual está dividida en siete etapas como se muestra en la figura 10.

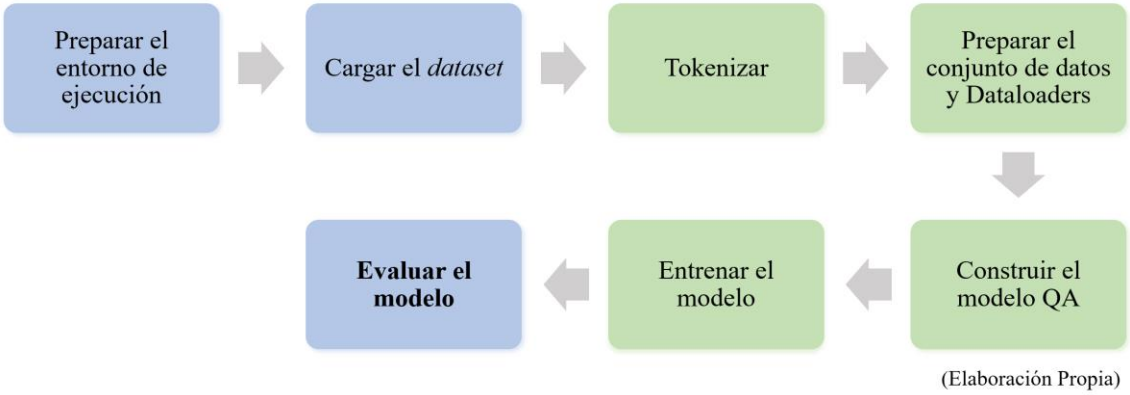

Figura 10. Esquema de las etapas del proceso de creación del modelo

Una vez construido el modelo inicial, se van a ir ajustando diferentes parámetros o variando partes de los procesos, hasta dar con el modelo con mayor exactitud en la tarea de contestar a las preguntas. Aquellas etapas de color azul se mantendrán fijas en los diferentes procesos de creación que se van a llevar a cabo, mientras que las etapas de color verde son dónde se van a producir las diferentes variaciones en el código para evaluar cómo afectan a la *accuracy* del modelo.

La primera etapa en la creación del modelo consiste en **preparar el entorno de ejecución**. Para ello es fundamental asegurarse de que esta seleccionado alguno de los aceleradores de GPU disponibles tanto en Google Colab o Kaggle. Además, en esta parte del código se importan las diferentes librerías que se van a necesitar durante la ejecución del programa.

Entre ellas destacan la librería *Json* para trabajar con bases de datos en este formato, *Torch* para poder crear modelos en PyTorch y la de los *Transformers*. De esta última además se importan diversas clases y funciones relacionadas con los modelos BERT y sus tokenizadores, en específico los enfocados a las tareas de QA. Además, se define la variable asociada al modelo base del cual vamos a partir para realizarle el fine-tuning.

La siguiente etapa tiene como finalidad **cargar el *dataset*** de RadQA que se va a utilizar para entrenar el modelo. Para realizar esto, se utiliza la función *load* de la clase *Json*. En total se van a cargar dos bases de datos. La más grande se va a utilizar para al entrenamiento del modelo mientras que la otra servirá la validación de este. Estas bases de datos se mantienen separadas en todo momento para evitar así hacer *overfitting* sobre el modelo.

Ambos conjuntos de datos cuentan con la misma estructura. Primero un contexto, que contiene una de las secciones de los reportes radiológicos, a continuación, la pregunta y finalmente la respuesta. La respuesta a su vez es un diccionario que cuenta con tres atributos diferentes: un identificador (*answer_id*), el texto extraído del contexto aportado que sirve como respuesta a la pregunta, y el número del carácter inicial donde se encuentra la respuesta en el contexto.

Usando la longitud del texto extraído, se va a añadir un cuarto atributo al diccionario. Este es el número del carácter donde finaliza la respuesta, y será de utilidad para la medición de la exactitud de las respuestas, y por tanto del modelo. De esta forma la base de datos queda tal y como se muestra en el ejemplo de la Figura 11.

```
Context: final report
 chest single ap film:

 history: icd placement and shortness of breath.

 there is a right sided dual chamber icd with leads unchanged in location in
 this single view compared with the prior film of [**2101-5-25**]. no pneumothorax.
 there is cardiomegaly with upper zone redistribution, bilateral pleural
 effusions and associated bibasilar atelectases. consolidation at the lung
 bases cannot be ruled out.

Question: is there any significant change from prior visit?

Answer: {'answer_id': '827318_2_1_O_MG', 'text': 'icd with leads unchanged in location',
'answer_start': 124, 'answer_end': 160}
```

Figura 11. Ejemplo de la estructura de la base de datos

Entre ambos conjuntos de datos, el de entrenamiento y validación, hay 5534 pares de preguntas y repuestas; de las cuales solo vamos a trabajar con las 3934 que disponen de una respuesta conocida.

La tercera etapa consiste en **tokenizar** el dataset, que tal y como se ha comentado en el capítulo anterior, es un pilar fundamental para el procesamiento de lenguaje natural. Como nuestro modelo se va a crear partiendo de alguno perteneciente a la familia BERT por defecto la tokenización será del tipo subpalabras, en concreto se utilizará el algoritmo WordPiece.

A pesar de no poder elegir el algoritmo a utilizar, sí que hay diferentes hiperparámetros implicados en el proceso de tokenización que se podrán variar, como la longitud máxima de las secuencias, si se truncan, si se les realiza *padding* o si se somete a la secuencia a algún tipo de preprocesamiento. La Figura 12 muestra una posibilidad de cómo podría ser tokenizado el ejemplo mostrado en la Figura 11.

```
Secuencia a tokenizar: '[CLS] is there any significant change from prior visit? [SEP]
final report chest single ap film : history : icd placement and shortness of breath.
there is a right sided dual chamber icd with leads unchanged in location in this
single view compared with the prior film of [ * * 2101 - 5 - 25 * * ]. no
pneumothorax. there is cardiomegaly with upper zone redistribution, bilateral pleural
effusions and associated bibasilar atelectases. consolidation at the lung bases cannot
be ruled out. [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]'

Secuencia tokenizada:[101, 1110, 1175, 1251, 2418, 1849, 1121, 2988, 3143, 136, 102,
1509, 2592, 2229, 1423, 170, 1643, 1273, 131, 1607, 131, 178, 1665, 1181, 12693, 1105,
1603, 1757, 1104, 2184, 119, 1175, 1110, 170, 1268, 11641, 7058, 5383, 178, 1665,
1181, 1114, 4501, 16684, 1107, 2450, 1107, 1142, 1423, 2458, 3402, 1114, 1103, 2988,
1273, 1104, 164, 115, 115, 13075, 1475, 118, 126, 118, 1512, 115, 115, 166, 119, 1185,
185, 1673, 1818, 12858, 25632, 119, 1175, 1110, 3621, 2660, 3263, 6997, 1183, 1114,
3105, 4834, 1894, 1776, 2047, 16442, 1988, 117, 20557, 185, 1513, 12602, 174, 3101,
27262, 1105, 2628, 16516, 16531, 8009, 1197, 8756, 18465, 23105, 119, 20994, 1120,
1103, 13093, 7616, 2834, 1129, 4741, 1149, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figura 12. Ejemplo de la tokenización en la creación del modelo

En este ejemplo concreto se puede ver el uso de los diferentes separadores que se usan previos a la tokenización ([CLS] y [SEP]) parara diferenciar el contexto de la pregunta a realizar una vez ambos han sido juntados en una única secuencia. También se observa que se ha utilizado un esquema de relleno (*padding*) para que la secuencia tenga un determinado tamaño, y como queda esto tras ser tokenizado.

Esta es una de las cuatro etapas con diferentes libertades a la hora de crear el modelo, por lo que se en el capítulo de resultados se estudiarán las diferentes opciones y su impacto en la precisión del modelo.

La cuarta etapa sirve para **preparar el Conjunto de Datos y los Dataloaders**. En ella, los datos que van a ser utilizados para el entrenamiento y la validación, ya tokenizados, son organizados en un formato adecuado para poder ser procesados por el modelo. Para ello se crea una clase que implementa la interfaz de la clase *Dataset* de *Pytorch* y se utilizan instancias de esta clase para representar los conjuntos de datos de entrenamiento y validación.

En este caso se crea un diccionario compuesto por cinco elementos: *inputs_ids*, que contiene los identificadores de los tokens de la secuencia de entrada; *token_type_ids*, que contiene los identificadores para diferenciar a que están asociados los tokens (pregunta, contexto o relleno); *attention_mask*, que contiene los ids de la máscara de atención; y finalmente *start_positions* y

end_positions. Este proceso se realiza tanto para los datos de entrenamiento como los datos de validación. La Figura 13 ejemplifica lo que vería el modelo como secuencia de entrada.

```
'input_ids': tensor([ 101, 1110, 1175, 1251, 2418, 1849, 1121, 2988, 3143, 136, 102,
1509, 2592, 2229, 1423, 170, 1643, 1273, 131, 1607, 131, 178, 1665, 1181, 12693, 1105,
1603, 1757, 1104, 2184, 119, 1175, 1110, 170, 1268, 11641, 7058, 5383, 178, 1665, 1181,
1114, 4501, 16684, 1107, 2450, 1107, 1142, 1423, 2458, 3402, 1114, 1103, 2988, 1273,
1104, 164, 115, 115, 13075, 1475, 118, 126, 118, 1512, 115, 115, 166, 119, 1185, 185,
1673, 1818, 12858, 25632, 119, 1175, 1110, 3621, 2660, 3263, 6997, 1183, 1114, 3105,
4834, 1894, 1776, 2047, 16442, 1988, 117, 20557, 185, 1513, 12602, 174, 3101, 27262,
1105, 2628, 16516, 16531, 8009, 1197, 8756, 18465, 23105, 119, 20994, 1120, 1103,
13093, 7616, 2834, 1129, 4741, 1149, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0]),

'token_type_ids': tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0]),

'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0]),

'start_positions': tensor(38), 'end_positions': tensor(46)
```

Figura 13. Ejemplo del conjunto de datos que se usará para entrenar el modelo

A continuación, usando estos conjuntos de datos se generan los *dataloaders*, para poder cargar y procesar los datos en lotes. Para el proceso de entrenamiento se determina un tamaño de lote (*batch size*) de 16 y se activa el modo de mezcla, para que en cada época se mezclen los datos. Por su parte, el *dataloader* de la validación también tiene un tamaño de lote de 16 pero mantiene desactivado el modo mezcla, para mantener el orden de los datos durante la evaluación. En el capítulo de resultados se estudiará si cambiar el *batch size* tiene algún efecto en la exactitud de las respuestas del modelo.

La quinta etapa se utiliza para acabar de definir y **construir el modelo de QA**. El primer paso para ello es cargar el modelo BERT pre-entrenado que se especificó en la primera etapa. Se va a usar este modelo como base para el entrenamiento porque ya contiene unos pesos y una arquitectura orientada a tareas de NLP. Nuestro modelo, contará con dos capas de redes neuronales. La última es la encargada de devolver la respuesta proporcionando la posición del

inicio y final de esta en la secuencia original. Además, se define una capa de dropout con una tasa del 0,1 para evitar el sobreajuste durante el entrenamiento.

Una vez definida la arquitectura ya se está listo para **entrenar el modelo de preguntas y respuestas**. Para ello lo primero es definir el optimizador a utilizar. El optimizador es el encargado de ajustar los hiperparámetros del modelo con la finalidad de minimizar la función de pérdidas durante el proceso de entrenamiento.

El optimizador utilizado ha sido *AdamW*, una variación del *Adam* (*Adaptative Moment Estimation*) en el que el decaimiento de peso (*weight decay*) es aplicado de una manera más efectiva. Dentro del optimizador se podrá variar la tasa de aprendizaje, la cual determina la magnitud de las actualizaciones de los parámetros durante el proceso, y el decaimiento de peso, que se utiliza para evitar el sobreajuste del modelo.

Además, se usa la clase *ExponentialLR* de *Pytorch* para crear un programador de tasas de aprendizaje (*schedular rate*). Con esto se pretende ajustar la tasa de aprendizaje al multiplicarla cada época la tasa actual por un factor exponencial. Por ejemplo, si gamma vale 0,9 cada época se disminuiría la tasa en un 10%.

A continuación, ya se está listo para iniciar el bucle de entrenamiento del modelo. Se ejecutará el número de épocas que se hayan especificado. Además de actualizar los parámetros del modelo, en este bucle se muestra la media de la precisión del modelo, tanto su precisión exacta como su puntuación F1, y las pérdidas de este.

La etapa de **evaluar el modelo** se ejecuta intercalada con la del entrenamiento dado que evalúa el modelo tras cada época, para poder ver así también su evolución. Para la evaluación del modelo se usa el *dataset* de validación y se pueden obtener las mismas métricas que en el entrenamiento: la precisión del modelo y sus pérdidas. La Figura 14 muestra un ejemplo de la respuesta que otorga uno de los modelos que se crearán en el próximo capítulo. Además, incluye la su evaluación atendiendo a dos métricas diferentes.

```
Context: impression

 1.  multiple small nodular densities, predominantly in the upper lobes,
     not significantly changed since the prior study.
 2.  mostly resolved spiculated nodule in the right lung base.
 3.  minimal reduction in mediastinal adenopathy.
 4.  persistent right middle lobe atelectasis.
 5.  resolved pericardial effusion.
 6.  multiple small low attenuation hepatic lesions unchanged.

Question: was there any change in mediastinal lad?

Expected Answer: minimal reduction

Predicted Answer: minimal reduction in mediastinal adenopathy

Exact Match: 0

F1 Score: 0.5714285714285715
```

Figura 14. Ejemplo de una respuesta dada por el modelo y su evaluación

Para comprender adecuadamente esta etapa, es necesario entender las diferentes métricas que vamos a utilizar para evaluar el modelo tanto en el entrenamiento como en la validación. Estas métricas son explicadas en la sección a continuación.

3.4 Métricas

Al evaluar modelos de preguntas y respuestas, es importante tener en cuenta tres métricas clave: la Coincidencia Exacta (EM, por sus siglas en inglés), la puntuación F1 y la función de pérdidas [26]. Las dos primeras son métricas ampliamente utilizadas para evaluar el rendimiento de modelos de preguntas y respuestas, mientras que la función de pérdidas es una métrica más técnica que se utiliza principalmente durante el entrenamiento del modelo.

Las dos primeras métricas calculan para cada par de preguntas y repuestas que conforman una base de datos. En aquellos casos donde es posible que existan varias repuestas correctas para una pregunta dada, porque así está anotado en el contexto, se calcula la puntuación máxima entre todas las respuestas correctas posibles. Las puntuaciones EM y F1 globales de un modelo se calculan promediando todas las puntuaciones individuales del *dataset*.

La Coincidencia Exacta, o *Exact Match*, es una métrica básica pero estricta que otorga una puntuación de 1 si la predicción del modelo coincide plenamente con la respuesta. En el caso de que tan solo un carácter sea diferente la puntuación será de 0. Por otra parte, en la puntuación F1 importa tanto la precisión como la exhaustividad (*recall* en inglés). En este caso la puntuación se calcula sobre la cantidad palabras individuales de la predicción que coinciden con la respuesta correcta, es decir, qué cantidad de palabras comparten.

La precisión es la proporción entre el número de palabras compartidas y el total de palabras en la predicción, mientras que la exhaustividad es la proporción entre el número de palabras compartidas y el total de palabras en la verdad. La puntuación F1 se calcula como la media armónica de la precisión y la exhaustividad; tal y como se muestra en la Ecuación 1.

$$\boldsymbol{F1 = 2 \cdot \frac{precisión \cdot exhaustividad}{precisión + exhaustividad}} \quad \textbf{(1)}$$

En aquellos casos donde la Coincidencia Exacta sea igual a la unidad, la puntuación F1 no aportará más información dado que al haber sido una predicción perfecta su valor también será 1, tal y como se muestra en la Figura 15.

```
Question: is there evidence of a tumor within the brain?

Expected Answer: no areas of abnormal enhancement are seen throughout the brain
to suggest a mass or tumor

Predicted Answer: no areas of abnormal enhancement are seen throughout the brain
to suggest a mass or tumor

Exact Match: 1

F1 Score: 1.0
```

Figura 15. Ejemplo de una predicción correcta dada por el modelo QA

Por otra parte, la *F1 Score* sí que será más interesante para los casos donde la Coincidencia Exacta sea nula dado que se pueden dar tres opciones. La primera de ellas es que la respuesta sea completamente errónea por lo que también obtendría un valor de 0 en la F1, tal y como se muestra en la Figura 16.

```
Question: Is the right cvl placed correctly?

Expected Answer: right subclavian venous approach was utilized for insertion
of a swan-ganz catheter which is in the left pulmonary artery

Predicted Answer: unchanged position

Exact Match: 0

F1 Score: 0.0
```

Figura 16. Ejemplo de una predicción completamente errónea dada por el modelo QA

Las otras dos opciones se dan cuando el modelo predice una respuesta que está parcialmente bien, obteniendo entonces una puntuación entre 0 y 1 para el F1, pero de 0 para la EM al no estar completamente correcta. Esta situación se puede dar cuando la respuesta otorgada por el modelo acorta o expande la respuesta correcta. En la Figura 17 se observa un ejemplo de respuesta otorgada por el modelo dónde la predicción es un fragmento más corto de la respuesta correcta.

Question: can the patient's shortness of breath be explained by any infiltrations in the lungs?

Expected Answer: right mid lung opacity is concerning for early pneumonia

Predicted Answer: right mid lung opacity

Exact Match: 0

F1 Score: 0.6153846153846153

Figura 17. Ejemplo de una predicción parcialmente correcta (acortada) dada por el modelo QA

Por otra parte, en la Figura 18 se muestra un caso donde la respuesta correcta es un fragmento de la respuesta predicha por el modelo, es decir, el modelo ha dado una respuesta más larga de la estrictamente necesaria para contestar la pregunta. Este tipo de respuestas son consideradas erróneas dado que los modelos se entrenan para que extraigan la respuesta más concisa encontrada en el texto para la pregunta dada.

Question: has cerebeller hemorrhage progressed?

Expected Answer: no significant change in the size Predicted

Predicted Answer: no significant change in the size Predicted of the right cerebellar hemorrhage

Exact Match: 0

F1 Score: 0.7142857142857143

Figura 18. Ejemplo de una predicción parcialmente correcta (expandida) dada por el modelo QA

Ambas métricas comentadas son fundamentales para evaluar el rendimiento de los modelos tanto en su etapa de entrenamiento cómo en la validación, dado que otorgan una visión general de la precisión de las respuestas pudiendo así compararla entre distintos modelos. Queda por comentar la importancia de la función de pérdidas durante el proceso de entrenamiento y que opciones de esta tercera métrica vamos a implementar en nuestros modelos.

Las funciones de pérdidas son ecuaciones matemáticas utilizadas para calcular en cuánto se desvían las predicciones realizadas por el modelo de los valores reales. Si las pérdidas son elevadas están indicando que el modelo está generando predicciones con un error significativo, mientras que cuánto más bajas sean las pérdidas más precisas serán las respuestas. El objetivo es minimizarlas lo máximo posible.

Internamente, el modelo utiliza la función de pérdidas para entrenar los parámetros como los pesos o sesgos, teniendo una impacto significativo en el proceso de descenso del gradiente. Existen diferentes funciones de pérdidas para usar en el entrenamiento de los modelos. Cada una de ellas implementa un método distinto para penalizar los errores del modelo, por lo que es

importante conocer sus ventajas y desventajas a la hora de elegir la más acertada. En este trabajo se va a probar la entropía cruzada y su evolución, la *Focal Loss*.

La entropía cruzada (en inglés, *Cross Entropy Loss*) es una función de pérdidas usada tradicionalmente en tareas de clasificación. Cómo su nombre indicia se basa en el concepto estadístico de la entropía y cuantifica el grado de incertidumbre presente en el valor predicho por el modelo. La Ecuación 3 muestra la relación entre la clase real de los datos (p) y la probabilidad predicha por el modelo de que pertenezcan a esa clase (q). En los modelo QA esta clasificación es binaria dado que se determina para cada token del texto si pertenece a la clase Respuesta o la clase No-Respuesta.

$$\boldsymbol{H(p,q) = -\sum_x p(x)\, log(\, q(x)\,)} \quad \textbf{(2)}$$

A pesar de ser una función de pérdidas con buenos resultados, tiene el problema de que puede heredar un sesgo si existe un desequilibrio entre las proporciones de las clases, pero ese no es nuestro caso al disponer únicamente de dos clases que van a estar presentes en todos los ejemplos. Otro inconveniente de la misma es que no distingue entre ejemplos difíciles y fáciles. Los ejemplos difíciles son aquellos en los que el modelo comete repetidamente grandes errores, mientras que los ejemplos fáciles son aquellos que se clasifican fácilmente. Dado que se desconoce cuántas de las preguntas y respuestas de nuestra base de datos se pueden clasificar como fáciles o difíciles, esto sí que podría ser un inconveniente para el entrenamiento de nuestro modelo.

La función de pérdidas *Focal Loss* se centra justo en este asunto, dado que se centra en aquellos ejemplos en los que el modelo se equivoca en lugar de en los que sí puede predecir con seguridad, garantizando que las predicciones sobre ejemplos difíciles mejoren con el tiempo al no estar confiándose demasiado con los fáciles. En la Ecuación 3 se muestra cómo se calculan las pérdidas con esta función. En ella se observa como con el uso de un parámetro gamma, se controla la concentración y el peso de las muestras. A mayor gamma, mayor peso para las predicciones inciertas.

$$\boldsymbol{Focal\ Loss\ (p_t) = -\sum_t (1-p_t)^{\gamma}\, log(p_t)} \quad \textbf{(3)}$$

Una vez conocidas que opciones disponemos para la elección de la función de pérdidas para la creación del modelo y las formas existentes para medir su precisión en las respuestas, en el siguiente capítulo se van a crear diferentes modelos analizando los factores involucrados y como afectan a los resultados que otorgan los modelos.

Capítulo 4. Resultados

Con el código para crear el modelo listo y el *dataset* de ejemplo cargado se inicia el proceso de búsqueda del mejor modelo posible, es decir, aquel que tenga una mayor exactitud con el conjunto de datos de validación. Para ello se van a ir variando diferentes aspectos del proceso de creación cómo se ha comentado en el capítulo anterior. Estas variaciones se pueden agrupar en tres categorías en función a que parte del proceso afectan: la variación de los hiperparámetros del entrenamiento del modelo, la variación del modelo utilizado como base y la variación en la forma de tokenizar.

La búsqueda del mejor modelo se va a hacer de forma secuencial en las tres categorías mencionadas, es decir, primero se variarán únicamente los hiperparámetros del entrenamiento hasta encontrar la combinación que mejor modelo generan para después usar esa configuración para buscar que modelo base es el más adecuado, y posteriormente, la forma de tokenizar óptima.

4.1 Variación de los hiperparámetros del entrenamiento

En total vamos a estudiar la influencia de seis "hiperparámetros" en nuestro proceso de entrenamiento del modelo. Estos hiperparámetros son los siguientes:

1. **Épocas** (*Epochs*): Representan la cantidad de veces que el modelo iterará completamente los datos de entrenamiento.
2. **Tasa de Aprendizaje** (*Learning Rate*): Determina el tamaño de los pasos que el algoritmo de optimización da para ajustar los pesos del modelo en cada iteración.
3. **Tamaño de Lote** (*Batch Size*): Indica el número de ejemplos utilizados durante el entrenamiento para cada paso de optimización.
4. **Tasa de Programación del Aprendizaje** (*Schedular Rate*): Establece la magnitud de las actualizaciones de los parámetros durante el proceso.
5. **Decaimiento de Pesos** (*Weight Decay*): Agrega un término a la función de pérdidas para penalizar los valores grandes en los pesos del modelo.
6. **Función de Pérdidas** (*Loss Function*): Mide la discrepancia entre la predicción del modelo y respuesta correcta. Existen diferentes funciones para utilizar.

La configuración inicial de los hiperparámetros es la que se muestra en la Tabla 1. Se han elegido estos valores al considerarse apropiados para el *dataset*, capacidad de almacenamiento y procesado del GPU o el tipo de modelo a utilizar, entre otras consideraciones.

Epochs	Learning Rate	Batch Size	Schedular Rate	Weight Decay	Loss Function
4	2,00E-05	16	0,9	2,00E-02	focal (1)

Tabla 1. Configuración inicial de los parámetros

En un primer momento, se van a iterar 4 veces los datos de entrenamiento al considerarse un número de épocas lo suficiente elevado para observar el desempeño inicial del modelo o si hay algún indicio de sobreajuste o subajuste, pero a la vez lo suficientemente bajo para destinar inicialmente mucho tiempo al entrenamiento sin conocer correctamente como va a resultar este. El *learning rate* por su parte tiene un valor pequeño para evitar movimientos demasiado bruscos en la optimización mientras que el tamaño de lote también es pequeño debido a las limitaciones en la memoria del GPU durante el entrenamiento al estar usando los aceleradores gratuitos, como se ha comentado anteriormente .

Por otra parte, la tasa de programación es de 0,9 indicando que en cada época la tasa de aprendizaje se reducirá un 10%, permitiendo de esta forma un ajuste gradual de la misma a medida que el modelo encuentra una solución óptima. El decaimiento de pesos por su parte también tiene inicialmente un valor pequeño para prevenir el sobreajuste.

Finalmente, se va a usar de base la función de pérdidas *Focal Loss* con gamma 1, por su habilidad para centrarse en las muestras más difíciles. El hiperparámetro gamma tiene un valor inicial de 1 para no reducir drásticamente el impacto de las muestras más fáciles.

Con los valores de los hiperparámetros del entrenamiento comentados se crea el modelo de preguntas y respuestas. En la Tabla 2 se muestra el rendimiento de este primer modelo a lo largo de las 4 épocas. La columna de *Train Accuracy* (Tr. Acc) hace referencia a la precisión del modelo con los datos de entrenamiento mientras que la de *Train Loss* (Tr. Loss) representa la pérdida del modelo para el entrenamiento también. La columna de *Validation Accuracy* (Val. Acc) muestra la precisión del modelo para el conjunto de datos de validación.

EPOCH 1			EPOCH 2			EPOCH 3			EPOCH 4		
Tr. Acc	Tr. Loss	Val. Acc	Tr. Acc	Tr. Loss	Val. Acc	Tr. Acc	Tr. Loss	Val. Acc	Tr. Acc	Tr. Loss	Val. Acc
0,3548	2,4700	0,5229	0,5497	1,4451	0,5659	0,6676	1,0418	0,5752	0,7563	0,7670	0,5830

Tabla 2. Resultados de la configuración inicial de los hiperparámetros

Ambas medidas de precisión representan el *Exact Match* del modelo. En esta etapa inicial de creación del modelo usaremos únicamente este indicador de precisión, ya que cómo se verá más adelante aquellos modelos con mayor *Exact Match* generalmente suelen ser también los que mayor puntuación F1 tienen.

De esta tabla se obtienen dos conclusiones que se van a mantener bastante constantes en el resto de pruebas en la creación del modelo. La primera es que la precisión con los datos de entrenamiento mejora considerablemente con el paso de las épocas, pasando de acertar un 35,45% de las respuestas en la primera época a un 75,36% en la cuarta. Esta mejora también se observa en la disminución en las pérdidas.

La otra observación es que para el caso de la precisión en los datos de validación no ocurre lo mismo, ya que empieza en un buen 52,29% de aciertos y tras tres épocas solo alcanza un 58,30% de acierto en las preguntas. Es por esto por lo que se va a intentar buscar aquella combinación de valores para los hiperparámetros que mejoren esta precisión, ya que al fin y al cabo esta muestra como de generalizable es el modelo para nuevas entradas.

En la Figura 19 se observa la evolución de la precisión del modelo para los datos de entrenamiento y sus pérdidas durante la primera época. La primera tiene una tendencia ascendente mientras que la segunda es exponencialmente negativa. Esto puede indicar que de disponer de una base de datos mayor quizás se podrían alcanzar mejores precisiones.

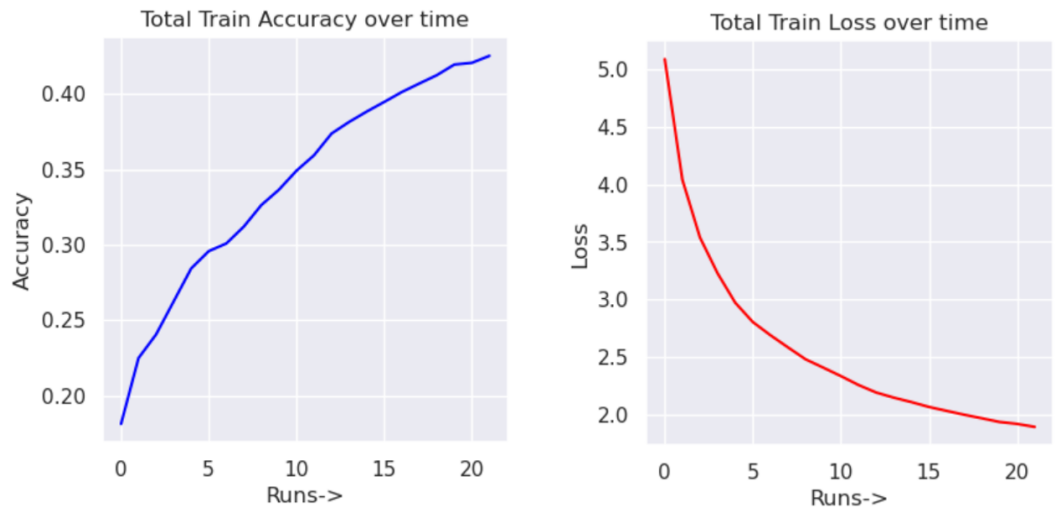

Figura 19. Evolución del *Train Accuracy* y *Train Loss* en el transcurso de la primera época

En la Tabla 3 muestra las distintas configuraciones que se han probado para encontrar los valores de los parámetros que crean el modelo de mayor precisión. En total se han creado 17

modelos diferentes. Aquellos parámetros que varían su valor de un modelo a otro están remarcados de color en la tabla para facilitar la lectura y el entendimiento de esta.

	PARÁMETROS						
MODELO	**Epochs**	**Learning Rate**	**Batch Size**	**Schedular Rate**	**Weight Decay**	**Loss f.**	**Val. Acc última Epoch**
1	4	2,00E-05	16	0,9	2,00E-02	focal (1)	0,5830
2	4	4,00E-05	16	0,9	2,00E-02	focal (1)	0,5841
3	4	8,00E-05	16	0,9	2,00E-02	focal (1)	0,5750
4	4	1,00E-05	16	0,9	2,00E-02	focal (1)	0,5380
5	4	4,00E-05	16	0,9	4,00E-02	focal (1)	0,5458
6	4	4,00E-05	16	0,9	1,00E-02	focal (1)	0,5822
7	4	4,00E-05	8	0,9	2,00E-02	focal (1)	0,5419
8	4	4,00E-05	32	0,9	2,00E-02	focal (1)	sin memoria
9	4	4,00E-05	16	1	2,00E-02	focal (1)	0,5812
10	4	4,00E-05	16	0,8	2,00E-02	focal (1)	0,5796
11	4	4,00E-05	16	0,99	2,00E-02	focal (1)	0,5726
12	4	4,00E-05	16	0,95	2,00E-02	focal (1)	0,5695
13	8	4,00E-05	16	0,9	2,00E-02	focal (1)	0,5820
14	8	4,00E-05	16	0,9	1,00E-02	focal (1)	0,5751
15	8	4,00E-05	16	1	1,00E-02	focal (1)	0,5976
16	8	4,00E-05	16	1	1,00E-02	Cross Entropy	0,5757
17	8	4,00E-05	16	1	1,00E-02	focal (2)	0,5730

Tabla 3. Configuración de los parámetros en las diferentes pruebas y la *Validation Accuracy* resultante en la cuarta época de la prueba

El primer parámetro en estudiar su efecto fue la tasa de aprendizaje. Al duplicarlo, la precisión aumentó unas centésimas; y al volver a duplicarlo disminuyó, quedándose por tanto con un valor de 4,00E-05. Con el decaimiento de pesos ocurrió lo contrario, al aumentarlo disminuyó la precisión. Se probó entonces a disminuirlo, pero la precisión también lo hizo, aunque muy ligeramente; así que se optó por dejarlo momentáneamente por su valor inicial a la espera de la posibilidad de que al cambiar otro parámetro una disminución del *weight decay* fuera positiva.

A continuación, se varío el tamaño de los lotes disminuyéndolos a la mitad resultando en una de las peores precisiones del modelo hasta la fecha. Se intentó entonces fijar el tamaño de los lotes en 32 pero debido a las limitaciones de espacio impuestas por el entorno de ejecución y acelerador de GPU de Kaggle no se dispuso de memoria suficiente para entrenar el modelo. Es por tanto que el *batch size* óptimo resultó ser el inicial de 16.

El siguiente parámetro del que se estudió su influencia a la hora de crear el modelo fue la tasa de programación del aprendizaje. En el modelo 9 se probó con un valor de 1, significando esto que no habrá actualizaciones de la tasa de aprendizaje en las diferentes operaciones, y se obtuvo una precisión mínimamente inferior a la de los otros modelos. Se probaron otros valores resultando en que la mejor tasa es la de 0,9.

En todas estas pruebas las conclusiones sacadas de la prueba piloto se repetían: la precisión en el entrenamiento empieza baja y aumenta rápidamente entre épocas, habiendo modelos con exactitudes del 85%; mientras que la precisión en la validación siempre empieza alrededor del 50% y se ha conseguido aumentar tras 4 iteraciones hasta el 58,41%. Es por esto por lo que en los últimos modelos se ha duplicado el número de épocas para ver si se consigue aumentar la *accuracy* o se mantiene esta tendencia. Eso se puede observar en la Tabla 4.

MODELO	Val. Acc Epoch 4	Val. Acc Epoch 5	Val. Acc Epoch 6	Val. Acc Epoch 7	Val. Acc Epoch 8
13	0,5554	0,5687	0,5695	0,5773	0,5820
14	0,5656	0,5726	0,5742	0,5843	0,5751
15	0,5843	0,5929	0,5859	0,5703	0,5976
16	0,5781	0,5929	0,5703	0,5671	0,5757
17	0,5578	0,5703	0,5828	0,5664	0,5730

Tabla 4. *Validation Accuracy* para los modelos entrenados en más de cuatro épocas

La tabla muestra como el mejor modelo es el 15 con una precisión en las respuestas del 59,76%. También muestra como aumentar las épocas, y por tanto el tiempo necesario para entrenar el modelo, no ha supuesto una gran diferencia. En el caso de la *train accuracy* sí que se notan estas iteraciones extras dado que ahora todos los modelos tienen una precisión superior al 90% lo que podría indicar que el modelo está realizando un sobreajuste sobre la muestra de los reportes.

El último parámetro que queda por variar es la definición de la función de pérdidas. Dado que entre los conjuntos de datos con los que se entrenan los modelos se han podidos observar algunos contextos muy cortos que prácticamente representan en su totalidad la respuesta a la preguntada formulada tiene sentido haber optado por utilizar la *Focal Loss* con gamma 1, al reducir el impacto de estos ejemplos sencillos de clasificar en comparación con los ejemplos más largos y complejos.

Aun así, se va a probar si hay otra función que beneficie más a la precisión del modelo. Por una parte, se va a probar el efecto de duplicar la gamma de la *Focal Loss*, aumentando el peso para las predicciones inciertas; y por otro lado se va a probar con la función de pérdidas de la entropía cruzada. Ambos casos resultaron en una menor *accuracy* por lo que la hipótesis inicial era correcta.

En conclusión, y tras todas las pruebas, el mejor modelo resultante es el 15 que cuyos parámetros tienen los valores mostrados en la Tabla 5. Este modelo tiene una precisión exacta para el conjunto de datos de validación de 59,76% en comparación con el 58,30% del modelo inicial.

Modelo	Epochs	Learning Rate	Batch Size	Schedular Rate	Weight Decay	Loss Function
15	8	4,00E-05	16	0,9	1,00E-02	focal (1)

Tabla 5. Configuración final de los parámetros

Esta variación tan pequeña se debe a dos factores: la acertada elección de los valores iniciales y la poca variación en la precisión al aumentar las iteraciones de los datos. Dada que para variar esta tendencia habría qua variar o contar con otro dataset, en la siguiente sección se va a estudiar si la elección de otro modelo base puede mejorar esta precisión.

4.2 Variación del modelo base

El modelo que se ha usado como base para crear los diferentes modelos de las pruebas de los hiperparámetros del entrenamiento ha sido el *bert-base-uncased* [15]. Cómo se ha comentado anteriormente en el trabajo, la familia BERT ha sido entrenada para tareas de preguntas y respuestas usando el *dataset* SQuAD.

BERT fue publicado originalmente en dos versiones de diferentes tamaños (*base* y *large*), tanto para entradas de texto con formato (*cased*) como sin formato (*uncased*). Se entiende por texto con formato a aquel que tiene mayúsculas o signos diacríticos como tildes o diéresis. Poco después, se publicaron versiones *cased* y *uncased* para chino y "multilingües", apto para más de 100 idiomas diferentes. Posteriormente se lanzaron otros 24 modelos más pequeños.

Se eligió el modelo *base-uncased* como base para la creación de nuestro modelo propio al tratarse del modelo en inglés más básico de los existentes, por su tamaño reducido y al trabajar con texto sin formato. A parte, se han utilizado la configuración de los hiperparámetros resultantes de la sección anterior, recogidos en la Tabla 5.

Para encontrar el mejor modelo base disponible, se va a mantener el uso de 4 épocas únicamente debido a que como se ha visto anteriormente el aumento de estas favorece mayoritariamente a la precisión en el entrenamiento y no tan notablemente a la validación. Los

resultados de esta primera prueba con el modelo base están recogidos en la Tabla 6, dónde se muestra que la *Exact Match* del modelo creado es del 57,10%.

EPOCH 1			EPOCH 2			EPOCH 3			EPOCH 4		
Tr. Acc	Tr. Loss	Val. Acc	Tr. Acc	Tr. Loss	Val. Acc	Tr. Acc	Tr. Loss	Val. Acc	Tr. Acc	Tr. Loss	Val. Acc
0,4302	1,7569	0,5539	0,6355	0,9333	0,5492	0,752	0,569	0,5812	0,8225	0,3756	0,5710

Tabla 6. Resultados de la configuración de los hiperparámetros óptimos con el modelo base

El siguiente modelo que se decidió probar fue el *bert-large-uncased*, una versión de mayor tamaño que el modelo BERT inicial. La versión *large* tiene alrededor de 340 millones de parámetros frente a los 110 millones del *base*. Estos parámetros son los valores ajustables que el modelo aprende durante el proceso de entrenamiento por lo que a mayor volumen de este mayor éxito con tareas complejas. La creación de nuestro modelo de QA a partir de la versión *large* no fue posible debido a la falta de espacio en el acelerador de GPU de Kaggle usado en el proceso.

Siendo conocedores de la limitación existente con el tamaño de los modelos, se decidió optar únicamente por aquellos con menor volumen de parámetros. Es por esto por lo que el siguiente modelo a usar de base fue el *bert-base-uncased*, igual a la inicial con la diferencia de que este sí que diferencia entre mayúsculas y demás formatos posibles en las secuencias de entrada. La precisión en la validación de esta prueba fue ligeramente más positiva en las cuatro épocas que el modelo sin formato. Resultando en un 57,89% de accuracy frente al 57,10% previo.

A continuación, se probó a crear un modelo partiendo del *bert-base-multilingual-cased*, dado que a pesar de que los reportes de la base de datos están únicamente en inglés, podría ser interesante el efecto que tiene crear un modelo que además del inglés entienda otro centenar de idiomas. Se ha optado por la versión *cased* del mismo al haberse obtenido mejores precisiones con este tipo de modelos en la prueba anterior.

El modelo resultante de esta prueba tiene una precisión inferior a los anteriores, de 56,95% en la cuarta etapa, por lo que en aquellos casos dónde los contextos a introducir en el modelo QA (los reportes en nuestro caso) estén en un único idioma conocido, es mejor centrarse en modelos preentrenados en ese mismo idioma si existen.

Una vez probados los modelos BERT originales, se van a probar modelos creados a partir de un BERT y que han sido entrenados con diferentes bases de datos para distintas materias. Estos modelos pueden estar creados por organizaciones, tanto públicas como privadas, o por usuarios

particulares, siendo un ejemplo de estos los modelos que se han ido creando a lo largo de este trabajo.

Al estar creando un modelo de preguntas y respuestas para informes radiológicos, aparentemente la opción más idónea sería partir de un modelo entrenado para este tipo de reportes que no haya usado la misma base de datos. De no existir también podrían ser útiles modelos específicos para reportes médicos en general, del ámbito sanitario o incluso relacionado con la biología dado que estos modelos ya dispondrán de tokens específicos para un vocabulario mucho más técnico; facilitando y mejorando el procesamiento y entendimiento de las secuencias de entrada, resultando en una mejor precisión del modelo.

En total se seleccionaron tres modelos existentes que podrían mejorar los resultados de los BERTs básicos. Todos son de libre uso y están disponibles en la web de Hugging Face. El primero de estos es el modelo RaDBERT desarrollado por Stanford AIMI [27]. La universidad de Standford ha creado el centro AIMI (del inglés, *Artificial Intelligence in Medicine & Imaging*) para desarrollar, evaluar y difundir modelos de inteligencia artificial para beneficiar la atención médica de los pacientes.

Este modelo parte del BioBERT, el cual está especializado en terminología biomédica, y ha sido entrenado con diversos *dataset*: uno con importes radiológicos, otro de artículos de la Wikipedia y otro llamado *pubmed*, el cual contiene citas y *abstracts* de literatura biomédica, pero no los *papers* completos [28]. De esta forma se consigue que el modelo conozca la estructura de los reportes y vocabulario específico del tema.

Tras crear nuestro modelo partiendo de RaDBERT se obtiene una precisión del 55,70%; inferior a la conseguida con los modelos generales. Esto puede deberse a diversos factores como a que tipos de informes radiológicos está preentrenado RaDBERT, que modelo inicial BERT se usó para su creación, la cantidad de parámetros de este, etc. La universidad de Standford no ha aportado más información respecto a este tema por lo que no se puede saber a que se debe esta "no" mejora de los resultados.

El siguiente modelo por probar tiene el nombre de ClinicalBERT y ha sido creado por el equipo medicalai en Hugging Face, compuesto por tres usuarios. En este caso, este modelo parte del BERT original y ha sido entrenado con un *dataset* variado, compuesto por un corpus de más de mil millones de palabras, donde se incluyen una gran variedad de enfermedades [29]. El modelo también ha sido *fine-tuned* con más de 3 millones de expedientes de pacientes extraídos de un registro electrónico de salud.

El modelo creado partiendo del ClinicalBERT resultó en un *Exact Match* para los datos de validación del 27,42%, la peor precisión de todos los modelos creados hasta el momento. Esta se

podría deber a diferentes causas. Al no disponer de las bases de datos utilizadas, no se puede saber la calidad de estas, o como ha sido el proceso de entrenamiento, puesto que un mal entrenamiento podría haber creado un modelo sobreajustado para un *dataset* diferente al nuestro, lo que explicaría el mal resultado.

También podría deberse a que el entrenamiento del modelo Clinical se ha realizado sobre un conjunto de datos que no estaba enfocado a la tarea de preguntas y respuestas, pudiendo de esta forma haber hecho perder al modelo la buena resolución de este tipo de tareas que tienen los BERT gracias a su entrenamiento con el *dataset* SQuAD.

El último modelo para probar fue creado por Kirill Trapeznikov, científico de alto nivel de STR. El modelo recibe el nombre de *biobert_v1.1_pubmed_squad_v2* [30], y es de todos los probados el que mejor precisión ha obtenido con un 66,71% tal y como se muestra en la Tabla 7.

MODELO	Tr. Acc	Tr. Loss	Val. Acc
bert-base-uncased	0,8225	0,3756	0,5710
bert-large-uncased	sin memoria		
bert-base-cased	0,8067	0,4176	0,5789
bert-base-multilingual-cased	0,805	0,4332	0,5695
StanfordAIMI/RadBERT	0,7746	0,4937	0,5570
medicalai/ClinicalBERT	0,3025	2,2162	0,2742
ktrapeznikov/biobert_v1.1_pubmed_squad_v2	0,8909	0,2130	0,6671

Tabla 7. Modelos utilizados como base y sus resultados

Este mejor desempeño se debe a cómo está creado el mismo, recogiendo las conclusiones obtenidas de los otros dos modelos creados por Standford y el grupo medicalai. Para empezar el modelo parte de un BioBERT creado a partir de la versión 1.1 del conjunto de datos *pubmed*, similar a como se creó el RadBERT; para a continuación realizarle un ajuste fino con el *dataset* SQuAD v2.

Lo que se consigue de esta forma es que un modelo especializado para un vocabulario más técnico siga siendo capaz de resolver tareas de preguntas y respuestas, evitando de esta forma lo que le sucedió al ClinicalBERT. Además, es un modelo *cased*, los cuales ya habíamos comprobado con los BERT originales que eran con los que mejores resultados obteníamos.

Es por todo esto, que se ha decido usar cómo modelo inicial para la creación de nuestro modelo de QA el *biobert_v1.1_pubmed_squad_v2*. Para crear el modelo final, falta únicamente analizar el efecto que tiene la forma de tokenizar en la precisión del modelo, tal y como se va a comentar la siguiente sección.

4.3 Variación en la tokenización

Como se ha comentado anteriormente, a la hora de crear el modelo no se puede elegir el algoritmo de tokenización, al ir ligado al modelo que se utilice como base, pero sí que se pueden variar diferentes hiperparámetros implicados en este proceso. En el caso de los modelos BERT este algoritmo es el WordPiece.

De entre los diferentes parámetros que se utilizan para configurar el proceso de tokenización, en este trabajo se va a estudiar cómo afectan los siguientes ocho:

1. **Función** (*Function*): Determina la función específica de tokenización que se va a utilizar.
2. **Longitud Máxima** (*Max Length*): Indica la longitud máxima permitida para una secuencia de tokens después de la tokenización.
3. **Truncamiento** (*Truncation*): Establece si las secuencias que exceden la longitud máxima permitida serán recortadas/reducidas o no.
4. **Relleno** (*Padding*): Señala si hay que agregar tokens de relleno para igualar las longitudes de las secuencias inferiores a la longitud máxima especificada.
5. **Limpiar el Texto** (*Clean Text*): Indica si se ha de "limpiar" la secuencia antes de la tokenización. Generalmente se entiende por "limpiar" eliminar caracteres especiales, emojis, puntuación, etc.
6. **Convertir a Minúsculas** (*Do Lower Case*): Determina si hay que convertir el texto a minúsculas antes de la tokenización.
7. **Lado del Relleno** (*Padding Side*): Establece en qué lado se deben agregar los tokens de relleno en caso de que se utilicen. Puede ser a la izquierda o a la derecha.
8. **Desplazamiento** (*Stride*): Controla la cantidad de tokens que se superponen entre la secuencia truncada y la desbordada, en el caso de que la secuencia de entrada sea superior a la longitud máxima.

La configuración inicial de los parámetros, la cual es la que se había usado en todas las pruebas realizadas hasta ahora, es la que se muestra en la Tabla 8. La mayoría de estos valores han sido elegidos al ser los típicos o los que vienen por defecto al usar la función *BertTokenizeFast*.

Function	Max Length	Truncation	Padding	Clean Text	Do Lower Case	Padding Side	Stride
BertTokenizerFast	512	true	true	true	true	right	0

Tabla 8. Configuración inicial de los parámetros de la tokenización

Se ha decidido usar esta función desde la creación del primer modelo al tratarse del tokenizador para modelos BERT más optimizado. También se fijó el máximo de tokens por secuencia en 512. Aquellas secuencias que superaran este umbral han sido truncadas sin ningún solape entre tokens, mientras que las que no lo alcanzaban han sido rellenadas con tokens por la derecha de estas. Además, las secuencias originales han sido preprocesadas, "limpiándolas" y pasándolas completamente a minúsculas. Esta configuración contrasta con el hecho de que los modelos que mejor funcionaban en la sección anterior eran aquellos que sí distinguían entre mayúsculas y minúsculas.

Dado que este conjunto de parámetros son los últimos que faltan por determinar a la hora de configurar el mejor modelo QA posible, para los modelos que se creen en esta sección además de utilizar el *Exact Match* también comentaremos su *F1 Score* tal y como se muestra en la Tabla 9. En ella se puede observar la puntuación F1 para el conjunto de datos de validación. También se mantiene el EM tanto para el entrenamiento como para la validación para comparar con los modelos anteriores. En esta tabla se muestran estas métricas para el modelo generado a partir de los parámetros iniciales de tokenización y partiendo de los hiperparámetros de entrenamiento y modelos iniciales definidos en las secciones anteriores.

EPOCH 1			EPOCH 2			EPOCH 3			EPOCH 4		
Tr. Acc	Val. Acc	F1 Score	Tr. Acc	Val. Acc	F1 Score	Tr. Acc	Val. Acc	F1 Score	Tr. Acc	Val. Acc	F1 Score
0,5412	0,6429	0,6703	0,7287	0,6007	0,6357	0,8296	0,6375	0,6460	0,8897	0,6632	0,7018

Tabla 9. Resultados de la configuración inicial de la tokenización para el mejor modelo inicial

En la tabla se observa como la precisión para esta primera configuración de la tokenización aumenta hasta un 70,18% si tenemos en cuenta la precisión general de las repuestas en vez de considerar únicamente las respuestas exactas. En nuestro caso, para los diferentes modelos que evaluemos siempre va a ocurrir que la puntuación F1 sea mayor (o igual) que EM. Lo que sí que será diferente entre modelos es cuanto de mayor es esta diferencia entre ambas métricas.

En la Tabla 10 se observa la puntuación para los diferentes modelos que se han creado y cuál ha sido la configuración de la tokenización usada en cada uno de ellos. Al igual que en la Tabla 3, aquellos parámetros que varían de un modelo a otro están sombreados para mostrar el efecto de dicha variable tiene sobre la puntuación.

	PARÁMETROS									
MODELO	**Function**	**Max Length**	**Truncation**	**Padding**	**Clean Text**	**Do Lower Case**	**Padding Side**	**Stride**	**EM**	**F1**
1	BertTokenizerFast	512	true	true	true	true	right	0	0,6632	0,7018
2	BertTokenizerFast	256	true	true	true	true	right	0	0,6500	0,7132
3	BertTokenizerFast	256	false	false	true	true	right	0	error	
4	BertTokenizerFast	512	true	true	false	true	right	0	0,6281	0,6719
5	BertTokenizerFast	512	true	true	true	false	right	0	0,6445	0,6982
6	BertTokenizerFast	512	true	true	true	true	left	0	0,6195	0,6433
7	BertTokenizerFast	512	true	true	true	true	right	100	0,6523	0,6855
8	BertTokenizerFast	256	true	true	true	true	right	100	0,6625	0,7205
9	BertTokenizer	256	true	true	true	true	right	100	0,6398	0,7151

Tabla 10. Configuración de los parámetros de la tokenización en las diferentes pruebas y la *Exact Match* y *F1 Score* resultante en la cuarta época de la prueba

Para la creación del segundo modelo se redujo a la mitad la longitud máxima hasta 256 tokens posibles resultando en un empeoramiento en la EM, pero una mejora en la puntuación F1, alcanzando un 71,32%. A pesar de que con secuencias más pequeñas ha mejorado la precisión del modelo también se probó cual sería el efecto al realizar lo contrario, es decir, tokenizar secuencias más largas. La Figura 20 muestra la distribución de las longitudes de los contextos de dónde el modelo extrae las respuestas.

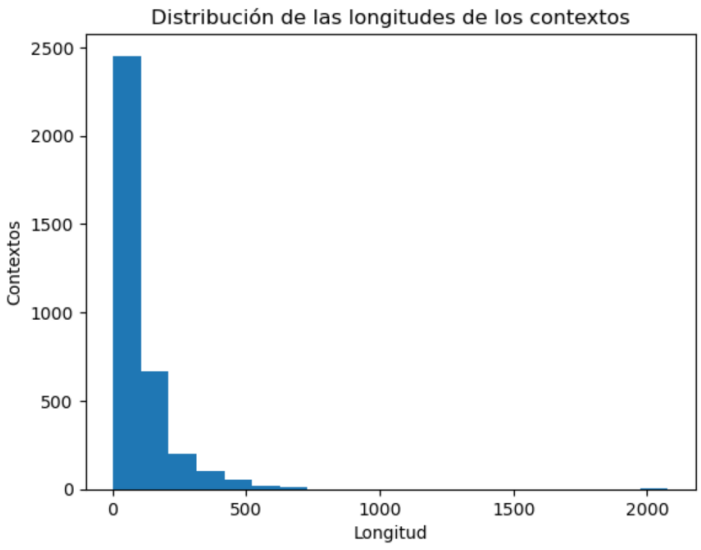

Figura 20. Distribución de las longitudes de los contextos

En un primer momento se eligió la longitud máxima de las secuencias en 512 al ser más elevado que la mayoría longitudes de los contextos. Para la creación de este tercer modelo, se decidió que era más lógico eliminar el truncamiento antes que duplicar la longitud máxima, puesto que marcaría más la diferencia. De esta forma, se introducirán en el modelo las secuencias de

texto completas, sin ser compartimentadas. Con esta configuración ha sido imposible realizar la tokenización.

La causa de esto se encuentra en la forma en la que se ha programado la creación y el entrenamiento de los modelos puesto que se ha configurado para que el tamaño de las secuencias con las que trabaja el modelo tenga que ser idénticas. Es por esto por lo que, para poder crear correctamente los modelos, todas aquellas secuencias que sean superiores a la longitud máxima tendrán que ser truncadas; mientras que las que no la alcancen necesariamente tendrán que ser rellenadas.

La siguiente comprobación fue el efecto que tiene en la precisión el hecho de limpiar el texto de caracteres especiales o innecesarios. En este caso tanto la EM como la F1 disminuyeron, resaltando entonces la importancia de este paso previo a la tokenización a la hora de obtener mejores resultados. En línea con esto, si se opta por crear el modelo sin convertir el texto completo a minúsculas, la precisión de las respuestas disminuye también.

En la sexta prueba se varió la ubicación del relleno a la izquierda, quedando cómo se muestra en la Figura 21. Este cambió tampoco mejoró la precisión del modelo dado que disminuyó alrededor de un 5% hasta una puntuación F1 de 64,33%, por debajo de la EM de los mejores modelos hasta la fecha.

```
Secuencia a tokenizar: '[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[SEP][CLS] is there any significant change from prior visit? [SEP] final report chest
single ap film : history : icd placement and shortness of breath. there is a right
sided dual chamber icd with leads unchanged in location in this single view compared
with the prior film of [ * * 2101 - 5 - 25 * * ]. no pneumothorax. there is
cardiomegaly with upper zone redistribution, bilateral pleural effusions and
associated bibasilar atelectases. consolidation at the lung bases cannot be ruled
out.'
```

Figura 21. Ejemplo de una secuencia rellenada por la izquierda

Hasta ahora ninguno de los modelos creados hacía uso del desplazamiento, al tener todos *Stride*=0, por lo que en el siguiente modelo se va a estudiar el efecto de incluirlo en el tokenizador. Se ha optado por usar un valor de 100 tokens, representando alrededor de un solape del 20 % entre secuencias. El resultado volvió a ser negativo. Ambas métricas empeoraron respecto al mejor modelo creado.

La disminución es más crítica en la puntuación F1 mientras que en la Coincidencia Exacta se obtiene la segunda mejor puntuación hasta la fecha con un 65,23%. Este buen resultado en una de las dos métricas fue el detonante para comprobar que ocurriría con un mayor solape. Para ello se mantuvo el valor del desplazamiento, pero se redujo a la mitad la longitud máxima, convirtiéndose entonces en un solape efectivo de casi el 40% de la longitud de la secuencia. Este modelo mejoró la EM del modelo anterior, quedando detrás del primero por un margen pequeño. Por otra parte, la puntuación F1 se alzó hasta un 72,05% de tokens correctos en las predicciones, siendo esta la mejor precisión para un modelo creado hasta el momento.

Finalmente, queda por ver el efecto que tiene variar la función del Tokenizador. En vez de usar la versión rápida y optimizada se va a usar la función *BertTokenizer*. Esta versión más general de la función es más antigua y estable por lo que puede ser interesante la forma en la que afecte en la tokenización. Este modelo obtiene muy buenos resultado en la puntuación F1, de un 71,51% pero no tanto para la EM que desciende hasta un 63,88%; quedándose por tanto el anterior modelo a este cómo el mejor modelo de todos los creados.

Este modelo final tiene una *Exact Match* de 66,25% y una puntuación F1 de 72,05%. En la Tabla 11 se puede ver cómo queda la configuración óptima de los parámetros de tokenización utilizados. Se puede observar cómo los parámetros óptimos no son muy diferentes a los elegidos inicialmente.

Function	Max Length	Truncation	Padding	Clean Text	Do Lower Case	Padding Side	Stride
BertTokenizerFast	256	true	true	true	true	right	100

Tabla 11. Configuración final de los parámetros de la tokenización

A la hora de tokenizar se ha demostrado que es importante limpiar y pasar a minúsculas las secuencias antes de trabajar con ellas y que el tamaño de las secuencias ha de ser el mismo y por ello es importante truncar y rellenar las secuencias. Además, el uso del solapamiento para aquellos casos donde haga falta truncar las secuencias mejora los resultados del modelo dado que tiene un mejor contexto para entender y encontrar la respuesta.

Capítulo 5. Conclusiones

Tras el estudio de cómo influyen los diferentes parámetros involucrados en la creación de un modelo de preguntas y respuestas se han podido determinar cuál es la combinación óptima para el caso de los reportes radiológicos. En este proceso de creación se ha observado como la precisión del modelo en el conjunto de validación no varía mucho entre épocas por lo que la elección de los parámetros iniciales es crucial para unos buenos resultados. Para la elección de los parámetros del entrenamiento y la tokenización es importante conocer los parámetros a configurar y en que rango de valores podría encontrarse su valor óptimo.

También se ha visto cómo la mayor influencia en la calidad del modelo viene dada por la elección del modelo inicial. Esta conclusión tiene sentido dado que este trabajo parte de la idea de crear un modelo QA propio partiendo de un *dataset* especifico para la tarea a realizar, aunque de dimensiones reducidas, favoreciendo de esta forma la implementación de esta tecnología por organizaciones sin tantos recursos o datos. Es por esto por lo que al estar creando nuestro modelo mediante la realización de un *fine tuning* a otro modelo en vez de crear un modelo de cero, lo cual además de necesitar grandes volúmenes de datos necesitaría una capacidad de procesamiento considerable, tiene sentido que sea el factor que más incluya.

En el trabajo se ha visto como con la creación del modelo partiendo de uno general se obtienen resultados decentes pero que existen dos aspectos claves a la hora de elegir el mejor modelo base disponible para la creación del nuestro. El primero de estos aspectos es que el modelo contenga vocabulario especifico para el ámbito en el que se va a utilizar esta tecnología. En el caso de este trabajo el vocabulario ha sido del sector biomédico, pero en función del uso del modelo este vocabulario podría ser jurídico, legislativo, académico, etc. El segundo aspecto es elegir un modelo que esté entrenado para resolver la tarea específica en la que se va a utilizar. En el caso de la creación de modelos para preguntas y respuestas es buena opción elegir modelos que tomen de base algún BERT o que hayan sido entrenados con el conjunto de datos SQuAD o parecidos.

De esta forma en el trabajo se ha hablado de las distintas opciones que existen para el uso de modelos de QA por parte de organizaciones, habiéndose explicado cómo se podría implementar en el ámbito médico mediante el uso de reportes radiológicos. Esta explicación puede servir a su vez de guía para que cualquier otro tipo de empresa, organización o individuo que quiera crear un modelo de este tipo para un uso concreto pueda hacerlo replicando lo indicado.

Capítulo 6. Bibliografía

[1] J. McCarthy, «WHAT IS ARTIFICIAL INTELLIGENCE?,» 2007.

[2] IBM, «What is machine learning? | IBM,» 2023. [En línea]. Available: https://www.ibm.com/topics/machine-learning. [Último acceso: 26 junio 2023].

[3] H. Jair Escalantes y E. F. Morales, «Chapter 6 - A brief introduction to supervised, unsupervised, and reinforcement learning,» de *Biosignal Processing and Classification Using Computational Learning and Intelligence*, Academic Press, 2022, pp. 111-129.

[4] A. P. López-Monroy y J. García-Salinas, «Chapter 9 - Neural networks and deep learning,» de *Biosignal Processing and Classification Using Computational Learning and Intelligence*, Academic Press, 2022, pp. 177-196.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser y I. Polosukhin, «Attention is all you need,» *Advances in neural information processing systems*, vol. 30, 2017.

[6] A. Karpathy, «Andrej Karpathy blog,» 21 Mayo 2015. [En línea]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/. [Último acceso: 15 Agosto 2023].

[7] J. Alammar, «The Illustrated Transformer,» 27 Junio 2018. [En línea]. Available: https://jalammar.github.io/illustrated-transformer/. [Último acceso: 14 Agosto 2023].

[8] J. Vig, «GPT-2: Understanding Language Generation through Visualization,» *Towards Data Science*, 2019.

[9] M. Saeed, «A Gentle Introduction to Positional Encoding in Transformer Models, Part 1,» *Attention*, 2022.

[10] T. Beysolow II, Applied Natural Language Processing with Python, San Francisco: Apress, 2018.

[11] G. Grefenstette, «Tokenization,» de *Syntactic Wordclass Tagging*, Dordrecht, Springer Netherlands, 1999, pp. 117-133.

[12] Hugging Face, «Summary of the tokenizers,» 2023. [En línea]. Available: https://huggingface.co/docs/transformers/tokenizer_summary#:~:text=More%20specifically%2C%20we%20will%20look,is%20used%20by%20which%20model.. [Último acceso: 20 julio 2023].

[13] Standfor Online, «Stanford CS224N: NLP with Deep Learning | Winter 2019 | Lecture 2 – Word Vectors and Word Senses [Video],» Youtube, 2019.

[14] S. Vivek, «Extractive vs Generative Q&A — Which is better for your business?,» *Towards Data Science*, 2023.

[15] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for,» *CoRR*, vol. abs/1810.04805, 2018.

[16] P. Rajpurkar, J. Zhang, K. Lopyrev y P. Liang, «SQuAD: 100,000+ Questions for Machine Comprehension of Text,» *CoRR*, vol. abs/1606.05250, 2016.

[17] OpenAi, «Pricing,» [En línea]. Available: https://openai.com/pricing. [Último acceso: 5 Agosto 2023].

[18] S. Soni, M. Gudala, A. Pajouhi y K. Roberts, «{R}ad{QA}: A Question Answering Dataset to Improve Comprehension of Radiology Reports,» *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pp. 6250-6259, junio 2022.

[19] Radiological Society of North America (RSNA) and American College of Radiology (ACR), «How to read your radiology report,» 1 junio 2022. [En línea]. Available: https://www.radiologyinfo.org/en/info/article-read-radiology-report. [Último acceso: 12 julio 2023].

[20] A. Johnson , T. Pollard y R. Mark, «MIMIC-III Clinical Database (version 1.4),» *PhysioNet*, 2016.

[21] Haystack, «What is Haystack? | Haystack,» 2022. [En línea]. Available: https://haystack.deepset.ai/overview/intro. [Último acceso: 10 agosto 2023].

[22] Haystack , «Annotation Tool | Haystack Documentation,» 6 agosto 2023. [En línea]. Available: https://docs.haystack.deepset.ai/docs/annotation#user-guide. [Último acceso: 15 agosto 2023].

[23] P. Kanani y M. Padole, «Deep Learning to Detect Skin Cancer using Google Colab,» *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 8, nº 6, 2019.

[24] Google Collaboratory, «Colab subscription pricing,» 10 agosto 2023. [En línea]. Available: https://colab.research.google.com/signup. [Último acceso: 10 agosto 2023].

[25] Kaggle, «Efficient GPU Usage Tips and Tricks,» [En línea]. Available: https://www.kaggle.com/page/GPU-tips-and-tricks. [Último acceso: 7 Agosto 2023].

[26] Cloudera Fast Forward Labs, «Evaluating QA: Metrics, Predictions, and the Null Response,» *NLP for Question Answering*, 2020.

[27] P. Chambon, T. S. Cook y C. P. Langlotz, «Improved fine-tuning of in-domain transformer model for inferring COVID-19 presence in multi-institutional radiology reports,» *Journal of Digital Imaging*, 2022.

[28] National Center for Biotechnology Information, «PubMEd,» [En línea]. Available: https://www.citethisforme.com/cite/sources/websiteautociteeval. [Último acceso: 28 Agosto 2023].

[29] K. Huang, J. Altosaar y R. Ranganath, «ClinicalBERT: Modeling Clinical Notes and Predicting Hospital Readmission,» *CoRR*, vol. abs/1904.05342, 20219.

[30] K. Trapeznikov, «ktrapeznikov/biobert_v1.1_pubmed_squad_v2 · Hugging Face,» [En línea]. Available: https://huggingface.co/ktrapeznikov/biobert_v1.1_pubmed_squad_v2. [Último acceso: 20 Agosto 2023].