eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Revisión de modelos de aprendizaje en el ámbito de pregunta-respuesta

Autor

Iñigo Pikabea Mentxaka

2022

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Revisión de modelos de aprendizaje en el ámbito de pregunta-respuesta

Autor

Iñigo Pikabea Mentxaka

Director

Josu Ceberio Uribe

Resumen

A lo largo de estos últimos años se ha popularizado el uso de aplicaciones capaces de responder a preguntas. En ellas, varias técnicas de aprendizaje profundo han sido implementadas y el rendimiento de estas ha incrementado considerablemente. Cada mes, aparecen modelos capaces de mejorar el rendimiento de los anteriores, lo que dificulta tener una idea adecuada de cada uno de ellos. Por esa razón, este trabajo recoge las técnicas más aplicadas y analiza y resume cómo funcionan este tipo de sistemas. Primero, se explicará la evolución de los modelos de pregunta-respuesta. Después, se analizarán los distintos tipos ya existentes, mostrando cómo estos son evaluados y entrenados. Finalmente, se evaluará experimentalmente el rendimiento de sistemas entrenados con recursos escasos y en un idioma menos hablado, como el euskera, con el fin de observar la adaptación de los mismos.

Palabras clave: *Fine-tuning*, *retiever-reader*, *OpenQA*, BERT, análisis bibliográfico.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

El impacto que podría tener el funcionamiento de un sistema de pregunta-respuesta bien entrenado podría cambiar la forma de investigar, de estudiar y de comprender las cosas. Cada vez más actividades involucran el uso de asistentes virtuales, motores de búsqueda, consultores de manuales y otro tipo de servicios. A la hora de buscar cualquier tipo de información, sistemas como Google o Siri se encargan de responder a la pregunta, ya sea por voz o por texto, y el nivel de exigencia por parte del usuario va en continuo aumento. Por otro lado, los manuales de maquinarias y otros tipos de tecnologías son cada vez más complejos, y un mismo técnico no es capaz de abordar y entender todos los mecanismos y métodos posibles para arreglarlos. También, se investiga cada vez más en todo tipo de temas, como la salud, informática, física, etc, y en muy poco tiempo se publican muchos artículos. Observando lo anterior, un sistema capaz de buscar la información y comprender el texto para concluir y responder a lo deseado resulta muy útil. Sobre lo mencionado anteriormente, un modelo de pregunta-respuesta, o QA (*Question-answering*), podría utilizarse para responder a las consultas relacionadas con los manuales y directamente resolver la petición, sin tener que dedicar tiempo en leer y comprender el manual solicitado en toda su extensión. En investigación, en vez intentar encontrar la respuesta a la duda planteada mediante la búsqueda y la lectura, se le podría preguntar al modelo directamente, y en cuestión de segundos ser respondido con la información solicitada. Como se puede observar, el potencial de un buen uso de sistemas de QA bien entrenados es muy elevado.

Por otro lado, debido al potencial que tienen, se publican nuevos modelos continuamente, lo cual dificulta la comprensión de cada uno de ellos así como la tipología y disponibilidad de los mismos. Por este motivo, el principal objetivo de este trabajo es explicar qué

tipos de modelos existen y cómo funcionan, analizando su comportamiento y realizando experimentos para entender sus prestaciones y limitaciones.

Concretamente, se van a analizar los modelos de aprendizaje profundo en el ámbito de pregunta-respuesta, desde una perspectiva de recuperación de información, o *information retrieval*. Dentro de los sistemas encargados en rastrear y recuperar la información solicitada dentro de un corpus de texto, como podría ser la web o una base de datos, estos modelos son los que componen la última fase y se encargan en devolver la respuesta, pasado un contexto como parámetro. Los modelos estudiados pertenecen al campo de la inteligencia artificial y estos requieren por lo general una fase de entrenamiento para que aprendan las relaciones entre palabras, y, junto a otro tipo de arquitecturas y métodos, como capas de atención y embeddings, son capaces de responder a preguntas con una precisión considerablemente buena. Quedan fuera de este trabajo los modelos con propósitos conversacionales.

El presente documento se organiza de la siguiente manera: Primero, se dará una breve evolución de los sistemas de QA, comentando cuáles fueron los primeros en crearse, la arquitectura que compartían, y cómo poco a poco ha ido reemplazándose por modelos de atención. Además, se comentará el estado del arte, explicando las arquitecturas utilizadas actualmente y los modelos que están generando mayor impacto. En el siguiente capítulo, se analizarán los distintos modelos de aprendizaje existentes, exponiendo las distintas métricas que hay para medir el rendimiento de los modelos, y mostrando las distintas bases de datos que se usan para entrenar a estos. A continuación, se presentará la sección experimental que consta de dos partes. El primero tiene como objetivo estudiar la capacidad de adaptación de modelos cuando se limita el entrenamiento. El segundo estudia la precisión de modelos en euskera, observando las métricas que obtienen en comparación con modelos entrenados en idiomas más extendidos. Finalmente, concluirá el trabajo, indicando las limitaciones de estos modelos y aclarando por donde podría seguir la investigación de cara al futuro.

2. CAPÍTULO

Planificación y objetivos del proyecto

En este capítulo, se presenta la planificación que se ha elaborado para llevar el proyecto a buen puerto. Primero se ha elaborado una planificación, en la cual se han agrupado las actividades a realizar en paquetes de trabajo, y se han estimado las horas que serán necesarias para terminar cada una de ellas. También se han planteado los objetivos y se han analizado los riesgos. Después, se ha hecho un seguimiento y control para ver si todo lo organizado ha ido cumpliéndose, comparando las horas reales con las estimadas.

2.1. Planificación

El trabajo de fin de grado se ha planificado de tal manera que cada semana haya horas dedicadas a la escritura del informe, a la lectura de artículos para comprender el ámbito en el que se está trabajando, y a la programación para la prueba y evaluación de modelos. En esta sección se explicará con mayor detalle cómo ha sido organizada cada tarea para que esta consiga entregarse bien y a tiempo.

El trabajo de fin de grado se realizará en un centro tecnológico, concretamente en Tecnalia. Esto quiere decir que la ejecución del trabajo planificado puede ser alterado por los inconvenientes que puedan surgir en temas relacionados con la petición de recursos, gestión y temas burocráticos no planteados previamente.

2.1.1. Descripción de los objetivos

Se distinguen distintos tipos de objetivos: el principal (objetivo final a cumplir), y objetivos más precisos, de menor alcance, pero vitales para conseguir el objetivo final. Plantear objetivos limitados es útil para no perder la dinámica de trabajo, y mantener la motivación.

- **Objetivo principal:** Analizar distintos modelos en el ámbito de QA y evaluar su rendimiento.
- **Objetivo 1:** Extraer información de artículos y libros, y redactar un resumen agrupando los distintos modelos existentes con objetivos y métodos en común.
- **Objetivo 2:** Analizar las métricas existentes y las distintas bases de datos en el ámbito.
- **Objetivo 3:** Experimentar los modelos estudiados observando su comportamiento en situaciones con recursos escasos.

2.1.2. Requisitos

1. Buena redacción. Al escribir el análisis, este tendrá que tener una buena introducción, una explicación simple de cada modelo, que lo distingue de los demás, qué resultados genera y que casos prácticos útiles tiene.
2. Se deberá de detallar correctamente las métricas utilizadas para entender cómo de útil o eficiente es el modelo analizado.
3. Acceso a recursos. Para entrenar a la IA, se tiene que disponer de bases de datos y de recursos computacionales suficientes para ejecutar dicho entrenamiento. También es necesario el acceso a recursos bibliográficos para estudiar el estado actual y previo de los modelos.
4. Entrega del trabajo antes de la fecha establecida. En caso contrario, el trabajo se evaluará en la siguiente convocatoria.

2.1.3. Estructura de desglose del trabajo

Se han estructurado las actividades principales a realizar en paquetes de trabajo. Esto agrupa grandes cantidades de actividades pequeñas en conjuntos. Simplificado de esta

forma, ayuda a ver cuánto se va a dedicar principalmente a cada una de estos tipos de tareas. En la Figura 2.1 se puede observar que tareas componen dicha estructura.

Figura 2.1: Estructura de desglose del trabajo general.

Se han identificado tres tipos de trabajo principales: Investigación (R), programación (P), e informe (I). A continuación, se explicaran las diferentes que las componen.

En un proyecto es imprescindible adquirir los conocimientos necesarios para una correcta ejecución del trabajo a realizar, y más aún en un trabajo de investigación. En nuestro caso, es muy importante conocer bien el ámbito que se está estudiando para analizar y evaluar correctamente los modelos y casos que se quiere investigar. Por este motivo, se ha creado la rama de investigación, que recoge todas las actividades relacionadas con la extracción de información y reflexión para conocer bien un tema.

- **(R). Recogida de información:** Dentro de la rama de investigación, el paquete de trabajo de recogida de información agrupa todo lo relacionado con la extracción de conocimiento. No se habla solo de lectura de artículos, también se refiere a visualización de vídeos, lectura de discusiones en foros, consulta a gente especializada, etc. Esta tarea crea el contenido necesario para escribir y redactar una buena memoria.
- **(R). Reflexión:** Este apartado solo está dedicado a, una vez conocida la información, reflexionar acerca de lo leído, visto o escuchado y reflexionar sobre ello. Aquí nace la curiosidad, necesaria para seguir el interés y continuar leyendo con un mejor razonamiento y comprensión.

En la rama de programación se probará todo lo estudiado y se comprobará y analizará el correcto funcionamiento y la utilidad de los modelos.

- **(P). Otros:** Paquete de trabajo dedicado a la creación de código necesario para hacer posible la evaluación y entrenamiento de modelos. Esta tarea podría ser, por

ejemplo, quitar código no interesante de la base de datos, programar las métricas, etc.

- **(P). Entrenamiento:** Paquete de trabajo dedicado para el entrenamiento de modelos. Esta tarea, dependiendo de los recursos que ofrezca la empresa externa, será uno de los paquetes de trabajo en el que más horas se invertirán. Sin embargo, si los modelos no podrán ser entrenados en servidores potentes, solo se podrán entrenar modelos con pocos parámetros y con un *batch size* y *epoch size* pequeños, y por ende, no se podrá hacer un análisis tan detallado y las horas dedicadas en este paquete no serán tantas.

Finalmente, la última rama será dedicada al informe del proyecto. Se intentará agrupar estos paquetes de tal forma que cada semana al menos un 20 % sea dedicado a la mayoría de las ramas.

- **(I). Redacción:** Este paquete de trabajo estará dedicado solamente a escribir. Aquí se harán borradores sobre la información extraída al momento, y más adelante se pasará a limpio.
- **(I). Revisión:** Este paquete de trabajo se usará para corregir lo redactado en los borradores y en el propio documento en limpio para comprobar que todo esté escrito como es deseado.

En las Figuras 2.2 y 2.3 se han definido las horas estimadas del proyecto y el porcentaje de dedicación de cada una de las tareas. Cada paquete de trabajo contiene una serie de horas a realizar. Como se puede observar, el paquete de trabajo asociado a la recogida de información es el que mayor tiempo ocupa. Esto es así porque al ser un trabajo de fin de grado analítico, la mayoría de horas se dedican a obtener conocimientos sobre el tema a tratar. Si fuera un trabajo práctico, lo más seguro es que el mayor porcentaje de horas se lo lleve la rama de programación. No es nuestro caso.

Aunque la mayoría de horas serán dedicadas a la extracción de información, el último tramo del proyecto será dedicado mayormente a la redacción del informe. Aunque está pensado que cada semana se dedique unas horas a la rama de Informe, lo más seguro es que en el último tramo del proyecto todo lo relacionado con el estudio en el ámbito y la programación ya haya sido realizado.

La planificación planteada es la siguiente: Todas las semanas se dedicará mínimo a trabajar en todas las ramas, con el motivo de no dejar ningún paquete de trabajo en el olvido.

	15-31 de Marzo	1-15 de Abril	15-30 de Abril	1-15 de Mayo	15-31 de Mayo	1-15 de Junio	15-26 de Junio
R.Recogida de información							
R.Reflexión							
P.Entrenamiento							
P.Otros							
I.Redacción							
I.Revisión							

Tabla 2.1: Diagrama de Gantt sobre la realización de la actividad en base al tiempo.

La mayor parte del tiempo en las primeras semanas van a ser dedicadas a los paquetes de trabajo sobre la investigación. Ya que todavía no se han adquirido conocimientos, difícil será programar sobre algo que no se sabe, y aún menos, escribir un informe sobre un ámbito desconocido. En las semanas intermedias llegará la mayor parte de actividades relacionadas con la programación. Una vez extraído lo necesario, se comenzará a programar para analizar los modelos y evaluar sus resultados. Finalmente, las últimas semanas se dedicarán mayormente a la memoria, por la razón explicada en el párrafo anterior. En la Tabla 2.1 se muestra la realización de cada tarea dependiendo de la fecha.

Cabe destacar que aunque se hable sobre que unas semanas tendrán mayor dedicación en ciertos paquetes de trabajo que en otros, dichos paquetes de trabajo respetarán las horas dedicadas y el porcentaje de dedicación respecto al resto de paquetes.

Sumando todas las horas por paquete de trabajo, se ha estimado que las horas dedicadas al proyecto será un total de 330.

2.1.4. Supuestos

- El director aceptará la supervisión del trabajo y dará el visto bueno a la defensa del mismo en la convocatoria de junio.
- Los recursos que ofrece la empresa serán suficientes como para poder entrenar los modelos.
- Se podrá acceder a datos de entrenamiento, necesarios para realizar la experimentación.

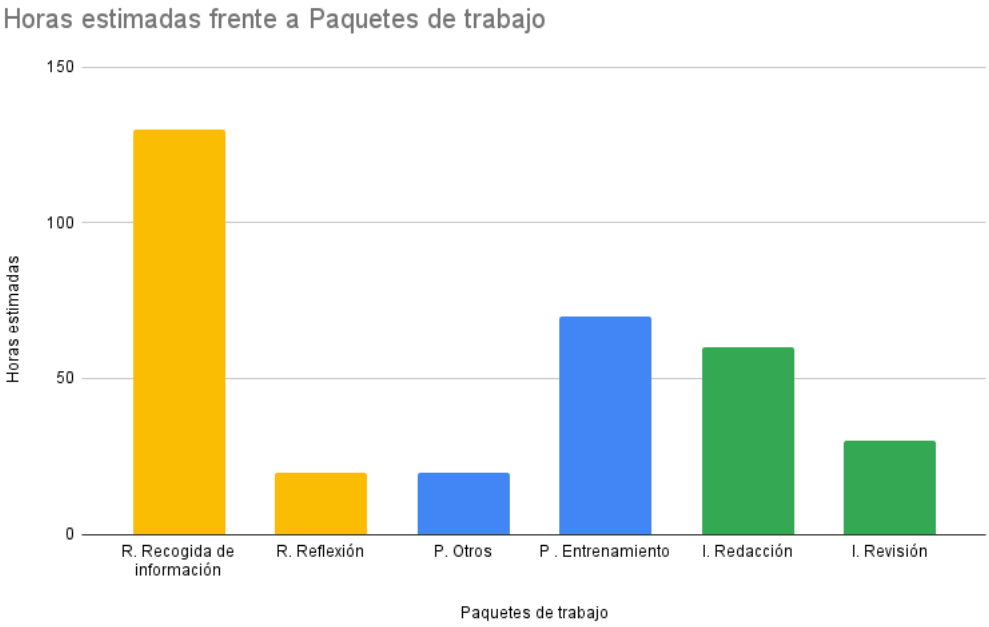

Figura 2.2: Horas dedicadas a cada paquete de trabajo.

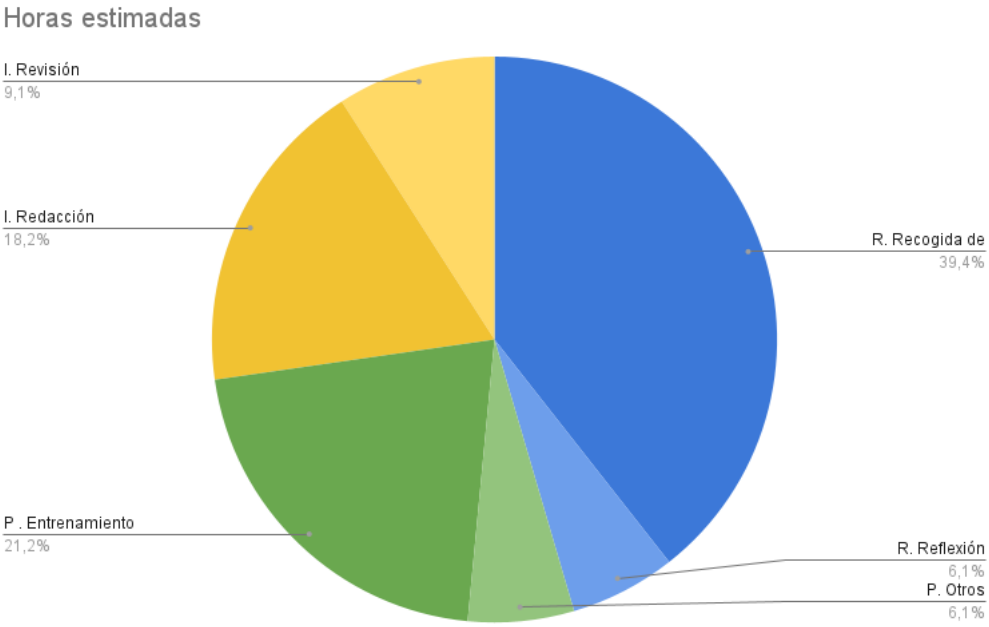

Figura 2.3: Porcentaje de horas dedicadas por paquete de trabajo.

2.1.5. Fechas de terminación de los entregables

En la Tabla 2.2 se indica el plazo limite por cada hito. Una vez pasada la fecha, no será posible realizar la actividad relacionada.

Hito	Fecha
Solicitar la defensa	20/06/2022
Subir el trabajo a la plataforma	26/06/2022
Defensa del TFG	4/07/2022 - 13/07/2022

Tabla 2.2: Fechas de hitos principales.

2.1.6. Sistemas de información y de comunicaciones

En el caso de entrenamiento de modelos, la empresa externa ha facilitado el uso de servidores en los que poder aumentar la velocidad de entrenamiento y memoria a la hora de escoger modelos con mayor número de parámetros. Estos servidores son accedidos mediante una máquina virtual. El tutor en la empresa ha facilitado el uso de dichos servidores para el trabajo final de grado. También, la empresa ha prestado un ordenador portátil para trabajar tanto desde la empresa como desde casa. No conformes, la empresa ha cedido una dirección de correo electrónico para habilitar las comunicaciones entre alumno y empresa.

En las comunicaciones, la aplicación Teams de Microsoft es utilizada tanto para las reuniones como para los mensajes informales. El correo electrónico de Outlook es utilizado para intercambio de mensajes formales. OneDrive es y será la aplicación más usada para compartir archivos de texto y comentar artículos. Finalmente, se usa la plataforma GitLab de Tecnalia para compartir el código.

2.1.7. Análisis de riesgos

- **Retraso en la empresa con las peticiones**: Normalmente las empresas, tanto pequeñas como grandes, suelen priorizar el trabajo urgente y lo que les beneficia a ellos. Por este motivo, es posible que a la hora de solicitar más información acerca de un tema, solicitar algún recurso más o simplemente pedir el rellenar algún tema burocrático, esta entidad tarde en contestar y retrase toda la planificación hecha. En el peor de los casos, este retraso podría conducir a no poder entregar el trabajo a

tiempo. Por este motivo, será imprescindible la optimización del tiempo. Siempre hay que tener trabajo pendiente que no dependa de la empresa externa para que, en caso de retrasos, uno no se estanque y se quede a la espera.

- **Longitud del trabajo**: Puede que el trabajo sea demasiado largo y no de tiempo a terminar todo lo que estaba planificado. Por otro lado, el trabajo puede quedar demasiado simplificado y que se quede escaso. Esto puede derivar en un suspenso. Por este motivo, es de vital importancia estructurar bien los paquetes de trabajo y hacer una buena estimación del resultado final.

2.2. Seguimiento y control

Tras definir toda la planificación y la organización de cómo va a abordarse el proyecto, después de completar el mismo se ha redactado como ha ido todo y si se ha cumplido con los objetivos y las horas dedicadas por paquetes de trabajo.

2.2.1. Objetivos del proyecto cumplidos

Cuatro eran los objetivos principales, siendo uno de ellos el objetivo principal. Todos ellos han sido cumplidos.

- Se ha logrado recoger la información suficiente como para poder realizar un estudio sobre los distintos modelos existentes, explicando como funcionan, cómo se entrenan, y mostrando varios ejemplos de ellos.

- También se ha conseguido analizar las principales métricas y bases de datos utilizados en la actualidad. En las bases de datos, también se ha mostrado varios ejemplos en una tabla indicando a qué sección pertenecen.

- Se han realizado los experimentos deseados. En concreto han sido 2: entrenar a modelos quitando partes de la base de datos y entrenar a modelos preentrenados en euskera para fine-tunearlo en el mismo idioma. Todo con el fin de estudiar el rendimiento de sistemas con recursos limitados.

2.2.2. Dedicación real por paquetes de trabajo

Se podría decir que se ha estimado bien las horas por paquete de trabajo. En la recogida de información, se ha superado las horas por 15 ya que el análisis bibliográfico no era sencillo. La tarea de agrupar la información necesaria y sacar los estudios desde varias fuentes ha resultado ser algo más costosa de lo esperado. Sin embargo, se esperaba que la cantidad de horas dedicadas a la programación iban a ser mayores, pero no. Esto se debe a plataformas como 'HuggingFace' y librerías como 'Pytorch' que facilitan la programación con miles de funciones ya programadas. El resto de paquetes han sido cumplidos con cierta precisión, respetando el horario y las semanas en las que se iban a hacer. En la Tabla 2.3 se muestra la diferencia de horas reales y estimadas por paquete de trabajo.

Paquetes de trabajo	Horas estimadas	Horas reales	Diferencia
R. Recogida de información	130	145	15
R. Reflexión	20	25	5
P. Otros	20	16	-4
P . Entrenamiento	70	58	-12
I. Redacción	60	64	4
I. Revisión	30	24	-6
Total	**330**	**332**	2

Tabla 2.3: Comparación de horas estimadas frente a horas reales.

En la Tabla 2.4 se puede observar de manera más visual la cantidad de horas invertidas en el proyecto.

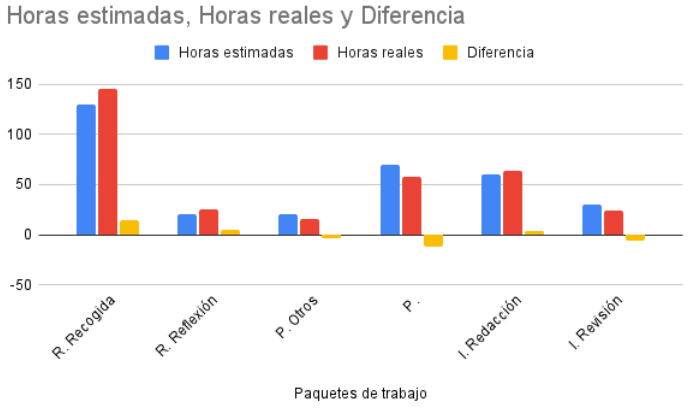

Figura 2.4: Muestra gráfica de diferencia entre horas.

3. CAPÍTULO

Antecedentes

Todos los modelos que, a día de hoy, generan respuestas muy bien logradas y con alta precisión, funcionan de tal manera porque el sector ha ido evolucionando y ha ido creando nueva tecnología capaz de superar a la anterior. Como es comprensible, toda la ciencia que hay detrás de estos modelos no ha nacido de la noche a la mañana. Por este motivo, en esta sección haremos un breve resumen sobre la historia y evolución de los modelos QA. Se comenzará explicando los primeros modelos que aparecieron dentro de esta área, y se terminará comentando el estado del arte, diciendo que modelos están teniendo mayor impacto a día de hoy.

En el año 1961 se creó el primer modelo QA. Este modelo se llamaba BASEBALL [Green et al., 1961], y fue creado para responder preguntas acerca de los partidos de béisbol (p.e. ubicación, nombre del equipo y tiempo de juego). Este prototipo, con la información que tenía guardado en una forma estructurada, recogía la pregunta dada y lo transformaba siguiendo unos métodos lingüísticos concretos para así llegar a la respuesta encontrada en el mencionado almacenamiento de información. Más adelante, en 1973 en concreto, se publicó LUNAR [Woods, 1973]. Modelo capaz de ayudar a investigadores en geología lunar. Este trataba de responder con la información que se almacenaba en una base de datos proveniente de la NASA. En 1993, apareció MURAX [Kupiec, 1993], que dado una enciclopedia en inglés, procuraba responder a preguntas puramente académicas. Como se puede observar, todos los modelos que salieron al principio tenían un enfoque de recogida de información, o *information retrieval* . En el año 1999, hubo un gran avance cuando se sacó un modelo capaz de extraer de un conjunto de artículos los 5 fragmentos con mayor probabilidad de acierto en la respuesta. En este problema se usó un gran

conjunto de documentos no estructurados como fuente de información. Esto tuvo un gran impacto en la investigación de sistemas de dominio abierto.

La investigación siguió avanzando, pero la popularidad de la *World Wide Web* hizo que los investigadores se decantasen más por en el uso de esta tecnología para extraer información [Brill et al., 2002]. Se intentaba recoger las respuestas mediante motores de búsqueda como Google.com, Ask.com, etc. En 2001, nació MULDER [Kwok et al., 2001], sistema que usaba el motor de búsqueda de Google para dar la respuesta. Este primero traducía las preguntas del usuario con analizadores sintácticos y generaba una serie de consultas, que se las pasaba a Google y utilizaba un componente de extracción de texto para los resultados devueltos. Los motores de búsqueda tienen la habilidad de encontrar información rápidamente dentro de un mar de datos. El problema está en que es muy difícil extraer la información correcta ya que está lleno de datos mal escritos e información irrelevante.

3.1. Arquitectura de los modelos clásicos

En los modelos convencionales, normalmente el proceso de extraer la información se dividía en 3 fases: análisis de la pregunta, búsqueda de información y extracción de la respuesta [Harabagiu et al., 2003]. La primera fase se centraba en facilitar el trabajo de la búsqueda y extracción procesando y entendiendo la pregunta. La segunda fase, buscaba la información en el que podría estar el resultado con sistemas de búsqueda (basados en la frecuencia del término, normalmente) y con motores de búsqueda. Finalmente, el tercer paso se encargaba en extraer la respuesta con los documentos pasados por la segunda etapa. A continuación se explicarán con mayor detalle.

1. Análisis de la pregunta: Los objetivos de esta fase eran dos: facilitar la búsqueda de información y mejorar la capacidad de extraer la respuesta correcta. En el primer objetivo, técnicas lingüísticas como *POS tagging*, o *part-of-speech tag*, *stemming* y *parsing* eran métodos que mejoraban la precisión. Por otro lado, habitualmente solía suceder que el término de la respuesta no era el mismo que el reformulado. A esto se le llama *term mismatch* y sigue siendo un problema importante en la búsqueda de información. Por otro lado, en la clasificación de preguntas, se intentaba predecir de qué tipo es la pregunta. Venía muy bien para identificar las intencionalidades del usuario. Se solía utilizar las distinciones de tipo *Where*, *When*, *Who*, etc, para identificar qué tipo de respuesta pedía el usuario.

2. Búsqueda de información: En esta fase el modelo buscaba documentos en el que podría estar la respuesta y devolvía una pequeña cantidad de ellos. El valor que indicaba esta cantidad dependía del algoritmo y de la capacidad del sistema. Los algoritmos de búsqueda dividen en varios grupos:

   - Modelos Booleanos: El modelo booleano es el más simple de todos. La pregunta se transforma en la forma de expresiones booleanas de términos, que se combinan con operadores como *AND*, *OR* y *NOT* para coincidir exactamente con documentos vistos como un conjunto de palabras.
   - Modelos de Espacio Vectorial: Estos modelos representan las preguntas como un vector de palabras, y los comparan con los documentos con algoritmos como la similaridad del coseno y la distancia euclídea. Estos vectores son de dimensión n, siendo n el número de palabras en el diccionario.
   - Modelos probabilísticos: Estos buscan integrar relaciones de probabilidad entre las palabras. BM25 [Robertson and Zaragoza, 2009] es un ejemplo de ello. Este modelo es sensible a la frecuencia del término y el tamaño del documento. Sigue siendo uno de los modelos de búsqueda más efectivos y actualmente lo usan varios motores de búsqueda.
   - Modelos de Lenguaje [Banerjee and Han, 2009]: Este tipo de modelos es muy popular y *query likelIhood model* [Manning et al., 2008] es el más utilizado. Este crea un modelo probabilístico de lenguaje LM_d para cada documento d y lo relaciona con la pregunta dada q: $P(q|LM_d)$.

   Aun utilizando técnicas elaboradas para la búsqueda, muchas veces sigue siendo costosa y poco eficiente. Por este motivo, se post-procesa el resultado de la búsqueda para filtrar y limpiar ruido y documentos que no interesan.

3. Extracción de la respuesta: Por último, esta frase devuelve la respuesta correcta. El correcto funcionamiento de esta última fase dependía mucho de cómo se había hecho el trabajo previo, desde el análisis de la pregunta, hasta el post-proceso de la búsqueda de información. También dependía de la complejidad de la pregunta, El tipo de respuesta esperado y el método de extracción escogido. Dos tipos de preguntas habían sido estudiadas en sistemas tradicionales de QA. Uno de ellos es la pregunta de tipo factoide donde las respuestas suelen ser una simple muestra dentro del documento. Como ejemplo, las preguntas de tipo *where*, *when*, *who*, etc. Por otro lado, estaban las listas de preguntas, donde la respuesta solía ser una lista

de factoides que aparecían tanto en un mismo documento como en un grupo de documentos.

3.2. Redes neuronales profundas y modelos de atención

En esta década, las técnicas de aprendizaje profundo se han aplicado con éxito en los sistemas mencionados anteriormente. Parece ser que los modelos de aprendizaje funcionan bien en cualquier área. No se puede hacer un resumen de todos los modelos de aprendizaje ya que abarcaría demasiado espacio, y por este motivo esta sección se centrará en los modelos de comprensión de texto. Es decir, los modelos centrados en entender el texto. Son útiles en los modelos de QA porque mejoran la precisión de las respuestas, ya que entienden lo que se le ha pasado como parámetro.

Para las tareas *seq2seq*, es decir, tareas en el que se le pasa una secuencia y devuelve otra, las RNN [Sherstinsky, 2020] y LSTM [Hochreiter and Schmidhuber, 1997] eran las que más se usaban. Las Redes Neuronales Recurrentes (RNN) tienen una arquitectura especial respecto a las redes neuronales típicas (perceptrón). La función de activación de un perceptrón multicapa solo funciona en una dirección, la dirección de avance desde la capa de entrada a la capa de salida, es decir, no recuerdan los valores anteriores. Las redes RNN son similares, pero incluyen conexiones 'hacia atrás', un tipo de retroalimentación entre las neuronas dentro de una capa. Esto seguía generando problemas, ya que la memoria que este modelo tenía era de corto plazo. Cuando se insertaba una secuencia larga, al terminar de procesarla el modelo no recordaba muy bien el texto del inicio. Por este motivo, se comenzó a usar los modelos LSTM (*Long Short Term Memory*). Estos modelos tenían una arquitectura que permitía una memoria a mayor plazo. Estos modelos tenían 2 problemas principales, las carencias en la comprensión de texto, y la velocidad de entrenamiento. No entendían bien el texto ya que la sentencia se procesaba de un lado a otro (de izquierda a derecha). Es cierto que se crearon modelos bi-LSTM [Huang et al., 2015] capaces de leer la sentencia por ambos lados (de izquierda a derecha y de derecha a izquierda) pero seguía sin entender bien el contexto. El problema de la falta de velocidad en el entrenamiento sucedía porque el texto se procesaba de forma secuencial. Todos estos problemas se eliminaron cuando apareció el modelo de atención Transformer [Vaswani et al., 2017].

A pesar de que fue creado para traducción, la arquitectura de este modelo fue muy útil por varios motivos. Fue capaz de computar las secuencias en paralelo, con lo que la velocidad de entrenamiento se multiplicó. Como no se leía de forma secuencial, también entendía

el contexto mucho mejor, ya que procesaba todo el texto a la vez. Pero, lo más importante de esta arquitectura fue la atención. Un mecanismo capaz de procesar la atención que se le debe de prestar a cada token respecto a otros y como de relevantes son. Usaba un mecanismo *encoder-decoder*. Una parte se dedicaba a codificar el texto y la otra en descodificar y sacar la respuesta.

La atención en los Transformers se le llamó *Scaled Dot-Product Attention*, y funcionaba de la siguiente manera: La entrada consiste en *Queries* y *Keys* de dimensión d_k, y *Values* de dimensión d_v. Se calculan los productos punto de la consulta con todas las *Keys*, divididos cada uno de ellos por $\sqrt{d_k}$, y se aplica una función *softmax* para obtener los pesos de los *Values*. En la práctica, se calcula la función de atención sobre un conjunto de *Queries* simultáneamente, empaquetadas en una matriz Q. Las *Keys* y *Values* también se empaquetan juntos en las matrices K y V . Se calcula la matriz de salida como:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.1}$$

Este modelo tuvo una repercusión muy importante, y varios modelos fueron publicados siendo modificaciones del Transformer. Uno de ellos es BERT [Devlin et al., 2018].

BERT (*Bidirectional Encoder Representation from Transformers*) es el modelo de comprensión más utilizado en el momento. BERT está diseñado como la parte de codificación del Transformer. Este modelo rinde muy bien porque tiene una cantidad de parámetros muy grande y porque su entrenamiento se puede hacer con una gran cantidad de información y de forma no supervisada. Normalmente, este modelo se utiliza como base para hacer tareas específicas, así como resumir, clasificar opiniones en base el sentimiento positivo o negativo, o, en nuestro caso, responder preguntas. Se aprovechan de su gran capacidad de entender texto, para poder entrenarlo para tareas concretas como análisis de sentimientos, clasificación, o, en este caso, responder preguntas. BERT es pre-entrenado con dos tipos de ejercicios. Uno de ellos es el de desenmascarar las palabras enmascaradas. Es decir, se le pasa una secuencia de texto, y aleatoriamente se enmascaran ciertas palabras. La tarea del modelo es intentar adivinar qué palabras son las que están ocultas. La segunda tarea es la de clasificación. Dadas dos secuencias A y B, el objetivo consiste en predecir si B es una continuación de A. Como se puede observar, al ser una tarea no supervisada y sencilla, se le puede pasar una gran cantidad de datos no estructurados. Este

modelo, en QA, normalmente se utiliza para entrenar a modelos de extracción de texto. A BERT, se le suele añadir una capa de más para que sea capaz de responder preguntas.

3.3. Sistemas modernos y estado del arte

Actualmente, los sistemas de pregunta respuesta más utilizados siguen el formato *Retriever reader*. *Retriever* es el encargado de buscar información y devolver documentos, mientras que el *reader* procesa dichos documentos filtrados, y los modelos atencionales explicados en la sección anterior serán los encargados en devolver la respuesta. En el capítulo de análisis, se explicará de forma más detallada los distintos tipos de modelos que usan este formato y su comportamiento. La idea principal de este trabajo es analizar los tipos de modelos *reader* hay en la tarea de responder preguntas. En la Figura 3.1 se muestra de forma visual el comportamiento de estos modelos. Como se ha recalcado en la introducción, estos algoritmos no están pensados para mantener una conversación, sino en responder una pregunta recuperando la información de algún lugar, ya sea internet o un corpus de texto. A esta perspectiva se le llama *Information Retrieval* [Manning et al., 2008].

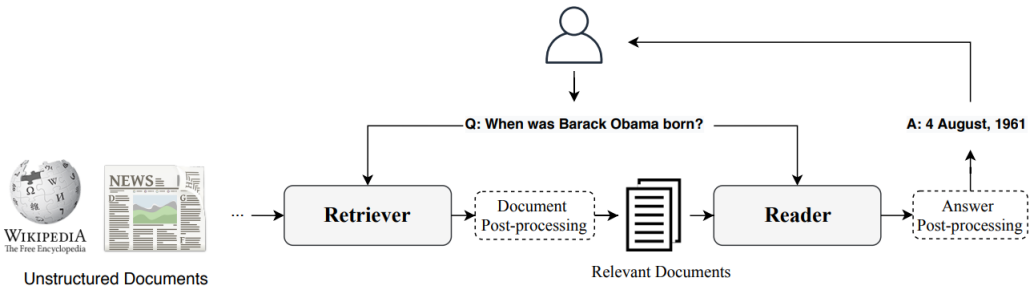

Figura 3.1: Ilustración de la arquitectura *Retriever-Reader* del sistema. Los módulos marcados con líneas discontinuas son auxiliares. [Zhu et al., 2021]

Aun y todo, uno no se puede olvidar de los últimos avances dentro del campo del NLP, los gigantes modelos de lenguaje [Banerjee and Han, 2009]. Estos son modelos estadísticos probabilísticos que determinan la probabilidad de que una determinada secuencia de palabras aparezca en una frase en función de las palabras anteriores. Uno de los ejemplos es GPT3 [Brown et al., 2020], T5 [Raffel et al., 2019], PaLM [Chowdhery et al., 2022] y Deepmind GATO [Reed et al., 2022]. Este último también completa tareas en el campo de los videojuegos, robótica, etc. Lo que se quiere recalcar en este párrafo es que estos

modelos de lenguaje tienen tantos parámetros y se han entrenado con tales cantidades de texto que son capaces de completar tarea en las que no se les ha entrenado concretamente, y con una tasa de acierto iguales o superiores a los modelos entrenados para dichas tareas. Por ejemplo, GPT-3 es un modelo que su principal objetivo es predecir cuál es la siguiente palabra, dada una secuencia de inicio. A este modelo, si como secuencia se le pasa una pregunta y se le pide que responda, el modelo contesta sin haber utilizado ninguna técnica de *Fine-tuning* (entrenamiento de modelos genéricos para una tarea especifica) para que sea capaz de hacerlo. Lo curioso es que, cuanto más se escala el modelo, más potente parece ser. A día de hoy, no se ha encontrado el límite en el que con más parámetros el modelo empeore.

4. CAPÍTULO

Revisión y análisis

Dentro del campo de QA, existe una gran cantidad de sistemas capaces de crear una respuesta basada en la pregunta. Dan Jurafsky, profesor de la universidad de Stanford, distingue en su popular libro 'Speech and Language Processing' [Jurafsky and Martin, 2000] dos tipos de sistemas. Los sistemas basados en la recuperación de información (*Information Retrieval*) y los basados en conocimiento (*Knowledge-based*).

- ***Information Retrieval***: Se basa en la gran cantidad de texto existente en la web o en colecciones de artículos científicos como PubMed. Ante una pregunta del usuario, se utiliza la recuperación de información para encontrar documentos relevantes. A continuación, los algoritmos neuronales de comprensión lectora leen estos documentos recuperados y extraen o generan una respuesta directamente de los tramos de texto.

- ***Knowledge Based***: Este tipo de sistema construye, en cambio, una representación semántica de la consulta, como la asignación de ¿Qué países limitan con Francia? a la representación lógica: $\lambda x.pais(x) \wedge limite(x, Francia)$, o ¿Cuándo nació Will Smith? a la relación con hueco: $ano-de-nacimiento(WillSmith, ?x)$. Estas representaciones de significado se utilizan para consultar bases de datos de hechos, y de ahí recibir la respuesta.

Ya que varios expertos como Yan LeCun comentan que el aprendizaje semi-supervisado puede ser la mejor forma de entrenar a sistemas inteligentes [LeCun and Misra, 2021], se

ha optado por enfocar el trabajo en analizar sistemas con la perspectiva de recuperación de información, ya que los métodos de entrenamiento y los algoritmos de búsqueda son más propensos a trabajar con datos no estructurados, y este tipo de aprendizaje normalmente requiere de texto crudo. Se ha elegido este enfoque por su mayor potencial a largo plazo.

Se va a hacer un análisis en base a tres ejes fundamentales: Tipología de modelo, en el que se comenta que tipos de modelos existen y cómo funcionan, métricas, en el que se explican cómo se mide el rendimiento de estos sistemas, y bases de datos, fundamentales para un buen entrenamiento.

4.1. Modelos de aprendizaje profundo

Antes de explicar los tipos de modelos que existen, es importante saber en qué tipo de dominio se están desarrollando, abierto o cerrado. Se describen de la siguiente forma:

- Dominio abierto: Los sistemas de este tipo de dominio son capaces de responder a cualquier pregunta que se le plantee, pero la calidad que ofrecen sus respuestas no son tan buenas como los que devuelven los sistemas de dominio cerrado si se les plantea una pregunta de su dominio. Se basan principalmente en la ontología general y el conocimiento del mundo.
- Dominio cerrado: Sistemas capaces de responder a preguntas con mayor precisión dentro de su ámbito. Son peores devolviendo respuestas en preguntas genéricas o de otro tema no relacionado con el suyo.

Aunque algunos redactores de artículos distinguen dichos modelos en base al dominio en el que se desarrollan, formalmente se clasifican en función a su comportamiento, y aunque haya muchas formas de actuar diferentes, se podrían agrupar estos modelos en dos tipos. Los modelos extractivos, y los modelos generativos. Ambos tienen como propósito dar una respuesta en base a un contexto.

Los modelos extractivos son aquellos que extraen información dentro de un contexto para responder a la pregunta pasada como parámetro. Es decir, dadas dos entradas, la pregunta y el contexto, el modelo tiene que predecir dónde se incluye la respuesta a dicha pregunta. Para ello, se suele optar por un modelo de comprensión de texto (por ejemplo BERT) pre-entrenado y se le añade dos capas. Una para predecir la probabilidad de cada token de ser el comienzo de la respuesta, y otra capa para la probabilidad del token final. Se

puede pensar al token como la unidad para procesamiento semántico. En procesamiento de lenguaje natural, el proceso de convertir nuestras secuencias de caracteres, palabras o párrafos en inputs para el modelo se llama tokenización.

Entrando en mayor detalle, el entrenamiento del modelo BERT para la tarea específica de extraer texto se hace de la siguiente forma: Entrenamos dos nuevos conjuntos de parámetros para la tarea , un vector inicial $S \in R^H$ y un vector final $E \in R^H$, donde H es el tamaño oculto de sobre una cantidad específica. Entrenamos S y E haciendo pasar la salida de BERT por una capa lineal. Concretamente, dado $T_i \in R^H$ (último vector oculto de BERT para el token de entrada i), la probabilidad de que la palabra i sea el inicio del tramo de respuesta se calcula como un producto escalar entre T_i y S, seguido de un *SoftMax* sobre todas las palabras del párrafo. El vector que indica el final se calcula de forma similar, y el tramo de puntuación máxima se utiliza como predicción [Schwager and Solitario, 2019].

$$P_i^S = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}} \tag{4.1}$$

$$P_i^E = \frac{e^{E \cdot T_i}}{\sum_j e^{E \cdot T_j}} \tag{4.2}$$

La imagen 4.1 extraída del artículo de BERT [Devlin et al., 2018], muestra de forma visual el preentrenamiento y el *fine-tuning* del modelo. *Fine-tuning* se refiere al entrenamiento de un modelo de lenguaje preentrenado para resolver tareas mas especificas.

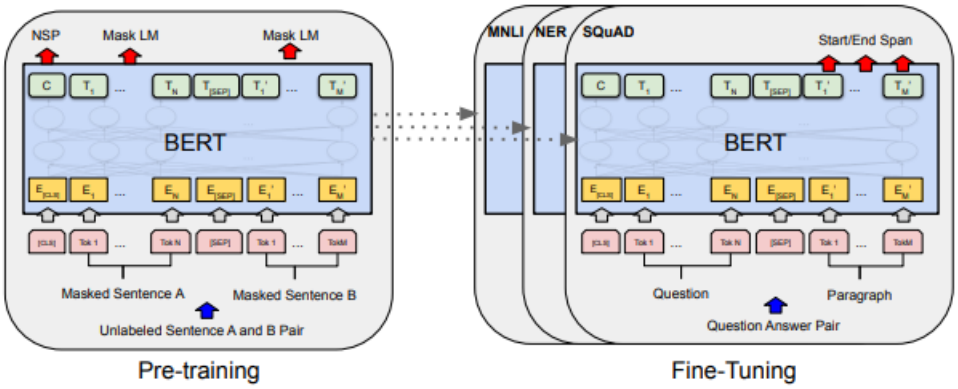

Figura 4.1: Entrenamiento de BERT.

Últimamente, los modelos extractivos no sólo predicen la posición de la respuesta dentro del contexto, sino que también son capaces de predecir si la respuesta está incluida o no. Esto es posible gracias a bases de datos, explicado en la siguiente sección.

La mayoría de modelos de aprendizaje QA suelen ser extractivos, ya que estos son los más sencillos de entrenar. No precisamente por su poca complejidad (ya que montarlo de cero es requiere de capacidades), sino por el gran trabajo que se ha hecho en distintas plataformas como HuggingFace [Wolf et al., 2020] facilitando a los desarrolladores e investigadores en la programación creando librerías específicas para la creación, entrenamiento y predicción de modelos.

Uno de los modelos extractivos usados es DS-QA [Lin et al., 2018]. Este modelo se divide en dos partes: el selector de párrafos y el lector de párrafos (*reader*).

- Selector de párrafos: Dada la pregunta q y el párrafo recuperado P, el selector de párrafos mide la distribución de probabilidad $Pr(p_i|q,P)$ sobre todos los párrafos recuperados, que se utiliza para seleccionar el párrafo que realmente contiene la respuesta a la pregunta q.
- Lector de párrafos: Dada la pregunta q y un párrafo p_i, el *reader* calcula la probabilidad $Pr(a|q,p_i)$ de extraer la respuesta a mediante una red *Long-Short Term Memory*, o LSTM [Hochreiter and Schmidhuber, 1997] multicapa.

En resumen, la probabilidad $Pr(a|q,P)$ de extraer la respuesta a dada la pregunta q se puede calcular de la forma:

$$Pr(a|q,P) = \sum_{p_i \in P} Pr(a|q,p_i) Pr(p_i|q,P) \tag{4.3}$$

Otro ejemplo, y aún más conocido, sería DrQA [Chen et al., 2017]. Este también consta de dos partes, el recuperador y el lector. El recuperador actúa en base a los resultados devueltos mediante TF-IDF [Sammut and Webb, 2010]. Es decir, utiliza dicha técnica para devolver los documentos con mayor probabilidad de que contenga la respuesta. TF-IDF (*Term Frecuency and inverse document frecuency*) premia a los documentos que más contengan el término (TF), en el caso de que este término no sea frecuente en el resto (IDF). Se computa de la siguiente forma.

$$\mathbf{tf}(t,d) = \frac{f_d(t)}{\text{máx}_{w \in d} f_d(w)} \tag{4.4}$$

$$\mathbf{idf}(t,D) = \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \tag{4.5}$$

$$\mathbf{tfidf}(t,d,D) = \mathbf{tf}(t,d) \cdot \mathbf{idf}(t,D) \tag{4.6}$$

$f_d(t) :=$ frecuencia del termino t en el documento d.

$D :=$ Corpus de documentos.

Después, descompone los documentos recuperados en párrafos y extrae varias características que consisten en *Part-of-Speech* (POS), *Entity Recognition* (NE), *Term Frecuency* (TF), etc. A continuación, el *reader*, usa un Bi-LSTM [Huang et al., 2015] multicapa para predecir la respuesta. En este proceso, para que las puntuaciones de las respuestas sean comparables entre párrafos, adopta la función exponencial no normalizada y añade *argmax* a todos los tramos de respuesta para obtener el resultado final. En la Figura 4.2 se muestra una representación visual de este modelo.

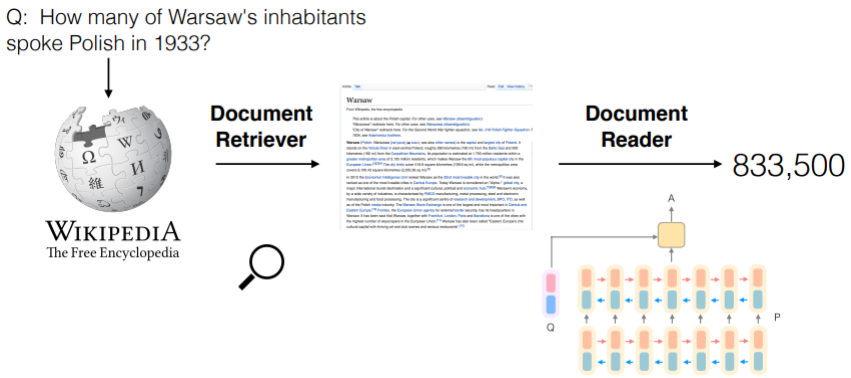

Figura 4.2: Modelo DrQA [Chen et al., 2017].

Sin embargo, estos no son los únicos modelos de respuesta existentes, también están los modelos generativos.

La idea principal de los *readers* generativos es el generar nuevo texto, como bien indica su nombre, y no solo extraer partes del contexto. Esta tarea es algo más compleja, ya que

el modelo no se centra en un problema de clasificación con varias clases, sino en generar nuevos tokens.

En un modelo generativo, el codificador, similar al modelo extractivo, toma una pregunta q y un contexto c como entrada y emite una representación contextual h. El decodificador toma los tokens de respuesta generados previamente como entrada y realiza la atención sobre h, para después generar un token. Formalmente, dada una tupla (q,c,a), el objetivo de entrenamiento es minimizar la siguiente función de pérdida [submission, 2021]:

$$\mathcal{L}_{Gen} = \sum_{i=1}^{K} \log \mathbf{P}(a_i|\mathbf{h}, a_{:i}) \tag{4.7}$$

Donde K es el número de tokens en la respuesta a, a_i es el i-ésimo token en a, y a_0 corresponde a el token de comienzo de la secuencia (BOS).

Es complicado dar una explicación genérica a cómo son entrenados estos modelos generativos, ya que se estaría generalizando demasiado. Por este motivo, a continuación se explican varios ejemplos de dichos modelos y cómo estos han sido entrenados. Un modelo *reader* generativo es RAG [Lewis et al., 2020]. Este modelo se aprovecha de las ventajas de los modelos de lenguaje *seq2seq* para Fine-Tunear estos y conseguir buenos resultados en la tarea de generar respuestas.

En concreto, usa el modelo de lenguaje BART [Lewis et al., 2019]. BART (*Bidirectional and Auto-Regresive Tranformers*) es un modelo de lenguaje que aprovecha las capacidades tanto de BERT como de GPT. En BERT, se enmascaran tokens aleatorios y el documento se codifica de forma bidireccional. Los tokens que faltan se predicen de forma independiente, por lo que el BERT no puede ser usado fácilmente para la tarea de generación. Por otro lado, en GPT los tokens se predicen de forma autorregresiva, lo que significa que se puede utilizar para generación. Sin embargo, las palabras sólo pueden condicionar el contexto hacia la izquierda, por lo que no puede aprender interacciones bidireccionales. BART combina las capacidades de ambos modelos. Primero pasa la secuencia de inicio por una función de ruido que lo modifica. La tarea de dicho modelo de lenguaje es predecir el texto original.

En concreto, BART usa cinco funciones distintas para alterar la secuencia.

- *Token Masking*: Como en BERT, se enmascaran palabras aleatorias de la secuencia.

- *Token Deletion*: Tokens aleatorios son eliminados. A diferencia del anterior, este no tiene un marcador indicando que la palabra está enmascarada ([MASK]), con lo que dificulta la tarea al modelo.

- *Text Infilling*: También es un enmascaramiento de palabras, pero se le añade una dificultad mayor. No se enmascaran palabras sueltas, sino grupos de palabras. Es decir, varias sub-secuencias son enmascaradas en un mismo marcador [MASK].

- *Sentence permutation*: Un documento se divide en frases basadas en puntos finales, y estas frases se barajan en un orden aleatorio. Con esto se quiere decir que no se permutan palabras, sino sentencias. Esto ayuda a no cambiar el significado de la frase.

- *Document Rotation*: Se elige un token al azar, y el documento o sentencia se rota para que comience dicho token. Esta tarea entrena al modelo para identificar el inicio del documento.

En la Figura 4.3 facilita la comprensión de la explicación previa, ya que lo muestra de forma visual.

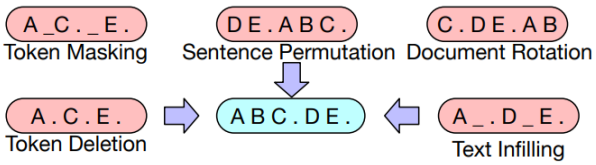

Figura 4.3: Funciones de ruido de BART.

Volviendo a RAG, se comentaba que este utilizaba las ventajas que aportaba BART para entrenarla para responder preguntas. Aunque este modelo RAG también posea una arquitectura de *retrieving* para escoger documentos, en concreto DPR [Karpukhin et al., 2020], se centrara al lector en explicar la parte de generación. El componente de generación utiliza la siguiente fórmula: $p_\theta(y_i|x,z,y_{1:i-1})$. Parametrizado por θ que genera un token actual basado en un contexto de los anteriores $i-1$ tokens $y_{1:i-1}$, el input original x y un fragmento recuperado por el *retriever* z. En concreto, se usa el modelo 'BART-large' pre-entrenado. Dicho modelo contiene 400 millones de parámetros. Para juntar el contexto del documento extraído con la pregunta, simplemente se concatenan y se pasan como input.

El sistema ha sido entrenado en conjunto (*end2end*) sin supervisar si el modelo está devolviendo el top-k de documentos correctamente. Dando pares de entrada/salida (x_j, y_j) del corpus para *fine-tuning*, el objetivo es minimizar el negativo del logaritmo de probabilidad marginal (*negative marginal log-likelihood*) de cada uno $\sum_j -\log p(y_j|x_j)$ usando el descenso del gradiente estocástico con Adam [Kingma and Ba, 2014].

Otro ejemplo de modelo generativo sería FID [Izacard and Grave, 2020]. Este usa modelos de lenguaje como T5 y BART (modelos *seq2seq*) para codificar el fragmento y la pregunta, para devolver un resultado. En concreto, inserta la concatenación de el pasaje, la pregunta y el título del documento como input. También añade los token *Question:*, *Title:* y *Context:* justo antes de la pregunta, el título y el contexto. Esto se hace para que el modelo de lenguaje entienda lo que se pretende hacer. Los algoritmos que usa este sistema para buscar y recoger los documentos son BM25 y DPR.

El modelo de lenguaje mencionado anteriormente, T5 [Raffel et al., 2019], o *Text-to-Text Transfer Transformer* (véase la Figura 4.4), es una arquitectura basada en Transformer que utiliza un enfoque de texto a texto. Cada tarea - incluida la traducción, la respuesta a preguntas y la clasificación- se plantea alimentando el texto del modelo como entrada y entrenándolo para que genere algún texto final. Esto permite utilizar el mismo modelo, la misma función de pérdida, los mismos hiperparámetros, etc. en todas las tareas. Comparándolo con BERT, hay dos diferencias principales. La primera es que se añade un decodificador a la arquitectura, ya que BERT solo tiene la parte de codificación. La segunda es que se modifica la tarea de rellenar los espacios en blanco por una mezcla de tareas alternativas de entrenamiento.

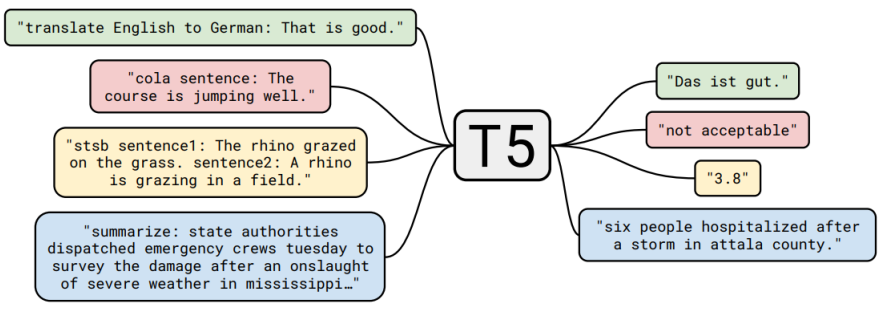

Figura 4.4: Diagrama de T5.

Es de interés comentar que existen modelos híbridos. Estos se aprovechan de ambas tecnologías, de la extractiva y generativa. Un ejemplo claro sería S-net [Tan et al., 2017]. Este,

emplea un modelo de extracción de pruebas para predecir primero el límite del tramo de texto como prueba de la respuesta y luego lo introduce en un modelo generativo *seq2seq* para obtener la respuesta final. Explicado con más detalle, primero usa modelos bi-GRU [Cho et al., 2014] y match-LSTM [Wang and Jiang, 2015] para hallar la posición de la respuesta. Es interesante mencionar que el modelo también hace un ranking se párrafos indicando cual es el más probable de tener la respuesta. Esto se debe a que como entrada hay varios párrafos y no solo uno. Esto es posible gracias a la base de datos MS-MARCO [Bajaj et al., 2016], que contiene varios contextos para una misma pregunta. Después, con la representación de las posiciones dado por el extractivo y la representación vectorial de la pregunta y los contextos, se insertan en el modelo GRU con atención para decodificar y generar la respuesta. A continuación, en la Figura 4.5, se puede observar el funcionamiento de esta última fase.

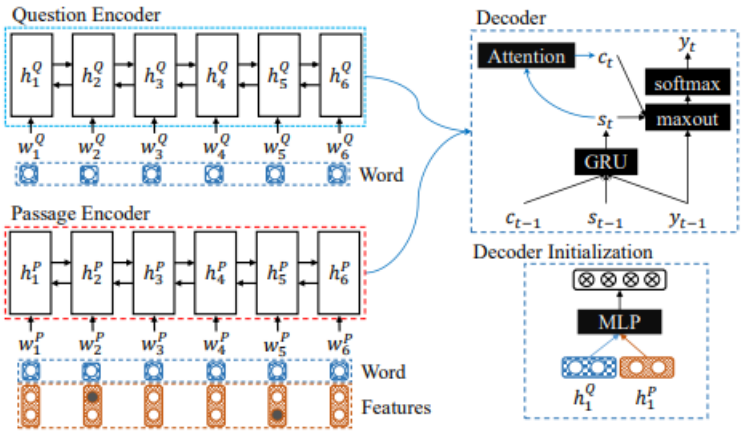

Figura 4.5: Fase de síntesis de la respuesta del modelo S-net.

4.1.1. Extractivos vs Generativos

Varios investigadores del Arizona state university publicaron un estudio en el que diferencian los modelos extractivos y los generativos provenientes de modelos de lenguaje preentrenados [Luo et al., 2022], no por su arquitectura sino por su rendimiento. Varias conclusiones sacadas por los autores son interesantes y se van a comentar a continuación.

1. El primer experimento revela que la elección de PrLM (modelo de lenguaje preentrenado) afecta al rendimiento. En concreto, para T5, el lector generativo es mejor que el extractivo, pero para BART, los lectores extractivos son mejores que los generativos.

2. El segundo experimento indica que los *readers* extractivos son generalmente mejores que los generativos, siendo el modelo extractivo fine-tuneado en T5 el que mejor performance obtiene entre el resto de PrLM.

3. Los modelos generativos rinden mejor en contextos largos, mientras que los extractivos lo hacen en contextos más cortos.

4. Los codificadores de modelos de codificación y decodificación de los PrLM son también muy buenos extractores de respuesta. De hecho, los modelos extractivos entrenados con codificadores de T5 y BART dan mejores resultados que el modelo *decorder-only* de RoBERTa [Liu et al., 2019].

4.2. Análisis de métricas

Las métricas son aquellos datos numéricos que nos sirven para analizar el rendimiento de un determinado modelo. Digamos que, gracias a las métricas, podemos saber si estamos cumpliendo un objetivo. La calidad de un sistema o modelo, es decir, si el modelo es bueno o no, lo define la métrica. No hay un método genérico que indique la calidad de un sistema, y por este motivo se investiga mucho en crear métricas que intenten estimar de la mejor forma posible.

En el campo del procesamiento de lenguaje natural hay muchos tipos de métricas que miden distintas cosas dependiendo de lo que se esté tratando, pero las más populares son dos: F1 y Exact Match. Ambas métricas son comparaciones entre la perdición y el resultado real, pero estas dos métricas se miden de forma distinta.

Exact match es la métrica más sencilla. Consta de lo siguiente: En el caso de que la respuesta predicha sea exactamente igual a la real, este devolverá un punto. En caso contrario, devolverá cero.

F1 es algo menos simple. Esta métrica tiene en cuenta tanto la precisión como la exhaustividad (*recall*) de la siguiente forma:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.8}$$

La precisión y el recall se calcula de la forma:

$$Precision = \frac{TP}{TP + FP} \tag{4.9}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.10}$$

TP: True Positive; TN: True Negative; FP: False Positive; FN: False Negative

Siendo TP los casos buenos acertados, FP los falsos positivos, TN los negativos acertados, y FN los falsos negativos. Dicho de otra forma, la precisión cuantifica el número de predicciones de clases positivas que realmente pertenecen a la clase positiva, y el *recall* cuantifica el número de predicciones de clases positivas realizadas de entre todos los ejemplos positivos del conjunto de datos.

Concretamente, en los modelos QA, F1 mide el número de palabras compartidas entre la predicción y la real: la precisión es la relación entre el número de palabras compartidas y el número total de palabras de la predicción, y la recuperación es la relación entre el número de palabras compartidas y el número total de palabras de la respuesta real.

A parte de estas métricas también existen otras que se están estudiando y comprobando su buen rendimiento y estimación. El paper publicado por varios estudiantes de la universidad de California [Chen et al., 2019] explica varios sistemas de medición más:

- **BLEU** [Papineni et al., 2002] es una métrica basada en la precisión desarrollada para evaluar la traducción automática. BLEU puntúa un candidato calculando el número de n-grams compartidas con la referencia. n varía desde 1 hasta N.
- **METEOR** [Banerjee and Lavie, 2005] es una métrica que busca relaciones entre unigramas (1-gram), pero también compara entre los sinónimos y varias modificaciones léxicas más.
- **ROGUE-L** [Lin, 2004] es una métrica basada en la subsecuencia más larga en común (LCS) que busca la mayor subsecuencia de tokens coocurrentes entre el texto predicho y el real.
- **BERTscore** [Zhang et al., 2019] aplica la métrica en un enfoque distinto. Dicha métrica obtiene de forma separada la representación BERT tanto de la predicción como la respuesta real, y calcula la similaridad del coseno entre ambos vectores. La similaridad del coseno se calcula de la siguiente forma:

$$sim(A,B) = \frac{A \cdot B}{||A||||B||} \tag{4.11}$$

Normalmente, para observar el rendimiento de una métrica, se suele comparar con las medidas hechas por un humano. Se estima que una buena métrica debería de tener mayor correlación con la puntuación dada por un humano. el paper mencionado anteriormente [Chen et al., 2019] estudió y evaluó varias métricas para ver su capacidad. Los resultados aparecen en la Tabla 4.1.

Métricas	NarrativeQA	SemEval	ROPES
BLEU-1	0.617	0.443	-
BLEU-4	0.563	0.437	-
METEOR	**0.752**	**0.642**	-
ROUGE-L	0.707	0.570	-
BERTScore	0.733	0.406	0.448
F1	-	-	**0.591**

Tabla 4.1: Correlación entre las métricas y los juicios humanos mediante el coeficiente de correlación de Spearman (ρ) en distintas bases de datos. El guión '-' indica que la métrica no ha sido usada.

El problema de estas métricas es que ninguna de ellas se adapta bien en sistemas generativos, ya que frases con palabras distintas pueden tener el mismo significado, y esto dificulta su evaluación. Por este motivo, se creó **KPQA** [Lee et al., 2020]. Esta métrica usa un modelo BERT para representar la importancia que tiene cada token en la frase. Viendo la importancia, se evaluará cada token junto a su grado de importancia. En la Figura 4.7 se muestra de forma gráfica el funcionamiento de esta métrica.

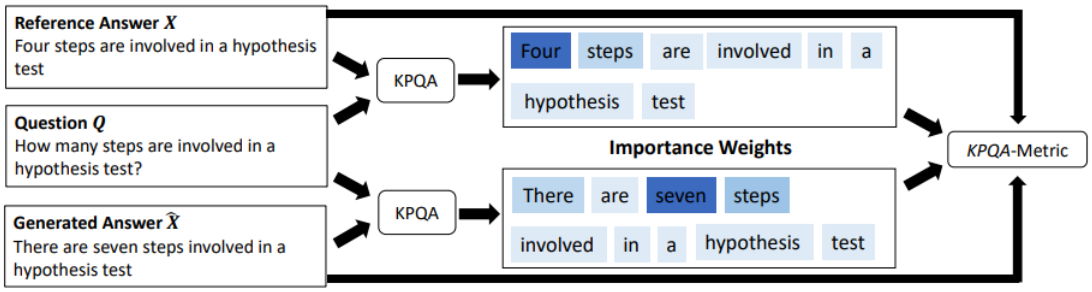

Figura 4.6: Uso de la métrica KPQA. El grado de azul indica la importancia del token en la frase.

4.3. Análisis de bases de datos

El rendimiento de un modelo se mide con las métricas, pero para que estos modelos lleguen a ser buenos tendrán que entrenarse en una base de datos de calidad. Un modelo de aprendizaje necesita de datos para aprender y comprender la tarea que quiere realizar, y sin un buen corpus de información este no podrá aprender. Por este motivo, se ha creado una sección específica para tratar las distintas bases de datos existentes.

Las bases de datos están estructuradas de distinta forma dependiendo de lo que se quiere hacer. Por ejemplo, en los grandes modelos de lenguaje, lo único que necesita es texto no estructurado. Este se refiere a un contenido escrito que carece de metadatos y que no puede indexarse ni asignarse fácilmente a los campos estándar de las bases de datos. Aun y todo, la tarea de entrenamiento no es tan sencilla ya que la dificultad se encuentra en recoger la cantidad de datos que estos modelos necesitan para rendir como rinden los mejores modelos. Por otro lado y volviendo al campo estudiado en este trabajo, las bases de datos usados en QA suelen utilizar los parámetros 'contexto', 'pregunta' y 'respuesta'. Contexto se refiere a un párrafo donde suele estar la respuesta incluida. Dependiendo de la base de datos, se suele tener varias preguntas en un mismo contexto, varias respuestas posibles, etc.

La base de datos más utilizada es **SQuAD** [Rajpurkar et al., 2016]. Esta base de datos tiene 2 versiones. La primera versión contiene más de 100.000 preguntas. La segunda versión es más útil ya que añade la posibilidad de que la respuesta no esté contenida en el contexto, creando 50.000 preguntas más. En total, hay más de 150 mil preguntas para poder entrenar modelos de comprensión.

Aparte de SQuAD, existen más bases de datos populares útiles. Algunos se centran en una única respuesta, mientras otros facilitan respuestas parecidas pero igual de validas. Varias bases de datos también poseen un apartado de acertar la respuesta, dando varias soluciones, con el objetivo de adivinar cual de ellas es la correcta .El artículo de Fengbin Zhu y varios investigadores más de la universidad nacional de Singapur [Zhu et al., 2021] hicieron un breve resumen explicando distintos tipos de bases de datos populares. La representación creada se expone en la Figura 4.7 y 4.8, y en la Tabla 4.2. En dicha tabla también se ha añadido alguna base de datos más, como ElkarHizketak [Otegi et al., 2020]. La Figura 4.7 muestra la cantidad de preguntas (medido en miles, o k) por cada base de datos y la Figura 4.8 muestra el porcentaje de dominios tratados.

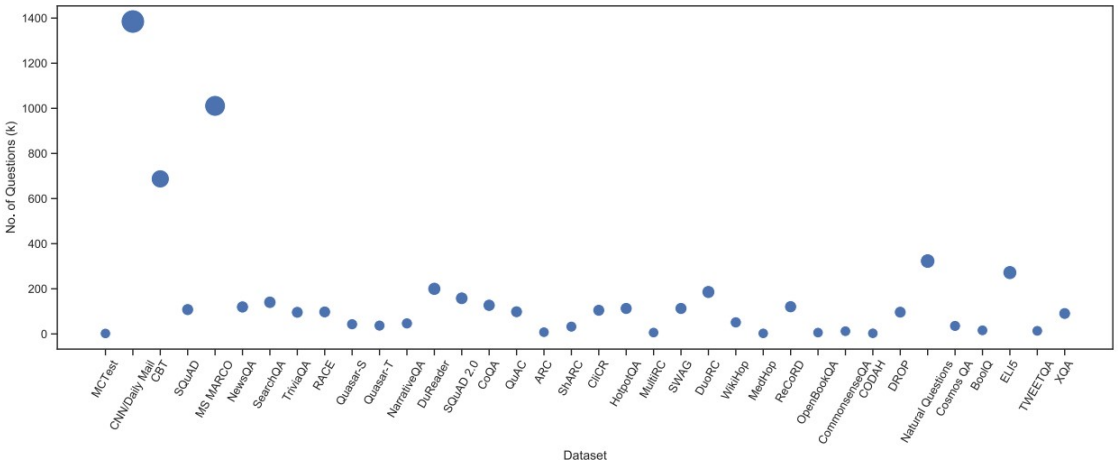

Figura 4.7: Cantidad de preguntas en cada base de datos.

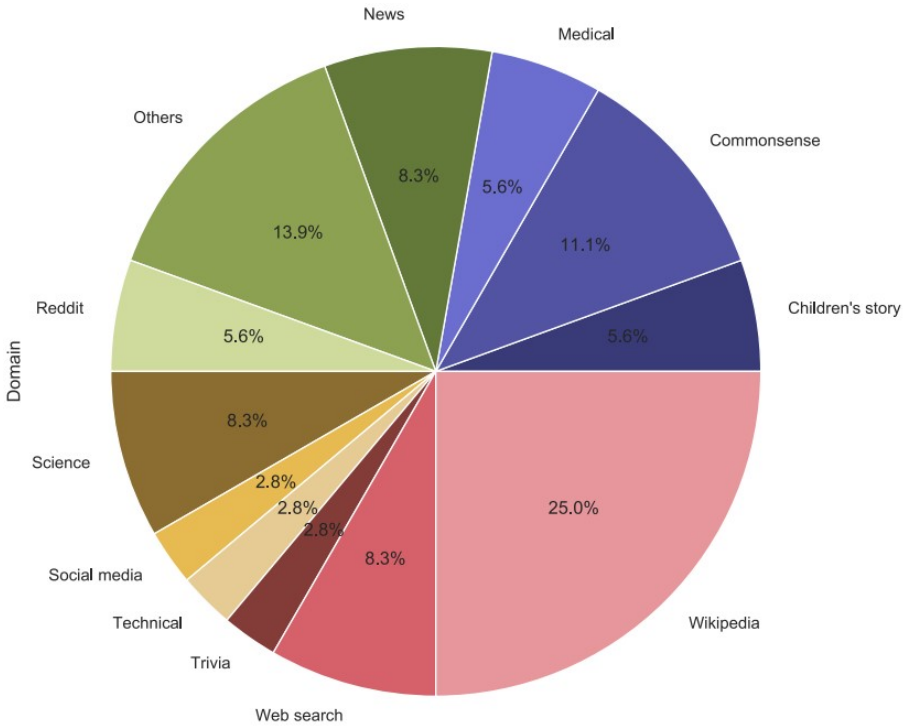

Figura 4.8: Distribución de las bases de datos con respecto al dominio de la información.

Base de datos	Dominio	#Q (k)	Tipo de respuesta	OpenQA
MCTest [Richardson et al., 2013]	Cuento para niños	2.0	Múltiples respuestas	No
CNN/Daily Mail [Hermann et al., 2015]	Noticias	1,384.8	Entidades	Si
CBT [Hill et al., 2015]	Cuento para niños	687.3	Múltiples respuestas	No
SQuAD	Wikipedia	108.0	Fragmento	Si
MS MARCO [Bajaj et al., 2016]	Búsqueda Web	1,010.9	Forma libre Booleano No contestable	Si
NewsQA [Trischler et al., 2016]	Noticias	119.6	Fragmentos No contestable	Si
SearchQA [Dunn et al., 2017]	Búsqueda Web	140.4	Fragmentos	Si
TriviaQA [Joshi et al., 2017]	Trivia	95.9	Fragmentos Forma libre	No
RACE [Lai et al., 2017]	Ciencia	97.6	Múltiples respuestas	No
Quasar-T [Dhingra et al., 2017]	Reddit	43.0	Forma libre	Si
Quasar-S	Tecnico	37.0	Entidades	Si
NarrativeQA [Kočiský et al., 2017]	Otros	127.0	Forma libre	No
DuReader [He et al., 2017]	Busqueda Web	200.0	Forma libre Booleano	Si
SQuAD 2.0	Wikipedia	158.0	Fragmentos No contestable	Si
CoQA [Reddy et al., 2018]	Otros	127.0	Forma libre No contestable Booleano	No
QuAC [Choi et al., 2018]	Wikipedia	98.4	Fragmentos No contestable Booleano	Si
ARC [Yadav et al., 2019]	Ciencia	7.7	Múltiples respuestas	No
CliCR [Šuster and Daelemans, 2018]	Médico	104.9	Fragmentos	No
HotpotQA [Yang et al., 2018]	Wikipedia	113.0	Fragmentos Booleano No contestable	Si
MultiRC [Khashabi et al., 2018]	Otros	6.0	Múltiples respuestas	No
SWAG [Zellers et al., 2018]	Sentido común	186.0	Forma libre	No
DuoRC [Saha et al., 2018]	Otros	186.0	Forma libre Fragmentos	No
WikiHop [Tu et al., 2019]	Wikipedia	51.3	Múltiples respuestas	Si
ReCoRD [Zhang et al., 2018]	Noticias	120.0	Múltiples respuestas	No
OpenBookQA [Mihaylov et al., 2018]	Ciencia	5.9	Múltiples respuestas	No
ElkarHizketak [Otegi et al., 2020]	Wikipedia en vasco	1.3	Fragmentos	Si

Tabla 4.2: Bases de datos más populares y sus características. **#Q (k)** indica la cantidad de preguntas de las bases de datos en miles. **OpenQA** indica si es de dominio abierto o no.

5. CAPÍTULO

Experimentación

Durante la elaboración de la idea del trabajo y mientras se analizaban distintos tipos de modelos, surgían varias preguntas. ¿Hasta qué punto era posible entrenar modelos con pocos recursos? ¿Cuál será el rendimiento de un modelo QA entrenado en un idioma con menos datos de entrenamiento como podría ser el euskera? De estas preguntas surge la motivación para hacer dos experimentos. Un experimento en el cual se estudie la capacidad de respuesta en modelos entrenados con recursos limitados, y otro para comprobar la eficiencia de modelos entrenados en euskera. En las próximas dos secciones se estudiará cada uno de estos planteamientos.

Esta sección se ha creado con el propósito de aplicar a la práctica lo aprendido y explicado en el análisis. Se utilizarán algunos de los modelos analizados, se evaluarán con las métricas descritas, y se entrenará con bases de datos mencionadas anteriormente.

5.1. Experimento 1: Entrenamiento con recursos limitados

El objetivo general de este experimento es el observar el rendimiento de un sistema QA entrenado con recursos escasos como podría ser un portátil o con una base de datos con poca información. La motivación viene al observar que las pequeñas empresas a veces no tienen material e información suficiente como para crear modelos capaces de responder preguntas con una buena precisión, ya sea en un dominio que interese a la propia empresa o de dominio abierto. Por este motivo, se quiere estudiar su viabilidad.

En concreto, lo que se va a hacer es entrenar a DistilBERT [Sanh et al., 2019] preentrenado para completar la tarea de responder preguntas en distintas bases de datos. Este modelo es un modelo de lenguaje con la misma arquitectura que BERT, pero con un 60 % menos de parámetros. El punto fuerte de este modelo es que con un entrenamiento más rápido, y con menos parámetros, el rendimiento de este es del 97 % respecto al BERT base. A diferencia de un entrenamiento normal, se extraerá la mayoría de información de dichas bases de datos para que estos tengan menos cantidad de datos para poder entrenarse. Se va a entrenar en dos ordenadores distintos, un ordenador portátil y un ordenador de sobremesa. La idea es eliminar la suficiente cantidad de información en cada base de datos como para que el proceso de entrenamiento se pueda hacer en menos de 2 horas. Al finalizar cada proceso, se evaluará con las métricas f1 y exact match, aparte de ver gráficamente cómo ha ido evolucionando el modelo en cada paso global, o *global step*.

5.1.1. Escenario

Utilizaremos dos ordenadores. Por una parte, el primer ordenador es un portátil con procesador 'INTEL core i7-7500U', 16GB de RAM y no tiene tarjeta gráfica dedicada. Por otro lado, el segundo ordenador es de sobremesa con procesador 'AMD RYZEN 7 3800X', 16GB de RAM y una tarjeta gráfica 'NVIDIA GEFORCE RTX 3060 TI'. De forma separada ejecutaremos modelos sacados de huggingface. En concreto, los modelos son 'distilbert-base-uncased' y 'distilbert-base-multilingual-cased'. Se entrenan en dos bases de datos, en 'SQuAD' y en 'TydiQA' [Clark et al., 2020]. La segunda base de datos mencionada es plurilingüe, por este motivo se han cogido dos modelos DistilBERT diferentes, ya que el primero este preentrenado en un corpus con texto puramente en inglés, y el segundo con un corpus en varios idiomas. Se recogerán partes aleatorias de la base de datos de tal forma que el entrenamiento no dure más que 2 horas. Finalmente, los resultados obtenidos se observarán principalmente en las métricas F1 y Exact Match.

En todos los modelos, los parámetros de entrenamiento se han definido de la siguiente manera:

- *Evaluation strategy: epoch*.
- *Learning rate*: $2e-5$.
- *Training batch size*: 16.
- *Evaluation batch size*: 16.

- *Epoch size*: 3.
- *Weight decay*: 0,01.

5.1.2. Resultado

Una vez entrenado a los modelos en las bases de datos SQuAD y TydiQA, los resultados obtenidos se pueden observar en la Tabla 5.1. A la hora de ejecutar el modelo en el portátil, se ha observado que para que el entrenamiento no dure más de dos horas no se podía coger más de 10.500 ejemplos. Por este motivo, se han extraído 3.500 ejemplos para que mantenga los mismos parámetros de entrenamiento que el ordenador sobremesa y pueda hacer 3 pasadas en la base de datos, o (*epoch*). Dicho de otra forma, se va a entrenar con el 3 % de la base de datos SQuAD, y con el 1.5 % de TydiQA. Gracias a tener una tarjeta gráfica, el ordenador de sobremesa pueda tratar el corpus completo en ambas bases de datos.

Modelo, dataset y tipo de entrenamiento	**F1**	**Exact Match**
DistillBERT SQuAD no limitado	0.8701	0.5618
DistilBERT SQuAD limitado	0.6157	0.2674
DistilBERT TydiQA no limitado	0.7490	0.6305
DistilBERT TydiQA limitado	0.5267	0.2455

Tabla 5.1: Comparación del resultado de cada modelo entrenado con recursos limitados y con el corpus completo. Se usa la métrica F1 y exact match (EM).

Viendo los resultados obtenidos, se puede concluir varias cosas. La primera es que, observando la métrica de F1, el modelo entrenado en el corpus completo, ha obtenido un resultado alto. Sin embargo, en la otra métrica, el resultado no ha sido tan bueno. Esto se debe a que la respuesta dada por el modelo se acerca mucho a la respuesta real, pero no la consigue extraer de principio a fin. Por otro lado, se puede ver como, aun entrenando con tan solo el 3 % y el 1.5 % de la base de datos SQuAD y TydiQA, el resultado obtenido no es mucho peor. La Tabla mencionada anteriormente muestra como la diferencia de la métrica F1 en DistilBERT SQuAD es de 0,2544, y en DistilBERT TydiQA es de 0,2223. Esto se debe a que el modelo adapta su mayor parte al en los primeros pasos, o *steps*. Esto se puede observar en la Figura 5.1. En esta Figura se ve como la función de pérdida devuelve unos valores muy grandes al comienzo, y luego va decrementando cada vez más lento. Por este motivo apenas existe diferencia en los resultados de las métricas.

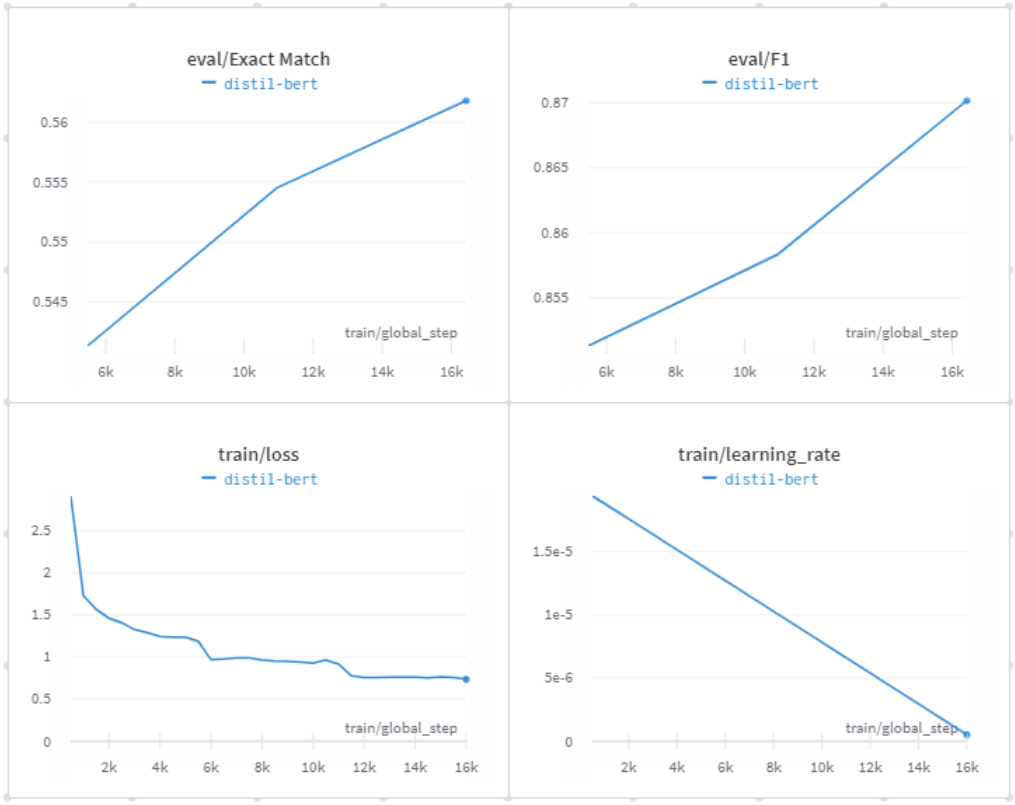

Figura 5.1: Gráfica de la evolución de los parámetros evaluadores en el primer experimento respecto al modelo DistilBERT entrenado con el corpus de SQuAD completo.

5.2. Experimento 2: Rendimiento de modelos QA en euskera

Lo que se va a estudiar en este experimento es la eficacia de los modelos en euskera. Normalmente, cuando se entrenan modelos para la tarea de responder preguntas, como se ha dicho anteriormente se suelen coger modelos de lenguaje preentrenados. El problema está en que estos modelos son preentrenados en un idioma popular como podría ser el inglés o el castellano. En otras ocasiones el corpus facilitado para el entrenamiento no solo contiene un idioma, sino varios. Los modelos que generaban los mejores resultados en la actualidad no disponía de su versión únicamente en euskera. Por este motivo, el grupo IXA preentrenó un modelo en este idioma [Agerri et al., 2020]. El objetivo de este experimento es comprobar el rendimiento de este modelo comprado con una versión en un idioma popular como el inglés o el castellano. Los 3 modelos están creados con la misma arquitectura, BERT_{BASE}. Es decir, los 3 modelos contienen 12 capas de codificación de transformadores, un tamaño oculto de 768 y 12 cabezas de auto-atención (self-attention

heads), con un total de 110 millones de parámetros. La diferencia es que estos están entrenados en un corpus distinto. El paper original [Devlin et al., 2018] explica que el modelo ha sido entrenado con más de 3 mil millones de palabras, mientras que el corpus de preentrenamiento de BERTeus contiene aproximadamente 200 millones de tokens. En este experimento se entrenará a BERTeus para la tarea de responder preguntas, y mediante las métricas de F1 y exact match se comprobará la calidad del modelo comparado con BERT entrenado tanto en el inglés como en castellano.

5.2.1. Escenario

Para entrenar el modelo BERT$_{eus}$ se utilizará la base de datos llamado ElkarHizketak [Otegi et al., 2020]. Este contiene más de 1.500 preguntas. Existe la posibilidad de que la respuesta no esté contenida en el contexto, con la que aumenta la dificultad de entrenamiento. El modelo preentrenado con texto puramente en castellano se llama BETO [Cañete et al., 2020]. Como no existe un modelo BETO entrenado en un dataset que tenga en cuenta el hecho de que no exista la respuesta, se ha tenido que entrenar por primera vez. Se ha utilizado la segunda version del dataset 'SQuAD-es' [Casimiro Pio et al., 2019]. Ambos entrenamientos se ejecutarán en un servidor que contiene cuatro 'NVIDIA TESLA V100 SXM2' de 16GB RAM trabajando en paralelo. Para el entrenamiento se usarán estos argumentos:

- *Evaluate during training*: True.
- *Max seq length*: 128.
- *Train epochs*: 5.
- *Evaluate during training steps*: 1000.
- *Train batch size*: 128.
- *Eval batch size*: 64.

5.2.2. Resultado

Después de ejecutar el entrenamiento, se han obtenido dos tipos de resultados. El primero, expuesto en la Figura 5.2 muestra la evolución del modelo BERT$_{eus}$ durante todo el

proceso de entrenamiento. Por otro lado, la Tabla 5.2 muestra una comparativa de los resultados obtenidos en BERT$_{eus}$ con los obtenidos en BETO y en el modelo BERT original. comparativa.

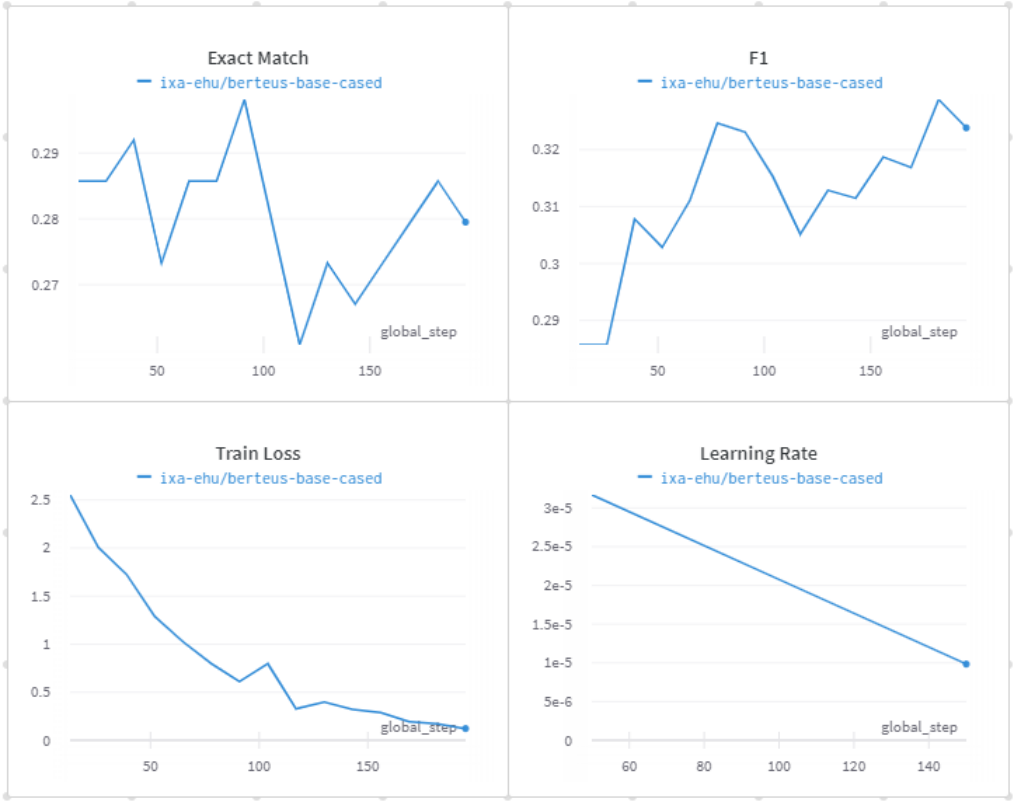

Figura 5.2: Gráfica de la evolución de los parámetros evaluadores de BERT$_{eus}$ en función de los pasos globales. *Global step* indica la cantidad de *batch* que el modelo ha ejecutado.

Una vez visto el resultado obtenido, se han sacado las siguientes conclusiones:

Modelo	**F1**	**EM**
BERT$_{esp}$ (BETO SQuADes2.0)	0.6270	0.5460
BERT$_{eng}$ (BERT SQuAD2.0)	**0.7011**	**0.6595**
BERT$_{eus}$ (IXA/BERTeusElkarHizketak)	0.3238	0.2795

Tabla 5.2: Comparación de las métricas F1 y Exact Match entre los modelos en idiomas populares y BERT$_{eus}$.

- Como el primer experimento bien indicaba, el hecho de tener preentrenado un modelo de lenguaje que comprende el texto pasado facilita notablemente la eficiencia del modelo final. Este razonamiento se obtiene comprobando las gráficas devueltas por el entrenamiento. Se puede observar que en los primeros pasos, o *steps*, el modelo ya obtenía buenos resultados, y a medida de que ha ido avanzando, el incremento en los resultados obtenidos ha ido desacelerando.

- Como era de esperar, el modelo no ha dado mejores resultados que en los otros idiomas. La diferencia en la métrica de F1 de BERT_{eus} comparado con BERT es de 0,3773, y comparado con BETO es de 0,3032. Sea por la cantidad del corpus de preentrenamiento, por la calidad del *fine-tuning*, o por la complejidad del lenguaje, el modelo no ha podido obtener resultados cercanos a el de los otros idiomas.

- Aunque la métrica F1 haya ido mejorando a lo largo de las evaluaciones, la métrica exact match no ha incrementado su valor. De hecho, la última evaluación ha devuelto peor valor que la primera. La escasez de preguntas en la base de datos en euskera seguramente ha sido el causante de este resultado.

6. CAPÍTULO

Conclusiones y trabajo futuro

En este trabajo, se planteaba la importancia de implementar modelos de aprendizaje profundo capaces de responder a preguntas viendo la cantidad de actividades en las que están involucrados los sistemas de pregunta respuesta, o QA. También se comentaba la cantidad de modelos que están siendo publicados últimamente y la necesidad de analizar y hacer un resumen sobre estos para facilitar su comprensión. El principal objetivo de este trabajo era exponer los distintos tipos de sistemas QA, ya que estos están dando resultados notablemente buenos. Se ha cumplido dicho objetivo, ya que se ha explicado cómo estos modelos funcionan, como se mide la calidad de estos, y como se entrenan y con qué tipo de datos. También se han podido probar distintos experimentos para poner a prueba su rendimiento en situaciones con datos escasos. Las conclusiones principales han sido las siguientes:

- Los modelos de lenguaje preentrenados mejoran considerablemente la calidad del output. Estos modelos de lenguaje, al ser modelos preentrenados con una cantidad inmensa de datos, son considerados modelos que comprenden texto. Como se comentaba en el capítulo de antecedentes, se ha visto que en distintas bases de datos probadas la comprensión de estos modelos las calificaciones de los mismos eran muy altas. También, en los experimentos se ha podido comprobar que tener un modelo preentrenado es útil ya que desde las primeras evaluaciones los sistemas QA daban resultados importantes. Si no fuera por la comprensión de texto, un modelo con pesos iniciados aleatoriamente tendría resultados cercanos a cero en sus primeros pasos.

- Los modelos generativos de texto rinden mejor en los contextos largos que los *readers* extractivos. Sucede lo contrario en los contextos cortos. En el apartado de análisis de modelos se vio que se podían separar en dos tipos, extractivos y generativos. A parte de mostrar cómo estos modelos funcionan, se hizo un análisis sobre el rendimiento de los modelos, y se concluyó que los *readers* extractivos generalmente trabajan mejor que los generativos, pero estos últimos son mejores en contextos de mayor extensión.

- Existen varias métricas capaces de medir el rendimiento de los modelos. Pero las métricas para calificar la calidad de un modelo generativo no son tan buenas como el de los extractivos. Es cierto que es más complejo medir la calidad de texto generado, pero puede que este sea una de las razones por las cuales los generativos funcionen algo peor.

Viendo que en la práctica estos sistemas pueden ser utilizados para infinidades de cosas, todavía queda camino de estudio para corregir limitaciones que estos modelos poseen:

- Es necesario generar cada vez más cantidad de datos para preentrenar y fine-tunear modelos. Sin embargo, el *fine-tuning* se hace mediante aprendizaje supervisado. Esto quiere decir que los datos tienen que ser estructurados de una forma y creada por humanos para que el modelo pueda aprender de él. Sin embargo, investigadores reconocidos internacionalmente, como Yan LeCun, proponen que quizás para una mayor inteligencia y una mejora del rendimiento, el modelo debería entrenarse mediante clasificación semi-supervisada [LeCun and Misra, 2021]. Por este motivo, uno de los limitantes es la falta de investigación en encontrar métodos de aprendizaje semi-supervisado para el ámbito de pregunta-respuesta.

- Euskera es un idioma que no posee de grandes cantidades de información para procesar y entrenar a modelos, ya que es un idioma que no es hablado internacionalmente. La cantidad de datos en internet es menor. Por este motivo, la calidad de los resultados no son tan buenos como en modelos con otros idiomas más populares.

Dentro del procesamiento de lenguaje natural, el campo de pregunta respuesta está avanzando a gran velocidad y hay mucha cantidad de trabajo por delante. Por esa razón, se plantean dos posibles trabajos a futuro.

A corto plazo, sería interesante observar el rendimiento de los modelos entrenados en distinto lenguaje pero con una misma cantidad de datos, tanto en el preentrenamiento como

en el *fine-tuning*. En este trabajo, el segundo experimento ha analizado el resultado de los modelos actuales en euskera, pero estos modelos no habían sido entrenados con la misma longitud del corpus de entrenamiento. Se propone medir lo explicado anteriormente para comprobar si las métricas devuelven un resultado similar o no, y así comprobar si la cantidad de datos es el único influyente o si también lo son la arquitectura, el tipo de entrenamiento y las métricas de evaluación.

A largo plazo, puede que cuando se encuentre una métrica capaz de medir bien la calidad de texto generado, y no solo extraído, sea un paso adelante para crear métodos de aprendizaje semi-supervisado y de esta forma crear modelos que aprendan de sus propios fallos con apenas intervención humana. Es posible que el objetivo de encontrar un modelo QA capaz de generar texto y no solo extraerlo con una perspectiva de aprendizaje semi-supervisado sea una forma de superar muchos límites que tienen actualmente los modelos de aprendizaje en este ámbito. Consiguiendo esto y acelerando su entrenamiento en hardware potente, es posible que se llegue a crear un modelo mucho mejor que los anteriores.

A parte, crear modelos capaces de generar preguntas pasado un contexto como parámetro podría ser una de las piezas clave a la hora de crear ese modelo QA ideal. Como trabajo a futuro, si un modelo es capaz, de forma semi-supervisada, crear preguntas, generar respuestas, y medir la calidad de sus repuestas de una forma eficaz, la calidad de los resultados sería lo suficientemente bueno como para poder aplicar a estos en la práctica de forma más fiable.

BIBLIOGRAFÍA

[Agerri et al., 2020] Agerri, R., Vicente, I. S., Campos, J. A., Barrena, A., Saralegi, X., Soroa, A., and Agirre, E. (2020). Give your text representation models some love: the case for basque.

[Bajaj et al., 2016] Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., and Wang, T. (2016). Ms marco: A human generated machine reading comprehension dataset.

[Banerjee and Han, 2009] Banerjee, P. and Han, H. (2009). Language modeling approaches to information retrieval. *JCSE*, 3:143–164.

[Banerjee and Lavie, 2005] Banerjee, S. and Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

[Brill et al., 2002] Brill, E., Dumais, S., and Banko, M. (2002). An analysis of the AskMSR question-answering system. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 257–264. Association for Computational Linguistics.

[Brown et al., 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

[Casimiro Pio et al., 2019] Casimiro Pio, C., Marta R., C.-j., and Jose A. R., F. (2019). Automatic Spanish Translation of the SQuAD Dataset for Multilingual Question Answering. *arXiv e-prints*, page arXiv:1912.05200v1.

[Cañete et al., 2020] Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H., and Pérez, J. (2020). Spanish pre-trained bert model and evaluation data. In *PML4DC at ICLR 2020*.

[Chen et al., 2019] Chen, A., Stanovsky, G., Singh, S., and Gardner, M. (2019). Evaluating question answering evaluation. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 119–124, Hong Kong, China. Association for Computational Linguistics.

[Chen et al., 2017] Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading wikipedia to answer open-domain questions.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation.

[Choi et al., 2018] Choi, E., He, H., Iyyer, M., Yatskar, M., Yih, W.-t., Choi, Y., Liang, P., and Zettlemoyer, L. (2018). Quac : Question answering in context.

[Chowdhery et al., 2022] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. (2022). Palm: Scaling language modeling with pathways.

[Clark et al., 2020] Clark, J. H., Choi, E., Collins, M., Garrette, D., Kwiatkowski, T., Nikolaev, V., and Palomaki, J. (2020). Tydi qa: A benchmark for information-seeking question answering in typologically diverse languages.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., Google, K. T., and Language, A. I. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

[Dhingra et al., 2017] Dhingra, B., Mazaitis, K., and Cohen, W. W. (2017). Quasar: Datasets for question answering by search and reading.

[Dunn et al., 2017] Dunn, M., Sagun, L., Higgins, M., Guney, V. U., Cirik, V., and Cho, K. (2017). Searchqa: A new q&a dataset augmented with context from a search engine.

[Green et al., 1961] Green, B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball. page 219. ACM Press.

[Harabagiu et al., 2003] Harabagiu, S., Maiorano, S., and Pasca, M. (2003). Open-domain textual question answering techniques. *Natural Language Engineering*, 9:231 – 267.

[He et al., 2017] He, W., Liu, K., Liu, J., Lyu, Y., Zhao, S., Xiao, X., Liu, Y., Wang, Y., Wu, H., She, Q., Liu, X., Wu, T., and Wang, H. (2017). Dureader: a chinese machine reading comprehension dataset from real-world applications.

[Hermann et al., 2015] Hermann, K. M., Kočiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend.

[Hill et al., 2015] Hill, F., Bordes, A., Chopra, S., and Weston, J. (2015). The goldilocks principle: Reading children's books with explicit memory representations.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[Huang et al., 2015] Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional lstm-crf models for sequence tagging.

[Izacard and Grave, 2020] Izacard, G. and Grave, E. (2020). Leveraging passage retrieval with generative models for open domain question answering.

[Joshi et al., 2017] Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. (2017). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension.

[Jurafsky and Martin, 2000] Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, USA, 1st edition.

[Karpukhin et al., 2020] Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering.

[Khashabi et al., 2018] Khashabi, D., Chaturvedi, S., Roth, M., Upadhyay, S., and Roth, D. (2018). Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262, New Orleans, Louisiana. Association for Computational Linguistics.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

[Kočiský et al., 2017] Kočiský, T., Schwarz, J., Blunsom, P., Dyer, C., Hermann, K. M., Melis, G., and Grefenstette, E. (2017). The narrativeqa reading comprehension challenge.

[Kupiec, 1993] Kupiec, J. (1993). Murax. pages 181–190. ACM Press.

[Kwok et al., 2001] Kwok, C., Etzioni, O., and Weld, D. S. (2001). Scaling question answering to the web. *ACM Transactions on Information Systems*, 19:242–262.

[Lai et al., 2017] Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. (2017). Race: Large-scale reading comprehension dataset from examinations.

[LeCun and Misra, 2021] LeCun, Y. and Misra, I. (2021). Self-supervised learning: The dark matter of intelligence.

[Lee et al., 2020] Lee, H., Yoon, S., Dernoncourt, F., Kim, D. S., Bui, T., Shin, J., and Jung, K. (2020). Kpqa: A metric for generative question answering using keyphrase weights.

[Lewis et al., 2019] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

[Lewis et al., 2020] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks.

[Lin, 2004] Lin, C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

[Lin et al., 2018] Lin, Y., Ji, H., Liu, Z., and Sun, M. (2018). Denoising distantly supervised open-domain question answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1736–1745, Melbourne, Australia. Association for Computational Linguistics.

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

[Luo et al., 2022] Luo, M., Hashimoto, K., Yavuz, S., Liu, Z., Baral, C., and Zhou, Y. (2022). Choose your qa model wisely: A systematic study of generative and extractive readers for question answering.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

[Mihaylov et al., 2018] Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. (2018). Can a suit of armor conduct electricity? a new dataset for open book question answering.

[Otegi et al., 2020] Otegi, A., Agirre, A., Campos, J. A., Soroa, A., and Agirre, E. (2020). Conversational question answering in low resource scenarios: A dataset and case study for Basque. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 436–442, Marseille, France. European Language Resources Association.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

[Raffel et al., 2019] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer.

[Rajpurkar et al., 2016] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text.

[Reddy et al., 2018] Reddy, S., Chen, D., and Manning, C. D. (2018). Coqa: A conversational question answering challenge.

[Reed et al., 2022] Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Bruce, J., Razavi, A., Edwards, A., Heess, N., Chen, Y., Hadsell, R., Vinyals, O., Bordbar, M., and de Freitas, N. (2022). A generalist agent.

[Richardson et al., 2013] Richardson, M., Burges, C. J., and Renshaw, E. (2013). MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203, Seattle, Washington, USA. Association for Computational Linguistics.

[Robertson and Zaragoza, 2009] Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

[Saha et al., 2018] Saha, A., Aralikatte, R., Khapra, M. M., and Sankaranarayanan, K. (2018). Duorc: Towards complex language understanding with paraphrased reading comprehension.

[Sammut and Webb, 2010] Sammut, C. and Webb, G. I., editors (2010). *TF–IDF*, pages 986–987. Springer US, Boston, MA.

[Sanh et al., 2019] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

[Schwager and Solitario, 2019] Schwager, S. and Solitario, J. (2019). Question and answering on squad 2.0: Bert is all you need.

[Sherstinsky, 2020] Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306.

[submission, 2021] submission, A. A. (2021). Machine reading comprehension: Generative or extractive reader?

[Tan et al., 2017] Tan, C., Wei, F., Yang, N., Du, B., Lv, W., and Zhou, M. (2017). S-net: From answer extraction to answer generation for machine reading comprehension.

[Trischler et al., 2016] Trischler, A., Wang, T., Yuan, X., Harris, J., Sordoni, A., Bachman, P., and Suleman, K. (2016). Newsqa: A machine comprehension dataset.

[Tu et al., 2019] Tu, M., Wang, G., Huang, J., Tang, Y., He, X., and Zhou, B. (2019). Multi-hop reading comprehension across multiple documents by reasoning over heterogeneous graphs.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

[Wang and Jiang, 2015] Wang, S. and Jiang, J. (2015). Learning natural language inference with lstm.

[Wolf et al., 2020] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

[Woods, 1973] Woods, W. A. (1973). Progress in natural language understanding-an application to lunar geology. page 441. IEEE Computer Society.

[Yadav et al., 2019] Yadav, V., Bethard, S., and Surdeanu, M. (2019). Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics.

[Yang et al., 2018] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. (2018). Hotpotqa: A dataset for diverse, explainable multi-hop question answering.

[Zellers et al., 2018] Zellers, R., Bisk, Y., Schwartz, R., and Choi, Y. (2018). Swag: A large-scale adversarial dataset for grounded commonsense inference.

[Zhang et al., 2018] Zhang, S., Liu, X., Liu, J., Gao, J., Duh, K., and Van Durme, B. (2018). Record: Bridging the gap between human and machine commonsense reading comprehension.

[Zhang et al., 2019] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2019). Bertscore: Evaluating text generation with bert.

[Zhu et al., 2021] Zhu, F., Lei, W., Wang, C., Zheng, J., Poria, S., and Chua, T.-S. (2021). Retrieving and reading: A comprehensive survey on open-domain question answering.

[Šuster and Daelemans, 2018] Šuster, S. and Daelemans, W. (2018). Clicr: A dataset of clinical case reports for machine reading comprehension.