UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Adaptación de modelos de lenguaje grandes para la generación de lenguaje natural a partir de palabras clave en sistemas aumentativos y alternativos de comunicación

TRABAJO FIN DE GRADO

Grado en Ciencia de Datos

Autor: Silvia Alegre Villa

Tutor: Jorge Civera Saiz

Curso 2023-2024

Resum

Els Sistemes Augmentatius i Alternatius de Comunicació (SAAC) son eines essencials per a facilitar la comunicació de les persones amb dificultats en la utilització del llenguatge. Aquest sistemes permeten a l'usuari la selecció de pictogrames associats a paraules claus que conformaran l'oració que es desitja comunicar. Posteriorment, aquesta oració pot ser sintetitzada amb veu humana. Els recents avanços en l'àrea del processament del llenguatge natural i, en concret, la proliferació de models de llenguatge grans ofereixen noves perspectives per a millorar els SAAC. En particular, aquest treball explorarà com aquests models de llenguatge poden millorar l'expressivitat de la comunicació dels SAAC quan s'utilitzen per a la generació de llenguatge natural a partir de les paraules clau (pictogrames) seleccionades per l'usuari. D'aquesta manera, aquest treball evaluar'a el rendiment d'aquests models quan són adaptats per a la seua integració en els SAAC. Awuesta evaluació es durà a terme utilitzant conjunts de test reals en espanyol i anglés extrets del portal del Centre Aragonés per a la Comunicació Augmentativa i Alternativa.

Paraules clau: Aprentatje Automàtic, Transformers, Models de Llenguatge Gran

Resumen

Los Sistemas Aumentativos y Alternativos de Comunicación (SAAC) son herramientas vitales para facilitar la comunicación de las personas con dificultades en la utilización del lenguaje. Estos sistemas permiten al usuario la selección de pictogramas asociados a palabras clave que conformarán la oración que se desea comunicar. Posteriormente, esta oración puede ser sintetizada con voz humana. Los recientes avances en el área del procesamiento de lenguaje natural y, en concreto, la proliferación de modelos de lenguaje grandes ofrece nuevas perspectivas para mejorar los SAAC. En particular, este trabajo explorará cómo estos modelos de lenguaje pueden mejorar la expresividad de la comunicación de los SAAC cuando se utilizan para la generación de lenguaje natural a partir de las palabras clave (pictogramas) seleccionadas por el usuario. De esta forma, este trabajo evaluará el rendimiento de estos modelos cuando son adaptados para su integración en los SAAC. Esta evaluación se llevará a cabo utilizando conjuntos de test reales en español e inglés extraídos del portal del Centro Aragonés para la Comunicación Aumentativa y Alternativa.

Palabras clave: Aprendizaje Automático, Transformers, Modelos de Lenguaje Grandes

Abstract

Augmentative and Alternative Communication (AAC) systems are vital tools for facilitating communication for individuals with difficulties using language. These systems allow users to select pictograms associated with key words that will form the sentence that is wished to communicate. Then, the sentence can be synthesized with a human voice. Recent advances in the field of natural language processing, and specifically the proliferation of large language models, offer new perspectives for improving AAC systems. In particular, this work will explore how these language models can enhance the expressiveness of AAC communication when used to generate natural language from the key words (pictograms) selected by the user. In this way, this work will evaluate the

performance of these models when adapted for integration into AAC systems. This evaluation will be carried out using real test sets in spanish and english extracted from the portal of the Aragonese Center for Augmentative and Alternative Communication.

Key words: Machine Learning, Transformers, Large Language Models

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En este trabajo se explorará cómo los Modelos de Lenguaje Grandes (LLMs) pueden ser aplicados en el campo de los Sistemas Aumentativos y Alternativos de Comunicación (SAAC). Para ello, se entrenarán y evaluarán algunos de los LLMs más populares actualmente, con el objetivo de realizar tareas de generación de texto en este contexto específico.

En este primer capítulo se presenta la motivación y los objetivos principales del trabajo, así como una visión general de la estructura que sigue el documento.

1.1 Motivación

La comunicación y el lenguaje son dos pilares fundamentales de la sociedad actual, pues constituyen la base de las relaciones interpersonales, permitiendo el intercambio de ideas e información. Gracias a ello podemos transmitir a los demás nuestros pensamientos, emociones y necesidades, permitiéndonos participar en la vida en sociedad. Sin embargo, para algunas personas, el hecho de comunicarse de manera satisfactoria puede suponer un gran desafío, debido a distintas causas. En estos casos, los Sistemas Aumentativos y Alternativos de Comunicación (SAAC) juegan un papel crucial.

Tal y como se explica en el portal del Centro Aragonés para la Comunicación Aumentativa y Alternativa (ARASAAC), "*los Sistemas Aumentativos y Alternativos de Comunicación (SAAC) son formas de expresión diferentes del lenguaje hablado que tienen como objetivo aumentar el nivel de expresión (aumentativo) y/o compensar (alternativo) las dificultades de comunicación que presentan algunas personas en este área*" [2].

Hay distintas razones por las cuales una persona podría necesitar utilizar un SAAC. Entre ellas encontramos la parálisis cerebral, la discapacidad intelectual, los trastornos del espectro autista, algunas enfermedades neurológicas, las distrofias musculares o las afasias.

Aunque hay muchos tipos de SAAC, todos se caracterizan por estar basados en sistemas de símbolos, ya sean gráficos (fotografías, dibujos, pictogramas, palabras o letras) o gestuales (mímica o símbolos manuales).

En este trabajo nos centraremos en los comunicadores electrónicos. Los comunicadores electrónicos son herramientas tecnológicas que pueden ser utilizados en cualquier tipo de dispositivo electrónico. Por lo general, consisten en un tablero donde aparecen distintos símbolos gráficos (pictogramas) que representan palabras. Los pictogramas pueden ser organizados y adaptados dependiendo de las necesidades de cada persona, permitiendo que cada usuario añada aquellos que necesite. De esta manera, los usua-

rios seleccionan los pictogramas para formar mensajes, que luego son traducidos por el programa en forma de habla digitalizada o texto escrito. Un ejemplo de comunicador electrónico muy utilizado actualmente es el comunicador AsTeRISCS Grid, desarrollado por ARASAAC, ilustrado en la Figura 1.1.

Figura 1.1: Comunicador AsTeRISCS Grid desarrollado por ARASAAC. Extraído de [3]

Aunque los comunicadores electrónicos son herramientas verdaderamente útiles para mejorar la comunicación de las personas, presentan ciertas limitaciones. Una de las principales es la dificultad para formar frases complejas y gramaticalmente correctas, ya que, para facilitar y simplificar su uso, estos sistemas suelen incluir solamente palabras clave en su forma básica, sin distinguir número, género o tiempo verbal, cosa que restringe en gran medida la fluidez y expresividad de la comunicación.

Algunos de los comunicadores que existen actualmente en el mercado ya emplean diferentes métodos para abordar este problema, pero existe todavía un amplio margen de mejora. Las recientes innoviaciones en el campo de la inteligencia artificial y el procesamiento de lenguaje natural ofrecen nuevas oportunidades para superar estas limitaciones. En particular, los modelos de lenguaje grandes (LLMs) tienen un gran potencial para transformar el uso de estos sistemas, pudiendo proporcionar una generación de texto mucho más natural.

1.2 Objetivos

Los objetivos principales de este proyecto son los siguientes:

1. Investigar sobre los enfoques actuales para la generación de frases en el sector de los SAAC.

2. Adaptar y evaluar modelos de lenguaje grandes de acceso público que son utilizados actualmente para la generación de frases en herramientas SAAC.

3. Comparar los resultados que se obtienen con los LLMs respecto a otros comunicadores que permiten la generación de frases que se encuentran en el mercado.

1.3 Estructura de la memoria

Este documento está dividido en 5 capítulos. El Capítulo 1 proporciona la motivación y los objetivos del trabajo, estableciendo también el contexto de la investigación. El Capítulo 2 presenta los conceptos esenciales, comenzando con una visión general del aprendizaje automático y seguido de una explicación detallada de las redes neuronales, tranformers y modelos de lenguaje grandes. En el Capítulo 3 se detallan los modelos de lenguaje utilizados, los métodos para su ajuste y adaptación al contexto y las técnicas de evaluación. El Capítulo 4 describe los datos utilizados, el entrenamiento de los modelos y los resultados obtenidos. Finalmente, el Capítulo 5 ofrece un resumen de los resultados, evalúa los objetivos alcanzados y sugiere posibles direcciones para futuras investigaciones.

CAPÍTULO 2
Fundamentos

En este capítulo se abordan los conceptos fundamentales necesarios para comprender el funcionamiento de los modelos de lenguaje grandes. En primer lugar, se presenta una visión general del aprendizaje automático, seguido de una explicación detallada sobre las redes neuronales, con un efoque particular en su aplicación en el procesamiento de texto. A continuación, se explora la arquitectura de los transformers y se presentan los modelos de lenguaje grandes, explicando detalladamente su funcionamiento.

2.1 Aprendizaje automático

El aprendizaje automático (ML, por su nombre en inglés, *machine learning*) es una disciplina dentro de la inteligencia artificial que se centra en el desarrollo y estudio de algoritmos y modelos que permiten que los sistemas puedan relizar tareas específicas sin haber sido explícitamente programados para ello. Este término fue acuñado por Arthur Samuel en el año 1959, quien creo uno de los primeros programas exitosos de esta disciplina, conocido como *the Samuel Ckeckers-playing Program* [28] (el programa de juego de damas de Samuel).

Tom Mitchell [22] define el proceso de aprendizaje de los programas en el campo del *machine learning* de la siguiente manera:

> *"Se dice que un programa aprende de la experiencia E con respecto a alguna clase de tarea T, y medida de rendimiento P, si su rendimiento en tareas en T, medido por P, mejora con la experiencia E."*

Aunque la idea principal del aprendizaje automático es esta, econtramos diferentes enfoques dependiendo del tipo de tarea que se quiera llevar a cabo, de la naturaleza de esta, de la medida del rendimiento que se utiliza para evaluar el sistema y del tipo de entrenamiento o experiencia que le proporcionamos a este.

Generalmente, los enfoques para el entrenamiento de algoritmos se agrupan en:

- **Aprendizaje supervisado**, cuyo objetivo es, a partir de unos datos de entrenamiento, encontrar la función f que realice el mejor mapeo posible entre un conjunto de entradas X y sus salidas correspondientes Y, de manera que $(X, Y) = (X, f(X))$.
- **Aprendizaje no supervisado**, que trata de modelar la estructura subyacente de un conjunto de datos para identificar relaciones y patrones, permitiendo así un entendimiento más profundo de los mismos.

- **Aprendizaje semi-supervisado**, que cae entre el supervisado y el no supervisado y utiliza una porción de datos etiquetados y no etiquetados.

- **Aprendizaje por refuerzo**, donde el algoritmo aprende a través de retroalimentaciones que va recibiendo, ajustando sus acciones con el objetivo de maximizar una recompensa acumulada a lo largo del tiempo [21].

La tarea principal de este trabajo se llevará a cabo utilizando técnicas de aprendizaje supervisado.

Otro concepto importante dentro del aprendizaje automático es el aprendizaje profundo (*deep learning*). El aprendizaje profundo es subconjunto dentro del aprendizaje automático que emplea algoritmos basados en redes neuronales. Dentro de este encontramos métodos como las redes neuronales profundas, las redes neuronales recurrentes, las redes neuronales convolucionales y los transformers, entre otros. Estos métodos tienen aplicaciones significativas en una gran variedad de ámbitos, entre los que se encuentra el procesamiento de lenguaje natural, disciplina en la que se enmarca este trabajo. En las siguientes secciones explicaremos con detalle los conceptos de redes neuronales y transformers.

2.2 Redes neuronales

El primer modelo de red neuronal artificial fue creado en 1943 por Warren McCulloch y Walter Pitts, y se conoce como *McCulloch-Pitts neuron* [19]. Este era un modelo simplificado que imitaba el comportamiento de una neurona natural. Esto dio inicio a un proceso de investigación de las redes neuronales desde dos enfoques distintos: uno centrado en los procesos biológicos del cerebro y otro en la aplicación de las redes neuronales para la inteligencia artificial. Las redes neuronales modernas, aunque inspiradas en estas primeras ideas, han evolucionado significativamente y ya no se basan directamente en las inspiraciones biológicas iniciales.

Las redes neuronales modernas están compuestas por pequeñas unidades de cómputo conocidas como neuronas o nodos, conectadas entre si a través de enlaces para permitir la transmisión de señales entre estas. Los nodos están organizados en capas, de manera que un nodo en una capa está conectado a todos los nodos de la capa siguiente. Existen tres tipos de capa: capa de entrada (*input layer*), capas ocultas (*hidden layers*) y capa de salida (*output layer*).

La arquitectura más simple de red neuronal es la de perceptrón. Este tipo de modelo surgió como método para tareas de clasificación binaria, en las que se debe decidir si un determinado *input* pertenece o no a una clase, aunque se amplió para abarcar también tareas de clasificación multiclase. Es un tipo de clasificador lineal, por lo que hace sus predicciones basándose en una función de predicción lineal que combina un conjunto de pesos con el vector de entrada. Este tipo de arquitectura sirve como base para arquitecturas de redes neuronales mucho más complejas.

Tal y como vemos en la Figura 2.1, en la arquitectura de perceptrón simple encontramos solamente una neurona, que toma un vector x como entrada. La neurona calcula la combinación lineal de los elementos del vector $x_1, x_2, ..., x_n$ con los pesos correspondientes $w_1, w_2, ..., w_n$, añadiendo al resultado un valor conocido como umbral o *bias term*:

$$z = b + \sum_{i=1}^{n} w_i x_i \tag{2.1}$$

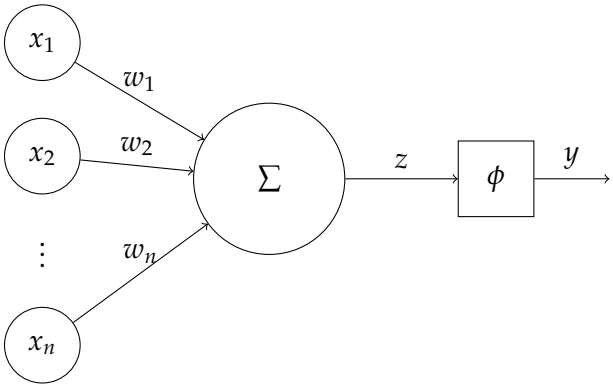

Figura 2.1: Esquema de perceptrón simple

A este resultado se le aplica una función ϕ conocida como función de activación, y finalmente se devuelve un solo valor y como salida:

$$y = \phi(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases} \tag{2.2}$$

El entrenamiento de las redes neuronales consiste en ajustar los distintos pesos de la red de manera que produzca las salidas más acertadas posibles. En el caso de las redes de perceptrón, al contar con solamente una neurona, este proceso resulta relativamente sencillo. El primer paso es inicializar el vector de pesos con valores aleatorios y calcular la salida de cada vector de entrada del conjunto de entrenamiento. A continuación, se comprueba si la predicción ha sido correcta. Si no lo ha sido, el vector de pesos se modifica utilizando la siguiente fórmula:

$$w = w - \lambda(\hat{y} - y)x \tag{2.3}$$

donde λ es la tasa de aprendizaje, $\hat{y}$ es el *output* predicho por el modelo y y es la clasificación real. Este proceso se puede realizar muestra a muestra (*online*) o para un subconjunto de muestras de entrenamiento (*batch*).

Así, estos pasos se repiten durante un número determinado de iteraciones o hasta que el modelo converge.

Este modelo de perceptrón tiene una limitación principal: el modelo solo converge si las dos clases en las cuales debe clasificar las muestras son linealmente separables. En caso de que no lo sean, los pesos oscilarán indefinidamente, hasta que el número máximo de iteraciones se alcance. Para solventar esta limitación existen modelos de redes neuronales mucho más complejos, con más neuronas y que utilizan funciones de activación no lineales. El modelo más conocido de este tipo es el de perceptrón multicapa (MLP, por su nombre en inglés: *Multi-Layer Perceptron*).

2.2.1. Perceptrón multicapa

El modelo de perceptrón multicapa es una evolución del modelo de perceptrón simple que tiene como objetivo el poder resolver problemas no lineales. La idea principal detrás de este modelo es la combinación de varios perceptrones simples en una única arquitectura más compleja. En esta arquitectura podemos encontrar un número elevado de neuronas, conectadas entre sí y organizadas en capas. Encontramos tres tipos de capas:

una capa de entrada (*input layer*), una o más capas ocultas (*hidden layers*) y una capa de salida (*output layer*) (ver Figura 2.2).

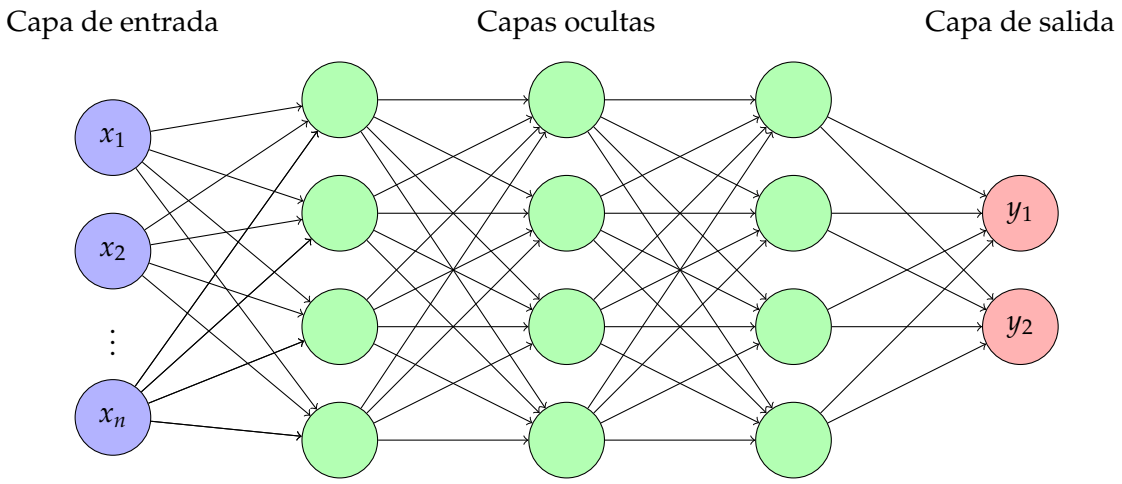

Figura 2.2: Esquema de perceptrón multicapa

Cada una de las neuronas de una capa está conectada a todas las neuronas de la capa siguiente, y cada uno de los enlaces tiene asociado un peso w. De manera similar al modelo de perceptrón simple, los datos se introducen en el modelo en forma de vector a través de la capa de entrada. El ajuste de pesos en el entrenamiento del modelo de perceptrón multicapa se realiza mediante el algoritmo de retropropagación (en inglés, *backpropagation algorithm*) [27]. El algoritmo de retropropagación se puede desglosar en cuatro fases:

1. **Computación hacia adelante (*feed-forward*)**: se realiza una pasada hacia adelante a través de la red, desde la capa de entrada hasta la capa de salida. En esta fase, se calculan las salidas de las neuronas utilizando la Ecuación 2.1 y aplicando sobre este resultado una función de activación no lineal, por ejemplo la función sigmoide, la tangente hiperbólica (*tanh*) o la rectificada lineal (*ReLU*). El uso de este tipo de funciones permite que el modelo pueda aplicarse a problemas no lineales. Al llegar a a la capa de salida, se calcula la predicción final del modelo.

2. **Retropropagación en la capa de salida**: a continuación, se calcula el error en la capa de salida comparando las predicciones del modelo con los valores reales. Posteriormente, se calculan los gradientes del error respecto a los pesos de esta capa.

3. **Retropropagación en las capas ocultas**: utilizando los gradientes obtenidos en la capa de salida, el error se propaga hacia atrás a través de las capas ocultas, ajustando los gradientes del error respecto a los pesos en cada una de estas.

4. **Actualización de los pesos**: finalmente, se actualizan los pesos de la red utilizando los gradientes calculados en las fases anteriores mediante el método de descenso de gradiente u otro algoritmo de optimización similar.

Este proceso se repite de manera iterativa hasta alcanzar el número máximo de iteracoines o hasta que el modelo converge.

2.2.2. Redes neuronales para secuencias de texto

El procesamiento de secuencias de texto utilizando redes neuronales es una técnica fundamental en el campo del procesamiento de lenguaje natural. Dado que las redes

neuronales operan utilizando vectores numéricos, es necesario transformar las secuencias de texto en vectores antes de poder procesarlas. Este proceso se realiza en varias etapas, que se detallan a continuación:

1. **Tokenización y conversión a índices**: el primer paso consiste en convertir la frase en una secuencia de palabras o tokens. Una vez obtenida la lista de tokens, se asigna a cada uno un número que servirá como índice.

2. **Transformación de palabras en vectores**: los índices numéricos se transforman en vectores utilizando *embeddings*. Los *embeddings* son representaciones vectoriales densas que capturan las características semánticas de las palabras.

3. **Creación de la matriz de *embeddings***: una vez que se han obtenido los vectores asociados a cada una de las palabras, se puede construir una matriz de *embeddings* que representa la frase completa. Esta matriz facilita la entrada secuencial de los vectores en la red neuronal.

Cada palabra en la secuencia de texto es procesada por la red neuronal de manera secuencial. En la siguiente sección se presentan las redes neuronales recurrentes (RNNs), un tipo de red neuronal que es ampliamente utlizado para el procesamiento de secuencias de texto.

Redes neuronales recurrentes (RNN)

Las redes neuronales recurrentes (en inglés, *recurrent neural networks* (RNN)) son un tipo de red neuronal que mapean desde un conjunto de entrada a otro de salida de manera que la predicción y_t depende no solo de la de entrada x_t sino también del estado oculto del sistema, h_t [23]. El estado oculto h_t es una representación interna de la red en el momento de tiempo t que captura información relevante que ha sido procesada anteriormente. Se actualiza con el tiempo a medida que se procesan los datos. Este tipo de modelos se pueden utilizar para la generación, clasificación y traducción de texto.

El proceso que siguen este tipo de redes es el siguiente. En primer lugar, se inicializa la red con un estado oculto h_0, que puede ser un vector de ceros o una inicialización aprendida. Para cada elemento en la secuencia de entrada, la red actualiza su estado oculto y produce una salida. Así, en el momento de tiempo t, la entrada x_t y el estado oculto previo h_{t-1} se combinan para generar el nuevo estado oculto h_t. Este se calcula utilizando la fórmula:

$$h_t = f(W_h h_{t-1} + W_x x_t + b_h) \tag{2.4}$$

donde W_h y W_x son las matrices de pesos asociadas al estado oculto y a las entradas respectivamente, f es una función no lineal y b_h es un vector de sesgos. La salida y_t se obtiene a partir del estado oculto h_t:

$$y_t = g(W_y h_t + b_y) \tag{2.5}$$

donde W_y es una matriz de pesos, g es la función de activación y b_y es un vector de sesgos [34].

El proceso se repite para cada elemento de la secuencia de entrada, propagando así la información relevante anterior a través de los estados ocultos.

Uno de los principales problemas de las RNN es la dificultad para recordar información a largo plazo, pues se ha comprobado que, si el modelo es entrenado con secuencias de entrada largas, la importancia de las primeras palabras que son procesadas tiende a ir perdiendose a medida que se procesa el resto de la secuencia. Esto limita la capacidad de este tipo de redes para capturar dependencias a largo plazo en secuencias largas.

2.2.3. Arquitectura *encoder-decoder*

Los modelos *encoder-decoder*, también conocidos como modelos *seq2seq* (*sequence-to-sequence*) son modelos capaces de generar secuencias de salida contextualmente apropiadas y de longitud arbitraria a partir de una secuencia de entrada. Estos modelos pueden generar secuencias de salida de longitud variable a partir de secuencias de entrada también de longitud variable, es decir, secuencias con longitudes distintas en la entrada y la salida. Son modelos ampliamente utilizados en tareas como el resumen de textos, la respuesta a preguntas, el diálogo y la traducción automática [17].

Este tipo de modelos están formados por dos componentes principales: el *encoder* (en español, codificador) y el *decoder* (decodificador). El *encoder* es la primera parte del modelo. Se encarga de mapear las secuencias de entrada a una representación intermedia contextualizada h en forma de vector. Este vector h captura la información relevante de cada elemento de la secuencia de entrada y sus contextos. A continuación, el *decoder* toma el vector de contexto h y, a partir de este, genera la secuencia de salida. Además, el *decoder* puede tener en cuenta también los estados del *encoder* gracias al mecanismo de atención, que se explicará en detalle en la sección 2.2.4 [31].

Esta arquitectura se puede implementar utilizando diversos tipos de redes neuronales, incluyendo los transformers, como veremos en la sección 2.3, o las redes neuronales recurrentes.

2.2.4. Atención

Tal y como se ha explicado en las secciones anteriores, en las redes neuronales clásicas, el cálculo de cada capa se realiza mediante una combinación lineal de los vectores de entrada y los correspondientes pesos, seguida de la aplicación de una función de activación. Esta operación se representa matemáticamente como $Z = \phi(XW)$, donde X es el vector de entrada, W es el vector de pesos, ϕ es la función de activación y Z son las salidas producidas en las capas [23].

Sin embargo, los modelos de redes neuronales pueden volverse aún más flexibles y potentes si permitimos que los pesos dependan de los *inputs*. Esta interacción multiplicativa, donde los pesos son dinámicos y dependen de las entradas, se conoce como mecanismo de atención. Formalmente, puede expresarse como $Z = \phi(XW(X))$.

De manera más general, el mecanismo de atención puede describirse mediante la fórmula $Z = \phi(VW(Q, K))$. En este contexto:

- Q (*queries*) es un conjunto derivado de X que describe qué es lo que busca cada palabra, es decir, con qué otro tipo de palabras puede estar relacionada.
- K (*keys*) es otro conjunto derivado de X que se usa para describir cuáles son las propiedades de las palabras.
- V (*values*) es un conjunto también derivado de X que describe cómo cada entrada debe ser transmitida hasta la salida.

Los vectores de *queries*, *keys* y valores se obtienen procesando las palabras mediante tres redes neuronales independientes.

Cuando se utiliza la atención para calcular una salida z_i, se utiliza la *query* q_i correspondiente y se compara con cada una de las claves k_j de todas las otras palabras de la secuencia, calculando cuál es su nivel de relación con cada una. Esto se realiza mediante el producto escalar entre el vector *query* y los vectores *key* de las otras palabras de la secuencia, de la siguiente manera:

$$\alpha_{ij} = softmax(q_i \cdot k_j) \tag{2.6}$$

Se utiliza la función *softmax* para normalizar los resultados y poder así crear el correspondiente vector de pesos. Así, el resultado se representa como α_{ij} y debe cumplir las siguientes condiciones:

$$0 \leq \alpha_{ij} \leq 1 \tag{2.7}$$

$$\sum_j \alpha_{ij} = 1 \tag{2.8}$$

El coeficiente α_{ij} determina cuánto peso se debe dar a cada valor v_j en la combinación final, y representa el nivel de importancia que tiene cada palabra de la secuencia sobre la palabra i. La salida z_i se calcula entonces como una suma ponderada de los valores v_j, donde los pesos son los coeficientes calculados:

$$z_i = \sum_j \alpha_{ij} v_j \tag{2.9}$$

Este enfoque permite que los *outputs* del modelo sean una combinación dinámica ponderada de los *inputs*, lo que hace que este tipo de sistemas sean mucho más efectivos para una amplia gama de tareas, como la traducción automática, el resumen de textos o la generación de texto entre otros. [23, 17]

2.3 Transformers

En esta sección presentaremos la arquitectura de transformers. Los modelos basados en transformers emplean una arquitectura *encoder-decoder* que utiliza el mecanismo de atención, descrito en la sección anterior. La arquitectura de los transformers fue introducida en el artículo "*Attention is All You Need*" [38], publicado en 2017. Esta supuso una gran revolución en diversas áreas del aprendizaje automático por su capacidad para manejar de manera efectiva secuencias largas y complejas [10]. Actualmente, hay una gran cantidad de modelos basados en esta arquitectura que se utilizan en el área del procesamiento de lenguaje natural.

2.3.1. Estructura del Transformer

Tal y como se ha mencionado anteriormente, la arquitectura de transformers utiliza el modelo *encoder-decoder* introducido en la sección 2.2.3. En este caso, se utilizan un total de seis bloques de *encoder* y seis de *decoder*, cada uno de los cuales contiene varias capas y mecanismos que facilitan el procesamiento eficiente de las secuencias de datos.

En general, los transformers utilizan tres tipos de capas: capas lineales simples, capas de *multi-head attention* y capas donde encontramos redes neuronales de tipo *feed forward*. En la figura 2.3 encontramos el esquema de la estructura de los *encoders* y *decoders*. Tal y como se puede observar, el *encoder* está formado por una primera capa de *multi-head attention* y, a continuación, una red neuronal de tipo *feedforward* completamente conectada, además de las capas de normalización que encontramos después de estas. Por lo que respecta al *decoder*, lo conforman dos capas de autoatención y una red neuronal, también con capas de normalización después de estas. [17, 5]

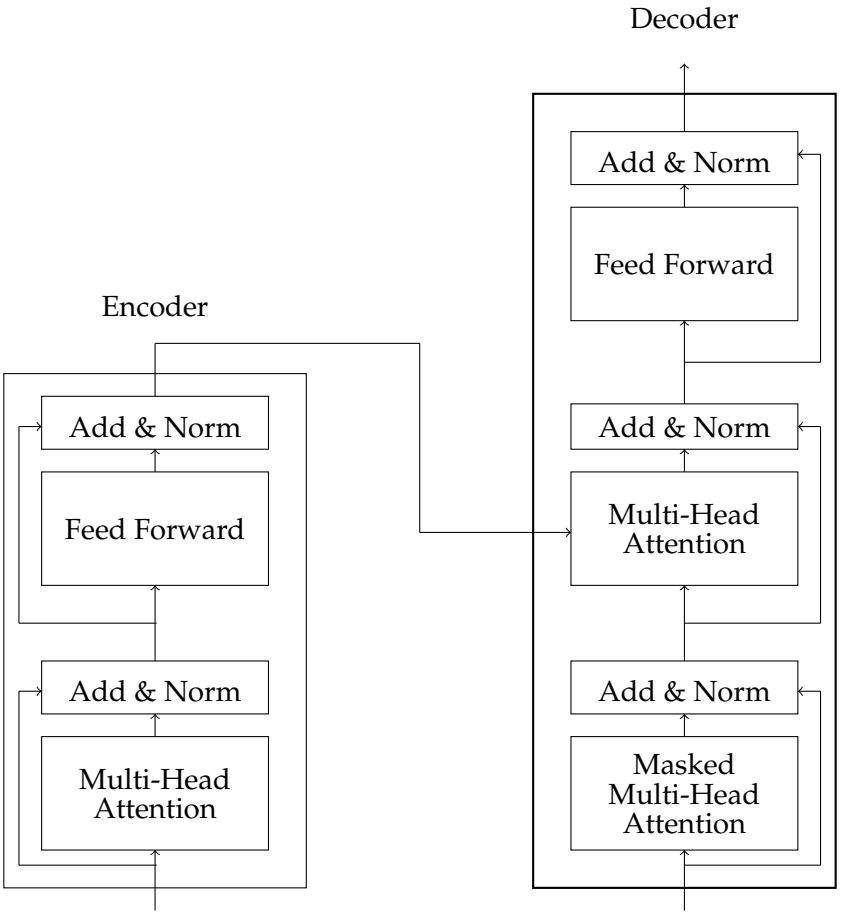

Figura 2.3: Esquema de la arquitectura de los transformers

La idea principal del transformer es, a lo largo de esta serie de capas, poder construir representaciones contextualizadas cada vez más acertadas de los significados de las palabras o *tokens* de las secuencias de entrada. Así, en las distintas capas del transformer, para obtener la representación de una palabra, se combina la información de la representación obtenida en la capa anterior con la información de las representaciones de las palabras vecinas. El objetivo es producir la versión contextualizada de cada palabra, representando lo que significa esta en el contexto particular en que se encuentra. [17]

A continuación se explica con más detalle los procesos que ocurren dentro de cada capa.

2.3.2. Capas de *multi-head attention*

Las capas de *multi-head attention* suponen la verdadera innovación de la arquitectura de transformers [17], pues es en estas donde el modelo aplica el mecanismo de atención. En el caso de los transformers, se utiliza una variación del mecanismo de atención expli-

cado en la sección 2.2.4. Esta variación se conoce como *scaled dot-product attention*. Igual que en el mecanismo general de atención, se utilizan tres matrices: la matriz de *queries* Q, la de claves K (por su nombre en inglés, *keys*) y la de valores V (*values*). En este caso, el valor de atención se calcula utilizando la siguiente fórmula:

$$A(Q, K, V) = softmax(\frac{QK^t}{\sqrt{d_k}}) \tag{2.10}$$

donde d_k es la dimensión del estado oculto de la fuente.

Como vemos, el cambio respecto al cálculo de atención descrito en la sección 2.2.4 reside en que en este caso se añade el factor $\frac{1}{\sqrt{d_k}}$, que se utiliza para escalar el resultado obtenido, y conseguir así que las gradientes sean más estables, ya que si la secuencia de entrada fuera muy larga, la función *softmax* puede devolver gradientes extremadamente pequeños, cosa que podria dificultar al modelo realizar un aprendizaje eficiente. [5]

Aunque esta forma de cálculo de la atención es empleada por los transformers, estos no se quedan solamente en este concepto, sino que van un paso más allá y utilizan un tipo de atención algo más complejo conocido como *multi-head attention*. Este proceso consiste en calcular distintas atenciones sobre las mismas *queries* y pares clave-valor. Se utiliza este tipo de atención ya que las distintas palabras dentro de la secuencia de texto pueden estar relacionadas entre sí de muchas maneras distintas de manera simultánea. Capturar todas estas relaciones entre palabras es muy complicado si se utiliza un solo valor de atención.

Así, en las capas de *multi-head attention* encontramos distintas capas de atención conocidas como *heads* (cabezas), que calculan valores de atención de manera paralela. Cada una de estas tiene un conjunto distinto de parámetros que se aprende durante el entrenamiento, con los cuales hará el cálculo de atención entre las palabras de entrada. Utilizando parámetros distintos en cada cabeza se consigue que cada una se centre en aspectos distintos de las relaciones entre las palabras. El cálculo de atención para una cabeza i se realiza mediante la siguiente fórmula:

$$H_i = A(QW_i^Q, KW_i^K, VW_i^V) \tag{2.11}$$

Donde W_i^Q, W_i^K y W_i^V son las matrices de parámtetros asociadas a la *query*, clave y valor respectivamente de la cabeza i. El valor de *multi-head attention* se obtiene mediante la concatenación de los resultados de las cabezas. Para un total de h cabezas de atención:

$$MultiHead(Q, K, V) = (H_1 \oplus H_2 \oplus ... \oplus H_h)W_O \tag{2.12}$$

La matriz de parámetros W_O proyecta la concatenación de los resultados de las h cabezas de vuelta al subespacio original. [9, 5, 17]

2.3.3. Capas de redes *Feed Forward*

Las capas *feed forward* en la arquitectura de transformers están compuestas por redes neuronales independientes de tipo *feed forward* (en español, redes de propagación hacia adelante). Este tipo de red neuronal es equivalente al modelo de perceptrón multicapa descrito en la Sección 2.2.1.

Así, una red neuronal de tipo *feed forward* es una red multicapa en la que, a diferencia de las redes neuronales recurrentes (RNN) explicadas en la sección 2.2.2, las conexiones entre neuronas no pueden formar ciclos. Esto implica que las señales siempre se propagan

en una única dirección: desde la capa de entrada, a través de las capas ocultas, hacia la capa de salida. En la arquitectura de transformers, estas redes están completamente conectadas, lo que significa que las neuronas de una capa reciben las salidas de todas las neuronas de la capa anterior y envían sus salidas a todas las neuronas de la capa siguiente. Además, típicamente contienen solamente una capa oculta.

En los transformers, estas capas complementan las salidas de la capa de atención. Mientras que la capa de atención procesa cada palabra de la secuencia relacionandola con las demás palabras, las capas *feed forward* procesan cada palabra de manera independiente. Este procesamiento independiente es crucial para mejorar la representación de las características extraídas de la capa de atención.

2.3.4. Capas de normalización

Las capas de normalización se encuentran detrás de tanto las capas de atención como las de redes *forward*. En ellas se aplica el proceso de normalización de capas (en inglés, *layer normalization* o *layer norm* [4]). Este tipo de normalización se utiliza para mejorar el rendimiento del entrenamiento de redes neuronales profundas, manteniendo los valores de la capa oculta dentro de un rango que facilita el entrenamiento basado en gradientes.

El proceso de normalización de capas toma como entrada un vector para cada palabra de la secuencia, y devuelve este mismo vector normalizado. El primer paso en el proceso es calcular la media μ y la desviación típica σ del vector, de la siguiente manera:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} x_i^l \tag{2.13}$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_i^l - \mu^l)^2} \tag{2.14}$$

Donde H es el número de unidades ocultas en la capa. A continuación, los componentes del vector se normalizan restando la media y desviación típica calculadas [17]:

$$\hat{x} = \frac{(x - \mu)}{\sigma} \tag{2.15}$$

Finalmente, se introducen parámetros de *bias* b y *gain* g:

$$z = g\hat{x} + b \tag{2.16}$$

2.4 Modelos de lenguaje grandes

Los modelos de lenguaje grandes (en inglés, *Large Language Models* o LLM) representan una de las innovaciones más significativas en el campo del procesamiento de lenguaje natural. Estos modelos están basados en redes neuronales profundas, y generalmente utilizan la arquitectura de transformers. Estos modelos cuentan con miles de millones de parámetros, y pueden ser entrenados con enormes cantidades de texto, lo que hace que sean capaces de procesar y generar lenguaje humano con una gran fluidez, naturalidad y corrección.

Las tareas sobre las que se aplican son casos de generación condicional, es decir, generación de texto condicionado a un fragmento de entrada, conocido como *prompt*. Los

LLMs son modelos que están diseñados fundamentalmente para predecir la siguiente palabra de una secuencia de palabras [17]. Aunque el ámbito de tareas sobre el que se pueden aplicar los LLMs pueda parecer reducido, la realidad es que la gran mayoría de tareas en el ámbito de NLP pueden ser enfocadas desde un punto de vista de generación condicional.

En la siguiente sección se explica cómo se deben introducir los datos de entrada al modelo mediante *prompting*.

2.4.1. *Prompting*

Prompting se refiere al proceso mediante el cual se introducen una secuencia de entrada o *query* al modelo a partir de la cual debe comenzar la tarea de generación de texto. Existen diversas maneras de crear estas *prompts*. Las más utilizadas son las siguientes:

- ***Zero-shot prompting***:
- ***In-context learning* (ICL)**:
- ***Chain-of-thought* (CoT)**:
- ***Self consistency***:
- ***Tree of thought* (ToT)**:

Por ejemplo, para la tarea de responder preguntas, podriamos plantearlo de la siguiente manera:

En primer lugar, se proporciona al modelo una *prompt* con la pregunta a contestar, con un formato que sugiera que después de la pregunta se espera una respuesta, por ejemplo:

"Pregunta: ¿Cuál es la capital de Francia? Respuesta: "

A continuación, el modelo calcula la probabilidad para cada palabra w dentro de su vocabulario de que esta sea la palabra que continua la *prompt*:

$$P(w|\text{Pregunta: ¿Cuál es la capital de Francia? Respuesta: })$$

Los vocabularios en los LLMs se refieren al conjunto de palabras sobre las que trabajan. Así, habiendo calculado las probabilidades para todas las posibles palabras, encontraremos algunas con probabilidades más altas, entre las cuales debería encontrarse la palabra *París*, que es la respuesta a la pregunta. El modelo deberia seleccionar esta palabra y, si se quiere generar una respuesta más larga, se calcula la siguiente probabilidad:

$$P(w|\text{Pregunta: ¿Cuál es la capital de Francia? Respuesta: París})$$

El proceso puede seguir hasta que lleguemos al número de palabras deseadas.

Inicialmente, este cálculo de probabilidades se realizaba de manera secuencial, basando las predicciones en las distribuciones de probabilidad de las palabras dentro de un texto. Actualmente se utiliza la arquitectura de transformers, que permite procesar

grandes cantidades de texto de manera eficiente y proporciona predicciones mucho más acertadas, pues tiene en cuenta el contexto en el que se encuentran las palabras y las relaciones entre estas [6].

Existen distintos enfoques para la selección de cuál es la palabra que sigue en la secuencia. Uno de los métodos más simples es utilizar el algoritmo de *greedy decoding* [11]. El algoritmo *greedy decoding* consiste en elegir la palabra con la probabilidad condicional más alta hasta el momento en cada instante.

$$\hat{w}_i = argmax_{w \in V} P(w | \{w_1, w_2, ..., w_{i-1}\}) \tag{2.17}$$

Con el tiempo se ha descubierto que este método llega a resultados subóptimos, pues al elegir siempre la palabra que es más probable que ocurra, tiende a generar textos excesivamente genéricos y repetitivos.

Existe una extension de este algoritmo conocida como *beam search*, la cual ha demostrado ser más efectiva, sobretodo en tareas de traducción automática. La diferencia entre el algoritmo *greedy decoding* y este, es que el *beam search*, en lugar de quedarse solamente con la palabra con mayor probabilidad y crear una única respuesta, el algoritmo guarda $K > 1$ posibles respuestas (hipótesis) durante la decodificación. Así, en cada iteración, el algoritmo elige las K hipótesis con mayor *score*:

$$p(w_1, w_2, ..., w_N) = \prod_{i=1}^{N} p(w_i \mid w_1, w_2, ..., w_{i-1}) \tag{2.18}$$

Cuando estas se terminan de generar, selecciona la hipótesis con mayor probabilidad.

Aunque el rendimiento de *beam search* es superior al *greedy decoding*, en los LLM se suelen utilizar métodos más complejos, que proporcionan resultados más sofisticados. Estos métodos se conocen como métodos de generación por *sampling*. Se explicarán con detalle en la siguiente sección.

2.4.2. Métodos de generación por *sampling*

En términos generales, el proceso de *sampling* consiste en seleccionar de manera aleatoria una serie de individuos de una muestra, teniendo (o no) en cuenta sus distribuciones de probabilidad. En el contexto de los LLM, esto se traduce en elegir las palabras que se generan en la secuencia de acuerdo con su probabilidad dentro del contexto definido por el modelo. De esta manera, es más probable seleccionar palabras con probabilidades altas y menos probable (aunque no imposible) elegir aquellas con probabilidades bajas.

El algoritmo de generación por *sampling* más sencillo se conoce como *random sampling*, el cual consiste en seleccionar la siguiente palabra de la secuencia de manera aleatoria según la distribución de probabilidades proporcionada por el transformer. Sin embargo, este enfoque es demasiado simplista para la generación de texto de calidad, ya que al seleccionar las palabras de manera aleatoria, incluso las palabras poco comunes, que tendrán probabilidades bajas, pueden ser elegidas, lo que podría resultar en la generación de frases incorrectas o extrañas [17].

Debido a estas limitaciones, en los LLM se suelen emplear algoritmos más sofisticados, los cuales analizaremos en las siguientes secciones.

Top-k sampling

El algoritmo de *top-k sampling* es una generalización del *greedy decoding* que consiste en seleccionar las K palabras con mayor probabilidad de la distribución y aplicar *random sampling* únicamente sobre este conjunto. Una vez se seleccionan las K palabras del conjunto, sus probabilidades deben ser normalizadas según el nuevo conjunto. Las probabilidades normalizadas se calculan de la siguiente manera:

$$\hat{p}_i = \frac{p_i * \mathbb{1}\{i \leq K\}}{\sum_{j=1}^{K} p_j} \tag{2.19}$$

Donde $\mathbb{1}$ es la función característica, que indica si una palabra está dentro o no del conjunto de las K más probables [24].

Nucleus sampling

El algoritmo de *nucleus sampling* [13], también conocido como *top-p sampling* también sigue la idea de eliminar de la distribución las palabras menos probables. Sin embargo, en lugar de seleccionar un número fijo de palabras, lo que hace es mantener el porcentaje p más importante de la muestra. Así, dada una distribución $P(x|x_{1:i-1})$, las *top-p* palabras del vocabulario $V^{(p)} \subset V$ se definen como el conjunto más pequeño que cumpla:

$$\sum_{x \in V^{(p)}} P(x|x_{1:i-1}) \geq p \tag{2.20}$$

Al medir la probabilidad acumulada en lugar de un número fijo de palabras, se espera que esta medida sea más robusta incluso en contextos diversos, que podrían tener resultados muy distintos si se usara un número fijo de palabras. De nuevo, tras seleccionar el subconjunto de palabras del vocabulario se deben normalizar sus probabilidades.

Las probabilidades normalizadas de cada palabra en el vocabulario se pueden calcular como:

$$\hat{p}_i = \frac{p'_i}{\sum_{j=1}^{|V|} p'_j} \tag{2.21}$$

Donde $p'_i = p_i * \mathbb{1}\{\sum_{j=1}^{i-1} p_j < P\}$ [24].

Tempered sampling

En el algoritmo de *tempered sampling* [24] no crea un subconjunto de palabras del vocabulario, sino que ajusta la distribución de probabilidad de las palabras mediante una transformación. Para ello, se utiliza un parámetro τ, que cumple $0 < \tau < 1$, según la siguiente fórmula:

$$\hat{p}_i = \frac{exp(log(p_i)/\tau)}{\sum_{j=1}^{|V|} exp(log(p_j)/\tau)} \tag{2.22}$$

Además, existe una variación de este algoritmo que se conoce como *tempered top-k sampling*, el cual combina las transformaciones definidas por el algoritmo *tempered sampling* y el *top-k*. Así, la probabilidad normalizada se calcula:

$$\hat{p}_i = \frac{p'_i}{\sum_{j=1} |V| p'_j} \tag{2.23}$$

donde $p'_i = exp(log(p_i)/\tau * \mathbb{1}\{i \leq K\}$.

2.4.3. Arquitecturas de los LLMs

Existen cuatro tipos principales de arquitecturas de los LLMs:

- **Arquitectura *encoder-decoder***: esta es la arquitectura explicada en la Sección 2.2.3, sobre la cual se basa también la arquitectura de transformers original, presentada en la Sección 2.3. Tal y como se explicó, consta de un codificador que se encarga de procesar la entrada de texto, transformándola a una representación interna que captura el significado y contexto del texto original. A continuación, el decodificador utiliza esta representación para generar el texto de salida.
- **Arquitectura *encoder-only***: en esta arquitectura se utiliza únicamente el componente de codificador del modelo *encoder-decoder*. Así, en este enfoque, el codificador crea una representación detallada de la entrada de texto, que puede ser utilizada posteriormente para tareas de comprensión del lenguaje, como la clasificación de texto o la extracción de información.
- **Arquitectura *decoder-only***: esta arquitectura utiliza únicamente el componente del decodificador. De esta manera, el modelo se entrena para generar texto de manera autoregresiva, es decir, prediciendo cada palabra o token en función de los tokens generados anteriormente. Este enfoque es adecuado para tareas que requieren generación continua de texto, como la completación de frases y la generación de contenido [18].
- **Arquitectura *Mixture of Experts* (MoE)**: introduce un enfoque en el que se utilizan varios expertos especializados para mejorar la eficiencia y el rendimiento del modelo. En lugar de activar todos los parámetros del modelo para cada tarea, solo un subconjunto de expertos se activa en función de la entrada específica. Este enfoque permite una mayor eficiencia en el procesamiento y optimiza el uso de los recursos computacionales.

CAPÍTULO 3
Adaptación y evaluación de modelos de lenguaje grandes

En este capítulo se presentan los modelos de lenguaje grandes utilizados en este trabajo. Además, se aborda cómo podemos adaptar estos modelos preentrenados para realizar tareas específicas, explorando las diferentes estrategias y enfoques disponibles. Se hará especial incapié en el método PEFT (*parameter-efficient fine-tuning*), centrándonos concretamente en LoRA (*Low-Rank Adaptation*), que es uno de sus tipos de enfoque.

3.1 Modelos de lenguaje utilizados

Para la realización de este trabajo se utilizarán modelos de lenguaje grandes preentrenados. Los modelos de lenguaje grandes preentrenados son modelos que han sido entrenados con grandes cantidades de datos generales, pero que aún no han sido ajustados para realizar tareas específicas. Este tipo de modelos han supuesto una revolución en el campo del procesamiento de lenguaje natural, ya que pueden ser adaptados para realizar una gran variedad de tareas específicas, incluso si estas difieren de los datos originales de entrenamiento. Utilizar LLMs preentrenados supone un ahorro de tiempo y recursos significativo, evitando la necesidad de entrenar un modelo desde cero con un gran conjunto de datos.

Algunos de los modelos más utilizados actualmente y que se utilizan en este trabajo son el modelo Gemma, desarrollado por Google, Llama-3, desarrollado por Meta, y los modelos GPT, desarrollados por la compañía *OpenAI*,. Estos tres modelos se explicarán en detalle en las siguientes secciones.

Gemma

Gemma [36] es un modelo de código abierto desarrollado por Google, basado en sus modelos privados Gemini [35]. Existen dos versiones de este modelo: uno con 7 billones de parámetros (modelo 7B), para despliegue en GPU y TPU, y otro con 2 billones (modelo 2B), diseñada para su uso en CPU. El modelo 7B está entrenado con 6T *tokens*, mientras que el 2B, con 3T. En ambos casos, los datos son principalmente en inglés y provienen de documentos web, documentos mátemáticos y código. Su entrenamiento se ha realizado mediante *fine-tuning*, utilizando la técnica de *Reinforcement Learning from Human Feedback* (RLHF) [8].

En cuanto a su arquitectura, esta está basada en el modelo de *transformer decoder only*, pero incorpora varias mejoras significativas:

- ***Multi-Query Attention*** [29]: es una variación de *multi-head attention* que permite que varias cabezas de las capas de atención compartan las mismas claves y valores, mejorando así la eficiencia computacional.

- ***Rotary Position Embeddings (RoPE)*** [33]: este tipo de *embeddings* mejora la representación de las posicones relativas en la secuencia, ya que se basan en la distancia relativa entre los *tokens*.

- ***GeGLU activations*** [30]: en lugar de utilizar la función de activación ReLU, se utiliza la función GeGLU, la cual ha demostrado mejorar el rendimiento del modelo en tareas de generación de texto.

- **Pre-normalización con RMSNorm** [41]: se normaliza la entrada a cada una de las capas con RMSNorm para estabilizar el entrenamiento. Este tipo de normalización es más eficiente que la de la arquitectura de transformers original.

LLaMA

LLaMA [37] es una familia de modelos de lenguaje *open-source* desarrollados por Meta. Estos modelos van desde los 7B a los 65B de parámetros. Están entrenados utilizando exclusivamente datos disponibles de manera pública, con un conjunto de datos de entrenamiento formado por fuentes diversas, que abarca una gran cantidad de dominios.

En cuanto a su arquitectura, al igual que Gemma, está basada en el modelo de *transformer decoder only*, pero incorpora varias mejoras. Algunas de estas mejoras las encontramos también en los modelos Gemma, como la pre-normalización con RMSNorm o el uso de *rotary position embeddings*. En lugar de la función de activación ReLU, LLaMA utiliza la función SwiGLU [30], que ha demostrado ser más eficiente en términos de rendimiento del modelo.

LLaMA incorpora también optimizaciones para reducir el tiempo de entrenamiento. Una de estas es la implementación eficiente del *causal multi-head attention*, que consiste en no guardar los valores de los pesos de atención ni calcular los *scores* entre claves y valores que están ocultas en la tarea, reduciendo así el coste computacional. Otra optimización es la reducción de las activaciones recalculadas mediante *checkpointing*, de manera que se guardan las activaciones costosas de calcular implementando manualmente la función hacia atrás para las capas transformadoras.

Concretamente, en este trabajo se utilizará el modelo Llama-3.

Modelos GPT

Los modelos GPT (*Generative Pre-trained Transformer*) han sido desarrollados por la compañía *OpenAI*, y son una de las familias de modelos de lenguaje de código abierto más utilizadas actualmente, especialmente debido a su interfaz de ChatGPT, que ofrece un diseño intuitivo para dar respuesta a preguntas realizadas por los usuarios. En este trabajo se utilizará específicamente el modelo GPT-4.

El modelo GPT-4 [1] es el último de la serie de modelos GPT. La principal innovación de este respecto a sus predecesores es que es un modelo multimodal. Esto significa que, además de aceptar entradas en formato de texto, también es capaz de procesar y analizar imágenes, lo que amplía significativamente su aplicabilidad en una gran variedad de tareas. GPT-4 puede alcanzar resultados a nivel humano en una amplia gama de tareas, tanto en ámbitos profesionales como académicos.

En cuanto a su entrenamiento, este modelo ha sido entrenado con un conjunto de datos mucho más grande que el de sus versiones anteriores. Este conjunto de datos cuenta con más de diez billones de palabras, que es diez veces más que el modelo anterior, GPT-3. Además, los datos utilizados abarcan una amplia variedad de fuentes tanto públicas como proporcionadas por terceros. La técnica utilizada para su entrenamiento fue *fine-tuning* utilizando el método de *Reinforcement Learning from Human Feedback* (RLHF).

Al contrario que los modelos anteriores, este modelo no será reentrenado para su utilización en los experimentos, sino que será utilizado mediante *prompting* desde la interfaz de ChatGPT.

3.2 Ajuste de LLMs

Tal y como se ha mencionado en la sección **??**, los LLMs preentrenados son modelos entrenados con grandes cantidades de datos genéricos que no han sido todavía ajustados para la realización de ninguna tarea específica. Por lo tanto, para aplicarlos a tareas concretas, es necesario ajustar estos modelos para adaptarlos a los requisitos específicos de cada tarea. Generalmente, para adaptar los modelos se utilizan técnicas de *transfer learning*.

3.2.1. Aprendizaje por transferencia

El concepto de aprendizaje por transferencia (en inglés, *transfer learning*) se refiere a la transferencia de conocimiento desde un dominio fuente, donde se dispone de una mayor cantidad de datos, hacia dominios más específicos, que cuentan con menos datos disponibles. Una definición formal dada en [39] es la siguiente:

> *"Dado un dominio fuente D_s con una tarea fuente correspondiente T_s y un dominio objetivo D_t con una tarea correspondiente T_t, el aprendizaje por transferencia es el proceso de mejora de la función predictiva objetivo $f_t(\cdot)$ mediante el uso de la información relacionada del dominio D_s y la tarea T_s, donde $D_s \neq D_t$ o $T_s \neq T_t$"*

Existen cuatro enfoques principales dentro del aprendizaje por transferencia: aprendizaje basado en instancias (*instance-based*), basado en características (*feature-based*), basado en parámetros (*parameter-based*) y basado en relaciones (*relational-based*) [39].

- **Aprendizaje basado en instancias**: la idea principal es utilizar una combinación de modelos preentrenados en el dominio fuente para etiquetar los datos no etiquetados del dominio objetivo. Para ello, se asignan pesos a los modelos del dominio fuente en función de su similitud con el dominio objetivo. Los resultados de los modelos se ponderan según estos pesos para estimar las etiquetas de los datos no etiquetados del dominio objetivo. Finalmente, se construye el modelo del dominio objetivo a partir de los datos etiquetados de este y las etiquetas estimadas.
- **Aprendizaje basado en características**: en este enfoque, se crean nuevos modelos específicos para las tareas, y se incorpora en estos las representaciones obtenidas con el modelo preentrenado, añadiéndolas como características adicionales. De esta manera, el modelo preentrenado sirve como punto de partida para el modelo específico. Utilizando las características extraídas del modelo preentrenado, el modelo específico puede transferir el conocimiento general obtenido por este al contexto del dominio específico de la tarea. Dentro de este enfoque, existen dos variantes:

asimétrico y simétrico [42]. En el enfoque asimétrico, se transforma las características extraídas por el modelo preentrenado para que se ajusten a las del dominio objetivo. Por el contrario, en el simétrico lo que se hace es buscar un espacio común de características latentes y transformar tanto las características del dominio fuente como del objetivo en una nueva representación de características.

- **Aprendizaje basado en parámetros**: consiste en modificar y reentrenar el modelo preentrenado para la tarea en particular, ajustando directamente sus parámetros. De esta manera, el modelo conserva el conocimiento general obtenido en el preentrenamiento y, tras la modificación y reentrenamiento, aprende nuevo conocimiento específico de la tarea.

- **Aprendizaje basado en relaciones**: este enfoque se centra principalmente en los problemas de dominios relacionales, transfiriendo la lógica de relaciones o reglas aprendidas en el dominio fuente al dominio objetivo. Estos métodos buscan preservar y adaptar las relaciones entre entidades para resolver tareas en el nuevo dominio.

Para los experimentos de este trabajo, se realizará la adaptación de los modelos de lenguaje preentrenados mediante aprendizaje por tranferencia basado en parámetros. Específicamente, se utilizará la técnica de *fine-tuning*.

3.3 *Fine-tuning*

El *fine-tuning* es una técnica detro del aprendizaje por transferencia basado en parámetros que consiste en ajustar los parámetros del modelo preentrenado utilizando para su entrenamiento un conjunto de datos adaptado a la tarea sobre la que se va a utilizar el modelo.

El principal inconveniente del método de *fine-tuning* es que puede ser un proceso muy lento, ya que es necesario ajustar todos los parámetros del modelo preentrenado, los cuales suelen ser numerosos. Existe un enfoque alternativo en el que se mantiene el modelo preentrenado intacto, y se añaden unos pocos parámetros adicionales que modifican su comportamiento interno para adaptarlo a la tarea específica. Durante el entrenamiento, solamente es necesario ajustar estos nuevos parámetros. Esta idea se conoce como adaptadores.

3.3.1. Adaptadores

El método de ajuste basado en adaptadores funciona añadiendo módulos específicos a un modelo preentrenado, de manera que durante el entrenamiento de este para alguna tarea específica se actualizan únicamente los parámetros de estos módulos, y no los del modelo completo. De esta manera, se incorporan solo unos pocos parámetros entrenables para cada nueva tarea, lo que permite un alto grado de compartición de parámetros [12]. Este enfoque puede ser utilizado como alternativa al *fine-tuning* completo del modelo, ya que, se ha demostrado que ambos métodos ofrecen un rendmiento comparable, siendo el ajuste con adaptadores mucho más eficientes en términos del número parámetros [14, 32].

Los módulos de los adaptadores deben cumplir con dos propiedades esenciales: contener un número reducido de parámetros y tener una incialización cercana a la identidad. los adaptadores deben ser pequeños en comparación con las capas del modelo original para asegurar que el modelo crezca de manera controlada al añadir nuevas tareas. La

inicialización cercana a la unidad es necesaria para mantener la estabilidad del entrenamiento. De esta manera, el modelo original no se verá afectado al inicio del entrenamiento. Durante este, los adaptadores pueden activarse para cambiar la distribución de las activaciones en la red. Además, los módulos pueden ignorarse si no son necesarios. Si la incialización se desviara demasiado de la identidad, el modelo podria fallar durante el entrenamiento [14].

En el ámbito del procesamiento de lenguaje natural, los adaptadores generalmente se insertan como módulos entre las capas del modelo. En el caso de la arquitectura de transformers, la idea es insertar dos redes neuronales MLP poco profundas como cuello de botella dentro de cada uno de los bloques del transformer: una después de la capa de *multi-head attention* y otra, después de la capa de red *feed forward*. Estas redes MLP deben tener lo que se conoce como *skip connections*, es decir, conexiones que permiten saltarse capas inermedias, para permitir la incialización de las redes como identidad [23].

3.4 *Parameter-efficient fine-tuning* (PEFT)

El *Parameter- Efficient Fine-Tuning* (PEFT) es uno de los métodos de ajuste basados en adaptadores que pueden aplicarse a modelos de lenguaje grandes. Este método incluye cuatro enfoques: basado en *promtps*, basado en reparametrización, adaptadores en serie y adaptadores en paralelo.

En este trabajo, se empleará el enfoque basado en reparametrización. Este se centra en la transformación de los pesos de la red mediante técnicas de bajo rango, de manera que se consigue reducir de manera eficaz el número de parámetros que necesitan ser entrenados, mientras se conserva la capacidad de manejar matrices de alta dimensión [16]. Dentro de este marco, existen varios métodos, entre los cuales se destaca LoRA (*Low-Rank Adaptation*), que es el enfoque que se utilizará en este proyecto.

3.4.1. *Low-Rank Adaptation* (LoRA)

El método de *Low-Rank Adaptation* (LoRA) fue presentado en el artículo *LoRA: Low-Rank adaptation of Large Language models* [15], desarrollado por un equipo de Microsoft. Como se mencionó en la sección anterior, este forma parte del enfoque PEFT basado en reparametrización de los modelos. LoRA se basa en congelar los pesos del modelo preentrenado durante el entrenamiento para luego ajustarlos mediante una matriz de parámetros adicionales que representan el cambio necesario en los pesos originales del modelo. La matriz de cambios es la que se ajusta en el entrenamiento. Generalmente, la matriz de pesos originales se denota como W y la matriz de cambios, $\triangle W$.

LoRA se basa en la idea de la descomposición en rangos de las matrices. El rango de una matriz hace referencia al número de columnas linealmente independientes de esta. Una columna es linealmente dependiente si puede ser obtenida como combinación lineal de otras columnas de la matriz. Esto implica que, si existen columnas linealmente dependientes en la matriz, estas se pueden eliminar sin necesidad de perder información, permitiendo así reducir el tamaño de esta.

Así, en lugar de optimizar los pesos de la matriz completa, LoRA utiliza lo que se conoce como *low rank decomposition* (en español, descomposición en rango bajo), de manera que la matriz $\triangle W$ se representa como el producto de dos matrices:

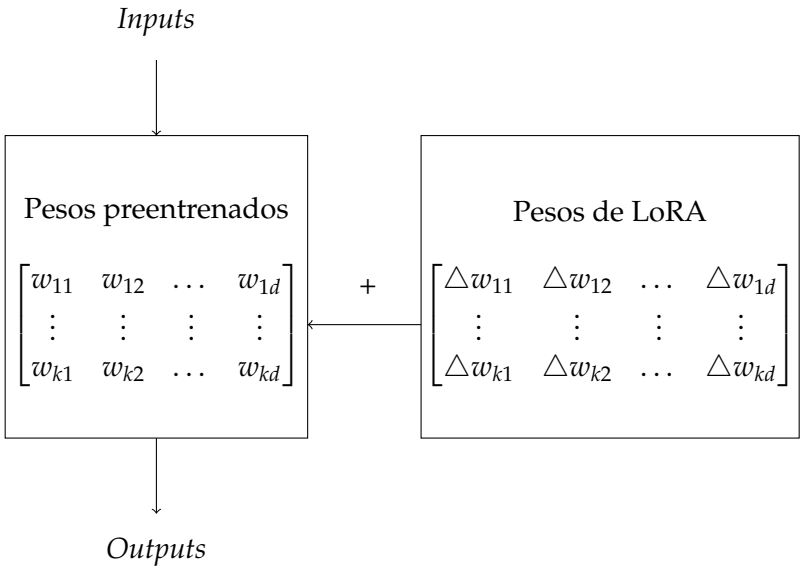

Figura 3.1: Esquema de las matrices de parámetros de LoRA

$$\triangle W = B \cdot A = \begin{bmatrix} b_{11} & \dots & b_{1r} \\ \vdots & \vdots & \vdots \\ b_{d1} & \dots & b_{dr} \end{bmatrix} \cdot \begin{bmatrix} a_{11} & \dots & a_{1k} \\ \vdots & \vdots & \vdots \\ a_{r1} & \dots & a_{rk} \end{bmatrix} \tag{3.1}$$

Aquí, la matriz B tiene dimensiones dxr, y la matriz A, rxk, donde d es el número de filas de la matriz de pesos original, k es el número de columnas y r, es un hiperparámetro (denominado *rank* o rango) que se debe determinar.

Este enfoque proporciona una mayor eficiencia computacional, ya que se está reduciendo el número de elementos en las matrices A y B con respecto a la matriz original. En el entrenamiento, la matriz A se inicializa a partir de una distribución gaussiana, mientras que la matriz B es inicializada a 0. A medida que trascurre el entrenamiento, los pesos se ajustan para adaptar el modelo a la tarea.

En las siguientes secciones presentaremos cuáles son los hiperparámetros que se deben ajustar en un entrenamiento con LoRA.

Rango

El primer hiperparámetro que se debe determinar en un entrenamiento con LoRA es el rango (*rank*, r). Este representa el rango de la matriz de parámetros original. Su valor exacto es desconocido, por lo que se deberá seleccionar al inicio del entrenamientol. Idealmente, r debe ser lo más pequeño posible, pero aún debe ser suficiente para capturar la información esencial. Se debe tener en cuenta que:

- Si r es demasiado pequeño, la matriz se reduce excesivamente, eliminando columnas que no son linealmente dependientes y, por tanto, perdiendo información necesaria.

- Si r es demasiado grande, se incluirán parámetros linealmente dependientes, cosa que puede reducir la efectividad computacional.

Alpha

Alpha es un parámetro que se utiliza para escalar los pesos entrenados, determinando la importancia que se les debe dar a los pesos aprendidos durante el entrenamiento en el modelo. Se debe tener en cuenta lo siguiente:

- Si $\alpha = r$, los nuevos pesos serán integrados con los del modelo original en una proporción de 1:1.
- Si $\alpha > r$, se está dando mayor peso a los nuevos parámetros aprendidos. Esto es recomendable únicamente cuando se cuenta con un conjunto de datos de entrenamiento grande y de alta calidad. De lo contrario, al darle más importancia al conocieminto aprendido del nuevo conjunto de datos en lugar de al conociemiento preexistente del modelo, podría resultar en incoherencias.
- Si $\alpha < r$, se prioriza el conocimiento preexistente del modelo sobre el nuevo. Esta configuración es aconsejable cuando el conjunto de datos de entrenamiento es pequeño, pues así se garantiza que el modelo conserve más el conociemiento original.

Dropout

El parámetro de *dropout* determina cuál la probabilidad de que uno de los parámetros ajustados en el entrenamiento sea configurado artificialmente a 0 para un lote de entrenamiento específico. Este procedimiento se realiza para evitar problemas de *overfitting*, donde el modelo se ajusta demasiado a los datos de entrenamiento y pierde capacidad de generalización.

3.5 Cuantización

La creciente complejidad de la arquitectura de los modelos de lenguaje grandes hace que se requiera una gran cantidad de recursos computacionales y de memoria para su implementación y utilización. Este requerimiento intensivo de recursos presenta un desafío significativo, especialmente en entornos con limitaciones en el hardware. Por esta razón, en estos contextos es esencial hacer uso de técnicas que permitan la ejecución eficiente de los modelos, sin comprometer sustancialmente su rendimiento. Una de las técnicas más efectivas en este contexto es la cuantización.

La cuantización es un proceso que consiste en convertir los valores de alta precisión de un modelo, representados normalmente en punto flotante de 32 o 64 bits, a representaciones de menor precisión, generalmente enteros de 8 o 16 bits [7]. Este mapeo de valores continuos de alta precisión a niveles discretos proporciona una gran mejora en términos de computación y memoria, ya que los requisitos de almacenamiento de un modelo de baja precisión son sustancialmente menores que los de uno de alta precisión. Además, también se pueden observar mejoras en cuanto al consumo de energía, los requisitos de ancho de banda de la memoria y la complejidad computacional.

De todas maneras, es importante tener en cuenta que la inferencia con baja precisión de bits puede conllevar a una pérdida de precisión en la tarea. Esta pérdida de precisión puede compensarse mediante varias técnicas.

Una de las técnicas de cuantización más utilizadas es la de precisión mixta [20]. Esta técnica consiste en utilizar una combinación de representaciones de alta y baja precisión dentro del modelo. Por ejemplo, los pesos del modelo pueden estar almacenados en alta

precisión, para asegurar la exactitud, mientras que los cálculos durante el entrenamiento se realizan con menor precisión para aumentar la velocidad y reducir el uso de la memoria. Posteriormente, los gradientes calculados en baja precisión se utilizan para actualizar los pesos originales.

Otra técnia ampliamente utilizada es la de cuantización post entrenamiento [40]. Esta técnica se aplica después de que los modelos hayan sido completament entrenados utilizando representaciones de alta precisión (en el caso de los modelos de lenguaje grandes preentrenados, después del preentrenamiento). El bojetivo principal es reducir el tamaño del modelo y los requisitos de cálculo para permitir una implementación más eficiente en cuanto a memoria y computación, sin tener que reentrenar el modelo desde cero. De esta manera, los pesos y, a veces, las activaciones del modelo son convertidos a representaciones de menor precisión, generalmente de 8 bits.

3.6 Medidas de evaluación

La correcta evaluación de los modelos de lenguaje grandes es crucial para poder medir su desempeño en la realización de tareas. A medida que los modelos se vuelven complejos y capaces de generar texto con mayor coherencia y fluidez, se vuelve indispensable contar con métricas de evaluación que permitan cuantificar su desempeño de manera precisa. Estas métricas no solo pueden ayudarnos a comparar modelos entre sí, sino que también proporcionan información sobre las áreas en las que un modelo puede necesitar mejoras.

En este trabajo, utilizaremos las métricas de BLEU y COMET para la evaluación de los modelos. Ambas se explicarán en detalle a continuación.

3.6.1. BLEU

BLEU (*Bilingual Evaluation Understudy*) [25] es una de las métricas más utilizadas actualmente para la evaluación de modelos en el campo del procesamiento de lenguaje natural. Esta métrica se basa en la premisa de que una frase generada por un modelo es mejor cuanto más se parezca a la frase que habría sido generada por una persona. Para poder calcularla, se necesita el conjunto de frases generadas por el modelo y un conjunto de frases de referencia reales generadas por personas.

El enfoque de BLEU consiste en utilizar una media ponderada de las coincidencias entre la frase generada por el modelo y las frases de referencia reales. Para ello, BLEU calcula la precisión modificada de n-gramas (p_n) entre las frase de referencia y la frase generada por el modelo. Para calcular esta precisión modificada, primero se cuentan cuántos n-gramas de la frase generada por el modelo coinciden con los n-gramas de las frases de referencia. Este conteo se realiza utilizando los *clipped n-gram counts*, que limitan la cantidad de veces que un n-grama en particular puede ser contado, basado en su frecuencia en las frases de referencia. A continuación, se suma el total de estos n-gramas limitados para todas las frases candidatas generadas por el modelo.

El cálculo de p_n se formaliza de la siguiente manera:

$$p_n = \frac{\sum_{C \in \{\text{Candidatos}\}} \sum_{\text{n-grama} \in C} Count_{clip}(\text{n-grama})}{\sum_{C \in \{\text{Candidatos}\}} \sum_{\text{n-grama} \in C} Count(\text{n-grama})} \tag{3.2}$$

donde el numerador representa la suma total de los *clipped n-gram counts* para todos los candidatos y el denominador es el conteo total de n-gramas en todas las frases candidatas.

BLEU incorpora también un factor de penalización por brevedad (*brevity penalty*), para evitar premiar las frases generadas que sean mucho más cortas que la frase de referencia. Este factor de penalización por brevedad se define como:

$$BP = \begin{cases} 1 & \text{si } c > r \\ e^{(1-r/c)} & \text{si } c \leq r \end{cases} \tag{3.3}$$

donde c es la longitud de la frase candidatas generadas por el m odelo y r es la longitud de la frase de referencia.

Finalmente, el valor de BLEU se obtiene combinando estas precisiones de n-gramas modificadas a través de diferentes valores de n:

$$BLEU = BP + exp(\sum_{n=1}^{N} w_n log p_n) \tag{3.4}$$

donde generalmente se utilizan los valores de $N = 4$ y $w_n = \frac{1}{N}$.

AÑADIR LIMITACIONES

3.6.2. COMET

COMET (*Crosslingual Optimized Metric for Evaluation of Translation*) [26] es una métrica avanzada utilizada para evaluar la calidad de las traducciones generadas por modelos de lenguaje en el campo de la traducción automática. A diferencia de BLEU, que se basa en coincidencias léxicas de n-gramas, COMET utiliza modelos de aprendizaje profundo para capturar de manera más efectiva la calidad de las traducciones.

COMET se presenta en dos tipos de arquitecturas: el *estimator model* y *translation ranking model*. La diferencia entre ambas es el objetivo de entrenamiento, ya que el *estimator model* está diseñado para predecir directamente un *score* de calidad para una traducción dada, mientras que el *translation ranking model* se entrena para minimizar la distancia entre una traducción generada por el modelo y su correspondiente frase de referencia y frase original en el idioma fuente.

Ambos modelos están formados por dos componentes principales:

- ***Cross-lingual encoder***: es un modelo pre-entrenado multilingüe, por ejemplo BERT, XLM o XLM-RoBERTa. Estos modelos contienen varias capas de codificación de *transformer* que generan representaciones (*embeddings*) de las secuencias de entrada. En el caso de COMET, estas secuencias serán las frases originales, las frases generadas por el modelo y las frases de referencia. Así, el codificador mapea todas estas a un mismo espacio de características, de manera que es posible hacer comparaciones más precisas.

- ***Pooling layer***: es una capa de agrupamiento que toma los *embeddings* más importantes generados por las capas del codificador y los combina en una única representación fija e_{x_j}. Esta representación se calcula como:

$$e_{x_j} = \mu E_{x_j}^T \alpha \tag{3.5}$$

donde μ es un parámetro entrenable, E_{x_j} es el vector de *embeddings* para el token x_j y α es un vector que corresponde a los pesos entrenables por capas.

Finalmente, estos *embeddings* son agrupados mediante *average pooling* para obtener una la representación final de cada frase.

Las representaciones obtenidas son las que se utilizan posteriormente para calcular el *score* (en el caso del *estimator model*) o el ranking (en el caso del *translation ranking model*) de las frases.

CAPÍTULO 4
Resultados experimentales

REESCRIBIR: En este capítulo se presenta el experimento principal de este trabajo, que consiste en el entrenamiento de dos modelos de lenguaje grandes preentrenados: Gemma y Llama-3, enfocados en la tarea específica de generar frases de lenguaje natural a partir de palabras clave en el contexto de los Sistemas de Comunicación Aumentativa y Alternativa. A continuación, se detallan los experimentos realizados, las medidas de evaluación empleadas y los resultados obtenidos.

4.1 Conjunto de datos

Para la realización de los experimentos, se han utilizado dos conjuntos de datos distintos, ambos extraídos del portal de la Universidad de Vigo. Ambos conjuntos contienen frases extraídas de los recursos del Portal Aragonés de la Comunicación Aumentativa y Alternativa, acompañada de la lista de lemas o palabras clave asociadas a cada frase. Estas palabras clave han sido seleccionadas por un grupo de anotadores expertos lingüistas.

El primer conjunto de datos incluye un total de 212 frases, todas ellas en español. El segundo conjunto consta de 260 frases, disponibles tanto en español como en inglés.

A continuación, se presentan algunos ejemplos de frases del primer conjunto de datos para ilustrar su formato:

El policía puso una multa. | policía, poner, un, multa

El lobo quiere comerlo. | lobo, querer, comer, lo

La niña escribe en la arena. | niña, escribir, arena

Los niños juegan en la piscina. | niño, jugar, piscina

Los cerditos viven felices. | cerditos, vivir, feliz

En cuanto al segundo conjunto de datos, se incluyen ejemplos bilingües:

Su mamá se enfadó. su, mamá, se, enfadar | His mother got angry. his, mother, get, angry

Pili ha metido un gol. Pili, haber, meter, un, gol | Pili has scored a goal. Pili, score, a, goal

La Luna está en el cielo. Luna, estar, cielo | The moon is in the sky. moon, be, sky

Juan encontró un castillo. Juan, encontrar, castillo | Juan found a castle. Juan, find, a, castle

No lo cambiaré. no, lo, cambiar | I will not change it. change, it, not

En el segundo conjunto de datos, se encontraron algunos valores nulos en las listas de palabras clave para ciertas frases en español. Estas frases fueron anotadas manualmente para completar la información faltante.

A partir de estos dos conjuntos, se han creado un conjunto de datos para el entrenamiento en español y otro, en inglés.

- **Conjunto en español**: se ha creado a partir de la unión de las frases en español de los dos conjuntos mencionados. Contiene un total de 340 frases, ya que entre los conjuntos originales algunas se encontraban repetidas.
- **Conjunto en inglés**: se ha creado a partir de las frases en inglés del segundo conjunto de datos, por lo que contiene un total de 260 frases.

4.1.1. Transformación de los datos

Tal y como se explicó en la Sección 2.4, para introducir los datos en el modelo de lenguaje es necesario que estos tengan una estructura de *prompt*, que indique al modelo que esperamos una determinada respuesta de él.

Por esta razón, las frases y sus correspondientes palabras clave han sido transformadas a formato *prompt* para crear los conjuntos de datos. Tanto el conjunto de frases en español como el de frases en inglés contienen una sola columna de tipo texto, que siguen el siguiente formato:

Lemmas: [lista de palabras clave]

Phrase: [frase generada]

Por ejemplo:

Lemmas: policía poner un multa

Phrase: El policía puso una multa.

Esta será la manera en que los datos entren al modelo durante el entrenamiento. Para los datos de test, se introducirán con el mismo formato pero sin incluir la frase que generan las palabras clave, pues esta debe ser generada por el modelos:

Lemmas: policía poner un multa

Phrase:

4.1.2. Partición de los datos en entrenamiento, test y validación

Para evaluar los modelos, se utilizará la técnica de *cross-validation*, que consiste en realizar diferentes particiones de los datos en conjuntos de entrenamiento y test, entrenando y evaluando cada partición por separado. Por esta razón, los datos se han dividido en un conjunto de entrenamiento y test para *cross-validation* y un conjunto de validación.

Los datos de validación no serán utilizados para el entrenamiento del modelo, y se utilizarán para la comparación final de resultados entre los modelos. El conjunto de datos

de validación ha sido seleccionado de manera aleatoria entre las frases que no presentaban valores nulos.

La Tabla 4.1 muestra la cantidad de datos utilizados para los experimentos en español y en inglés:

Tabla 4.1: Número de frases para validación y entrenamiento de los conjuntos de datos en español y en inglés

	Conjunto en español	Conjunto en inglés
Entrenamiento-Test	290	220
Validación	50	40

4.2 Resultados experimentales con Gemma y Llama-3

Los primeros experimentos de este trabajo han sido realizados utilizando los modelos de Gemma y Llama-3, concretamente Gemma-7B y Llama3-8B. Para el entrenamiento de ambos se ha utilizado el método de adaptación de LoRA, explicado en la Sección 3.4.1. Además,el testeo de los modelos ha sido realizado mediante *cross-validation*, realizando un total de diez particiones sobre los datos de entrenamiento. A partir de estas se ha creado el conjunto final de predicciones, sobre el cual se han calculado las métricas de BLEU y COMET.

Tabla 4.2: *Scores* de BLEU y COMET para el modelo Gemma-7B

		Español		Inglés	
Rank	*Alpha*	BLEU	COMET	BLEU	COMET
16	8	0	42.64	-	-
16	16	10.26	59.27	-	-
16	32	50.39	87.24	-	-
32	16	13.47	61.57	-	-
32	32	52.56	87.54	-	-
32	64	53.22	89.53	-	-
64	32	52.26	88.15	-	-
64	64	52.44	89.05	-	-
64	128	46.92	87.38	-	-

Los resultados obtenidos en el entrenamiento se muestran en la tabla

!!! INSERTAR TABLA/GRÁFICA RESULTADOS !!!

En estos experimentos observamos que con el modelo de Llama-3 se obtienen mejores resultados que con Gemma. Por esta razón, se ha decidido ampliar los experimentos con este modelo para comprobar si podemos obtener aún mejores resultados.

rango = 128, alfa = 128

!!! INSERTAR GRAFICAS/TABLAS !!!!

4.3 Validación de los modelos

4.4 Comparación con otros modelos

Tabla 4.3: *Scores* de BLEU y COMET para el modelo Gemma-7B

	Español		Inglés	
	BLEU	COMET	BLEU	COMET
Llama-3	-	-	-	-
Gemma	-	-	-	-
GPT-4	66.36	92.51	-	-
AsTeRICS Grid	26.89	78.36	-	-

CAPÍTULO 5
Conclusiones

????? ????????????? ????????????? ????????????? ????????????? ?????????????

5.1 Objetivos cumplidos

5.2

Bibliografía

[1] OpenAI (2023). Gpt-4 technical report, 2024.

[2] ARASAAC. Arasaac - portal de comunicación aumentativa y alternativa, 2024. Accessed: 2024-08-04.

[3] Aula Abierta ARASAAC. Asterics grid: Pantalla de inicio del comunicador, 2024. Accessed: 2024-08-04.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[5] Bohdan Bilonoh and Sergii Mashtalir. Parallel multi-head dot product attention for video summarization. In *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*, pages 158–162, 2020.

[6] Mikhail Burtsev, Martin Reeves, and Adam Job. The working limitations of large language models. *MIT Sloan Management Review*, page 7, November 2023.

[7] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018. IEEE, 2019.

[8] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[9] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. Multi-head attention: Collaborate instead of concatenate, 2021.

[10] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

[11] Jiatao Gu, Kyunghyun Cho, and Victor O. K. Li. Trainable greedy decoding for neural machine translation, 2017.

[12] Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jia-Wei Low, Lidong Bing, and Luo Si. On the effectiveness of adapter-based tuning for pretrained language model adaptation, 2021.

[13] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2020.

[14] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

[15] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. 2021.

[16] Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models, 2023.

[17] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Stanford University and University of Colorado at Boulder, third edition draft edition, 2023. Draft version.

[18] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences, 2018.

[19] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[20] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[21] Seyedeh Leili Mirtaheri and Reza Shahbazian. *Machine Learning*. CRC Press, 2022.

[22] Tom Mitchell. *Machine Learningl*. McGraw-Hil, 1997.

[23] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts; London, England, 2022.

[24] Moin Nadeem, Tianxing He, Kyunghyun Cho, and James Glass. A systematic characterization of sampling algorithms for open-ended language generation, 2020.

[25] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[26] Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. Comet: A neural framework for mt evaluation, 2020.

[27] Raúl Rojas. *The Backpropagation Algorithm*, pages 149–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[28] Claude Sammut and Geoffrey I. Webb, editors. *Samuel's Checkers Player*, pages 881–881. Springer US, Boston, MA, 2010.

[29] Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019.

[30] Noam Shazeer. Glu variants improve transformer, 2020.

[31] Anuroop Sriram, Heewoo Jun, Sanjeev Satheesh, and Adam Coates. Cold fusion: Training seq2seq models together with language models, 2017.

[32] Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning, 2019.

[33] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[34] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[35] Gemini Team. Gemini: A family of highly capable multimodal models, 2024.

[36] Gemma Team. Gemma: Open models based on gemini research and technology, 2024.

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2023.

[39] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016.

[40] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024.

[41] Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019.

[42] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021.

APÉNDICE A

Configuració del sistema

????? ????????????? ????????????? ????????????? ????????????? ?????????????

A.1 Fase d'inicialització

????? ????????????? ????????????? ????????????? ????????????? ?????????????

A.2 Identificació de dispositius

????? ????????????? ????????????? ????????????? ????????????? ?????????????

APÉNDICE B
??? ????????????? ????

????? ????????????? ????????????? ????????????? ????????????? ?????????????