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 evaluarà el rendiment d'aquests models quan són adaptats per a la seua integració en els SAAC. Aquesta 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 explora cómo los modelos de lenguaje grandes (LLMs) pueden ser aplicados en el campo de los Sistemas Aumentativos y Alternativos de Comunicación (SAAC) para la generación de frases de lenguaje natural a partir de palabras clave para su uso en comunicadores electrónicos. Para ello, se entrenarán y evaluarán algunos de los LLMs más destacados en la actualidad.

En este primer capítulo se presenta la motivación que impulsa el estudio 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. 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) [1], *"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].

Existen diversas 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 pue-

[1] https://arasaac.org

den ser organizados y adaptados dependiendo de las necesidades de cada persona, permitiendo que cada usuario añada aquellos que necesite. De esta manera, los usuarios seleccionan secuencias de 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 [2], 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 distintos tipos de Sistemas de Comunicación Aumentativa y Alternativa (SAAC) disponibles, explorando cómo las herramientas de aprendizaje automático e inteligencia artificial podrian optimizar su uso.

2. Adaptar y evaluar modelos de lenguaje grandes actuales y de acceso público para su implementación en la generación de frases dentro de herramientas SAAC.

[2] El comunicador AsTeRISCS Grid se puede utilizar *online* en `grid.asterics.eu`

3. Realizar una comparación de los resultados obtenidos utilizando LLMs respecto a otros comunicadores SAAC que permiten la generación de frases y que se encuentran actualmente en el mercado.

1.3 Estructura de la memoria

Este documento está dividido en 6 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, los experimentos que han sido realizados y los resultados obtenidos. El Capítulo 5 ofrece un análisis adicional de los resultados obtenidos con el modelo principal de este proyecto, presentando una comparación de resultados con otros modelos, con resultados de individuos humanos y un análisis de los errores más comunes. Finalmente, el Capítulo 6 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 en esta área, conocido como *the Samuel Ckeckers-playing Program* [30] (el programa de juego de damas de Samuel).

Tom Mitchell [24] 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 la tarea 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 programa 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 $(X, Y) = (X, f(X))$. Para ello, se utilizan datos etiquetados, es decir, datos que incluyen tanto las entradas como las salidas correctas asociadas.
- **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 estos. En este enfoque se utilizan datos no etiquetados.

- **Aprendizaje semi-supervisado**,combina elementos del aprendizaje supervisado y el no supervisado, utilizando tanto datos etiquetados como no etiquetados. Este enfoque permite aprovechar la información disponible en grandes volúmenes de datos no etiquetados, utilizando una cantidad menor de datos etiquetados para guiar el proceso de aprendizaje.
- **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 [23].

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 un 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

Las redes neuronales son un tipo de modelo que se inspira en la estructura y funcionamiento del cerebro humano para procesar información. El primer modelo de red neuronal artificial fue desarrollado en 1943 por Warren McCulloch y Walter Pitts, y se conoce como *McCulloch-Pitts neuron* [21]. Este modelo simplificado imitaba el comportamiento de una neurona natural y marcó el inicio de la investigación en redes neuronales. Aunque las redes neuronales modernas se basan en estas ideas iniciales, han evolucionado significativamente y ya no siguen de manera directa las inspiraciones biológicas originales.

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 [6]. Este tipo de modelo surgió como método para la resolución de tareas de clasificación binaria, en las que se debe decidir si una determinada entrada pertenece o no a una clase. Posteriormente se amplió para abarcar también tareas de clasificación multiclase. Es un tipo de clasificador lineal, por lo que hace sus predicciones basándose en funciones de predicción lineales. Este tipo de arquitectura ha servido como base para arquitecturas de redes neuronales mucho más complejas.

Tal y como se observa en la Figura 2.1, la arquitectura de perceptrón simple está formada por una única 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* b. El umbral es una constante que se incorpora para desplazar la función de activación hacia

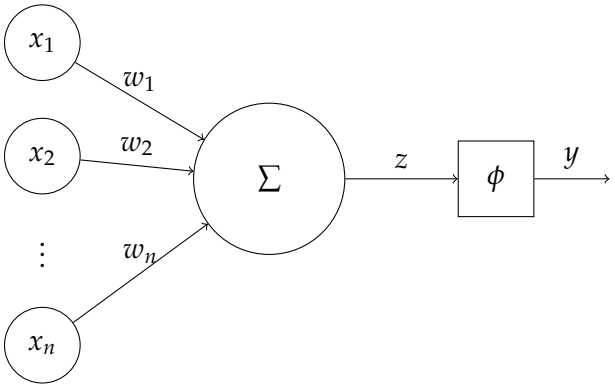


Figura 2.1: Esquema de perceptrón simple

la izquierada o derecha, lo cual puede ser fundamental para el éxito del aprendizaje. Así, la salida producida por la neurona se calcula como:

:

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

A este resultado se le aplica una función ϕ conocida como función de activación. 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 vectores de 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 el parámetro de 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, se repiten estos pasos durante un número determinado de iteraciones o hasta que el modelo converge.

La arquitectura de perceptrón simple tiene una limitación principal, y es que este tipo de modelos solo convergen si las 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 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 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).

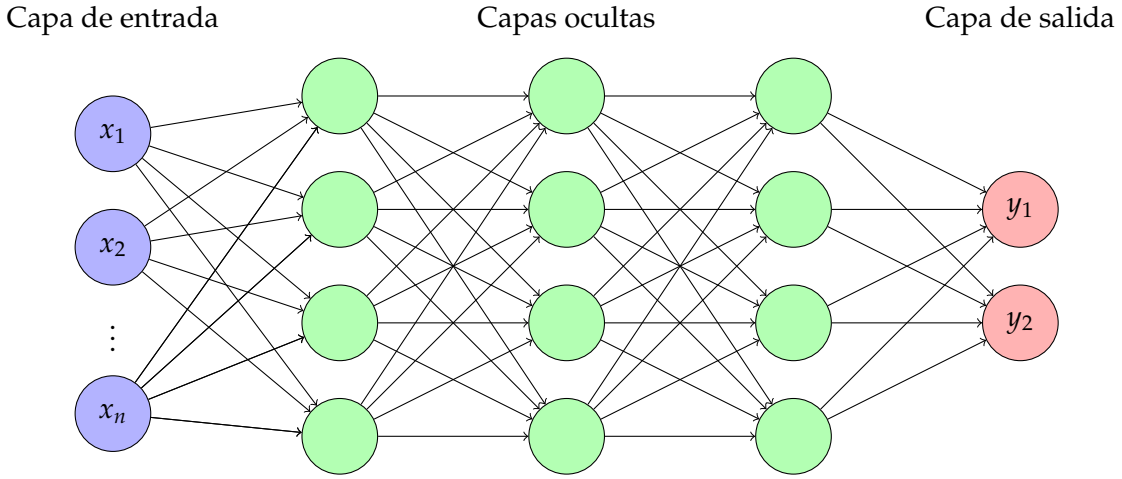


Figura 2.2: Esquema de perceptrón multicapa

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*) [29]. El algoritmo de retropropagación consta de 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. Algunas de las funciones de activación no lineales más utilizadas son 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 iteraciones 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 de este tipo 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 descomponer la frase en una secuencia de palabras o tokens. Una vez obtenida la lista de tokens, se asigna a cada uno un número que sirve como índice en un vocabulario.
2. **Transformación de palabras en vectores**: los índices numéricos se convierten 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 generado los vectores asociados a cada token, 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.

De esta manera, se consigue una representación en forma de vectores numéricos de las secuencias de texto, de manera que estas puedan ser procesadas por la red neuronal.

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, y los modelos *encoder-decoder*.

Redes neuronales recurrentes (RNN)

Las redes neuronales recurrentes (RNN, por sus siglas en inglés *recurrent neural networks*) son un tipo de red neuronal diseñadas para procesar secuencias de texto teniendo en cuenta la información aprendida en etapas anteriores. Esto se consigue mediante el mantenimiento de una variable de estado oculto h_t, que permite que la prediccióny_t depende tanto de la de entrada actual x_t como del estado oculto [25]. 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 y se actualiza a medida que se procesan nuevas entradas.

Las RNNs son útiles para tareas como generación de texto, clasificación y traducción de secuencias, ya que pueden capturar dependencias a lo largo del tiempo dentro de los datos secuenciales.

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:

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

donde W_y es una matriz de pesos, ϕ es la función de activación y b_y es un vector de sesgos [36].

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. Modelos *encoder-decoder*

Los modelos *encoder-decoder*, también conocidos como modelos *seq2seq* (*sequence-to-sequence*) son modelos capaces de generar secuencias de salida contextualmente apropiadas a partir de las secuencias de entrada. Estos modelos son capaces de manejar secuencias de longitud variable tanto en la entrada como en la salida. Son modelos ampliamente utilizados en tareas como el resumen de textos, la respuesta a preguntas, la generación de diálogo y la traducción automática [19].

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 transformar las secuencias de entrada a una representación intermedia h. 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 explica en detalle en la Sección 2.2.4 [33].

Esta arquitectura se puede implementar utilizando diversos tipos de redes neuronales, incluyendo los transformers, presentados en la Sección 2.3, o las redes neuronales recurrentes, vistas en el apartado anterior.

2.2.4. Atención

Tal y como se explicó en la Sección 2.2, en las redes neuronales clásicas, el cálculo en cada capa se realiza mediante la combinación lineal de los vectores de entrada y los correspondientes pesos, seguida de la aplicación sobre este resultado de una función de activación. Esta operación se puede representar matemáticamente como $Z = \phi(XW)$, donde X es la matriz formada por los vectores de entrada, W la matriz de pesos, ϕ es la función de activación y Z son las salidas de las capas [25].

Esta aproximación tiene limitaciones en términos de flexibilidad y capacidad de manejar datos complejos. El mecanismo de atención mejora esto al permitir que los pesos de la red dependan dinámicamente de los datos de entrada. Así, en lugar de utilizar una matriz de pesos fija, el mecanismo de atención ajusta los pesos en función de las características de cada entrada, lo que proporciona una forma más flexible de moldear las relaciones entre la entrada y la salida. Formalmente, esto puede expresarse como $Z = \phi(XW(X))$.

Aquí la matriz de pesos ya no es fija, sino que se adapta dinámicamente a la entrada X. Esta adaptación permite que el modelo enfoque su atención en diferentes partes de la entrada para cada salida, mejorando su capacidad para capturar información relevante y mejorando su precisión.

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

- Q (*queries* o consultas) es un conjunto de vectores derivados de X que representan lo que cada palabra "busca.en otras palabras de la secuencia, es decir, con qué otro tipo de palabras puede estar relacionada.
- K (*keys* o claves) es otro conjunto de vectores también derivado de X que describe las propiedades de cada palabra.
- V (*values* o valores) es otro conjunto de vectores derivado de X que contiene la información que cada palabra transmite hacia la salida.

Estos vectores de *queries*, *keys* y *values* se obtienen procesando las palabras a través de 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. 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 las salidas del modelo sean una combinación dinámica ponderada de las entradas, 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 [25, 19].

2.3 Transformers

En esta sección se presenta la arquitectura de transformer. Los modelos basados en transformers emplean una arquitectura *encoder-decoder* que utiliza el mecanismo de atención, descrito en la sección anterior. Esta arquitectura fue introducida en el artículo "*Attention is All You Need´´* [40], publicado en el año 2017, y ha supuesto una gran revolución en diversas áreas del aprendizaje automático por su capacidad para manejar de manera efectiva secuencias largas y complejas [11]. 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 la estructura del transformer original se utiliza 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 *feed-forward* 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 [19, 5].

La idea principal del transformer es, a lo largo de esta serie de capas, poder construir representaciones contextualizadas cada vez más precisas 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 [19].

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 [19], pues es en estas donde el modelo aplica el mecanismo de atención. En este caso, se utiliza una variación del mecanismo de atención explicado 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 y la de valores V. 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.

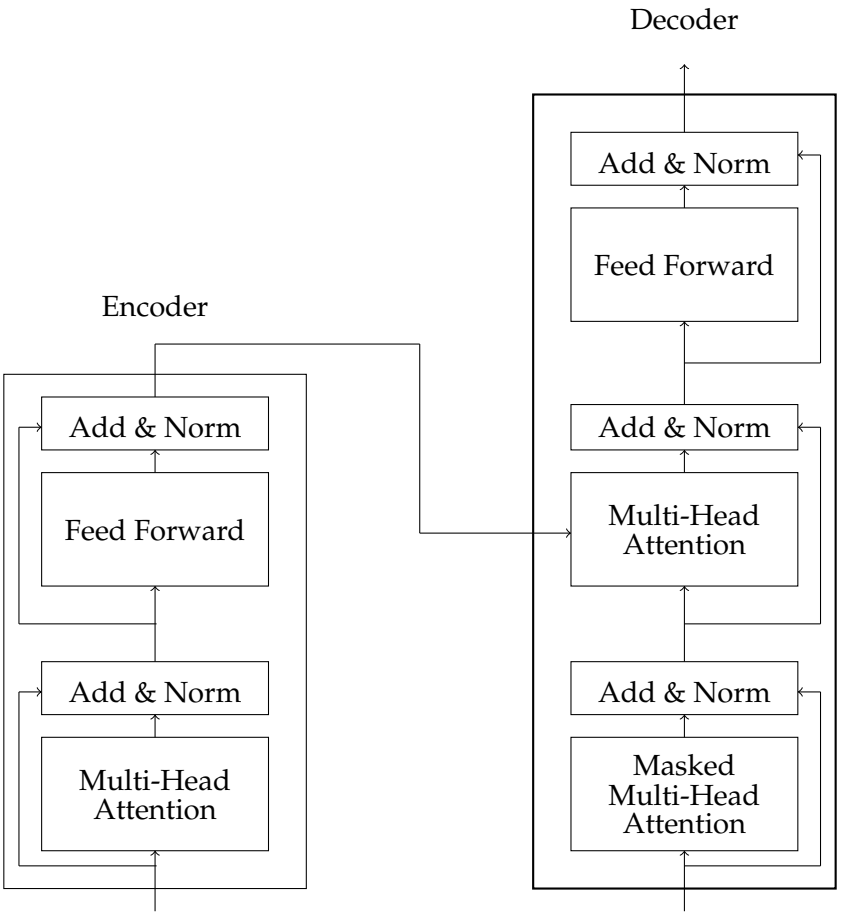


Figura 2.3: Esquema de la arquitectura de los transformers

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].

De todas maneras, los transformers 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 distintos 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 las *queries*, claves y valores, 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 [10, 5, 19].

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 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 *feed-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 [19]:

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

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 (LLM por su nombre en inglés, *large language models*) 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*. Así, los LLMs son modelos que están diseñados fundamentalmente para predecir la siguiente palabra de una secuencia de palabras [19]. 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 introduce 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*. Algunas de las técnicas más utilizadas son las siguientes [1]:

- ***Zero-shot prompting***: consiste en presentar al modelo la tarea que se desea realizar sin proporcionar ejemplos previos sobre los que pueda basarse. Los modelos de lenguaje grandes han demostrado funcionar bien con este tipo de *prompting* en tareas simples, aunque pueden tener dificultades en tareas más complejas. En estos casos, puede ser recomendable utilizar *few-shot prompting* en su lugar.

- ***Few-shot prompting***: en este caso, se en proporciona al modelo un número limitado de ejemplos que muestran cómo realizar la tarea. Esta técnica suele ser bastante efectiva, pero puede fallar en tareas que requieran un razonamiento complejo.

- ***Chain-of-thought* (CoT)** [42]: esta técnica permite realizar un razonamiento más complejo añadiendo pasos intermedios en la *prompt* que muestren el proceso de pensamiento necesario para resolver el problema. Puede combinarse con *few-shot promting* para tener mejor rendimiento en tareas complejas.

[1] Esta información se ha obtenido de Prompt Engineering Guide www.promptingguide.ai

- ***Self-consistency*** [41]: consiste en generar diferentes caminos de razonamiento utilizando *few-shot CoT*, y luego seleccionar la respuesta más coherente entre ellos. En esta técnica, el modelo crea varias respuestas posibles y elige la que mejor resuelve el problema.
- ***Tree of thought*** **(ToT)**: es una generalización de *CoT prompting* que se basa en la idea de construir un "árbol de pensamientos". Cada pensamiento de este árbol representa un paso intermedio hacia la solución del problema. El modelo evalúa continuamente el progreso a través de estos pasos intermedios y utiliza algoritmos de búsqueda para explorar los diferentes caminos hacia la solución.

2.4.2. Generación de respuestas

Una vez introducida la secuencia de entrada en forma de *prompt* al modelo, este debe generar la secuencia de salida. Como se ha mencionado anteriormente, los modelos de lenguaje grandes se utilizan en tareas de generación de texto condicionado a un fragmento de entrada. Es decir, el modelo debe generar un texto de respuesta que esté condicionado a la *prompt* proporcionada.

Para ilustrar este proceso, supongamos que se le introduce al modelo la siguiente *prompt*:

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

A partir de esta, el modelo deberá calcular 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 [7].

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* [13]. El algoritmo *greedy decoding* consiste en elegir la palabra d con la probabilidad condicional más alta hasta el momento en cada instante.

$$\hat{d_i} = argmax_{d \in V} P(d | \{d_1, d_2, ..., d_{i-1}\}) \tag{2.17}$$

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*:

$$p(d_1, d_2, ..., d_N) = \prod_{i=1}^{N} p(d_i \mid d_1, d_2, ..., d_{i-1}) \tag{2.18}$$

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.

2.4.3. 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 [19].

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 [26].

Nucleus sampling

El algoritmo de *nucleus sampling* [15], 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 conjunto de palabras más probables que acumulan una masa de probabilidad p. 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:

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

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:

$$\hat{p}_i = \frac{p'_i}{\sum_{j:d_j \in V^{(p)}} p'_j} \tag{2.21}$$

Donde $p'_i = p_i * \mathbb{1}\{d_i \in V^{(p)}\}$ [26].

Tempered sampling

En el algoritmo de *tempered sampling* [26] 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:

$$\hat{p}_i = \frac{p'_i}{\sum_{j=1}^{V^{(k)}} p'_j} \tag{2.23}$$

donde $V^{(k)}$ es el conjunto de las k palabras más probables y $p'_i = exp(log(p_i)/\tau * \mathbb{1}\{i \leq K\}$.

2.4.4. 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 [20].

- **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 hincapié 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, Llama3, 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 [38] es un modelo de código abierto desarrollado por Google, basado en sus modelos privados Gemini [37]. 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) [9].

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

- ***Multi-Query Attention*** [31]: 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)*** [35]: 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*** [32]: 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** [45]: 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 [39] 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 [32], 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 Llama3.

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 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 3.1, 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 [43] 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*) [43].

- **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 [46]. 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 [14]. 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 [16, 34].

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 [16].

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 intermedias, para permitir la inicialización de las redes como identidad [25].

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 [18]. 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* [17], 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 descomponer las matrices de parámetros en componentes de rango bajo. El rango de una matriz se define como el número de columnas linealmente independientes de esta. Una columna es considerada linealmente dependiente si puede ser obtenida como una combinación lineal de otras columnas de la matriz. Esto implica que, si existen columnas linealmente dependientes en la matriz, estas se pueden eliminar sin perder información esencial, 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 de cambios $\triangle W$ se representa como el producto de dos matrices de rango bajo B y A:

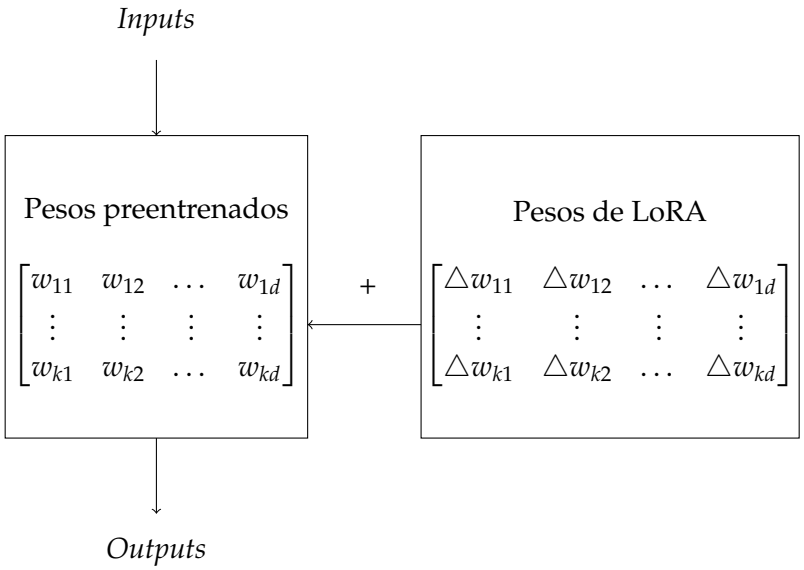


Figura 3.1: Esquema de las matrices de parámetros de LoRA

$$\triangle W = B \cdot A = \begin{bmatrix} b_{11} & \dots & b_{1r} \\ \vdots & \vdots & \vdots \\ b_{k1} & \dots & b_{kr} \end{bmatrix} \cdot \begin{bmatrix} a_{11} & \dots & a_{1d} \\ \vdots & \vdots & \vdots \\ a_{r1} & \dots & a_{rd} \end{bmatrix} \tag{3.1}$$

Aquí, la matriz B tiene dimensiones $k * r$, y la matriz A, $r * d$. Donde k es el número de columnas de la matriz de pesos original, d es el número de filas, y r es un hiperparámetro conocido como *rank* o rango, que debe ser determinado.

Dada una entrada x, la salida del modelo se calcula de la siguiente manera:

$$y = Wx + \triangle Wx = Wx + BAx \tag{3.2}$$

Durante 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. Además, la matriz $\triangle W$ se escala por α/r, donde α es un parámetro adicional que debe ser ajustado y r es el rango mencionado anteriormente. Este escalado controla el impacto de la matriz de cambios $\triangle W$ aprendida durante el entrenamiento de LoRA en el modelo final.

Este enfoque proporciona una mayor eficiencia computacional al reducir significativamente el número de elementos que deben ajustarse en las matrices A y B en comparación con la matriz de pesos original.

En las siguientes secciones se presentan los hiperparámetros que deben ajustarse en el 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 entrenamiento. 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

El parámetro *alpha* (α) se utiliza para escalar la matriz de cambios $\triangle W$, determinando la importancia relativa de los pesos aprendidos durante el entrenamiento en comparación con los pesos preexistentes del modelo. Como se mencionó anteriormente, la matriz de cambios $\triangle W$ se escala por α/r para poder controlar el impacto que esta tiene sobre el modelo final. Por tanto, en cuanto al parámetro α se deben tener en cuenta los siguientes aspectos:

- Si $\alpha = r$, los nuevos pesos serán integrados con los del modelo original en una proporción de 1:1. Esto implica que los pesos aprendidos tendrán la misma importancia que los pesos del modelo preentrenado.

- Si $\alpha > r$, se está dando mayor peso a los nuevos parámetros aprendidos. Esto es recomendable cuando se cuenta con un conjunto de datos de entrenamiento grande y de calidad. De lo contrario, al darle más importancia a lo aprendido a partir 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 y los dominios del modelo preexistente y la tarea a realizar son similares, pues así se garantiza que el modelo conserve más el conociemiento original.

Dropout

El parámetro de *dropout* determina la probabilidad de que, en cada iteración del entrenamiento, ciertas neuronas de la red sean desactivadas temporalmente, configurándolas artificialmente a 0. Este procedimiento se realiza para prevenir el sobreajuste del modelo, que puede ocurrir cuando este se ajusta demasiado a los datos de entrenamiento, provocando que pierda capacidad de generalización con datos nuevos.

De esta manera, un valor alto de *dropout* significa que un mayor número de neuronas se desactivarán durante el entrenamiento. Sin embargo, si el valor es excesivamente alto, el modelo podría no aprender lo suficiente sobre los datos, ya que se reducen demasiado las neuronas activas. Por otro lado, un valor muy bajo del *dropout* puede llevar a que el modelo pierda capacidad de generalización, ya que es más probable que se sobreajuste a los datos de entrenamiento.

Generalmente, se recomienda utilizar un valor de *dropout* entre 0.05 y 0.5, ajustándolo según el número de parámetros del modelo, el volúmen de datos de entrenamiento y la complejidad del problema.

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 [8]. 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 [22]. 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 técnica ampliamente utilizada es la de cuantización post entrenamiento [44]. 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 objetivo 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*) [27] 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:

$$p_n = \frac{\sum_{C \in \{\text{Candidatos}\}} \sum_{\text{n-grama} \in C} Count_{clip}(\text{n-grama})}{\sum_{C \in \{\text{Candidatos}\}} \sum_{\text{n-grama} \in C} Count(\text{n-grama})} \tag{3.3}$$

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:

$$BP = \begin{cases} 1 & \text{si } c > r \\ e^{(1-r/c)} & \text{si } c \leq r \end{cases} \tag{3.4}$$

donde c es la longitud de la frase candidatas generadas por el modelo 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:

$$BLEU = BP + exp(\sum_{n=1}^{N} w_n log p_n) \tag{3.5}$$

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

En cuanto a la interpretación, generalmente se considera que un *score* BLEU menor de 30 indica traducciones de mala calidad, entre 30 y 40 sugiere traducciones aceptables, y un *score* de 40 o superior indica un rendimiento bueno. La Tabla 3.1 proporciona una interpretación detallada de las puntuaciones de BLEU.

El uso de esta métrica es muy común en tareas de procesamiento de lenguaje natural debido a su facilidad de cálculo y la simplicidad en la interpretación de la puntuación que proporciona, la cual suele alinearse con la evaluación humana. Sin embargo, BLEU presenta algunas limitaciones importantes, principalmente porque su cálculo se basa unicamente en las coincidencias exactas entre las frases generadas y las de referencia, lo que

Tabla 3.1: Interpretación de las puntuaciones de BLEU, extraída de [12]

Puntuación BLEU	Interpretación
<10	Casi inútil
10 - 19	Resulta difícil captar la esencia
20-29	La esencia es clara, pero aparecen errores gramaticales
30-40	Comprensible, buenas traducciones
40-50	Traducciones de alta calidad
50-60	Traducciones de muy alta calidad. adecuadas y fluidas
>60	Calidad generalmente mejor que la humana

puede limitar su capacidad para capturar variaciones en la formulación del lenguaje. Por esta razón, es aconsejable utilizar también otras métricas que no dependan exclusivamente de coincidencias exactas para obtener una evaluación más completa y precisa de la calidad de las frases generadas.

3.6.2. COMET

COMET (*Crosslingual Optimized Metric for Evaluation of Translation*) [28] 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:

$$e_{x_j} = \mu E_{x_j}^T \alpha \tag{3.6}$$

  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. Este cálculo se realiza mediante una red neuronal *feed forward*, entrenada para minimizar el MSE (*Mean Squared Error* o error cuadrático medio). La salida de esta red neuronal es un valor entre 0 y 1 que representa la puntuación de COMET.

La métrica COMET es más compleja de interpretar en comparación con BLEU. Generalmente, se considera que un valor de COMET superior a 80 indica una traducción de alta calidad.

CAPÍTULO 4
Experimentos realizados

En este capítulo se presentan los experimentos realizados en este trabajo, los cuales se han llevado a cabo utilizando dos modelos de lenguaje grandes: Gemma y Llama3. Los experimentos se centraron principalmente en la exploración de los parámetros rango y *alpha* de la adaptación mediante LoRA, con el objetivo de encontrar la configuración que proporcionara los mejores resulados posibles para la tarea de generar frases de lenguaje natural a partir de listas de palabras clave, en el contexto de los Sistemas de Comunicación Aumentativa y Alternativa. Tras identificar la mejor configuración, se realizó un experimento adicional utilizando todos los datos de entrenamiento disponibles para evaluar la eficacia de los modelos con un conjunto de datos de test que no había sido utilizado en fases anteriores. Los experimentos se realizaron con conjuntos de datos en español y, a continuación, se realizaron también con el conjunto de datos en inglés para estudiar el efecto del lenguaje en los resultados. Todos los experimentos han sido realizados utilizando Google Colab [1].

A continuación, se detallan los experimentos realizados y los resultados obtenidos. También se incluyen comparaciones entre los modelos entrenados y otros sistemas externos como GPT-4, el comunicador electrónico AsTeRISCS Grid, y resultados obtenidos por personas en esta misma tarea.

4.1 Datos utilizados

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, y representan las palabras que un usuario introduciría a través de la interfaz de un comunicador electrónico con la intención de generar la frase correspondiente.

El primer conjunto de datos incluye un total de 212 frases, todas ellas en español. En la Tabla 4.1 se presentan algunos ejemplos de frases de este conjunto para ilustrar su formato. El segundo conjunto consta de 260 frases, disponibles tanto en español como en inglés. Se presentan ejemplos de este conjunto en la Tabla 4.2.

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.

[1] Google Colab es una plataforma en la nube que permite escribir y ejecutar código Python en *notebooks* interactivos, ofreciendo acceso a recursos de procesamiento como GPUs y TPUs.

Tabla 4.1: Ejemplos de frases del conjunto de datos en español

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

Tabla 4.2: Ejemplos de frases del conjunto de datos en inglés

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

A partir de estos dos conjuntos, se ha creado un conjunto de datos para los experimentos 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. Preprocesamiento 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. Durante la inferencia, se introducirán con el mismo formato pero sin incluir la frase que generan las palabras clave, pues esta debe ser generada por el modelo:

Lemmas: policía poner un multa

Phrase:

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

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 validación, entrenando y evaluando de manera independiente cada partición. Por esta razón, los datos se han dividido en un conjunto de entrenamiento y validación para *cross-validation* y un conjunto adicional de test. Este último no se utilizará durante la fase de entrenamiento, sino exclusivamente para la comparación final de resultados entre los modelos. El conjunto de datos de test ha sido seleccionado de manera aleatoria considerando solo las frases que no presentaban valores nulos.

La Tabla 4.3 muestra la cantidad de datos utilizados para los experimentos en español y en inglés:

Tabla 4.3: 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 Experimentos realizados con el conjunto en español

Los modelos utilizados para los experimentos de este trabajo son los modelos Gemma-7B y Llama3-8B. Ambos modelos están disponibles para su uso libre en el portal de *Hugging Face* [2].

[JUSTIFICAR ELECCIÓN MODELOS: Justifica la elección de estos modelos por razones computacionales (coste temporal y uso de memoria GPU)]

Para el entrenamiento de los modelos, se ha aplicado el método de adaptación LoRA, que se describe en detalle en la sección 3.4.1. Este método requiere la configuración de

[2] *Hugging Face* es una plataforma que ofrece herramientas y modelos de aprendizaje automático, especialmente para el procesamiento de lenguaje natural.

dos parámetros clave: el rango y el *alpha*. Dado que estos parámetros pueden influir significativamente en el rendimiento del modelo, se han realizado experimentos variando sus valores para identificar la configuración óptima que proporciona mejores resultados.

Para garantizar una evaluación robusta y fiable de los modelos, se ha utilizado la técnica de *cross-validation* con 10 particiones. Esto implica dividir el conjunto de datos en 10 partes iguales. En cada iteración, 9 partes se utilizan para entrenar el modelo y la parte restante, para validarlo. Las particiones se han realizado de manera que, al final del proceso, todos los datos hayan sido utilizados tanto para entrenamiento como para validación, asegurando que cada dato aparezce solo una vez en el conjunto de validación.

Se ha entrenado un modelo independiente para cada una de las 10 particiones. Una vez completado el entrenamiento y realizadas las predicciones de estos 10 modelos, se han combinado las predicciones de todos ellos en un solo conjunto de predicciones para calcular las métricas de rendimiento. Las métricas utilizadas para la evaluación son las métricas BLEU y COMET, ambas descritas en la Sección 3.6.

4.2.1. Primeros experimentos con Gemma y Llama3

En los primeros experimentos realizados, se ha comparado el rendimiento de los modelos Gemma-7B y Llama3-8B para evaluar cómo se comportan bajo diferentes configuraciones. El enfoque principal fue explorar el impacto del parámetro de rango en la adaptación LoRA, manteniendo un valor fijo para el parámetro *alpha*.

La Tabla 4.4 presenta los resultados obtenidos de BLEU y COMET mediante validación cruzada en 10 bloques sobre el conjunto de datos en español. Por restricciones computacionales, se ha explorado un conjunto limitado de valores de rango: 16, 32 y 64; y se ha adoptado la práctica común de fijar el parámetro *alpha* a dos veces el valor del rango [**?**].

Tabla 4.4: BLEU y COMET para los modelo Gemma-7B y Llama3-8B

		Gemma-7B		Llama3-8B	
Rank	*Alpha*	BLEU	COMET	BLEU	COMET
16	32	52.44	88.92	65.59	92.82
32	64	44.44	87.44	66.13	93.37
64	128	55.21	87.82	67.62	93.23

Los resultados de la Tabla 4.4 muestran que, en todas las configuraciones de rango probadas, el modelo Gemma presenta un rendimiento consistentemente inferior respecto al modelo Llama3. Esta diferencia de rendimiento sugiere que Llama3 puede ser más efectivo para las tareas evaluadas.

Dado que Llama3 ha demostrado ser superior en las pruebas iniciales, se ha decidido que este será el modelo principal a utilizar en los experimentos posteriores. Esta decisión busca maximizar la calidad de los resultados en los experimentos posteriores dadas nuestras capacidades computacionales limitadas.

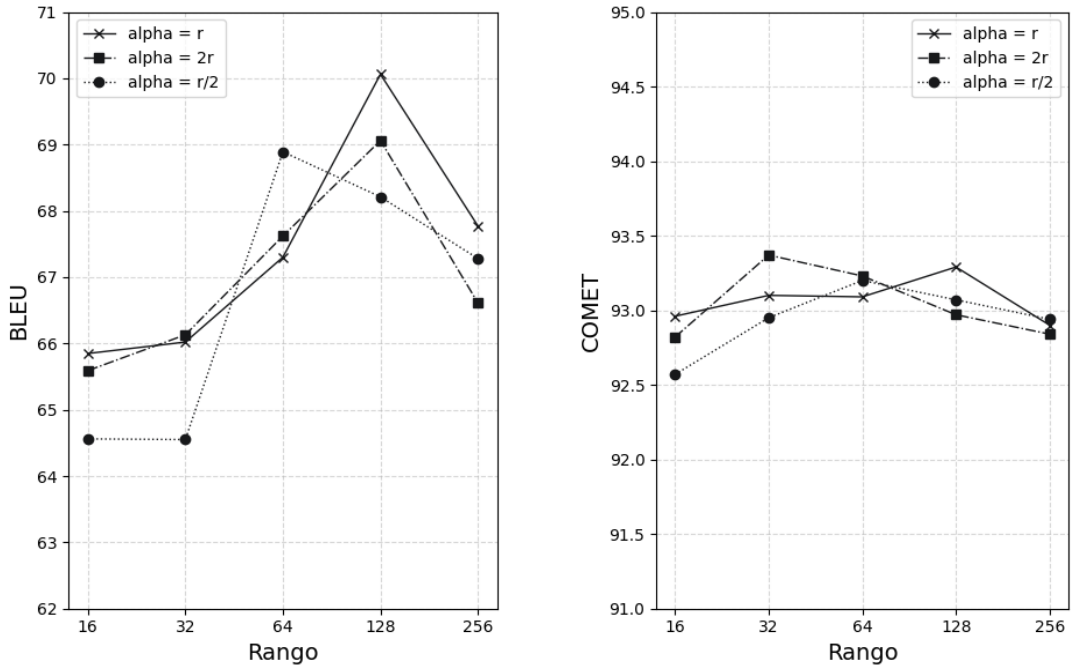
4.2.2. Exploración de valores de *alpha*

En esta fase de los experimentos, se ha investigado el impacto del parámetro *alpha* en el rendimiento del modelo. El parámetro *alpha* controla el peso relativo que se le asigna a los parámetros de LoRA en comparación con los parámetros del modelo preentrenado.

Así, para esta exploración se han realizado experimentos utilizando diferentes valores para el rango r de LoRA: 16, 32, 64 y 128. Para cada valor de r, se han probado tres configuraciones distintas de *alpha*:

1. ***alpha* = r/2**: en esta configuración, *alpha* es la mitad del valor del rango. De esta manera, se da más peso a los parámetros del modelo preentrenado y menos a los aprendidos durante el entrenamiento con LoRA.
2. ***alpha* = r**: en esta configuración, *alpha* es igual al valor del rango. De esta manera, se da el mismo peso a los parámetros del modelo preentrenado y a los aprendidos durante el entrenamiento con LoRA.
3. ***alpha* = 2r**: en esta configuración, *alpha* es el doble del valor del rango. De esta manera, se da más peso a los parámetros aprendidos durante el entrenamiento con LoRA que a los del modelo preentrenado.

Las gráficas de la Figura 4.1 presentan en el eje horizontal los valores de rango explorados. En el eje vertical, la gráfica de la izquierda muestra los valores de BLEU, mientras que la de la derecha indica los valores de COMET. Cada una de las curvas en la gráfica corresponde a una configuración diferente de *alpha*.



(a) Resultados de BLEU para Llama3-8B

(b) Resultados de COMET para Llama3-8B

Figura 4.1: Resultados obtenidos en las métricas BLEU y COMET con el modelo Llama3-8B tras ser entrenado mediante adaptación con LoRA con los valores de r 16, 32, 64, 128 y 256; y explorando distintas configuraciones de *alpha* con el conjunto de datos en español.

En cuanto a la métrica BLEU, los resultados muestran que todas las configuraciones de *alpha* siguen una tendencia ascendente hasta alcanzar su punto máximo, y a continuación disminuyen la puntuación. En el caso de $\alpha = r/2$, su valor óptimo se alcanza en en $r = 64$. En cambio, con $alpha = r$ y $alpha = 2r$ el valor óptimo se encuentra en r = 128. Este comportamiento sugiere que, cuando $\alpha = r/2$, es decir, cuando se da mayor importancia a los parámetros del modelo preentrenado que a los pesos ajustados con LoRA, el aumento del rango tiene un impacto limitado en el rendimiento, ya que no se aprovechan completamente los parámetros libres disponibles con valores de rango superiores a 64. En

general, las otras configuraciones de *alpha* muestran un desempeño superior, alcanzando su máximo en $r = 128$.

En lo que respecta a la métrica COMET, los resultados muestran que todas las configuraciones de *alpha* tienen valores muy similares, oscilando entre 92.5 y 93.5, y sin mostrar diferencias significativas en su comportamiento.

Así, tanto los resultados de BLEU como de COMET sugieren que la combinación óptima de parámetros es $r = 128$ con $\alpha = r$.

4.2.3. Validación del modelo final

Finalmente, se ha llevado a cabo el entrenamiento del modelo de Llama3 con la mejor combinación de parámetros obtenidos en los experimentos anteriores: r = 128 y *alpha* = r. En este caso, el modelo ha sido entrenado utilizando todo el conjunto de datos disponible para el experimento de validación cruzada.

Una vez completado el entrenamiento, se ha realizado la evaluación del modelo utilizando el conjunto de datos reservado específicamente para evaluación. De esta manera, se ha evaluado el desempeño del modelo en datos que no habían sido vistos por el modelo en ninguna fase de entrenamiento, lo que proporciona una medida de su capacidad de generalización.

Los resultados obtenidos con esta configuración final han sido los siguientes:

- BLEU = 62.90
- COMET = 89.15

Aunque los valores obtenidos para BLEU y COMET son inferiores a los resultados alcanzados durante la fase de experimentación anterior, estos resultados se encuentran dentro de un rango adecuado de las métricas. La disminución en estas puede deberse a la diferencia entre el conjunto de entrenamiento y el conjunto de validación. De todas maneras, los resultados siguen siendo prometedores y reflejan un buen desempeño del modelo en esta tarea.

4.3 Experimentos adicionales con el conjunto en inglés

Para evaluar la influencia del idioma en los resultados de los modelos, se han replicado los mismos experimentos realizados con el conjunto de datos en español utilizando el conjunto en inglés. De nuevo, se han utilizado los modelos Gemma-7B y Llama3-8B disponibles en el portal de *Hugging Face*.

Al igual que en los experimentos anteriores, se ha aplicado la adaptación LoRA y se ha llevado a cabo el entrenamiento utilizando validación cruzada con un total de 10 particiones.

En primer lugar, se han realizado los experimentos para ambos modelos explorando diferentes valores de r (16, 32 y 64) y utilizando una configuración fija de $\alpha = 2r$. Al igual que con el conjunto en español, los resultados obtenidos con el modelo Gemma son notablemente inferiores a los resultados obtenidos con el modelo Llama3. El mejor valor de rango resultó ser $r = 32$ en ambos casos. En el caso de Gemma, los valores de BLEU y COMET con esta configuración son de 54.32 y 88.88 respectivamente, mientras que los de Llama3 son 62.22 y 91.12.

En la Tabla 4.5 se muestran los mejores resultados obtenidos de BLEU y COMET para cada modelo e idioma. Se han tenido en cuenta lose experimentos realizados con los valores de rango 16, 32 y 64 y la configuración de $\alpha = 2r$. En el caso de los experimentos en español, el mejor valor de rango entre estos es $r = 32$ para ambos modelos. Para el conjunto en inglés es $r = 32$ para Llama3 y $r = 16$ para Gemma.

		BLEU	COMET
Español	Llama3	66.13	93.37
	Gemma	52.44	88.92
Inglés	Llama3	62.22	91.12
	Gemma	54.23	88.88

Tabla 4.5: Mejores resultados de BLEU y COMET en los experimentos de ambos idiomas y modelos

4.3.1. Exploración de valores de *alpha*

En los siguientes experimentos, se ha llevado a cabo una exploración de diferentes configuraciones del parámetro *alpha* utilizando el modelo Llama3-8B, ya que este es el modelo que ha demostrado ofrecer mejores resultados.

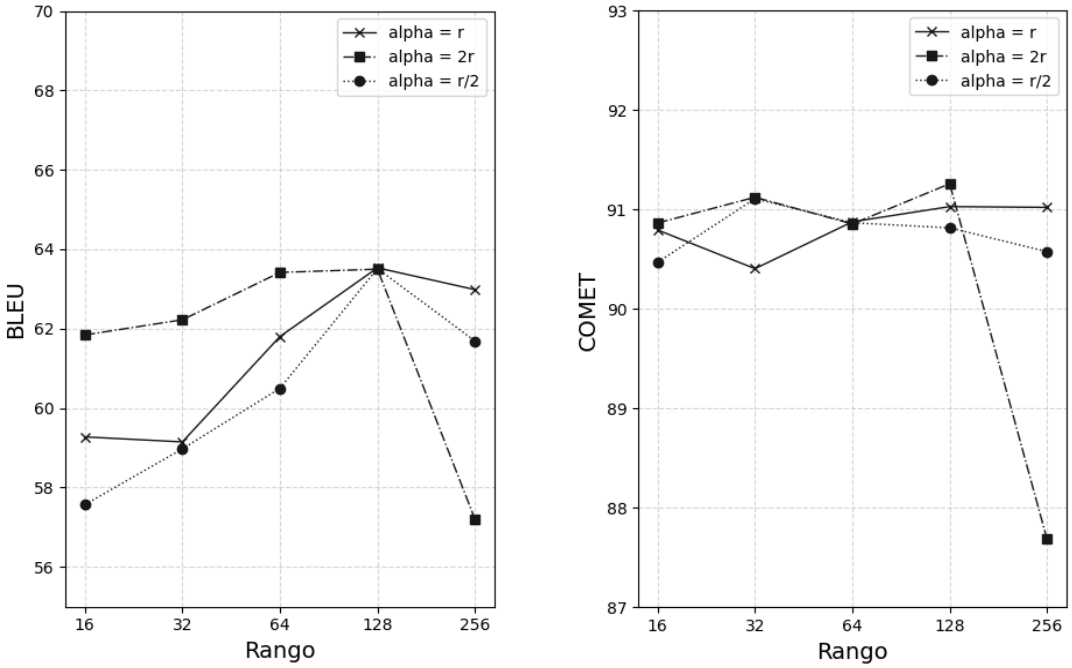
De manera similar a los experimentos realizados con el conjunto de datos en español, se han probado diferentes valores de rango: 16, 32, 64, 128 y 256. Para cada valor de rango, se han evaluado las siguientes configuraciones de *alpha*: $\alpha = r/2$, $\alpha = r$ y $\alpha = 2r$.

Las gráficas de la Figura 4.2 ilustran en el eje horizontal los valores de rango que han sido explorados. En el eje vertical, la gráfica de la izquierda muestra los valores de BLEU, mientras que la de la derecha presenta los valores de COMET. Cada una de las curvas representadas en las gráficas corresponde a una configuración diferente de *alpha*.

Al observar las puntuaciones de BLEU, se puede ver que las tres configuraciones de *alpha* presentan una tendencia ascendente hasta alcanzar su máximo en $r = 128$. Para $r = 256$, los valores de BLEU disminuyen significativamente en los tres casos, de manera similar a lo observado con el conjunto de datos en español. En este caso, la configuración con $r = 128$ y $\alpha = r$ obtiene la mejor puntuación de BLEU, aunque la diferencia con respecto a las otras configuraciones de *alpha* para el mismo valor de r es mínima.

En cuanto a la métrica COMET, las tres configuraciones siguen una tendencia similar a la observada con el conjunto de datos en español. No obstante, se observa un descenso notable en la puntuación de COMET para $r = 256$ y $\alpha = 2r$. El mejor valor de COMET se encuentra en $r = 128$, $\alpha = 2r$.

Considerando ambas métricas, la mejor configuración es $r = 128$, $\alpha = 2r$, ya que ofrece el mejor valor de COMET junto con un buen rendimiento de BLEU.



(a) Resultados de BLEU para Llama3-8B **(b)** Resultados de COMET para Llama3-8B

Figura 4.2: Resultados obtenidos en las métricas de BLEU y COMET con el modelo Llama3-8B tras ser entrenados mediante adaptación con LoRA con los valores de r = 16, 32, 64, 128 y 256; y explorando distintas configuraciones de *alpha* con el conjunto de datos en inglés.

4.3.2. Validación del modelo final

Igual que con el conjunto en español, el experimento final con los datos en inglés ha consistido en el entrenamiento de un modelo Llama3 utilizando la mejor configuración de parámetros identificada durante la experimentación previa, que fue $r = 128$ y $\alpha = 2r$.

El modelo ha sido entrenado utilizando la totalidad de datos de entrenamiento disponibles. Posteriormente, se ha realizado la validación del modelo utilizando el conjunto de datos de validación que se había reservado desde el inicio de los experimentos. Los resultados obtenidos en las métricas BLEU y COMET han sido los siguientes:

- BLEU = 56.70
- COMET = 87.96

De nuevo, se observa que los resultados obtenidos son inferiores a los obtenidos en los alcanzados en la fase de experimentación anterior.

En general, el rendimiento del modelo en inglés es inferior al del modelo en español. Esto podría deberse a la menor cantidad de datos disponibles en inglés.

CAPÍTULO 5
Análisis posteriores del modelo en español

En este capítulo se presentan una serie de análisis y experimentos adicionales que se han llevado a cabo con el objetivo de evaluar de manera más profunda la eficacia del modelo final en español. Este es el modelo Llama3 con $r = 128$ y $\alpha = r$.

En primer lugar, se ha realizado una comparación del rendimiento del modelo con el modelo GPT-4 y la aplicación de AsTeRISCS Grid, ampliamente utilizada en el ámbito de los SAAC. Además, se han realizado experimentos con particiapantes humanos para verificar si los resultados generados por el modelo son comparables a los obtenidos por personas. Finalmente, se ha realizado un análisis de los errores más comunes cometidos por el modelo, con el fin de identificar áreas de mejora y comprender mejor sus limitaciones.

5.1 Comparación con otros modelos

Una vez obtenidos los resultados finales del modelo Llama3-8B, se han realizado pruebas adicionales con el modelo GPT-4 y con la aplicación AsTeRICS Grid para comparar su rendimiento. Todas estas pruebas se han realizado utilizando el conjunto de validación reservado desde el incio de los experimentos.

El modelo GPT-4 fue seleccionado para esta comparación debido a su reconocimiento como uno de los modelos de lenguaje grandes más avanzados y potentes disponibles actualmente. Para evaluar su desempeño, se introdujeron los datos de entrenamiento a través de su interfaz *ChatGPT* mediante la técnica de *prompting*, y se le solicitó que generara las frases que se encontraban en el conjunto de test. Es importante destacar que este modelo es significativamente más grande en comparación con Llama3, ya que cuenta con 1.8 trillones de parámetros. Además, no podemos determinar con certeza si GPT-4 ha tenido acceso previo a los datos de test, ya que estos se encuentran accesibles libremente en internet. GPT-4 ha sido entrenado con una gran cantidad de información recopilada de internet, aunque los detalles específicos sobre qué datos han sido utilizados para su entrenamiento no han sido publicados.

Por otro lado, la aplicación AsTeRISCS Grid ha sido incluida en esta evaluación debido a su relevancia como herramienta de comunicación utilizada en el ámbito de los Sistemas de Comunicación Aumentativa y Alternativa. Esta aplicación es un comunicador electrónico desarrollada por ARASAAC, que incorpora la función de generar frases conjugadas a partir de las palabras clave introducidas por el usuario. Esta conjugación de las frases se realiza mediante una serie de reglas predefinidas.

La Tabla 5.1 presenta los resultados de BLEU y COMET obtenidos con los tres modelos (Llama3-8B, GPT-4 y AsTeRISCS Grid).

Tabla 5.1: Puntuaciones de BLEU y COMET de los distintos modelos utilizados para la comparación

	BLEU	COMET
Llama3	62.90	89.15
GPT-4	66.36	92.51
AsTeRICS Grid	26.89	78.36

Obsevamos que, aunque el modelo entrenado Llama3-8B no alcanza el nivel de rendimiento de GPT-4, sus resultados son notablemente superiores a los de AsTeRICS Grid, una herramienta ampliamente utilizada en el ámbito de los SAAC. Estos resultados sugieren que la integración de modelos de lenguaje grandes podría mejorar significativamente la calidad de las aplicaciones en este campo, proporcionando una alternativa más robusta y efectiva para la generación de frases a partir de las palabras introducidas por los usuarios y mejorando así la calidad de la comunicación de estos.

5.2 Experimentos con personas

Además de las pruebas realizadas con otros modelos, se han realizado también experimentos con un grupo de personas para poder comparar los resultados obtenidos por el modelo Llama3-8B con producciones humanas. Para ello, se explicó la tarea a un grupo de 5 personas, estudiantes de Ciencia de Datos o Informática con edades comprendidas entre 22 y 26 años. A continuación, se les proporcionó el conjunto de datos de entrenamiento para que pudiesen revisar ejemplos de la realización de la tarea, así como el conjunto de datos de validación sin incluir las frases generadas, solamente con las listas de palabras clave.

A partir de esta información, se les pidió a los participantes que escribieran las frases que consideraban que podrían generarse basándose en los ejemplos proporcionados. Posteriormente, se calcularon las punutaciones de BLEU y COMET para las frases producidas por cada participante, siguiendo el mismo procedimiento utilizado en los experimentos anteriores.

La Figura 5.1 presenta en el eje horizontal los valores de BLEU y en el vertical, los valores de COMET. Los puntos marcados por triángulos representan los resultados obtenidos por los distintos indiviudos, y el punto marcado con un círculo representa los resultados del modelo.

Los resultados obtenidos por el modelo final de Llama3 entrenado con el conjunto de datos en español son BLEU = 62.90 y COMET = 89.15. Comparando estos resultados con los obtenidos por personas, observamos que, aunque el modelo Llama3 presenta un rendimiento inferior al promedio de los resultados obtenidos por individuos, supera a dos de ellos en términos de BLEU y a uno en términos de COMET.

Esto sugiere que, a pesar de no alcanzar los niveles más altos de rendimiento humano, el modelo logra generar frases que se acercan a los estándares de calidad que se obtendrían si las frases fueran generadas por personas. Este desempeño cercano a los resultados humanos indica que el modelo Llama3 tiene una capacidad considerable para gnerar frases de manera efectiva y coherente.

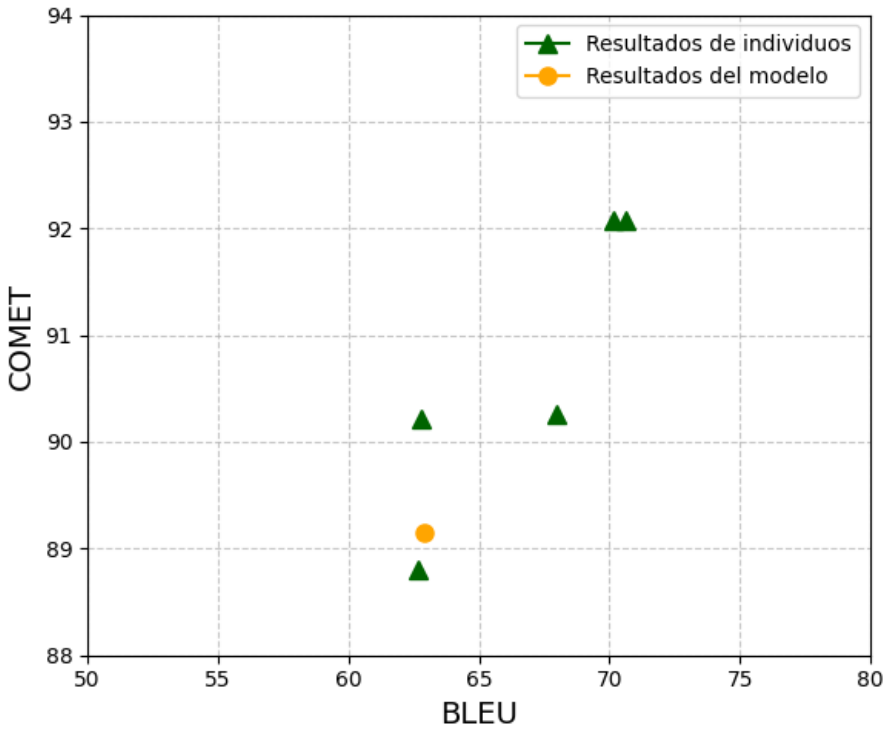


Figura 5.1: Resultados de BLEU y COMET obtenidos por los individuos y por el modelo en la tarea de generación de frases

5.3 Análisis de errores

En esta sección se realiza un análisis de los errores más comunes observados en la generación de frases por parte del modelo. El objetivo de este es identificar y comprender los errores recurrentes en las frases generadas, lo que proporcionará una base para futuras mejoras en el proceso.

El análisis se ha realizado comparando las frases generadas del conjunto de validación con las frases reales. Se observa que, de las 50 frases del conjunto, 18 fueron generadas correctamente, es decir, de manera idéntica a las frases reales. Esto significa que el modelo generó un total de 27 frases con algún tipo de error. A continuación se enumeran los errores principales que estas presentan:

- **Errores en los tiempos verbales**: se han identificado 9 frases en las que el único error se relaciona con la conjugación del tiempo verbal. En la Tabla 5.2 se presentan tres ejemplos de frases con este tipo de error. Se observa que son casos en los que, a partir de las palabras clave introducidas, resulta muy difícil determinar el tiempo verbal correcto de la frase. El modelo tiende a conjugar las frases en presente, ya que es el tiempo verbal más común, aunque en algunas ocasiones las frases deberían estar en pasado o en futuro. Este error también es frecuente en las generaciones de frases realizadas por personas. La ausencia de pistas claras sobre el tiempo verbal en las palabras clave puede llevar a asumir que el tiempo verbal de la frase debe ser presente.

- **Falta de preposición**: en 8 frases, los errores se deben a la falta o confusión de alguna preposición. Se detallan ejemplos de este caso en la Tabla 5.3.

Tabla 5.2: Frases generadas por Llama3 con errores en el tiempo verbal

Lemmas	Frase generada	Frase real
no, lo, cambiar	No lo cambio	No lo cambiaré
niños, se, reír	Los niños se ríen	Los niños se reían
perra, tener, cachorros	La perra tiene cachorros	La perra tuvo cachorros

Tabla 5.3: Frases generadas por Llama3 con errores en preposiciones

Lemmas	Frase generada	Frase real
yo, dormir, noche	Yo duermo la noche	Yo duermo por la noche
Blancanieves, correr, bosque	Blancanieves corre por el bosque	Blancanieves corre al bosque
Juan, se, encontrar, un, mago	Juan se encontró un mago	Juan se encontró con un mago

- **Confusión en el sujeto**: un total de 4 frases presentan errores solamente en el sujeto. Los ejemplos de este caso se presentan en la Tabla 5.4. Estos errores incluyen la falta del artículo del sujeto debido a que se interpreta el sustantivo como nombre propio.

Tabla 5.4: Frases generadas por Llama3 con errores en el sujeto

Lemmas	Frase generada	Frase real
Ratita, comprar, un, lazo	Ratita compra un lazo	La ratita compra un lazo
papá, abrir, puerta, de, coche, con, llave	Papá abre la puerta del coche con la llave	El papá abre la puerta del coche con la llave
Ratita, barrer, su, casita	Ratita barre su casita	La ratita barre su casita

- **Error en el número**: solo se ha identificado un caso en el que el error radica en el número gramatical. En este caso las palabras clave son " niño, jugar, piscina ´´. A partir de estas, el modelo generó " El niño juega en la piscina ´´ en lugar de " Los niños juegan en la piscina ´´, que era la frase real. Dado que las palabras clave no especificaban el número, es comprensible que el modelo asumiera el singular. Este error también fue cometido por todos los participantes humanos en el experimento.

Las 10 frases restantes presentan varios de los errores mencionados u otros problemas sintácticos. En la tabla 5.5 se presentan tres ejemplos de frases con distintos errores.

Tabla 5.5: Frases generadas por Llama3 con varios errores

Lemmas	Frase generada	Frase real
este, ser, Caperuza, Roja	Este ser Caperuza Roja	Esta es Caperuza Roja
Reyes, mago, ir, camello	3 Reyes magos irían al camello	Los Reyes Magos van en camello
Caperucita, abuela, ser, feliz	Caperucita es feliz con su abuela	Caperucita y la abuela fueron felices

CAPÍTULO 6
Conclusiones

En este trabajo se ha explorado la posibilidad de adaptar modelos de lenguaje grandes a tareas dentro del ámbito de los sistemas de comunicación aumentativos y alternativos. Se han entrenado dos modelos, Gemma y Llama3, utilizando un conjunto de datos en español y en inglés, con el objetivo de generar frases gramaticalmente correctas a partir de una serie de listas de palabras clave o lemas. Estas listas de palabras clave simulan las entradas que los usuarios introducirían a través de la interfaz de un comunicador electrónico.

El modelo Llama3 ha demostrado un rendimiento positivo en este contexto, logrando generar frases bien estructuradas y mostrando resultados muy prometedores y competitivos. Este modelo ha superado a Gemma en términos de precisión y coherencia gramatical. Por su parte, el modelo Gemma ha presentado un rendimiento inferior, lo que sugiere que este modelo tiene una capacidad de adaptación menos efectiva que Llama3.

Aunque los resultados obtenidos con Llama3 han sido satisfactorios, se ha observado que el tamaño del conjunto de datos ha impactado el rendimiento de ambos modelos. A pesar de esta limitación, Llama3 ha demostrado una notable capacidad para adaptarse y generar resultados de calidad.

Estos resultados muestran que los modelos de lenguaje grandes tienen un gran potencial en el campo de la comunicación aumentativa y alternativa. La capacidad de adaptación y el rendimiento prometedor de Llama3 indican que, con conjuntos de datos más extensos y representativos, estos modelos podrían ofrecer soluciones aún más efectivas para mejorar la comunicación de los usuarios con necesidades específicas.

6.1 Objetivos cumplidos

A lo largo del desarrollo de este trabajo, se han cumplido satisfactoriamente los objetivos propuestos inicialmente:

1. Se ha realizado una investigación sobre diferentes tipos de SAAC y cómo aplicar herramientas de Aprendizaje Automático e Inteligencia Artificial para mejorar su uso.

2. Se han adaptado dos modelos de lenguaje grandes de actualidad, Gemma y Llama3, para la tarea de generación de frases sintácticamente correctas a partir de palabras clave para su uso en comunicadores electrónicos. Tras la realización de los experimentos, se determinó que el modelo Llama3 ofrece mejores resultados que Gemma. Los resultados obtenidos resultan prometedores.

3. Se ha realizado una comparación de los resultados obtenidos con el modelo Llama3 frente al modelo GPT-4 y el comunicador AsTeRISCS Grid. Aunque el modelo desarrollado en este trabajo no supera el rendimiento de GPT-4, demuestra un rendimiento significativamente superior al de AsTeRISCS Grid, lo que muestra que puede suponer un avance importante en este sector.

6.2 Propuesta de trabajo futuro

Aunque los resultados en este trabajo son satisfactorios, se han identificado algunas áreas de mejora que no han podido ser abordadas debido a limitaciones de tiempo, recursos computacionales y costos económicos. Así, para continuar la investicación y mejorar los resultados, se proponen las siguientes líneas de trabajo futuro:

- **Realización de más repeticiones de experimentos**: resultaría interesante realizar más repeticiones de los experimentos realizados para poder obtener resultados más robustos, ya que las variaciones en el entrenamiento pueden influir significativamente en el rendimiento de los modelos. En este proyecto, las limitaciones de tiempo, recursos computacionales y costos económicos han impedido realizar un número más alto de experimentos.
- **Obtención de conjuntos de datos más grandes**: la aplicación de los conjuntos de datos utilizados podría tener un gran impacto positivo en el rendimiento de los modelos de lenguaje. Los conjuntos de datos de los que se disponían eran relativamente pequeños. Un conjunto más grande y diverso permitiría entrenar a los modelos de manera más completa, mejorando su capacidad de generalización y su habilidad para generar frases precisas y apropiadas al contexto.

6.3 Legado

Este trabajo supone una contribución al campo de los Sistemas de Comunicación Aumentativa y Alternativa (SAAC). Se ha demostrado que la adaptación de modelos de lenguaje grandes para la generación de frases a partir de palabras clave permite mejorar significativamente la eficacia de estas herramientas, facilitando una comunicación más natural y coherente para las personas que tienen dificultades en este área.

El código y los datos utilizados están disponibles en el repositorio de GitHub, permitiendo así a otros investigadores reproducir el análisis y trabajo realizado. Toda la documentación relevante está referenciada en la bibliografía.

6.4 Relación del trabajo desarrollado con los estudios cursados

El trabajo desarrollado en este proyecto está estrechamente relacionado con los estudios cursados e integra diferentes competencias y conocimientos adquiridos a lo largo de este.

En primer lugar, la obtención y transformación de diferentes conjuntos de datos, que es una tarea fundamental de la Ciencia de Datos, han sido necesarios para conseguir los datos para el entrenamiento de los modelos.

El entrenamiento de modelos de lenguaje grandes, que forma parte del campo del Procesamiento de Lenguaje Natural, se ha basado en conceptos estudiados en la carrera, por ejemplo el *fine-tuning* y la técnica de *cross-validation*.

Se ha profundizado también en técnicas avanzadas como la adaptacion de modelos mediante LoRA, una técnica no vista en profundidad durante la carrera pero que ha sido fundamental para el entrenamiento de los modelos en este proyecto.

Por último, el cálculo de métricas y creación de tablas y gráficas han sido cruciales para la interpretación de resultados. Los conceptos de visualización de datos aprendidos durante los estudios han resultado muy útiles para presentar de manera clara los resultados obtenidos.

Para ello han sido muy utiles los conceptos de visualización de datos aprendidos durante los estudios.

En resumen, este proyecto ha permitido aplicar de manera práctica los conocimientos adquiridos en el grado de Ciencia de Datos, desde la preparación de los datos y el entrenamiento de modelos hasta la evaluación y visualización de resultados.

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] H. D. Block. The perceptron: A model for brain functioning. i. *Rev. Mod. Phys.*, 34:123–135, Jan 1962.

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

[8] 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.

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

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

[11] 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.

[12] Google Cloud. Evaluate custom models with automl translation, 2024. Accessed: 2024-08-10.

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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

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

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

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

[27] 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.

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

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

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

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

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

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

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

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

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

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

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

[39] 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.

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

[41] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023.

[42] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

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

[44] 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.

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

[46] 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.