VIP Cheatsheet: Transformers y Grandes Modelos de Lenguaje

Afshine Amidi y Shervine Amidi
Traducido por Steven Van Vaerenbergh y Lara Lloret Iglesias

24 de marzo de 2025

Esta VIP cheatsheet ofrece un resumen del contenido del libro «Super Study Guide: Transformers & Large Language Models», que incluye ∼600 ilustraciones a lo largo de 250 páginas y profundiza en los siguientes conceptos.
Puedes encontrar más detalles en `https://superstudy.guide`.

1 Fundamentos

1.1 Tokens

❒ **Definición** – Un *token* es una unidad indivisible de texto, como una palabra, subpalabra o carácter, y forma parte de un vocabulario predefinido.

Nota: El token desconocido `[UNK]` *representa fragmentos de texto no reconocidos, mientras que el token de relleno* `[PAD]` *se utiliza para completar posiciones vacías y garantizar una longitud uniforme para las secuencias de entrada.*

❒ **Tokenizador** – Un *tokenizador* T divide el texto en tokens con un nivel de granularidad arbitrario.

Aquí están los principales tipos de tokenizadores:

Tipo	Beneficios	Desventajas	Ilustración
Palabra	• Fácil de interpretar • Secuencias cortas	• Vocabulario de gran tamaño • No se manejan las variaciones de las palabras	`oso` `de` `peluche`
Subpalabra	• Se aprovechan las raíces de las palabras • Embeddings intuitivos	• Secuencias de mayor longitud • Tokenización más compleja	`oso` `de` `pelu` `##che`
Caracter Byte	• Las palabras fuera del vocabulario no son un problema • Vocabulario de pequeño tamaño	• Secuencias de longitud mucho mayor • Los patrones son difíciles de interpretar debido a que son de nivel demasiadobajo	`o` `s` `o` ` ` `d` `e` ` ` `p` `e` `l` `u` `c` `h` `e`

Nota: Byte-Pair Encoding (BPE) y Unigram son tokenizadores de subpalabras ampliamente utilizados.

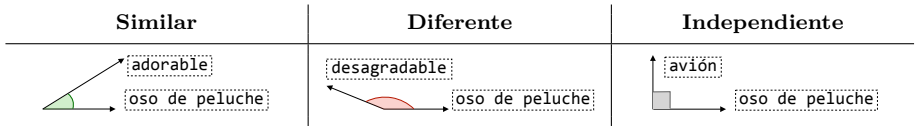
1.2 Embeddings

❒ **Definición** – Un *embedding* es una representación numérica de un elemento (por ejemplo, un token, una oración) y se caracteriza por un vector $x \in \mathbb{R}^n$.

❒ **Similitud** – La *similitud del coseno* entre dos tokens t_1 y t_2 se cuantifica de la siguiente manera:

$$\boxed{\text{similitud}(t_1, t_2) = \frac{t_1 \cdot t_2}{||t_1||\ ||t_2||} = \cos(\theta) \in [-1, 1]}$$

El ángulo θ caracteriza la similitud entre los dos tokens:

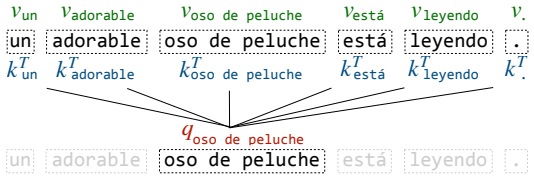


Nota: Vecinos más cercanos aproximados y hashing sensible a la localización son métodos que aproximan la operación de similitud de manera eficiente sobre grandes bases de datos.

2 Transformers

2.1 Atención

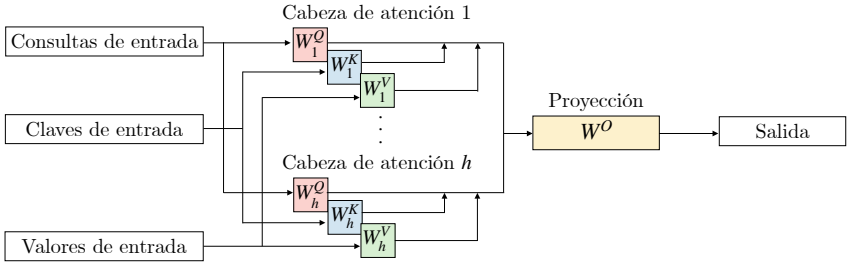
❒ **Fórmula** – Dada una *consulta* q, queremos saber a qué *clave* k debe prestar «atención» la consulta con respecto al *valor* asociado v.



La atención puede ser calculada de manera eficiente utilizando las matrices Q, K, V que contienen las consultas q, las claves k y los valores v respectivamente, junto con la dimensión d_k de las claves:

$$\boxed{\text{atención} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V}$$

❒ **MHA** – Una capa de *atención multi-cabeza* (***M**ulti-**H**ead **A**ttention*, MHA) realiza cálculos de atención a través de múltiples cabezas, y luego proyecta el resultado en el espacio de salida.

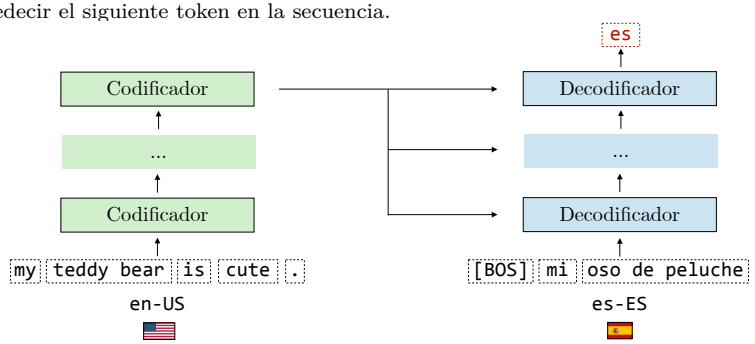


Está compuesto por h cabezas de atención, así como por las matrices W^Q, W^K, W^V que proyectan la entrada para obtener las consultas Q, las claves K y los valores V. La proyección se realiza utilizando la matriz W^O.

Nota: atención por consultas agrupadas (Grouped-Query Attention, GQA) y atención por múltiples consultas (Multi-Query Attention, MQA) son variaciones de MHA que reducen la sobrecarga computacional al compartir las claves y los valores entre las cabezas de atención.

2.2 Arquitectura

❐ **Resumen** – Un *Transformer* es un modelo de referencia que se basa en el mecanismo de auto-atención y está compuesto por codificadores y decodificadores. Los codificadores calculan representaciones significativas de la entrada, que luego son utilizadas por los decodificadores para predecir el siguiente token en la secuencia.

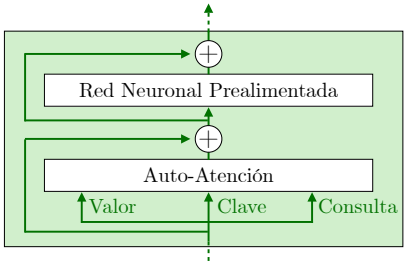


Nota: Aunque el Transformer se propuso inicialmente como un modelo para tareas de traducción, ahora se usa ampliamente en muchas otras aplicaciones.

❐ **Componentes** – El *codificador* y el *decodificador* son dos componentes fundamentales del Transformer y tienen roles diferentes:

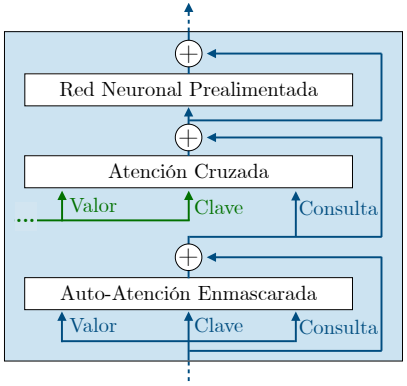
Codificador

Los embeddings codificados encapsulan el significado de la entrada.



Decodificador

Los embeddings decodificados encapsulan el significado tanto de la entrada como de la salida predicha hasta el momento

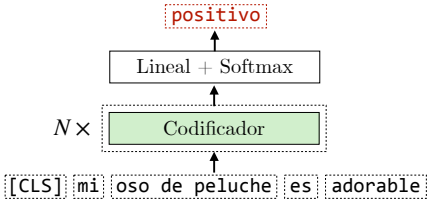


❐ **Embeddings de posición** – Los embeddings de posición informan sobre la posición del token en la oración y tienen la misma dimensión que los embeddings de los tokens. Pueden ser definidos de manera arbitraria o aprendidos a partir de los datos.

Nota: Los embeddings de posición rotatoria (Rotary Position Embeddings, RoPE) son una variación popular y eficiente que rota los vectores de consulta y clave para incorporar información de posición relativa.

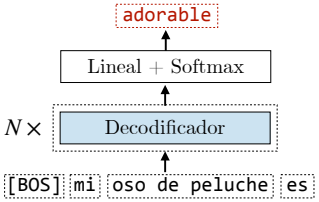
2.3 Variantes

❐ **Solo codificador** – BERT (***B**idirectional **E**ncoder **R**epresentations from **T**ransformers*) es un modelo basado en Transformers compuesto por una pila de codificadores que toman texto como entrada y generan embeddings significativos, los cuales pueden ser utilizados posteriormente en tareas de clasificación.



Se añade un token `[CLS]` al principio de la secuencia para capturar el significado de la oración. Su embedding codificado se utiliza a menudo en tareas posteriores, como la extracción de sentimientos.

❐ **Solo decodificador** – GPT (***G**enerative **P**re-trained **T**ransformer*) es un modelo basado en Transformers autorregresivos que está compuesto por una pila de decodificadores. A diferencia de BERT y sus derivados, GPT trata todos los problemas como problemas de texto a texto.



La mayoría de los grandes modelos de lenguaje actuales se basan en una arquitectura de solo decodificador, como los diferentes GPTs, LLaMA, Mistral, Gemma, DeepSeek, etc.

Nota: Los modelos codificador-decodificador, como T5, también son autoregresivos y comparten muchas características con los modelos formados solo por decodificadores.

2.4 Optimizaciones

❐ **Aproximación de la Atención** – Los cálculos de atención tienen una complejidad de $\mathcal{O}(n^2)$, lo que puede resultar costoso a medida que aumenta la longitud n de la secuencia. Existen dos métodos principales de aproximación:

- *Dispersión*: La auto-atención no ocurre a lo largo de toda la secuencia, sino sólo entre los tokens más relevantes.

- *Bajo rango*: La fórmula de atención se simplifica como el producto de matrices de bajo rango, lo que reduce la carga computacional.

❒ **Atención Flash** – La *atención flash* es un método exacto que optimiza los cálculos de atención aprovechando de manera inteligente el hardware de la GPU, utilizando la rápida *memoria estática de acceso aleatorio* (***S**tatic **R**andom-**A**ccess **M**emory*, SRAM) para las operaciones matriciales antes de escribir los resultados en la más lenta *memoria de alta ancho de banda* (***H**igh **B**andwidth **M**emory*, HBM).

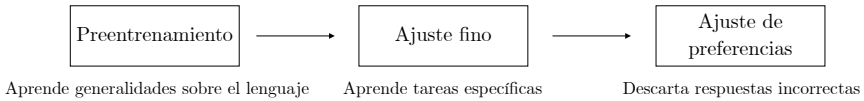
Nota: En la práctica, esto reduce el uso de memoria y acelera los cálculos.

3 Grandes modelos de lenguaje

3.1 Resumen

❒ **Definición** – Un *gran modelo de lenguaje* (***L**arge **L**anguage **M**odel*, LLM) es un modelo basado en Transformer con potentes capacidades de procesamiento de lenguaje natural. Se considera «grande» en el sentido de que típicamente contiene miles de millones de parámetros.

❒ **Ciclo de vida** – Un LLM se entrena en 3 etapas: preentrenamiento, ajuste fino y ajuste de preferencias.

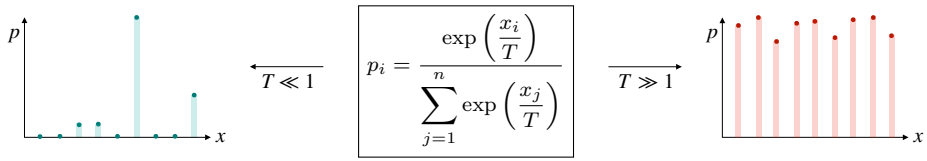


El ajuste fino y el ajuste de preferencias son enfoques posteriores al entrenamiento que buscan alinear el modelo para realizar tareas específicas.

3.2 Prompting

❒ **Longitud del contexto** – La *longitud del contexto* de un modelo es el número máximo de tokens que se pueden incluir en la entrada. Normalmente, varía desde decenas de miles hasta millones de tokens.

❒ **Muestreo de decodificación** – Las predicciones de tokens se muestrean a partir de la distribución de probabilidad predicha p_i, la cual es controlada por el hiperparámetro de temperatura T.



Nota: Las temperaturas altas generan salidas más creativas, mientras que las temperaturas bajas generan salidas más deterministas.

❒ **Cadena de pensamiento** – Una *cadena de pensamientos* (***C**hain-**o**f-**T**hought*, CoT) es un proceso de razonamiento en el que el modelo descompone un problema complejo en una serie de pasos intermedios. Esto ayuda al modelo a generar la respuesta final correcta. Un *árbol de pensamientos* (***T**ree **o**f **T**houghts*, ToT) es una versión más avanzada de la CoT.

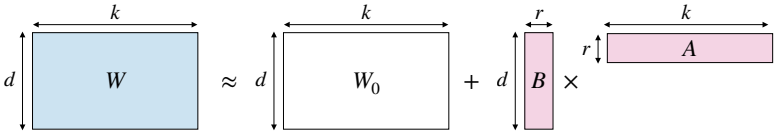
Nota: La auto-consistencia es un método que agrega respuestas a lo largo de los caminos de razonamiento de la CoT.

3.3 Ajuste fino

❒ **SFT** – El *ajuste fino supervisado* (***S**upervised **F**ine**T**uning*, SFT) es un enfoque posterior al entrenamiento que alinea el comportamiento del modelo con una tarea final. Se basa en pares de entrada-salida de alta calidad alineados con la tarea.

Nota: Si los datos del SFT son sobre instrucciones, entonces este paso se llama «ajuste de instrucciones».

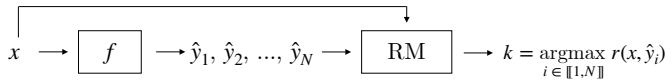
❒ **PEFT** – El *ajuste fino eficiente de parámetros* (***P**arameter-**E**fficient **F**ine**T**uning*, PEFT) es una categoría de métodos utilizados para llevar a cabo SFT de manera eficiente. En particular, la *adaptación de bajo rango* (***L**ow-**R**ank **A**daptation*, LoRA) aproxima los pesos aprendibles W fijando W_0 y aprendiendo matrices de bajo rango A y B en su lugar:



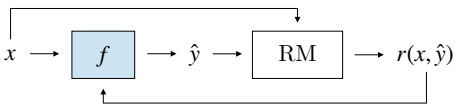
Nota: Otras técnicas PEFT incluyen ajuste de prefijos e inserción de capas adaptadoras.

3.4 Ajuste de preferencia

❒ **Modelo de recompensas** – Un *modelo de recompensas* (***R**eward **M**odel*, RM) es un modelo que predice cómo de bien una cierta salida $\hat{y}$ se alinea con el comportamiento deseado dado la entrada x. El muestreo ***B**est-**o**f-**N*** (BoN), también llamado muestreo por rechazo, es un método que utiliza un modelo de recompensas para seleccionar la mejor respuesta entre N generaciones.



❒ **Aprendizaje por refuerzo** – El *aprendizaje por refuerzo* (***R**einforcement **L**earning*, RL) es un enfoque que aprovecha el RM y actualiza el modelo f en función de las recompensas recibidas por sus salidas generadas. Si el RM se basa en las preferencias humanas, este proceso se llama *aprendizaje por refuerzo a partir de retroalimentación humana* (***R**einforcement **L**earning from **H**uman **F**eedback*, RLHF).

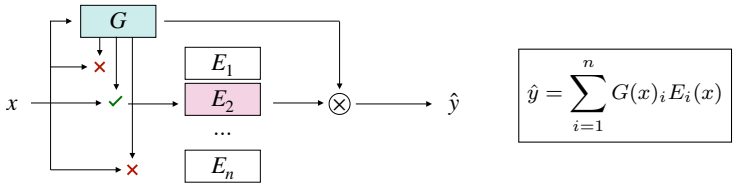


La *optimización de políticas próximas* (***P**roximal **P**olicy **O**ptimization*, PPO) es un algoritmo popular de RL que incentiva recompensas más altas mientras mantiene el modelo cercano al modelo base para evitar el hackeo de recompensas.

Nota: También existen enfoques supervisados, como optimización directa de preferencias (Direct Preference Optimization, DPO), que combinan RM y RL en un solo paso supervisado.

3.5 Optimizaciones

❒ **Mezcla de expertos** – Una *mezcla de expertos* (***M**ixture **o**f **E**xperts*, MoE) es un modelo que activa solo una porción de sus neuronas en el momento de la inferencia. Está basado en una puerta G y expertos $E_1, ..., E_n$.



$$\hat{y} = \sum_{i=1}^{n} G(x)_i E_i(x)$$

Los LLMs basados en MoE utilizan este mecanismo de puerta en sus Redes Neuronales Prealimentadas.

Nota: Entrenar un LLM basado en MoE es un desafío considerable, como se menciona en el artículo de LLaMA, cuyos autores decidieron no usar esta arquitectura a pesar de su eficiencia en el tiempo de inferencia.

❒ **Destilación** – La *destilación* es un proceso en el que un modelo estudiante (pequeño) S se entrena con las predicciones de un modelo maestro (grande) T. Se entrena utilizando la divergencia KL como función de pérdida:

$$\text{KL}(\hat{y}_T || \hat{y}_S) = \sum_i \hat{y}_T^{(i)} \log\left(\frac{\hat{y}_T^{(i)}}{\hat{y}_S^{(i)}}\right)$$

Nota: Las etiquetas de entrenamiento se consideran etiquetas suaves ya que representan las probabilidades de las clases.

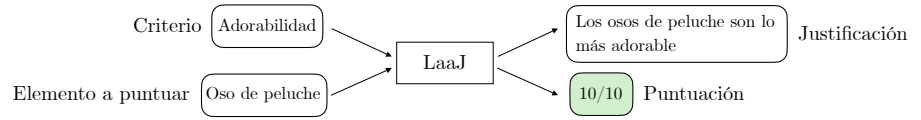
❒ **Cuantización** – La *cuantización de modelos* es un tipo de técnicas que reducen la precisión de los pesos del modelo, limitando su impacto en el rendimiento del modelo resultante. Como resultado, esto reduce el uso de memoria del modelo y acelera su inferencia.

Nota: QLoRA es una variante cuantizada comúnmente utilizada de LoRA.

4 Aplicaciones

4.1 LLM como juez

❒ **Definición** – *LLM como juez* (***LLM-as-a-Judge***, LaaJ) es un método que utiliza un LLM para puntuar las salidas dadas según ciertos criterios proporcionados. Cabe destacar que también es capaz de generar una justificación para su puntuación, lo que ayuda con la interpretabilidad.



A diferencia de las métricas de la era previa a los LLMs, como ROUGE (***R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation*), LaaJ no necesita ningún texto de referencia, lo que lo hace adecuado para evaluar cualquier tipo de tarea. En particular, LaaJ presenta una fuerte correlación con las calificaciones humanas cuando se basa en un modelo grande y potente (por ejemplo, GPT-4), ya que requiere capacidades de razonamiento para obtener buenos resultados.

Nota: LaaJ es útil para realizar rondas rápidas de evaluación, pero es importante monitorear la alineación entre los resultados de LaaJ y las evaluaciones humanas para asegurarse de que no haya divergencias.

❒ **Sesgos habituales** – Los modelos LaaJ pueden presentar los siguientes sesgos:

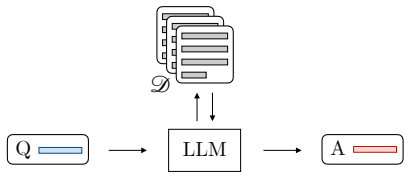
	Sesgo de posición	**Sesgo de verbosidad**	**Sesgo de auto-refuerzo**
Problema	Favorece la primera posición en comparaciones por pares	Favorece contenido más verboso	Favorece salidas generadas por sí mismos
Solución	Promediar la métrica en posiciones aleatorias	Añadir una penalización según la longitud de la salida	Usar un evaluador construido a partir de un modelo base diferente

Una solución a estos problemas puede ser ajustar un LaaJ personalizado, pero esto requiere mucho esfuerzo.

Nota: La lista de sesgos no es exhaustiva.

4.2 RAG

❒ **Definición** – La *generación aumentada por recuperación* (***R**etrieval-**A**ugmented **G**eneration*, RAG) es un método que permite a los LLMs acceder a conocimiento externo relevante para responder a una pregunta dada. Esto es particularmente útil si se desea incorporar información más allá de la fecha de corte del conocimiento con el que ha sido preentrenado el LLM.



Data una base de conocimiento $\mathscr{D}$ y una pregunta, un **R**ecuperador obtiene los documentos más importantes y luego **A**umenta el prompt (la instrucción) con la información relevante antes de **G**enerar la respuesta.

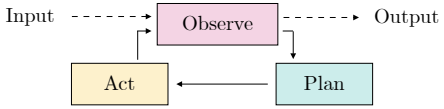
Nota: La etapa de recuperación normalmente se basa en embeddings de modelos de tipo solo codificadores.

❒ **Hiperparámetros** – La base de conocimiento $\mathscr{D}$ se inicializa dividiendo los documentos en fragmentos de tamaño n_c y convirtiéndolos en vectores de tamaño $\mathbb{R}^d$.

4.3 Agentes

❒ **Definición** – Un *agente* es un sistema que persigue objetivos y completa tareas de manera autónoma en nombre del usuario. Para ello, puede utilizar diferentes cadenas de llamadas a LLMs.

❒ **ReAct** – ReAct, abreviatura de **Re**ason (razonar) + **Act** (actuar), es un marco que permite utilizar múltiples cadenas de llamadas a LLMs para llevar a cabo tareas complejas:



Este marco se compone de los siguientes pasos:

- *Observar*: Sintetizar las acciones previas y expresar explícitamente lo que se conoce hasta el momento.
- *Planificar*: Detallar las tareas que deben realizarse y las herramientas que se deben utilizar.
- *Actuar*: Ejecutar una acción a través de una API o buscar información relevante en una base de conocimiento.

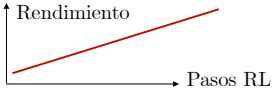
Nota: Evaluar un sistema agente es un desafío. Sin embargo, es posible hacerlo tanto a nivel de cada componente, mediante la evaluación de sus entradas y salidas locales, como a nivel del sistema completo, analizando las cadenas de llamadas.

4.4 Modelos de razonamiento

❒ **Definition** – Un *modelo de razonamiento* es un modelo que se basa en cadenas de razonamiento tipo CoT para resolver tareas más complejas en matemáticas, programación y lógica. Algunos ejemplos de modelos de razonamiento son los diferentes o de OpenAI, DeepSeek-R1 y Gemini Flash Thinking de Google.

Nota: DeepSeek-R1 muestra explícitamente su cadena de razonamiento entre las etiquetas `<think>`.

❒ **Escalado** – Se utilizan dos tipos de métodos de escalado para mejorar las capacidades de razonamiento:

	Descripción	**Ilustración**
Escalado durante el entrenamiento	Ejecuta RL durante más tiempo para permitir que el modelo aprenda a generar cadenas de razonamiento al estilo CoT antes de dar una respuesta.	
Escalado durante el test	Deja que el modelo reflexione más tiempo antes de dar una respuesta, utilizando palabras clave que gestionen el presupuesto, como «`Esperar`».