Facsímil 03

Arquitecturas y modelos

Capítulo 02

Transformer por dentro: de texto a tensores y atención

686f6c61 · 2026

Capítulo 02: Transformer por dentro: de texto a tensores y atención

El momento en que el texto deja de ser texto

En el capítulo anterior vimos el LLM desde fuera: contexto, parámetros, logits, escala y sistema completo. Ahora vamos a abrir una puerta más pequeña y más concreta: ¿qué ocurre justo después de tokenizar?

Para nosotros, una frase sigue siendo una frase. Para el modelo, ya no. La frase se convierte en una tabla de números. Esa tabla tiene filas, columnas, tamaño, memoria y operaciones permitidas. Cuando un modelo «lee» una oración, en realidad está multiplicando matrices, sumando vectores y calculando pesos de atención.

Este capítulo va de ese paso. No vamos todavía a exprimir Q, K, V, máscara causal y softmax con todo detalle; eso será el capítulo 03. Aquí queremos entender la carretera principal: texto → tokens → tensores → atención → representación contextual.

Qué no es un Transformer

Un Transformer no es una lista de reglas gramaticales. No tiene una tabla escrita a mano con «si aparece sujeto, busca verbo». Aprende transformaciones numéricas durante entrenamiento y las aplica a vectores.

Tampoco es una memoria que recorra el texto palabra por palabra como una persona leyendo en voz alta. Antes de los Transformers, muchas arquitecturas secuenciales procesaban una posición tras otra y llevaban un estado interno hacia delante.[1] Eso funcionaba, pero tenía un cuello de botella: si cada paso depende del paso anterior, paralelizar se vuelve más difícil.

Y no es «atención humana» en sentido psicológico. La palabra atención es útil, pero técnica: significa calcular pesos para mezclar información. Si un token atiende a otro, no está «pensando» en él. Está usando su vector para decidir cuánto de otro vector debe entrar en la nueva representación.

Qué sí es un Transformer

Un Transformer es una arquitectura de red neuronal diseñada para procesar secuencias mediante atención y operaciones paralelas sobre matrices.[2]

La idea central es sencilla de decir y potente de ejecutar: cada token empieza con un vector, y cada capa transforma ese vector usando información de otros tokens. Al final, el vector de cada posición ya no representa solo ese token aislado, sino ese token **dentro de su contexto**.

Por ejemplo, en estas dos frases:

> Me senté en el banco
> El banco aprobó el préstamo

El token «banco» podría empezar con un embedding parecido en ambos casos. Pero tras pasar por atención, su representación debería cambiar: en la primera frase se acerca a asiento; en la segunda, a entidad financiera. El Transformer no recibe una definición explícita: aprende a usar el contexto.

La atención moderna no apareció de la nada. Antes del Transformer, Bahdanau, Cho y Bengio ya habían mostrado que un modelo de traducción podía aprender a alinear partes de una frase origen con partes de una frase destino.[3] Luong, Pham y Manning estudiaron variantes eficaces de atención para traducción neuronal.[4] El salto del Transformer fue llevar esa intuición al centro de la arquitectura.

El paper original: por qué fue tan raro y tan importante

El paper **“Attention Is All You Need”** se envió a arXiv el 12 de junio de 2017 y fue presentado en NeurIPS 2017.[5] Lo firmaron ocho autores: **Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser e Illia Polosukhin**.

La propuesta era incómoda para su época porque quitaba dos muletas habituales en modelos de secuencia: recurrencia y convolución. Hasta entonces, para traducir o generar secuencias era normal procesar tokens en orden, mantener un estado oculto y apoyarse en atención como complemento. El Transformer dijo: probemos a construir toda la arquitectura alrededor de atención.

El paper original no nació pensando en chats modernos. Su tarea principal era **traducción automática**. La entrada era una frase en un idioma y la salida una frase en otro. La arquitectura completa tenía dos mitades:

PARTE DEL PAPER	QUÉ HACÍA	CÓMO LO LEEMOS HOY
Encoder	Leía toda la frase de entrada y construía representaciones contextuales.	Se parece a modelos de comprensión como BERT.
Decoder	Generaba la frase de salida token a token mirando lo ya generado.	Se parece a la familia GPT cuando hablamos de generación autoregresiva.
Atención encoder-decoder	Permitía que cada token generado mirase partes relevantes de la entrada.	Es una forma temprana de conectar generación con contexto fuente.
Multi-head attention	Usaba varias atenciones en paralelo para capturar relaciones distintas.	En el capítulo 03 veremos por qué varias cabezas miran patrones distintos.
Feed-forward, residual y normalización	Completaba cada bloque para hacerlo entrenable y expresivo.	En el capítulo 04 lo abriremos despacio.

Esa tabla esconde una distinción que conviene nombrar: no toda la atención es igual. Cuando un token mira a otros de **su misma secuencia** (los de su propio contexto), hablamos de *self-attention*: es la que usan tanto el encoder como, en generación, la familia GPT. Cuando los tokens que se generan miran a los de **otra secuencia** (por ejemplo, la frase de entrada que se está traduciendo), hablamos de *cross-attention*: es el puente encoder-decoder del paper original. Los LLMs decoder-only modernos son, en esencia, una pila de self-attention; la cross-attention reaparece en arquitecturas encoder-decoder y en algunos modelos multimodales, donde el texto atiende a una imagen.

La idea que debemos llevarnos no es «inventaron los LLMs de golpe». Es más precisa: inventaron una arquitectura que hacía mucho más natural entrenar modelos grandes de secuencias con paralelismo y atención directa. Los LLMs modernos heredan esa carretera y la llevan a otra escala.

De tokens a tensores

Empezamos con texto:

```
El banco aprobó el préstamo
```

El tokenizador lo convierte en identificadores. No importa ahora si salen cinco o seis tokens; usemos cinco para entender el mecanismo:

$$I = [i_1, i_2, i_3, i_4, i_5]$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
I	Secuencia de identificadores de token.	$[42, 981, 774, 42, 1503]$.
i_t	Identificador del token en la posición t.	$i_2 = 981$ para «banco».
t	Posición dentro de la secuencia.	$t = 1$ es el primer token.

Cada identificador selecciona una fila de la matriz de embeddings:

$$E \in \mathbb{R}^{V \times d_{\text{model}}}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
E	Matriz de embeddings aprendida.	Una tabla gigante de vectores.
V	Tamaño del vocabulario.	100 000 tokens posibles.
d_{model}	Dimensión interna del modelo.	4096 números por token.
$\mathbb{R}$	Conjunto de números reales.	Valores como 0,37 o -1,12.

Después de buscar esas filas, obtenemos una matriz:

$$X = E[I] + P_{1:n}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
X	Tensor de entrada al Transformer.	Una matriz de 5×4096.
$E[I]$	Filas de embedding seleccionadas por los IDs.	Los vectores de «El», «banco», «aprobó»...
$P_{1:n}$	Información de posición para las n posiciones.	Permite distinguir «perro muerde hombre» de «hombre muerde perro».
n	Número de tokens de la secuencia.	$n = 5$.

De dónde sale la posición

El sumando $P_{1:n}$ merece una parada, porque la atención tiene un punto ciego: por sí sola **no sabe el orden**. Si solo sumáramos embeddings de token, «el perro muerde al hombre» y «el hombre muerde al perro» entrarían como el mismo conjunto de vectores, porque la atención mira a todos los tokens a la vez, sin noción de antes o después. La información de posición es lo que rompe ese empate. Hay tres formas habituales de generarla:

FORMA	CÓMO FUNCIONA	DÓNDE SE USA
Sinusoidal	A cada posición le asigna un patrón fijo de senos y cosenos de distintas frecuencias. No se aprende, se calcula. La propuso Vaswani en el paper original.	El Transformer original.
Aprendida	Una fila de embedding por posición, entrenada como cualquier otro parámetro. Simple, pero limitada a la longitud vista en entrenamiento.	BERT, GPT-2.
Rotatoria (RoPE)	Rota los vectores Q y K un ángulo proporcional a la posición, de modo que la atención depende de la *distancia relativa* entre dos tokens. Generaliza mejor a contextos largos.[6]	La mayoría de LLMs modernos.

La idea de fondo es la misma en las tres: que el vector de cada token lleve, además de «qué palabra soy», un «en qué lugar estoy». Sin eso, el Transformer no sería más que una bolsa de palabras muy cara.

Este punto es importante: el Transformer no recibe una frase como texto. Recibe un tensor. Esa forma de pensar (representar datos como vectores, matrices y transformaciones diferenciables) es la base común del *deep learning* moderno.[7] En la práctica, si procesamos una sola frase con 5 tokens y $d_{\text{model}} = 4$ para simplificar, podríamos imaginar algo así:

TOKEN	VECTOR INTERNO SIMPLIFICADO
El	$[0{,}20,\ 0{,}10,\ -0{,}30,\ 0{,}40]$
banco	$[0{,}80,\ -0{,}20,\ 0{,}10,\ 0{,}50]$
aprobó	$[0{,}10,\ 0{,}70,\ 0{,}20,\ -0{,}10]$
el	$[0{,}18,\ 0{,}08,\ -0{,}25,\ 0{,}36]$
préstamo	$[0{,}60,\ 0{,}40,\ 0{,}70,\ 0{,}20]$

En un modelo real no habría 4 columnas, sino cientos o miles. Pero la idea es la misma: una fila por token, una columna por dimensión interna.

La atención como mesa de mezclas

La atención responde a una pregunta: para actualizar la representación de este token, ¿cuánto debería mezclar información de los demás tokens?

En una capa de atención simplificada, el modelo calcula tres proyecciones desde X:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
Q	Consultas (*queries*): qué busca cada token.	«banco» pregunta qué contexto le da sentido.
K	Claves (*keys*): qué ofrece cada token para ser encontrado.	«préstamo» ofrece pista financiera.
V	Valores (*values*): información que se mezclará si recibe atención.	Vector que aporta contenido al resultado.
W_Q, W_K, W_V	Matrices aprendidas que proyectan X.	Parámetros entrenados.

¿Por qué tres proyecciones distintas y no usar X directamente? Porque lo que un token **busca** no tiene por qué ser lo mismo que lo que **ofrece** ni lo que **aporta**. Separar Q, K y V le da al modelo esa libertad: «banco» puede *buscar* contexto financiero (su Q), *ofrecerse* como lugar físico (su K) y *aportar* otro matiz al mezclarse (su V). Con un solo vector, esos tres papeles colapsarían en uno y la atención sería simétrica y pobre. Tres matrices aprendidas permiten además que la relación entre dos tokens no sea recíproca: que A mire mucho a B no obliga a que B mire mucho a A.

Después se calculan puntuaciones comparando consultas con claves:

$$S = \frac{QK^T}{d_k}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
S	Matriz de puntuaciones de atención.	Una tabla $n \times n$.
QK^T	Producto entre consultas y claves.	Mide afinidad entre posiciones.
d_k	Dimensión de las claves.	64, 128 u otra dimensión por cabeza.
$\sqrt{d_k}$	Escalado para estabilizar valores.	Si $d_k = 64$, dividimos por 8.

Ese $\sqrt{d_k}$ del denominador no es un capricho. El producto QK^T suma d_k términos: cuanto mayor es d_k, más grandes tienden a ser las puntuaciones. Y si las puntuaciones que entran al softmax son muy grandes, el softmax se **satura**, reparte casi toda la probabilidad a un único token y deja a los demás casi en cero. En esa zona los gradientes son minúsculos y el modelo aprende muy despacio. Dividir por $\sqrt{d_k}$ mantiene las puntuaciones en un rango sano, donde el softmax sigue siendo sensible y el entrenamiento estable. Es un detalle pequeño con un efecto grande, y por eso aparece tal cual en el paper original.

Convertimos esas puntuaciones en pesos con softmax:

$$A = \text{softmax}(S)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
A	Matriz de atención.	Cada fila suma 1.
softmax	Convierte puntuaciones en pesos positivos.	2,0 se vuelve más importante que 0,2.
A_{ij}	Peso con el que el token i mira al token j.	«banco» mira a «préstamo» con peso alto.

Por último, mezclamos los valores:

$$O = AV$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
O	Salida de atención.	Nueva representación contextual.
A	Pesos de mezcla.	Cuánto mirar a cada token.
V	Valores disponibles para mezclar.	Información que aporta cada posición.

Leído sin símbolos: cada token produce una pregunta, compara esa pregunta con las claves de todos los tokens, convierte las coincidencias en pesos y mezcla valores según esos pesos. Eso es atención.

Varias cabezas a la vez

Lo que acabamos de describir es **una** cabeza de atención, pero un Transformer real no usa una sola: usa varias en paralelo, lo que se llama *multi-head attention*. Cada cabeza tiene sus propias matrices W_Q, W_K, W_V y, por tanto, aprende a mirar un patrón distinto: una puede especializarse en la concordancia entre sujeto y verbo, otra en a qué nombre se refiere un pronombre, otra en la puntuación. Se calculan todas a la vez, se concatenan sus salidas y una última proyección las combina. La intuición: en vez de una sola mesa de mezclas, hay un panel de varias mesas mirando relaciones diferentes del mismo texto, y luego se juntan. Veremos por dentro cómo y por qué en el capítulo 3.

Cómo funcionaría una pasada completa

Ahora juntemos las piezas sin perdernos. Imagina que queremos generar una respuesta a partir del prompt:

```
Resume este contrato en tres puntos
```

Una pasada de inferencia, simplificada, seguiría estos pasos:

1. **Tokenizar.** El texto se trocea en IDs: el modelo no recibe caracteres, recibe números.
2. **Buscar embeddings.** Cada ID selecciona una fila de la matriz E.
3. **Añadir posición.** A cada vector se le suma o aplica información de orden para distinguir la primera posición de la segunda.
4. **Construir el tensor X.** La entrada queda como una matriz $n \times d_{\text{model}}$.
5. **Proyectar a Q, K, V.** Cada capa aprende tres formas de mirar la misma entrada: qué busca, qué ofrece y qué información puede mezclar.
6. **Calcular atención.** Se comparan consultas y claves, se aplica softmax y se obtiene una matriz de pesos.
7. **Mezclar valores.** Cada token recibe una combinación ponderada de información del contexto.
8. **Pasar por residual, normalización y MLP.** La representación se estabiliza y se transforma dentro del bloque.
9. **Repetir capas.** Un modelo real tiene muchas capas; cada una refina las representaciones.

10. **Proyectar a logits.** El último vector se convierte en puntuaciones para todos los tokens del vocabulario.
11. **Elegir el siguiente token.** El sistema aplica temperatura, `top_p`, `top_k` u otra política de generación.
12. **Volver a empezar.** El token elegido se añade al contexto y la siguiente pasada genera el siguiente token.

Si lo escribimos como pseudocódigo conceptual:

```
ids = tokenizador(texto)
X = embeddings(ids) + posicion(ids)

para cada bloque Transformer:
    Q = X W_Q
    K = X W_K
    V = X W_V
    A = softmax(Q K^T / sqrt(d_k))
    X = normalizacion(X + A V)
    X = normalizacion(X + MLP(X))

logits = X_ultimo W_vocab
siguiente_token = muestrear(logits)
```

Ese pseudocódigo no sirve para entrenar un modelo real, pero sí para orientarse. Si alguna vez te pierdes, vuelve a la pregunta: ¿en qué punto estoy? ¿Tokenizando, mezclando contexto, transformando representación o eligiendo el siguiente token?

Un ejemplo pequeño con números

Usemos solo tres tokens para mirar la fila de atención de «banco»:

```
El banco aprobó
```

Imagina que, después de proyectar consultas y claves, las puntuaciones de «banco» hacia cada token son:

TOKEN AL QUE MIRA «BANCO»	PUNTUACIÓN
El	0,2
banco	2,0
aprobó	1,0

Aplicamos softmax:

$$\text{softmax}([0{,}2,\ 2{,}0,\ 1{,}0]) \approx [0{,}108,\ 0{,}652,\ 0{,}240]$$

VALOR	SIGNIFICADO	LECTURA
0,108	Peso hacia «El».	Aporta poco.
0,652	Peso hacia «banco».	Mantiene mucho de sí mismo.
0,240	Peso hacia «aprobó».	Aporta contexto de acción.

Ahora supongamos valores simplificados de dos dimensiones:

TOKEN	VALOR V
El	$[0{,}1,\ 0{,}0]$
banco	$[1{,}0,\ 0{,}2]$
aprobó	$[0{,}4,\ 1{,}0]$

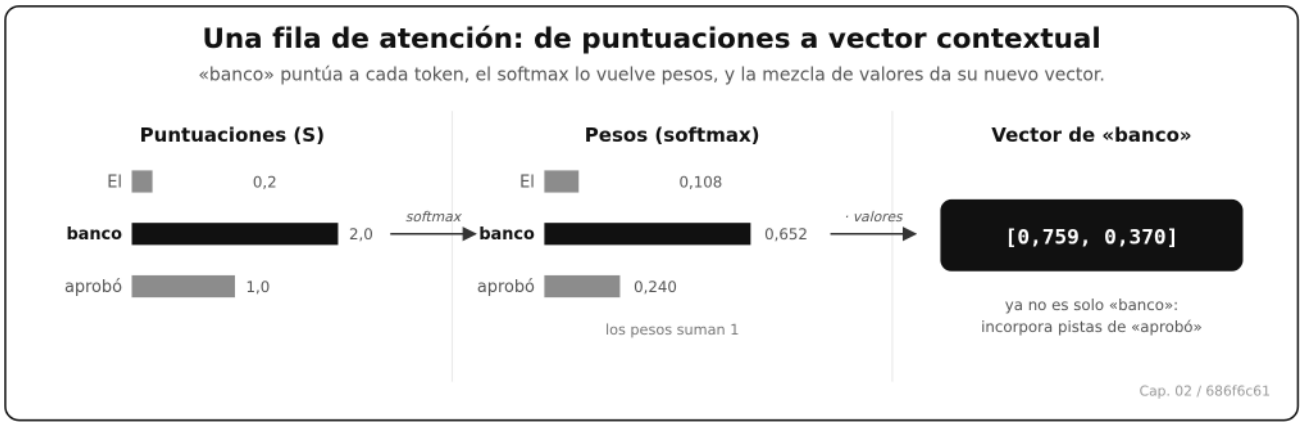
La nueva representación de «banco» es la mezcla ponderada:

$$o_{\text{banco}} = 0{,}108[0{,}1,\ 0{,}0] + 0{,}652[1{,}0,\ 0{,}2] + 0{,}240[0{,}4,\ 1{,}0]$$

Resultado:

$$o_{\text{banco}} \approx [0{,}759,\ 0{,}370]$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
o_{banco}	Vector contextualizado del token «banco».	Ya no es solo el embedding inicial.
$0{,}652\,[1{,}0,\ 0{,}2]$	Parte que conserva de sí mismo.	El token sigue siendo «banco».
$0{,}240\,[0{,}4,\ 1{,}0]$	Parte que recibe de «aprobó».	El verbo cambia el sentido posible.



Esto es pequeño, casi de juguete, pero contiene la intuición completa. La atención no reemplaza el token: lo contextualiza.

Modo ingeniero: la fila de atención en código

Ese cálculo a mano es, casi línea por línea, lo que hace una librería real. En `numpy` cabe en muy pocas líneas: puntuaciones, softmax estable y mezcla ponderada.

```
import numpy as np

def softmax(x):
    x = x - np.max(x)          # estabilidad: restar el máximo evita desbordes
    e = np.exp(x)
    return e / e.sum()

puntuaciones = np.array([0.2, 2.0, 1.0])        # banco -> [El, banco, aprobo]
valores = np.array([[0.1, 0.0],
                    [1.0, 0.2],
                    [0.4, 1.0]])

pesos = softmax(puntuaciones)                   # [0.108, 0.652, 0.240]
o_banco = pesos @ valores                       # mezcla ponderada
print(pesos.round(3), o_banco.round(3))         # [0.108 0.652 0.24] [0.759 0.37]
```

El `@` es el producto matriz-vector: multiplica cada peso por su fila de valores y los suma, justo lo que hicimos a mano. El único truco de ingeniería es el `x - np.max(x)` dentro del softmax: restar el máximo antes de exponenciar evita que `exp` desborde con puntuaciones grandes, sin cambiar el resultado. Esto es exactamente lo que practica el lab de este capítulo, pero con varios tokens y comprobando que los pesos suman 1.

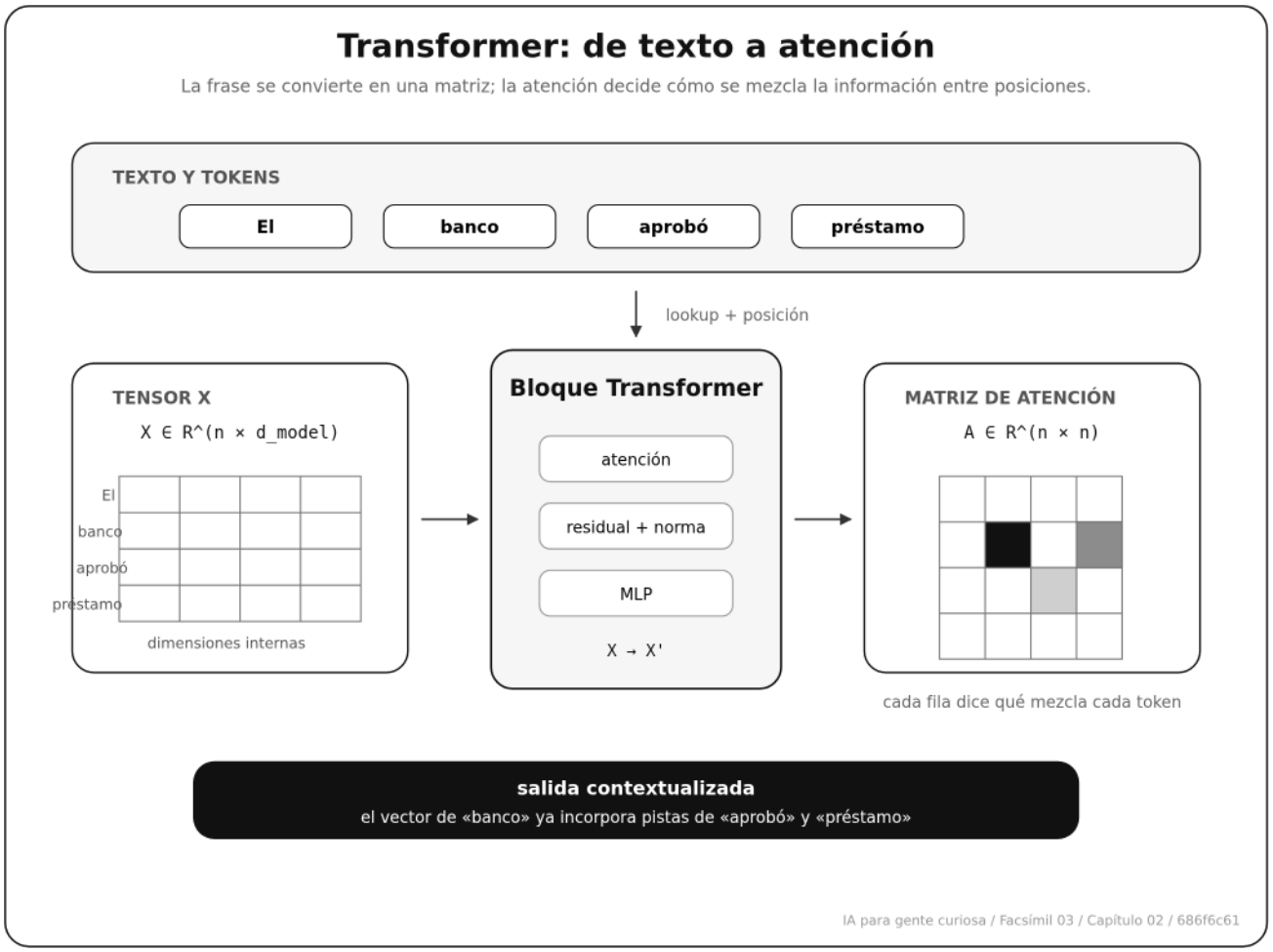
Dentro de un bloque Transformer hay más piezas además de atención. Las conexiones residuales permiten que la información original siga circulando a través de capas profundas, una idea que se volvió central tras ResNet.[8] La normalización por capa ayuda a estabilizar las activaciones dentro de la red.[9] En el capítulo 04 entraremos con calma en residual, normalización, MLP y logits.

Simuladores para verlo funcionando

Hay conceptos que se entienden mejor cuando los ves moverse. Estos recursos no sustituyen al capítulo, pero ayudan mucho a mirar la arquitectura con las manos:

RECURSO	QUÉ PUEDES MIRAR	CUÁNDO ABRIRLO
Transformer Explainer	Visualiza un GPT-2 pequeño en el navegador, con flujo de tokens, componentes internos y predicción del siguiente token.[10]	Después de «Cómo funcionaría una pasada completa».
LLM Visualization	Un recorrido 3D por un Transformer estilo GPT, con capas, atención y operaciones internas.[11]	Cuando quieras una intuición espacial de las capas.
The Illustrated Transformer	Explicación visual paso a paso del Transformer original, muy buena para fijar encoder, decoder y atención.[12]	Antes o después de leer el bloque del paper original.
The Annotated Transformer	Implementación anotada del paper, con código y explicación matemática.[13]	Cuando ya quieras pasar de dibujo a código.

Una forma útil de usarlos: primero abre Transformer Explainer y escribe una frase corta. Mira cómo cambia la distribución del siguiente token. Después vuelve al pseudocódigo de este capítulo y localiza en qué parte del simulador está cada paso. Si una visualización te abruma, no intentes entender todo a la vez: busca solo tokens, atención y logits.



Por qué el Transformer cambió el juego

La ventaja práctica del Transformer está en dos ideas que se refuerzan.

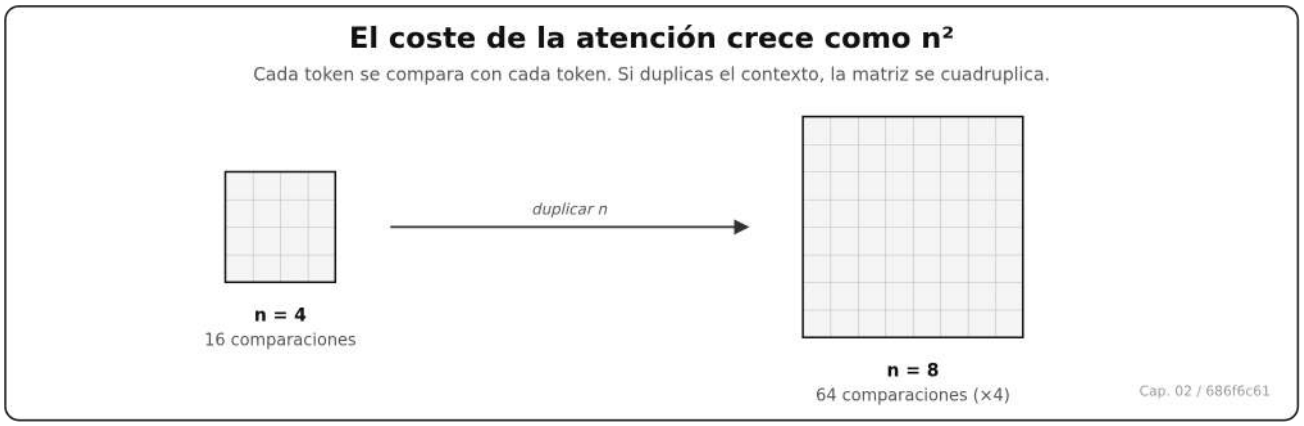
Primera: paralelismo. En una arquitectura recurrente clásica, procesar el token 200 depende de haber procesado antes los 199 tokens previos. En un Transformer, muchas operaciones se expresan como multiplicaciones de matrices. Eso encaja muy bien con GPU y TPU, que son máquinas excelentes para operar con bloques grandes de números.

Segunda: acceso directo al contexto. Si el token de la posición 200 necesita información de la posición 12, la atención puede crear una conexión directa dentro de la matriz. No necesita transportar todo a través de un único estado oculto. Esa es una de las razones por las que el Transformer se volvió la arquitectura dominante para modelos grandes de lenguaje.

Pero no todo es gratis. La atención estándar compara cada token con cada token. Si hay n tokens, la matriz de atención tiene $n \times n$ entradas:

$$\text{coste de atención} \propto n^2$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
n	Número de tokens de contexto.	1 000 tokens.
n^2	Comparaciones entre posiciones.	1 000 000 relaciones posibles.
$\propto$	«Proporcional a».	Si duplicas n, el coste crece aproximadamente por cuatro.



Esto explica por qué el contexto largo es tan valioso y tan caro. No basta con decir «metamos más texto». El diseño del sistema debe decidir qué entra, qué se resume, qué se recupera con RAG y qué conviene dejar fuera.

Además, no todos los Transformers leen igual. BERT usa una arquitectura orientada a entender texto mirando a ambos lados del contexto, muy útil para tareas de comprensión.[14] La familia GPT usa el enfoque autoregresivo: predice el siguiente token mirando el contexto disponible hacia atrás, que es la base de los LLMs generativos modernos.[15]

En el día a día

Cuando integras un modelo en una aplicación, rara vez manipulas Q, K, V a mano. Pero sí pagas sus consecuencias.

Diseñar prompts largos. Si añades instrucciones, documentos, historial y ejemplos, estás aumentando n. Más tokens significan más memoria, más latencia y más coste. Este capítulo te ayuda a entender por qué el contexto debe diseñarse, no solo rellenarse.

Elegir entre modelos. Dos modelos pueden tener calidades parecidas, pero arquitecturas internas distintas. Uno puede ser más rápido con contexto corto; otro puede estar optimizado para contexto largo. Cuando leas una ficha técnica, «context window», «attention implementation», «KV cache» y «throughput» dejan de ser palabras decorativas.

Depurar respuestas raras. A veces el problema no está en que el modelo «no sepa», sino en que la información relevante no está en el contexto, queda enterrada entre demasiado ruido o llega en un formato difícil de usar. Entender atención te recuerda que el modelo mezcla señales: si la señal útil está muy diluida, la salida también puede estarlo.

Conectar con RAG. En RAG, no meteremos todos los documentos en el prompt. Recuperaremos fragmentos. La razón no es solo comodidad: es arquitectura. El contexto es una ventana cara y limitada.

Por qué debería importarte

Porque el Transformer es la bisagra entre «texto» y «sistema de IA moderno». Si solo sabes que los LLMs predicen tokens, entiendes el comportamiento externo. Si además entiendes tensores y atención, empiezas a leer por dentro por qué el contexto importa, por qué el orden importa, por qué el coste crece y por qué algunos errores se arreglan cambiando el contexto en vez de cambiando de modelo.

También te prepara para el capítulo siguiente. Q, K, V, máscara causal y softmax no son nombres sueltos: son las piezas concretas que hacen posible esta mesa de mezclas.

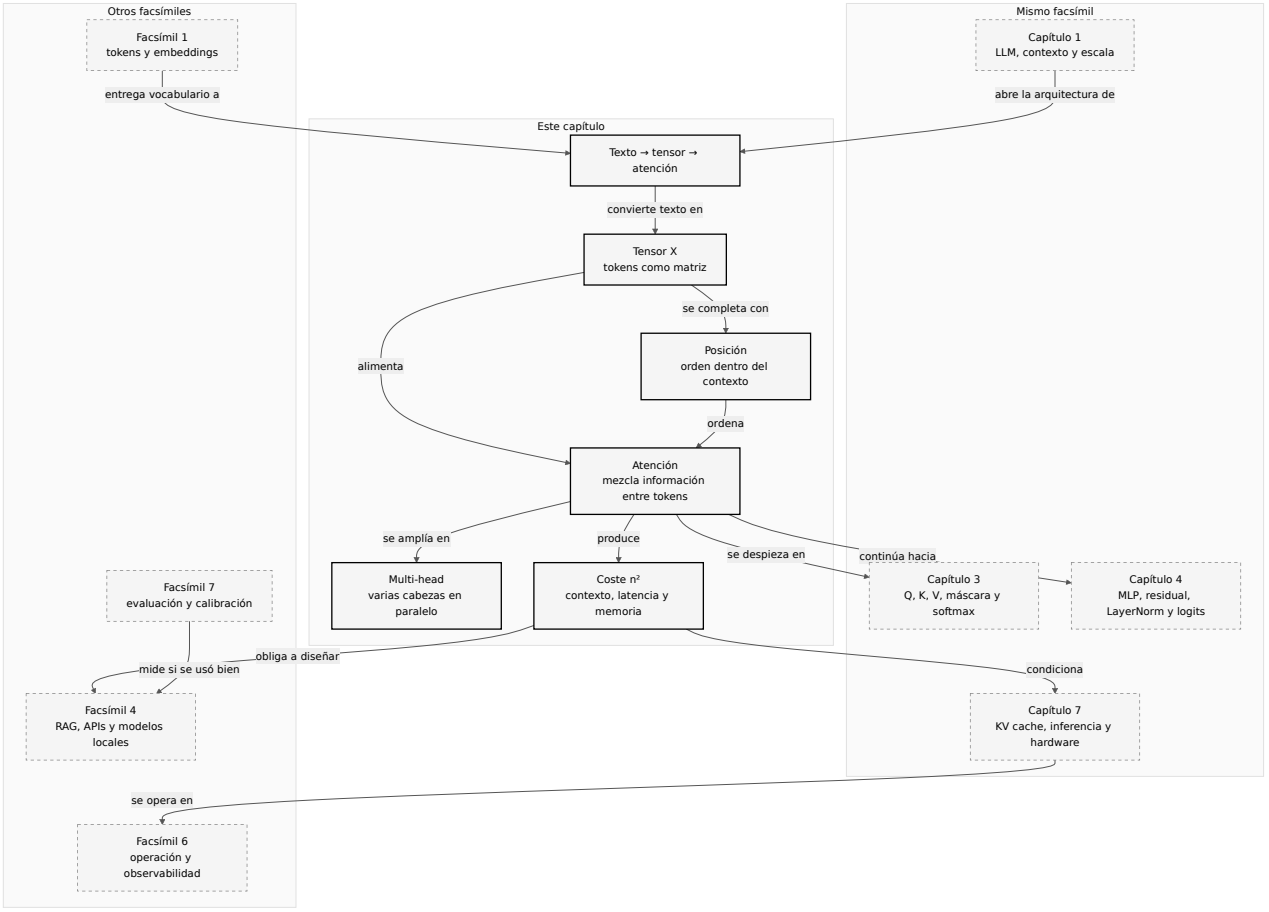
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Pensar que el tensor es un detalle de implementación	El tensor X define cuántos tokens hay, cuántas dimensiones maneja el modelo y qué coste tendrá procesar la entrada.	Cuando leas una arquitectura, pregunta siempre por las formas: n, d_{model}, capas y cabezas.
Creer que atención es explicación perfecta	Un peso de atención alto indica mezcla de información, pero no equivale automáticamente a una explicación humana completa.	Úsala como pista técnica, no como prueba definitiva de causalidad.
Olvidar la posición	Si solo sumas embeddings de tokens sin posición, pierdes orden. Y en lenguaje, el orden cambia el significado.	Repite mentalmente: token + posición = entrada útil para secuencia.
Meter todo en contexto por tranquilidad	Más contexto puede añadir ruido, coste y latencia. No todo lo disponible es útil.	Recupera, resume y ordena. El contexto es una superficie de diseño.
Separar demasiado los capítulos	Tokens, embeddings, atención, logits y evaluación parecen temas distintos, pero son una cadena.	Sigue el flujo: texto → tensor → atención → logits → decisión de producto.

Cómo encaja todo

Este mapa baja un nivel respecto al capítulo anterior. Ya no miramos solo el LLM como producto matemático: miramos cómo una frase se vuelve tensor, cómo la posición evita perder el orden y cómo la atención empieza a mezclar señales.

La conexión importante es práctica: cada vez que en facsímiles posteriores hablemos de contexto, RAG, coste o latencia, estaremos pagando decisiones que nacen aquí, en el tamaño de n, d_{model} y la matriz de atención.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Tensor	Estructura numérica con dimensiones. Una matriz $n \times d$ es un tensor de dos dimensiones.
d_{model}	Dimensión interna del modelo: cuántos números representan cada token dentro del Transformer.
Embedding posicional	Información que indica la posición de cada token para que el modelo no pierda el orden.
Atención	Mecanismo que calcula pesos para mezclar información entre posiciones de una secuencia.
Multi-head attention	Varias cabezas de atención en paralelo, cada una con sus proyecciones, que miran patrones distintos.
Matriz de atención	Tabla $n \times n$ donde cada fila indica cuánto mira un token a los demás.
Consulta, clave y valor	Tres proyecciones aprendidas que permiten preguntar, comparar y mezclar información.
Residual	Conexión que suma la entrada del bloque a su salida para no perder señal y entrenar mejor.
Normalización	Operación que estabiliza activaciones dentro de la red.
Paralelismo	Capacidad de ejecutar muchas operaciones a la vez, especialmente multiplicaciones de matrices.

Antes de pasar página

- [ ] ¿Puedo explicar por qué el Transformer recibe una matriz y no texto en bruto? (Si no, vuelve a «De tokens a tensores».)
- [ ] ¿Entiendo qué significa $X \in \mathbb{R}^{n \times d_{\text{model}}}$? (Si no, vuelve a «De tokens a tensores».)
- [ ] ¿Sé decir para qué sirve la información de posición? (Si no, vuelve a «De tokens a tensores».)
- [ ] ¿Distingo las tres formas de codificar la posición (sinusoidal, aprendida, RoPE)? (Si no, vuelve a «De dónde sale la posición».)

- [ ] ¿Puedo explicar la atención como una mezcla ponderada de valores? (Si no, vuelve a «La atención como mesa de mezclas».)
- [ ] ¿Entiendo por qué Q, K y V son tres proyecciones distintas y por qué se divide por d_k? (Si no, vuelve a «La atención como mesa de mezclas».)
- [ ] ¿Sé qué aporta multi-head attention frente a una sola cabeza? (Si no, vuelve a «Varias cabezas a la vez».)
- [ ] ¿Puedo calcular una softmax pequeña y leer sus pesos? (Si no, vuelve a «Un ejemplo pequeño con números».)
- [ ] ¿Puedo narrar una pasada completa desde el texto hasta el siguiente token? (Si no, vuelve a «Cómo funcionaría una pasada completa».)
- [ ] ¿He abierto al menos un simulador y sé localizar tokens, atención y logits? (Si no, vuelve a «Simuladores para verlo funcionando».)
- [ ] ¿Entiendo por qué el coste de atención crece con n^2? (Si no, vuelve a «Por qué el Transformer cambió el juego».)
- [ ] ¿Sé conectar este capítulo con Q, K, V y máscara causal del capítulo siguiente? (Si no, vuelve a «Cómo encaja todo».)
- [ ] ¿Puedo explicar por qué meter más contexto no siempre mejora una aplicación? (Si no, vuelve a «Dónde solía tropezar yo».)

En resumen

IDEA FUERZA	DETALLE
El texto entra al Transformer como tensor.	Después de tokenizar y buscar embeddings, la entrada tiene forma $n \times d_{\text{model}}$: filas para tokens, columnas para dimensiones internas.
La atención contextualiza cada token.	Cada posición calcula pesos sobre otras posiciones y mezcla valores para construir una representación dependiente del contexto.
El Transformer ganó por paralelismo y acceso directo.	Procesa secuencias con operaciones matriciales y permite que posiciones lejanas se conecten sin pasar por un único estado oculto.
El contexto largo tiene coste real.	La matriz de atención crece como n^2, así que diseñar contexto es una decisión técnica y de producto.

Para saber más

Alammar, J. (2018). *The Illustrated Transformer*. https://jalammar.github.io/illustrated-transformer/

Ba, J. L., Kiros, J. R. y Hinton, G. E. (2016). *Layer normalization*. https://arxiv.org/abs/1607.06450

Bahdanau, D., Cho, K. y Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. En *International Conference on Learning Representations*. https://arxiv.org/abs/1409.0473

Brown, T. B. y otros (2020). Language models are few-shot learners. En *Advances in Neural Information Processing Systems 33* (pp. 1877-1901). https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Bycroft, B. (2023). *LLM Visualization*. https://bbycroft.net/llm

Cho, A., Kim, G. C., Karpekov, A., Helbling, A., Wang, Z. J., Lee, S., Hoover, B. y Chau, D. H. (2025). Transformer Explainer: interactive learning of text-generative models. En *Proceedings of the AAAI Conference on Artificial Intelligence*. https://ojs.aaai.org/index.php/AAAI/article/download/35347/37502

Devlin, J., Chang, M. W., Lee, K. y Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. En *Proceedings of NAACL-HLT* (pp. 4171-4186). https://doi.org/10.18653/v1/N19-1423

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org

He, K., Zhang, X., Ren, S. y Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90

Luong, M.-T., Pham, H. y Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. En *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1412-1421). https://doi.org/10.18653/v1/D15-1166

Rush, A. M. (2018). *The Annotated Transformer*. https://nlp.seas.harvard.edu/annotated-transformer/

Sutskever, I., Vinyals, O. y Le, Q. V. (2014). Sequence to sequence learning with neural networks. En *Advances in Neural Information Processing Systems 27* (pp. 3104-3112). https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). https://papers.nips.cc/paper/7181-attention-is-all-you-need

Notas

1. Sutskever, I., Vinyals, O. y Le, Q. V. (2014). Sequence to sequence learning with neural networks. En *Advances in Neural Information Processing Systems 27* (pp. 3104-3112). https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks. Los modelos sequence-to-sequence con redes recurrentes fueron un paso clave para traducir y generar secuencias antes del dominio del Transformer. ↩

2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). https://papers.nips.cc/paper/7181-attention-is-all-you-need. El artículo presentó una arquitectura sin recurrencia ni convolución, basada en atención, capaz de entrenar modelos de secuencia de forma mucho más paralela. ↩

3. Bahdanau, D., Cho, K. y Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. En *International Conference on Learning Representations*. https://arxiv.org/abs/1409.0473. El trabajo introdujo un mecanismo de atención para traducción neuronal que permitía al decodificador consultar partes relevantes de la entrada. ↩

4. Luong, M.-T., Pham, H. y Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. En *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1412-1421). https://doi.org/10.18653/v1/D15-1166. El artículo comparó formas prácticas de calcular atención en traducción automática neuronal. ↩

5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). https://arxiv.org/abs/1706.03762. La página de arXiv muestra la fecha de envío, los autores y el resumen del trabajo. ↩

6. Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B. y Liu, Y. (2021). *RoFormer: Enhanced Transformer with Rotary Position Embedding*. https://arxiv.org/abs/2104.09864. RoPE codifica la posición rotando las consultas y las claves, lo que captura posiciones relativas; es la opción dominante en los LLMs recientes. ↩

7. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org. El libro presenta las redes profundas como composiciones de funciones diferenciables sobre tensores, justo el marco matemático que usamos aquí para leer el Transformer. ↩

8. He, K., Zhang, X., Ren, S. y Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90. Aunque ResNet nació en visión por computador, popularizó las conexiones residuales que ayudan a entrenar redes muy profundas. ↩

9. Ba, J. L., Kiros, J. R. y Hinton, G. E. (2016). *Layer normalization*. https://arxiv.org/abs/1607.06450. LayerNorm normaliza activaciones dentro de una capa y se convirtió en una pieza habitual de arquitecturas Transformer. ↩

10. Cho, A., Kim, G. C., Karpekov, A., Helbling, A., Wang, Z. J., Lee, S., Hoover, B. y Chau, D. H. (2025). Transformer Explainer: interactive learning of text-generative models. En *Proceedings of the AAAI Conference on Artificial Intelligence*. https://ojs.aaai.org/index.php/AAAI/article/download/35347/37502. El artículo describe la herramienta como una visualización web interactiva que ejecuta un modelo GPT-2 localmente en el navegador. ↩

11. Bycroft, B. (2023). *LLM Visualization*. https://bbycroft.net/llm. Visualización interactiva 3D de un modelo Transformer estilo GPT. ↩

12. Alammar, J. (2018). *The Illustrated Transformer*. https://jalammar.github.io/illustrated-transformer/. Recurso visual ampliamente usado para entender la arquitectura Transformer. ↩

13. Rush, A. M. (2018). *The Annotated Transformer*. https://nlp.seas.harvard.edu/annotated-transformer/. Implementación comentada de la arquitectura Transformer original. ↩

14. Devlin, J., Chang, M. W., Lee, K. y Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. En *Proceedings of NAACL-HLT* (pp. 4171-4186). https://doi.org/10.18653/v1/N19-1423. BERT popularizó el preentrenamiento bidireccional con Transformer encoder para comprensión de lenguaje. ↩

15. Brown, T. B. y otros (2020). Language models are few-shot learners. En *Advances in Neural Information Processing Systems 33* (pp. 1877-1901). https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html. GPT-3 mostró que un Transformer decoder autoregresivo a gran escala podía realizar muchas tareas mediante ejemplos en el propio prompt. ↩