Facsímil 03

# Arquitecturas y modelos

Capítulo 03

## Q, K, V, máscara causal y softmax

686f6c61 · 2026

# Capítulo 03: Q, K, V, máscara causal y softmax

## La mesa de mezclas ya tiene mandos

En el capítulo anterior dijimos que la atención era una mesa de mezclas: cada token decide cuánto tomar de otros tokens para construir una representación contextual. Pero dejamos sin abrir los mandos concretos.

Ahora entran tres letras que aparecen por todas partes cuando alguien explica Transformers: **Q**, **K** y **V**. Al principio parecen una contraseña. En realidad son tres formas distintas de mirar el mismo tensor: qué busca cada token, qué ofrece cada token y qué información se mezcla al final.

También aparece una regla silenciosa pero decisiva: la **máscara causal**. Sin ella, un modelo generativo haría trampas durante entrenamiento: para predecir el token 3 podría mirar el token 4, que en generación real todavía no existe.

## Qué no son Q, K y V

Q, K y V no son tres bases de datos internas. No hay una tabla llamada «preguntas», otra llamada «claves» y otra llamada «valores» con significado humano directo. Son proyecciones lineales aprendidas a partir del tensor de entrada.

Tampoco son conceptos nuevos separados del capítulo anterior. En el capítulo 02 ya teníamos una matriz $X$ con una fila por token. Q, K y V salen de esa misma matriz, multiplicándola por tres matrices de pesos distintas.

Y la máscara causal no es un filtro moral ni una regla de producto. Es una restricción geométrica sobre qué posiciones pueden verse entre sí. Su función es mantener honesta la generación de izquierda a derecha.

## Qué sí son Q, K y V

En self-attention, cada token produce tres vectores: una consulta (*query*), una clave (*key*) y un valor (*value*). El Transformer original formalizó este mecanismo como atención escalada por producto punto y lo convirtió en la pieza central de la arquitectura.[1]

La intuición:

| LETRA | NOMBRE | PREGUNTA INFORMAL |
|---|---|---|
| $Q$ | Query / consulta | ¿Qué estoy buscando desde esta posición? |
| $K$ | Key / clave | ¿Qué tipo de información ofrezco para que otros me encuentren? |
| $V$ | Value / valor | Si alguien me mira, ¿qué información le entrego? |

Como lectura pedagógica complementaria, Alyona Vert resume esta intuición de forma muy útil: Q busca, K permite decidir si algo merece atención y V transporta el contenido que se mezcla; además conecta esa separación con la KV cache que veremos en inferencia.[2]

Imagina la frase:

> El banco aprobó el préstamo

Para la posición «banco», la consulta puede aprender a buscar señales que aclaren el sentido. Las claves de «aprobó» y «préstamo» pueden ofrecer pistas financieras. Los valores son la información que realmente se mezcla cuando esas posiciones reciben peso.

## Las proyecciones lineales

Partimos de la matriz $X$:

$$X \in \mathbb{R}^{n \times d_{\text{model}}}$$

| SÍMBOLO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| $X$ | Tensor de entrada a la capa de atención. | 4 tokens por 3 dimensiones en el ejemplo pequeño. |
| $n$ | Número de tokens. | $n = 4$. |
| $d_{\text{model}}$ | Dimensión interna del modelo. | En un modelo real, 768, 4096 o más. |

Calculamos:

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

| SÍMBOLO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| $W_Q$ | Matriz aprendida para consultas. | Transforma cada fila de $X$ en lo que busca. |
| $W_K$ | Matriz aprendida para claves. | Transforma cada fila de $X$ en lo que ofrece. |
| $W_V$ | Matriz aprendida para valores. | Transforma cada fila de $X$ en lo que se mezclará. |
| $Q, K, V$ | Tres matrices derivadas de la misma entrada. | Tres versiones útiles del contexto. |

Estas multiplicaciones son capas lineales, una idea básica de redes neuronales profundas.[3] Lo interesante no es que la operación sea exótica, sino que el entrenamiento aprende qué proyecciones convienen para comparar tokens.

## Puntuaciones: comparar lo que busco con lo que ofreces

Para saber cuánto debe mirar cada token a cada otro token, comparamos consultas con claves:

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

| SÍMBOLO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| $S$ | Matriz de puntuaciones antes de softmax. | Una tabla $n \times n$. |
| $QK^T$ | Producto entre consultas y claves. | Afinidad entre cada par de posiciones. |
| $K^T$ | Transpuesta de $K$. | Convierte columnas en filas para multiplicar. |
| $d_k$ | Dimensión de las claves. | Si las claves tienen 64 dimensiones, $d_k = 64$. |
| $d_k$ | Factor de escala. | Evita puntuaciones demasiado grandes. |

El escalado por $d_k$ ayuda a que las puntuaciones no crezcan demasiado cuando la dimensión aumenta. Si las puntuaciones llegan enormes a softmax, la distribución puede volverse muy extrema y los gradientes menos útiles. Esto lo verás también al estudiar optimización y estabilidad numérica.

## La máscara causal: no mirar el futuro

En un modelo generativo autoregresivo, la posición $i$ solo puede mirar posiciones anteriores o la misma posición. No puede mirar posiciones futuras. La máscara causal se puede escribir así:

$$M_{ij} = \begin{cases} 0 & \text{si } j \leq i \\ -\infty & \text{si } j > i \end{cases}$$

| SÍMBOLO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| $M_{ij}$ | Valor de la máscara para la fila $i$, columna $j$. | Fila 2 no puede mirar columna 4. |
| $j \leq i$ | Posición visible. | El token actual puede mirar lo anterior. |
| $j > i$ | Posición futura. | Se bloquea antes de softmax. |
| $-\infty$ | Valor ideal que softmax convierte en peso 0. | El futuro queda con probabilidad cero. |

¿Por qué $-\infty$ y no simplemente poner el peso a cero después del softmax? Porque sumar $-\infty$ **antes** del softmax hace el trabajo solo: como $e^{-\infty} = 0$, esa posición sale con peso exactamente cero y el resto de la fila se normaliza solo entre las posiciones permitidas. Si en cambio calculáramos el softmax con todas las posiciones y luego borráramos las futuras, la fila ya no sumaría 1 y habría que renormalizar a mano. En código real no se usa $-\infty$ literal (produciría operaciones inválidas), sino un número muy negativo como $-10^9$: al exponenciar queda prácticamente cero, con el mismo efecto y sin romper la aritmética.

Aplicamos la máscara a las puntuaciones:

$$\tilde{S} = S + M$$

Después:

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

| SÍMBOLO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| $\tilde{S}$ | Puntuaciones ya enmascaradas. | Futuro sustituido por $-\infty$. |
| $A$ | Matriz final de atención. | Cada fila suma 1. |
| $\text{softmax}$ | Función que convierte puntuaciones en pesos positivos. | Muy usada en clasificación y en LLMs.[4] |

La máscara causal es una de las diferencias importantes entre un modelo que genera texto de izquierda a derecha y un modelo de comprensión que puede mirar ambos lados del texto. GPT usa atención causal para generación autoregresiva.[5] BERT, en cambio, usa una arquitectura bidireccional pensada para comprensión.[6]

Conviene poner nombre a las tres formas de atención que aparecen en la práctica, porque se confunden con facilidad:

| VARIANTE | QUIÉN MIRA A QUIÉN | MÁSCARA | DÓNDE SE USA |
|---|---|---|---|
| **Self-attention** | Los tokens de una secuencia se miran entre sí. | Sin máscara causal: cada token ve toda la secuencia. | El encoder de BERT, comprensión bidireccional. |
| **Self-attention enmascarada** | Los tokens de una secuencia se miran, pero solo hacia atrás. | Máscara causal: el futuro queda bloqueado. | El decoder de GPT, generación autoregresiva. |
| **Cross-attention** | Los tokens de una secuencia miran a los de **otra**. | Sin máscara causal (mira toda la otra secuencia). | Encoder-decoder (traducción) y modelos multimodales (texto que mira una imagen). |

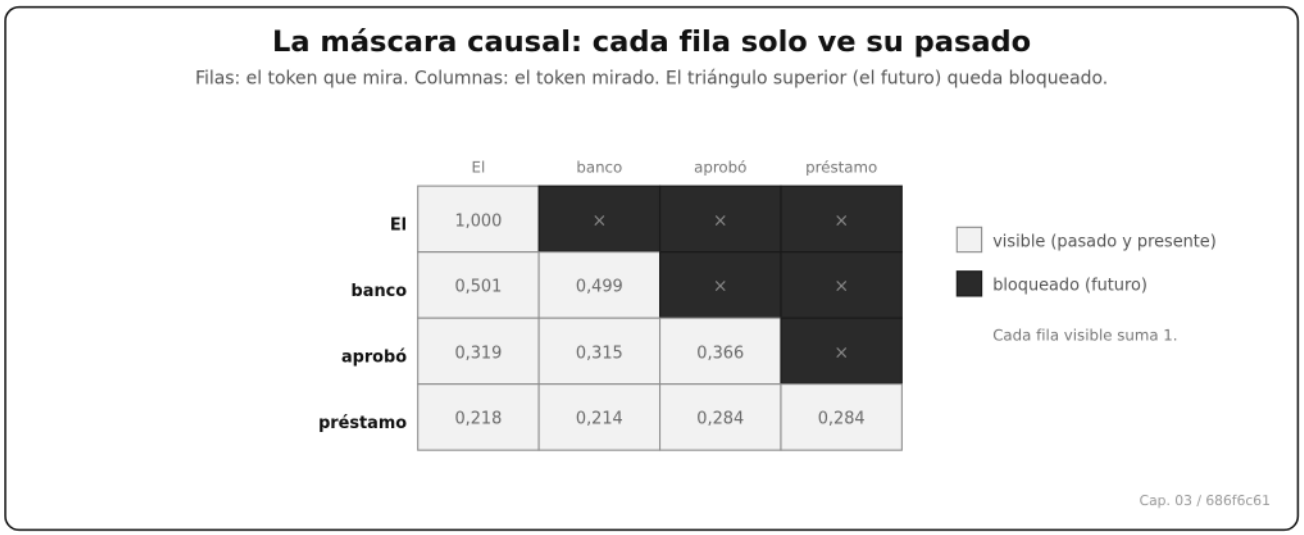
La diferencia entre las dos primeras es solo la máscara. La misma maquinaria de Q, K, V sirve para las tres; lo que cambia es a qué se permite mirar.

## Cómo leer la matriz sin perderse

La matriz de atención se lee **fila a fila**. Cada fila responde a una pregunta concreta: «desde este token, ¿cuánto miro a cada posición disponible?». Por eso no tiene sentido mirar un número aislado sin saber en qué fila vive.

En una frase de cuatro tokens, la máscara causal deja esta forma:

| FILA | TOKEN QUE MIRA | PUEDE MIRAR | NO PUEDE MIRAR TODAVÍA | LECTURA |
|---|---|---|---|---|
| 1 | El | El | banco, aprobó, préstamo | No hay pasado; solo se ve a sí mismo. |
| 2 | banco | El, banco | aprobó, préstamo | Tiene un poco de contexto, pero no sabe lo que viene después. |
| 3 | aprobó | El, banco, aprobó | préstamo | Puede usar el sujeto y el verbo, pero no el objeto futuro. |
| 4 | préstamo | El, banco, aprobó, préstamo | (ninguna) | Ya puede mirar toda la frase disponible. |




Después de aplicar softmax, cada fila suma 1. Eso significa que el token reparte toda su atención entre las posiciones permitidas. Si una fila tiene cuatro posiciones posibles, no significa que las cuatro importen igual; significa que el reparto completo se hace dentro de esas cuatro posiciones.

Una forma sana de leerlo es esta:

1. Elige una fila.
2. Mira qué columnas permite la máscara.
3. Aplica softmax solo a esas puntuaciones útiles.
4. Usa esos pesos para mezclar los valores $V$.

Si sigues esos cuatro pasos, la fórmula deja de ser una pared y se convierte en una receta.

## Valores: mezclar la información

Una vez tenemos los pesos de atención $A$, mezclamos los valores:

$$O = AV$$

| SÍMBOLO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| $O$ | Salida de la atención. | Nueva matriz con representaciones contextualizadas. |
| $A$ | Pesos de atención. | Cuánto mira cada token a cada posición permitida. |
| $V$ | Valores. | Información que se mezcla. |

Esta fórmula es la parte más importante para la intuición: $Q$ y $K$ deciden **cuánto** mirar; $V$ decide **qué información** se toma cuando miras.

### Cuánto cuesta esta operación

Merece la pena ver de dónde sale el coste. Para puntuar, $QK^T$ compara las $n$ consultas con las $n$ claves, cada una de dimensión $d_k$: son del orden de $n^2 \cdot d_k$ operaciones, y produce una matriz $n \times n$. La mezcla $AV$ vuelve a recorrer esa matriz $n \times n$ por cada dimensión de los valores, otro $n^2 \cdot d$. El término que manda es el $n^2$: cada token se compara con cada token. Por eso, como vimos en el capítulo anterior, **duplicar el contexto cuadruplica el trabajo de la atención**, y por eso casi toda la investigación de contexto largo busca formas de no pagar ese $n^2$ entero.

### De Q, K, V a la KV cache

Aquí está el germen de una optimización capital. En generación, el modelo produce un token, lo añade al contexto y repite. Pero gracias a la máscara causal, los tokens ya generados **solo miran hacia atrás**: su $K$ y su $V$ no cambian cuando llega un token nuevo. Recalcularlos en cada paso sería un derroche. La solución es guardarlos: la **KV cache** almacena las claves y los valores de todo el pasado, y en cada token nuevo solo se calcula su $Q$, su $K$ y su $V$ y se compara contra la caché. Sin máscara causal esto no sería posible, porque el pasado podría depender del futuro. Veremos el coste en memoria de esa caché en el capítulo 7.

## Un ejemplo pequeño con máscara causal

Usemos cuatro tokens:

```
El banco aprobó préstamo
```

Tras las proyecciones, supongamos que obtenemos estas matrices pequeñas:

| TOKEN | $Q$ | $K$ | $V$ |
|---|---|---|---|
| El | $[-0{,}01,\ -0{,}04]$ | $[0{,}18,\ -0{,}19]$ | $[-0{,}02,\ 0{,}24]$ |
| banco | $[0{,}47,\ 0{,}00]$ | $[0{,}17,\ -0{,}23]$ | $[0{,}51,\ -0{,}04]$ |
| aprobó | $[-0{,}03,\ 0{,}51]$ | $[0{,}51,\ 0{,}21]$ | $[0{,}23,\ 0{,}50]$ |
| préstamo | $[0{,}43,\ 0{,}58]$ | $[0{,}45,\ 0{,}25]$ | $[0{,}75,\ 0{,}16]$ |

La matriz de puntuaciones enmascarada queda así, redondeada:

| TOKEN QUE MIRA | EL | BANCO | APROBÓ | PRÉSTAMO |
|---|---|---|---|---|
| El | 0,004 | n/d | n/d | n/d |
| banco | 0,060 | 0,056 | n/d | n/d |
| aprobó | -0,072 | -0,087 | 0,065 | n/d |
| préstamo | -0,023 | -0,043 | 0,241 | 0,239 |

La marca «n/d» significa «posición futura bloqueada por la máscara». Cuando aplicamos softmax por fila:

| TOKEN QUE MIRA | EL | BANCO | APROBÓ | PRÉSTAMO |
|---|---|---|---|---|
| El | 1,000 | 0,000 | 0,000 | 0,000 |
| banco | 0,501 | 0,499 | 0,000 | 0,000 |
| aprobó | 0,319 | 0,315 | 0,366 | 0,000 |
| préstamo | 0,218 | 0,214 | 0,284 | 0,284 |

Mira la segunda fila. El token «banco» solo puede mirar «El» y «banco». Aunque «aprobó» y «préstamo» serían pistas útiles, todavía son futuro para esa posición en generación autoregresiva. Por eso su peso es 0.

Después multiplicamos por $V$:

$$O = AV$$

La salida redondeada es:

| TOKEN | SALIDA CONTEXTUALIZADA |
|---|---|
| El | $[-0{,}020,\ 0{,}240]$ |
| banco | $[0{,}245,\ 0{,}100]$ |
| aprobó | $[0{,}238,\ 0{,}247]$ |
| préstamo | $[0{,}383,\ 0{,}231]$ |

Esta tabla enseña una idea que cuesta ver al principio: el mismo mecanismo sirve para todos los tokens, pero cada fila tiene una frontera temporal distinta.

### Modo ingeniero: la atención causal en código

Todo lo anterior cabe en una función. Con `numpy`, la atención causal es: puntuar con $QK^T/\sqrt{d_k}$, sumar la máscara, aplicar softmax por fila y mezclar con $V$.

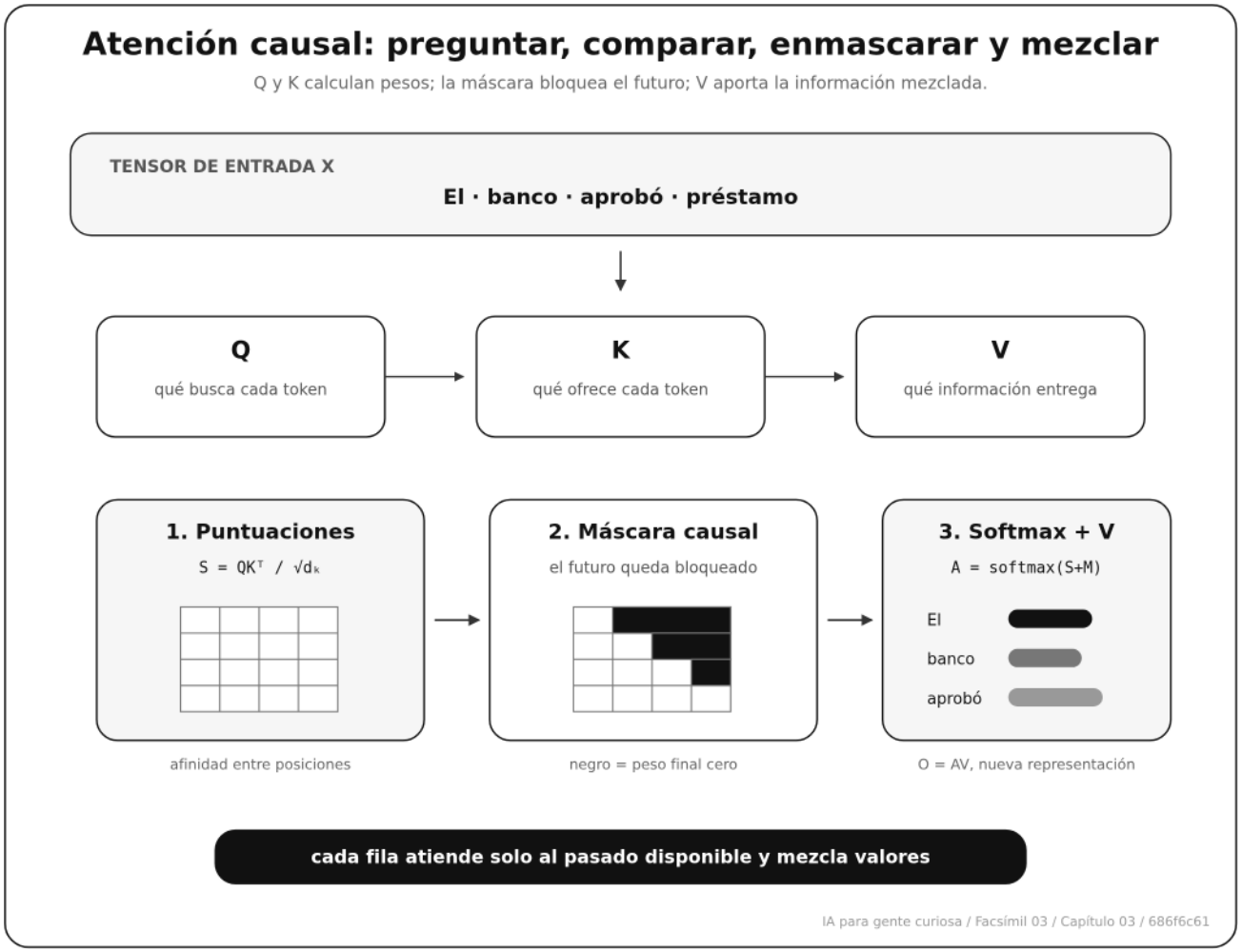
```
import numpy as np

def softmax_filas(S):
    S = S - S.max(axis=1, keepdims=True)           # estabilidad por fila
    e = np.exp(S)
    return e / e.sum(axis=1, keepdims=True)

def atencion_causal(Q, K, V):
    d_k = Q.shape[1]
    S = Q @ K.T / np.sqrt(d_k)                      # puntuaciones n x n
    n = S.shape[0]
    mascara = np.triu(np.full((n, n), -1e9), k=1)   # -1e9 en el triángulo superior
    A = softmax_filas(S + mascara)                  # el futuro queda en 0
    return A @ V                                    # mezcla de valores

# con las Q, K, V del ejemplo, A[1] (banco) sale [0.501, 0.499, 0, 0]
```

`np.triu(..., k=1)` construye la máscara: pone `-1e9` por encima de la diagonal (el futuro) y deja `0` en la diagonal y por debajo (el pasado y el presente). Tras sumar y aplicar softmax, esas posiciones futuras quedan en cero exacto. Es, línea por línea, lo que comprueba el lab de este capítulo: que ninguna fila reparte peso hacia el futuro.




## Varias cabezas: mirar relaciones distintas

El Transformer no usa una sola atención. Usa varias cabezas en paralelo:

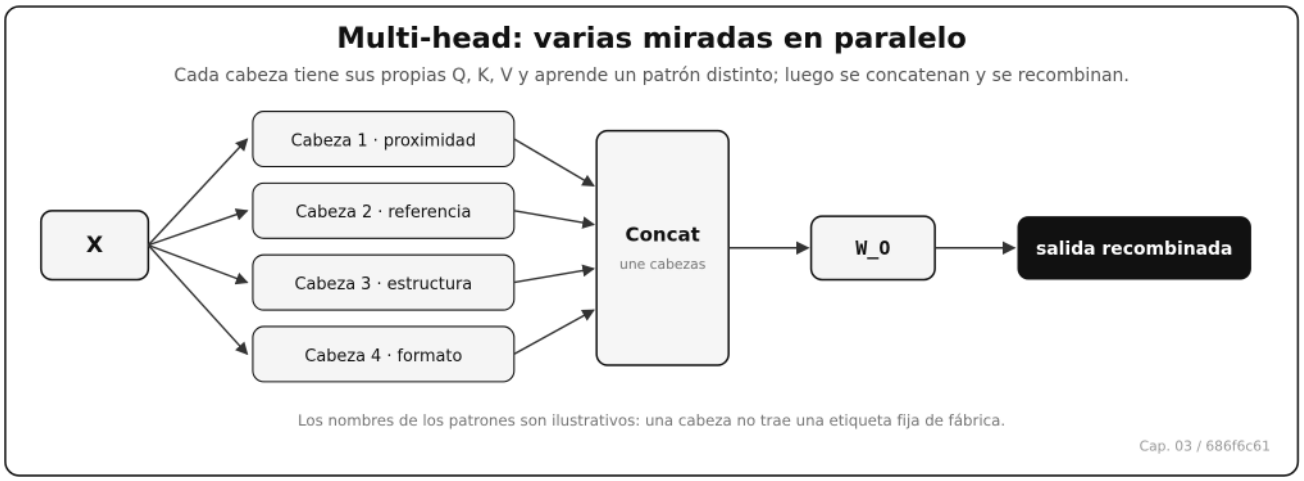
$$\text{head}_h = \text{Attention}(XW_Q^{(h)}, XW_K^{(h)}, XW_V^{(h)})$$

| SÍMBOLO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| $h$ | Índice de la cabeza de atención. | Cabeza 1, cabeza 2, etc. |
| $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)}$ | Proyecciones propias de la cabeza $h$. | Cada cabeza aprende una mirada distinta. |
| $\text{head}_h$ | Salida de una cabeza. | Una matriz contextual. |

Después concatena las cabezas y proyecta:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H) W_O$$

| SÍMBOLO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| $H$ | Número de cabezas. | 12, 32, 64... según modelo. |
| $\text{Concat}$ | Concatenación de salidas. | Une las cabezas por dimensión. |
| $W_O$ | Matriz de salida aprendida. | Recombina lo que miraron las cabezas. |




La intuición: una cabeza puede aprender relaciones de proximidad, otra relaciones sintácticas, otra dependencias de nombres, otra patrones de formato. No hay que imaginarlo como roles fijos garantizados, pero sí como capacidad de mirar varios tipos de relación en paralelo.

Recursos como *The Illustrated Transformer* ayudan a ver esta estructura con dibujo.[7] *The Annotated Transformer* la baja además a código comentado.[8]

### Cuando las cabezas comparten claves y valores

La atención multi-cabeza clásica (MHA) da a cada cabeza sus propias $Q$, $K$ y $V$. Funciona muy bien, pero tiene un coste oculto que se paga en inferencia: como vimos, $K$ y $V$ se guardan en la KV cache, y con muchas cabezas esa caché se multiplica. De ahí nacen dos variantes que dominan los modelos modernos:

| VARIANTE | QUÉ COMPARTEN LAS CABEZAS | EFECTO |
|---|---|---|
| **MHA** (multi-head) | Nada: cada cabeza tiene su $Q$, $K$, $V$. | Máxima expresividad, máxima KV cache. |
| **GQA** (grouped-query) | Varias cabezas comparten un mismo par $K$, $V$. | Casi la misma calidad con una caché mucho menor. |
| **MQA** (multi-query) | **Todas** las cabezas comparten un único $K$, $V$. | Caché mínima, ligeramente menos expresiva. |

Las consultas $Q$ siguen siendo independientes en las tres; lo que se comparte son las claves y los valores. Por eso GQA es hoy la opción por defecto en los LLMs de contexto largo: divide la KV cache por el factor de compartición casi sin tocar la calidad. Lo veremos con números en el capítulo 7.

## Qué aprende una cabeza de atención

Una cabeza no viene de fábrica con una misión escrita. No podemos decir «esta cabeza es la de fechas» como si fuera una etiqueta estable. Lo que sí podemos decir es que cada cabeza tiene sus propias matrices $W_Q^{(h)}$, $W_K^{(h)}$ y $W_V^{(h)}$, así que dispone de una forma propia de comparar y mezclar.

Eso permite que distintas cabezas se especialicen de manera útil:

| PATRÓN QUE PUEDE CAPTURAR | QUÉ SIGNIFICA EN UNA FRASE | POR QUÉ AYUDA |
|---|---|---|
| Proximidad | Mirar mucho a tokens cercanos. | Muchas dependencias locales viven cerca: artículo, nombre, puntuación, formato. |
| Referencia | Conectar una palabra con otra anterior. | Ayuda a mantener entidades: «Marta llegó. Ella...». |
| Estructura | Detectar patrones de frase o de código. | Un cierre de paréntesis, una coma o una indentación pueden depender de señales previas. |
| Tema | Mantener pistas semánticas repartidas por el contexto. | Si una conversación habla de medicina, finanzas o cocina, algunas posiciones arrastran ese marco. |
| Formato | Seguir listas, tablas, JSON o Markdown. | La forma del texto también es información que el modelo usa para continuar. |

La clave para el lector es no convertir esto en una caja negra. Una cabeza calcula pesos; varias cabezas calculan varios repartos; la proyección final recombina esos repartos. La complejidad emerge de repetir este patrón muchas veces, no de que una pieza aislada «entienda» el texto como una persona.

## Para verlo en pantalla

Aquí conviene jugar un poco:

| RECURSO | QUÉ MIRAR EN ESTE CAPÍTULO |
|---|---|
| Transformer Explainer | Escribe una frase corta y observa cómo la predicción del siguiente token cambia con el contexto. Busca la parte de atención y fíjate en qué posiciones reciben más peso.[9] |
| The Annotated Transformer | Localiza la función de atención y compara el código con la fórmula $\mathrm{softmax}(QK^T/\sqrt{d_k})V$. |
| The Illustrated Transformer | Repasa el dibujo de Q, K y V después de leer este capítulo. |

No intentes verlo todo a la vez. Una buena práctica es abrir el simulador con una frase de cuatro o cinco tokens y preguntar: ¿qué fila estoy mirando?, ¿qué columnas quedan permitidas?, ¿qué pesos suman 1?

## En el día a día

**Diseñar contexto.** Si sabes que cada fila de atención reparte peso entre posiciones disponibles, empiezas a cuidar dónde pones la información importante. No es lo mismo abrir un prompt con instrucciones claras que enterrarlas al final de un bloque enorme.

**Entender por qué existe la KV cache.** En inferencia, $K$ y $V$ de tokens ya procesados pueden guardarse para no recalcularlos una y otra vez. En el capítulo 07 conectaremos esta idea con memoria, latencia y hardware.

**Leer parámetros de generación.** Después de la atención y las capas siguientes llegan logits. Softmax reaparece cuando convertimos puntuaciones en probabilidades y elegimos tokens. Métodos como `top_p` y `top_k` se entienden mejor cuando ya has visto qué significa normalizar una distribución.[10]

**No sobrerreaccionar a un mapa de atención.** Mirar pesos de atención puede ser útil, pero no basta para explicar todo un comportamiento. La red tiene muchas capas, cabezas, MLPs y proyecciones. La atención es una ventana técnica, no una explicación completa del sistema.

## Por qué debería importarte

Porque Q, K, V y la máscara causal son el punto donde la arquitectura deja de ser una caja negra total. No necesitas memorizar cada matriz, pero sí entender la película: el modelo proyecta, compara, bloquea lo que no debe ver, normaliza y mezcla.

Si entiendes eso, muchas decisiones dejan de sonar arbitrarias: por qué los modelos autoregresivos generan token a token, por qué el contexto largo cuesta, por qué la KV cache importa, por qué el orden del prompt afecta y por qué softmax no significa «verdad», sino distribución de pesos.

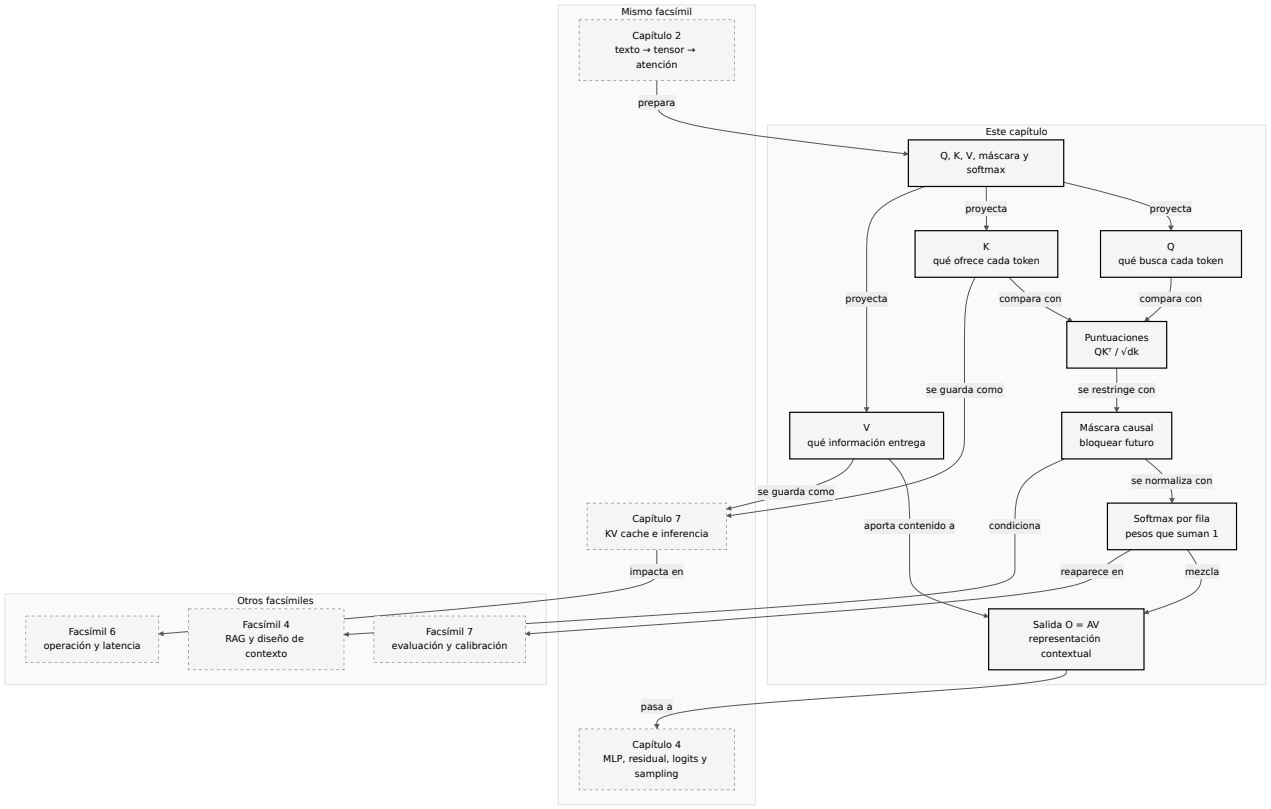
## Dónde solía tropezar yo

| ERROR | POR QUÉ ES UN ERROR | ANTÍDOTO |
|---|---|---|
| **Pensar que Q, K y V tienen significado humano fijo** | Son proyecciones aprendidas. Una dimensión o una cabeza no trae una etiqueta estable como «sujeto» o «fecha». | Usa las metáforas como andamio, no como definición literal. |
| **Olvidar que softmax se aplica por fila** | Cada token reparte su atención entre posiciones permitidas. Si normalizas toda la matriz de golpe, rompes la interpretación. | Pregunta siempre: ¿qué fila estoy leyendo? |
| **No ver la máscara causal** | Sin máscara, parece que todos miran a todos. En generación autoregresiva, eso sería mirar tokens futuros. | Dibuja el triángulo inferior de la matriz antes de calcular softmax. |
| **Confundir atención alta con importancia total** | Un peso alto indica mezcla en una cabeza y una capa, pero el comportamiento final depende de muchas operaciones. | Lee atención como señal local, no como explicación completa. |
| **Creer que V es menos importante que Q y K** | Q y K deciden pesos, pero V contiene lo que se mezcla. Sin V, no hay salida contextual. | Recita la película: comparar con $QK^T$, normalizar con softmax, mezclar con $V$. |

## Cómo encaja todo

Este mapa coloca Q, K, V y la máscara causal en la cadena completa. El capítulo anterior entrega el tensor; este capítulo decide qué posiciones pueden mirarse, cómo se normalizan los pesos y qué información se mezcla.

La conexión hacia delante es igual de importante: las mismas claves y valores que aquí parecen una fórmula serán memoria real en la KV cache, coste real en serving y criterio real cuando diseñemos contexto, RAG o evaluación.


Mismo facsímil
Capítulo 2
texto → tensor → atención
prepara
Este capítulo
Q, K, V, máscara y softmax
proyecta
proyecta
K
qué ofrece cada token
Q
qué busca cada token
proyecta
compara con
compara con
Puntuaciones
QKᵀ / √dk
se guarda como
se restringe con
V
qué información entrega
Máscara causal
bloquear futuro
se guarda como
se normaliza con
Capítulo 7
KV cache e inferencia
aporta contenido a
condiciona
Softmax por fila
pesos que suman 1
impacta en
reaparece en
mezcla
Otros facsímiles
Facsímil 6
operación y latencia
Facsímil 4
RAG y diseño de contexto
Facsímil 7
evaluación y calibración
Salida O = AV
representación contextual
pasa a
Capítulo 4
MLP, residual, logits y sampling

## Vocabulario aprendido

| TÉRMINO | DEFINICIÓN |
|---|---|
| **Query (Q)** | Proyección que representa lo que una posición busca en el contexto. |
| **Key (K)** | Proyección que representa lo que una posición ofrece para ser encontrada. |
| **Value (V)** | Proyección que contiene la información que se mezclará al final. |
| **Producto punto** | Suma de productos entre dos vectores; mide afinidad en atención. |
| **Máscara causal** | Regla que pone peso cero a posiciones futuras durante generación autoregresiva. |
| **Softmax por fila** | Normalización de cada fila de puntuaciones para convertirla en pesos que suman 1. |
| **Cabeza de atención** | Una atención independiente con sus propias proyecciones $Q$, $K$ y $V$. |
| **Multi-head attention** | Varias cabezas trabajando en paralelo y recombinadas al final. |
| **KV cache** | Memoria que guarda las claves y valores del pasado para no recalcularlos en cada token generado. |
| **GQA** | Variante en la que varias cabezas comparten un mismo par de claves y valores, reduciendo la KV cache. |

## Antes de pasar página

- [ ] ¿Puedo explicar Q, K y V sin decir solo «consulta, clave y valor»? (Si no, vuelve a «Qué sí son Q, K y V».)
- [ ] ¿Entiendo por qué $QK^T$ produce una matriz $n \times n$? (Si no, vuelve a «Puntuaciones».)
- [ ] ¿Puedo escribir la máscara causal y explicar qué significa $j > i$? (Si no, vuelve a «La máscara causal».)
- [ ] ¿Entiendo por qué la máscara suma $-\infty$ antes del softmax, y por qué en código se usa $-10^9$? (Si no, vuelve a «La máscara causal».)
- [ ] ¿Distingo self-attention, self-attention enmascarada y cross-attention? (Si no, vuelve a «La máscara causal».)
- [ ] ¿Sé por qué softmax se aplica por fila? (Si no, vuelve a «La máscara causal».)

- [ ] ¿Puedo explicar por qué $O = AV$ mezcla información? (Si no, vuelve a «Valores».)
- [ ] ¿Puedo recorrer el circuito Q → K → máscara → softmax → V sin saltarme pasos? (Si no, vuelve a «Cómo leer la matriz sin perderse».)
- [ ] ¿Entiendo por qué BERT y GPT no miran el texto de la misma forma? (Si no, vuelve a «La máscara causal».)
- [ ] ¿Sé explicar por qué aparecen ceros en la parte derecha de cada fila de la matriz de atención? (Si no, vuelve a «La máscara causal».)
- [ ] ¿Entiendo por qué la atención cuesta $n^2$ y cómo la KV cache lo alivia en generación? (Si no, vuelve a «Cuánto cuesta esta operación».)
- [ ] ¿Distingo MHA, GQA y MQA y por qué comparten claves y valores? (Si no, vuelve a «Cuando las cabezas comparten claves y valores».)
- [ ] ¿Sé conectar $K$ y $V$ con la KV cache? (Si no, vuelve a «De Q, K, V a la KV cache».)

## En resumen

| IDEA FUERZA | DETALLE |
|---|---|
| Q, K y V son tres proyecciones aprendidas de la misma entrada. | Q busca, K permite comparar y V aporta la información que se mezcla. |
| La atención compara todos los pares permitidos de posiciones. | $QK^T/\ \ d_k$ produce puntuaciones; softmax las convierte en pesos. |
| La máscara causal hace honesta la generación. | Una posición no puede mirar tokens futuros, así que esos pesos acaban siendo cero. |
| Multi-head attention permite varias miradas en paralelo. | Distintas cabezas pueden aprender patrones de relación diferentes y luego recombinarse. |

## Para saber más


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

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

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

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

Holtzman, A., Buys, J., Du, L., Forbes, M. y Choi, Y. (2020). The curious case of neural text degeneration. En *International Conference on Learning Representations*. https://openreview.net/forum?id=rygGQyrFvH

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

Vert, A. (2026, 13 de mayo). *AI 101: Your Ultimate Guide to Attention: Mechanism, QKV, and KV Cache*. Turing Post. https://www.turingpost.com/p/your-ultimate-guide-to-attention-mechanism-qkv-and-kv-cache

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. 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 paper define la atención como $\mathrm{softmax}(QK^T/\ \ d_k)V$, que es la fórmula que vamos a desmenuzar aquí. ↩

2. Vert, A. (2026, 13 de mayo). *AI 101: Your Ultimate Guide to Attention: Mechanism, QKV, and KV Cache*. Turing Post. https://www.turingpost.com/p/your-ultimate-guide-to-attention-mechanism-qkv-and-kv-cache. Consultado el 10 de junio de 2026. ↩

3. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org. Las capas lineales y las composiciones de funciones diferenciables forman la base matemática de las redes profundas. ↩

4. Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer. Bishop explica softmax como transformación de puntuaciones en probabilidades normalizadas, una pieza común en modelos probabilísticos y clasificación. ↩

5. 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 es un ejemplo de Transformer decoder autoregresivo a gran escala. ↩

6. 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 uso de Transformer encoder bidireccional para comprensión de lenguaje. ↩

7. Alammar, J. (2018). *The Illustrated Transformer*. https://jalammar.github.io/illustrated-transformer/. Recurso visual para seguir el flujo de Q, K, V y multi-head attention. ↩

8. Rush, A. M. (2018). *The Annotated Transformer*. https://nlp.seas.harvard.edu/annotated-transformer/. Implementación anotada del Transformer original con código y explicación matemática. ↩

9. 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. La herramienta permite visualizar componentes de un GPT-2 pequeño en navegador. ↩

.0. Holtzman, A., Buys, J., Du, L., Forbes, M. y Choi, Y. (2020). The curious case of neural text degeneration. En *International Conference on Learning Representations*. https://openreview.net/forum?id=rygGQyrFvH. El artículo analiza cómo distintas estrategias de muestreo afectan a la calidad del texto generado. ↩