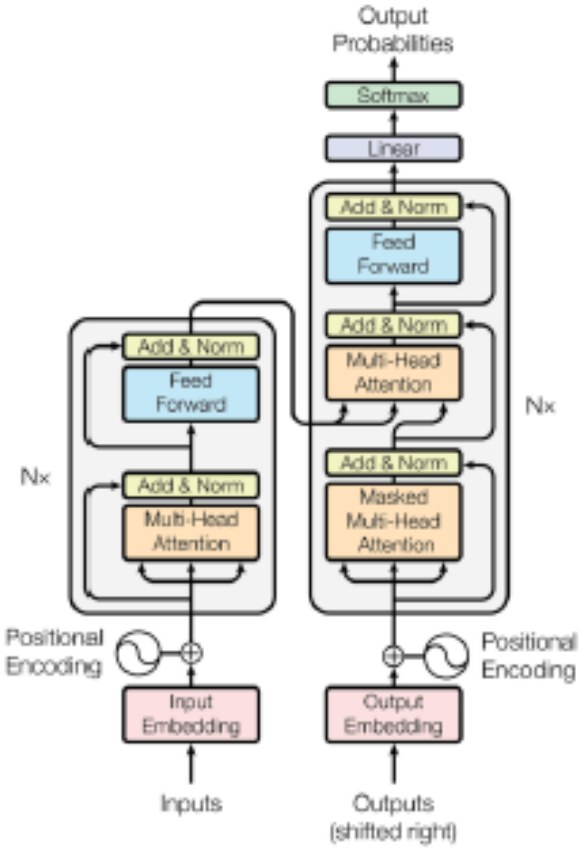
LLAMA desde cero
(LLAMA 1 y LLAMA 2)

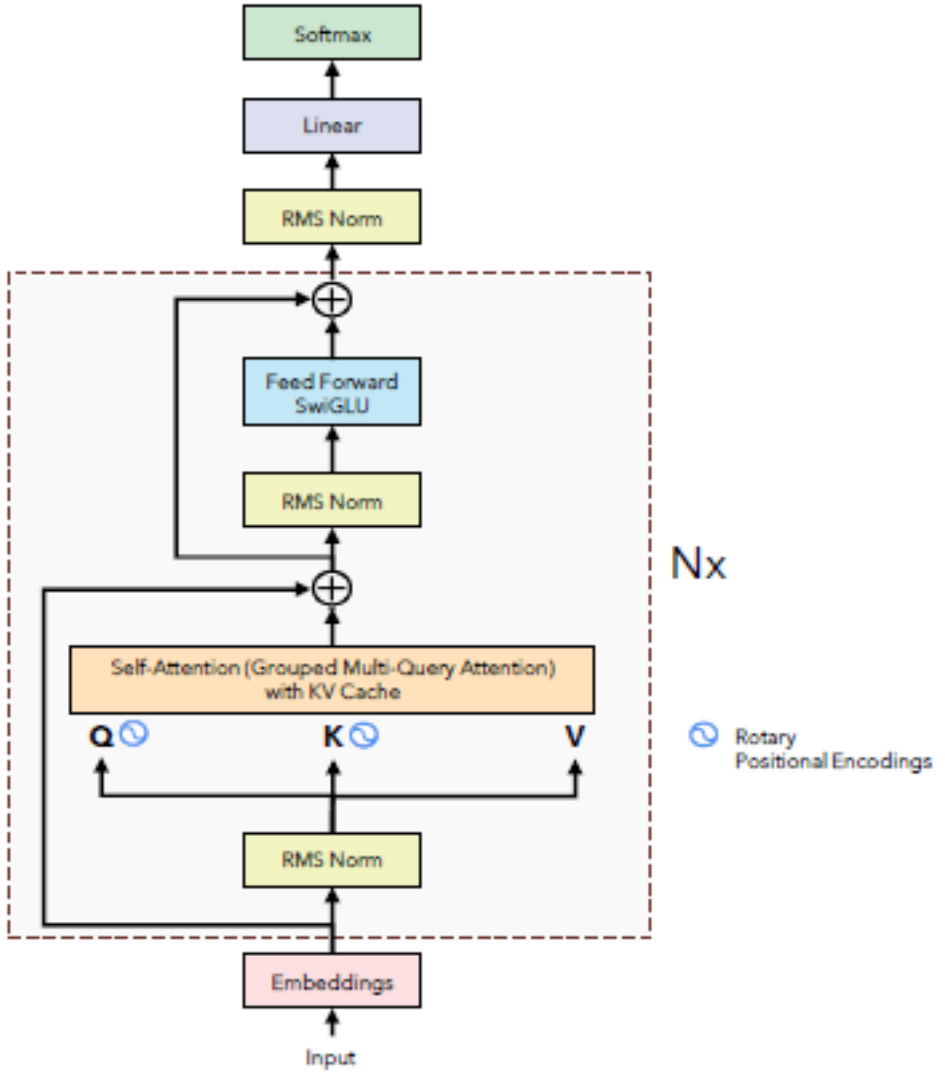
- Diferencias arquitectónicas entre el Transformer estándar y LLAMA.
- Normalización RMS (con revisión de la Normalización por Capas).
- Embeddings posicionales rotatorios.
- KV-Cache.
- Atención Multi-Query.
- Atención Multi-Query agrupada.
- Función de Activación SwiGLU.

$$e^{ix} = \cos x + i \sin x$$

Transformer vs LLaMA



Transformer
("Attention is all you need")



LLaMA

Modelos LlaMA 1

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

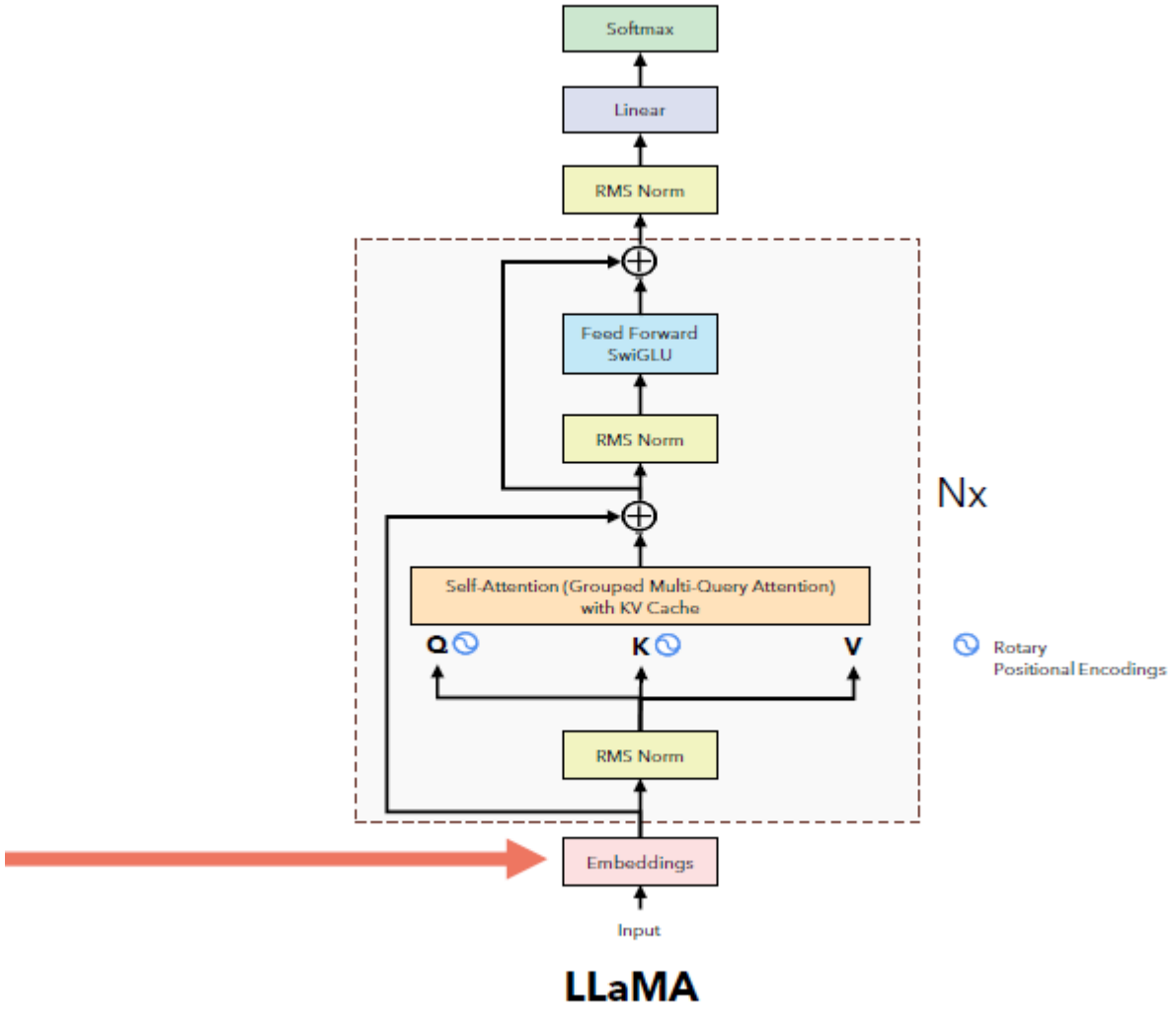
Table 2: **Model sizes, architectures, and optimization hyper-parameters.**

Modelos LlaMA 2

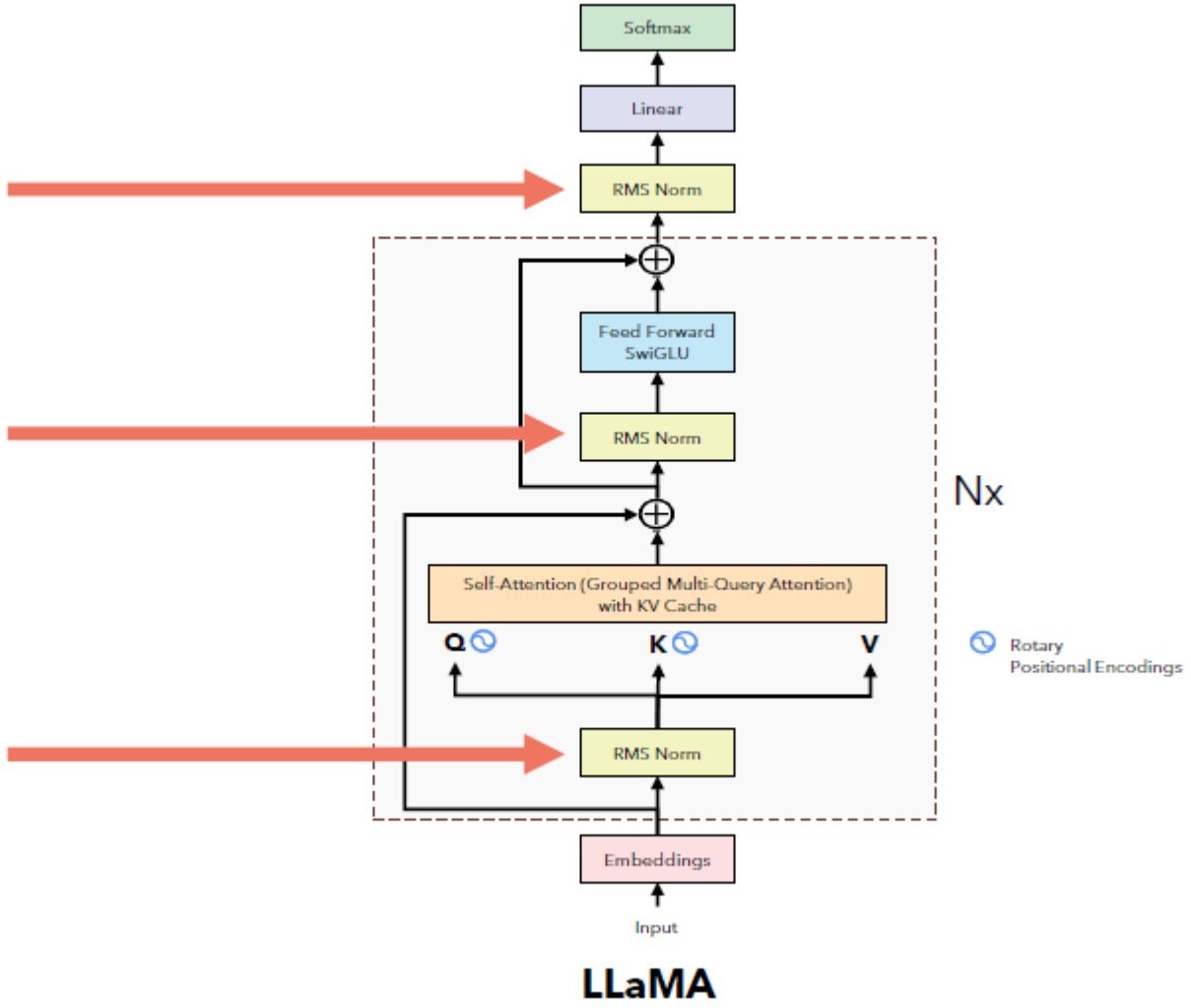
	Training Data	Params	Context Length	GQA	Tokens	LR
Llama 1	*See Touvron et al. (2023)*	7B	2k	✗	1.0T	3.0×10^{-4}
		13B	2k	✗	1.0T	3.0×10^{-4}
		33B	2k	✗	1.4T	1.5×10^{-4}
		65B	2k	✗	1.4T	1.5×10^{-4}
Llama 2	*A new mix of publicly available online data*	7B	4k	✗	2.0T	3.0×10^{-4}
		13B	4k	✗	2.0T	3.0×10^{-4}
		34B	4k	✓	2.0T	1.5×10^{-4}
		70B	4k	✓	2.0T	1.5×10^{-4}

Table 1: Llama 2 family of models. Token counts refer to pretraining data only. All models are trained with a global batch-size of 4M tokens. Bigger models — 34B and 70B — use Grouped-Query Attention (GQA) for improved inference scalability.

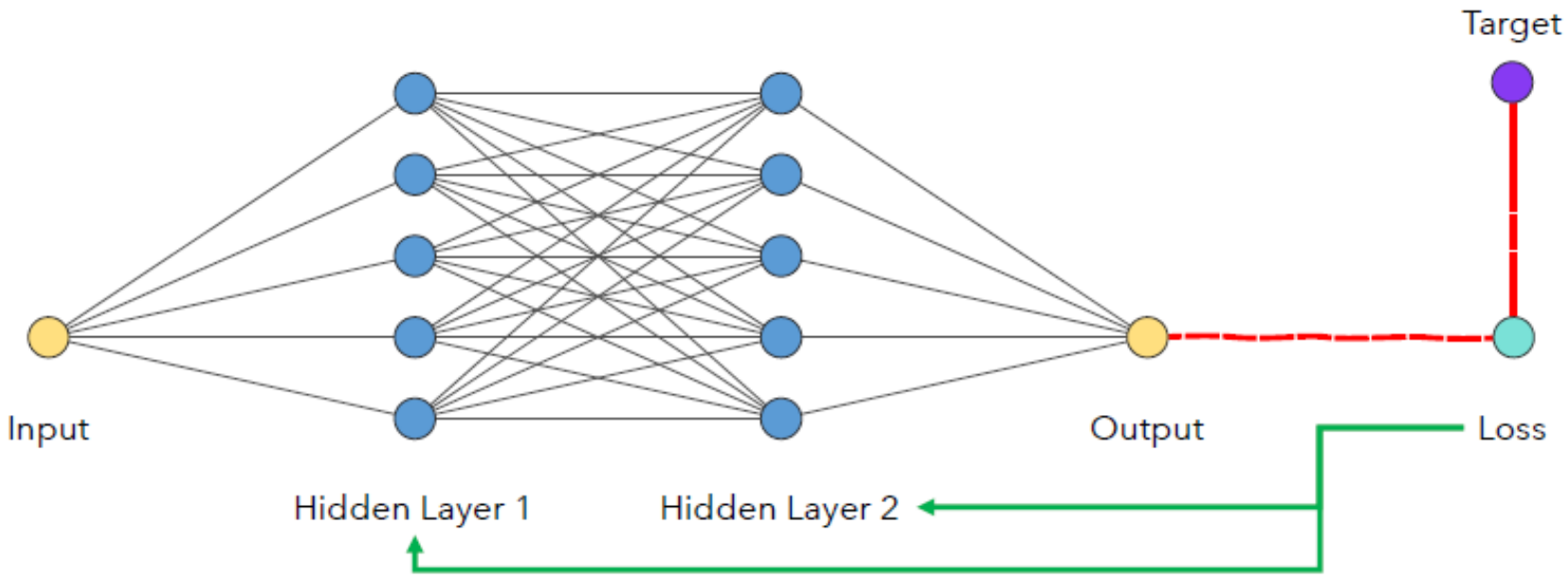
¡Revisemos los Embeddings!



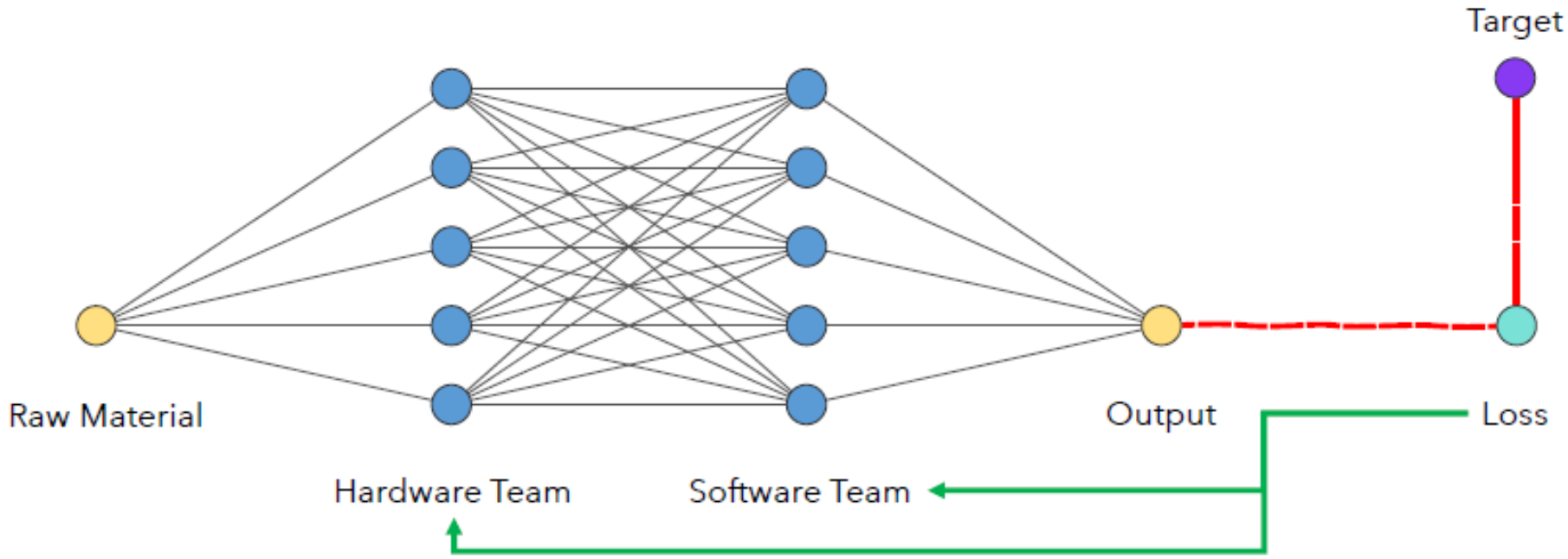
¿Qué es la normalización?



¿Qué es la normalización?
Revisemos las redes neuronales

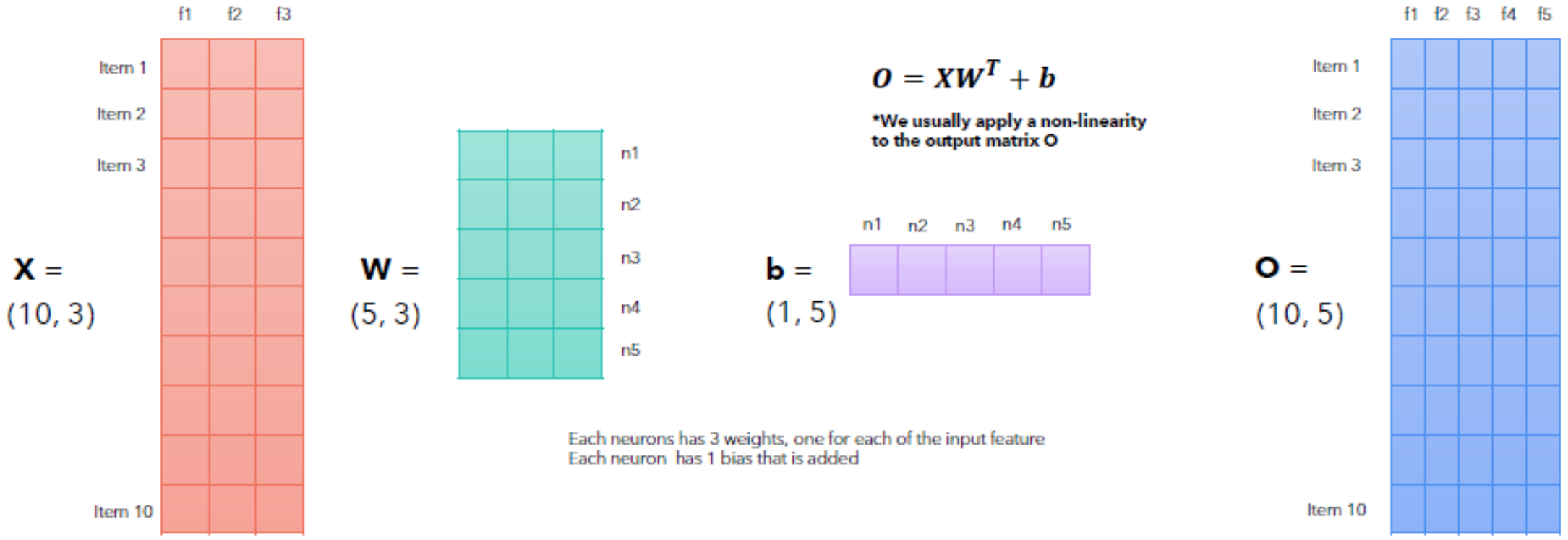


Un paralelo simple:
El mal CEO en una fábrica de teléfonos



¿Qué es la normalización?
Revisemos las **matemáticas** de las redes neuronales.

Supongamos que tenemos una capa lineal, definida como **nn.Linear(in_features=3, out_features=5, bias=**True). Esta capa lineal creará dos matrices, llamadas **W** (peso) y **b** (sesgo). Si tenemos una entrada **X** de forma (10, 3), la salida **O** será (10, 5). Pero, ¿cómo ocurre esto matemáticamente?

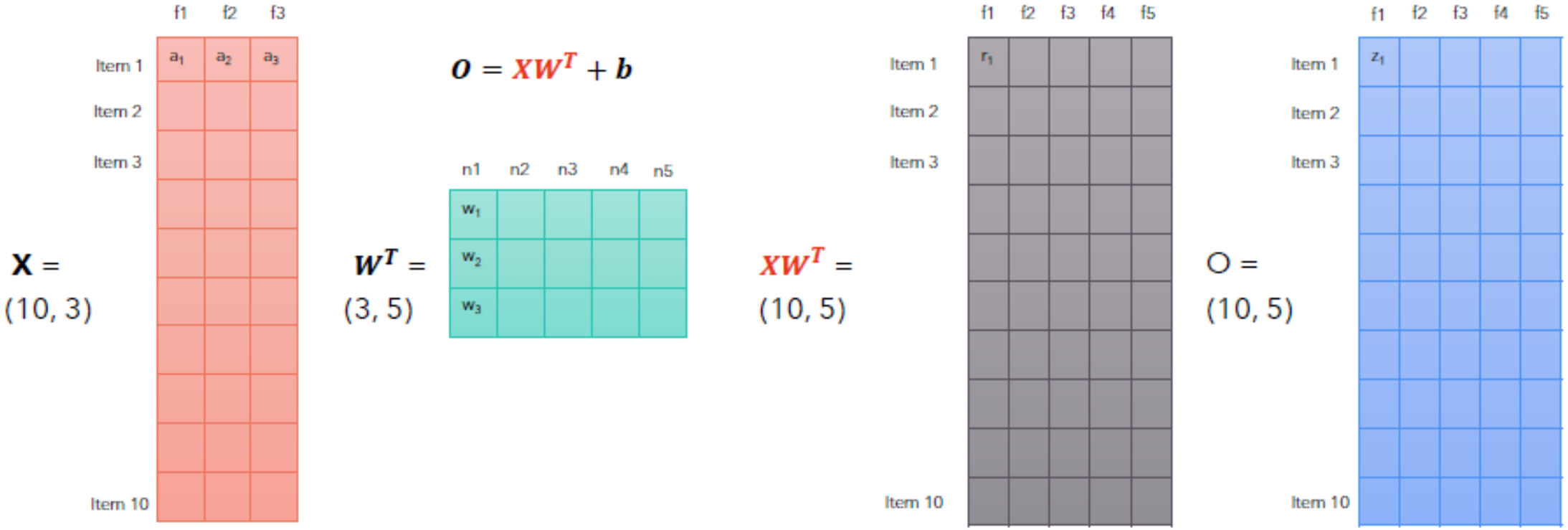


¡Revisemos las matemáticas de las redes neuronales!

$$z_1 = (r_1 + b_1) = (\sum_{i=1}^{3} a_i w_i + b_1)$$

La salida de la neurona 1 para el ítem 1 solo depende de las características del ítem 1. Usualmente aplicamos una no linealidad como la función ReLU a la salida $z1z\ 1$; $z1z\ 1$ se refiere a la activación de la neurona 1 respecto al ítem de datos 1.

$$\boldsymbol{O} = \boldsymbol{XW^T} + \boldsymbol{b}$$



Revisemos las matemáticas de las redes neuronales!

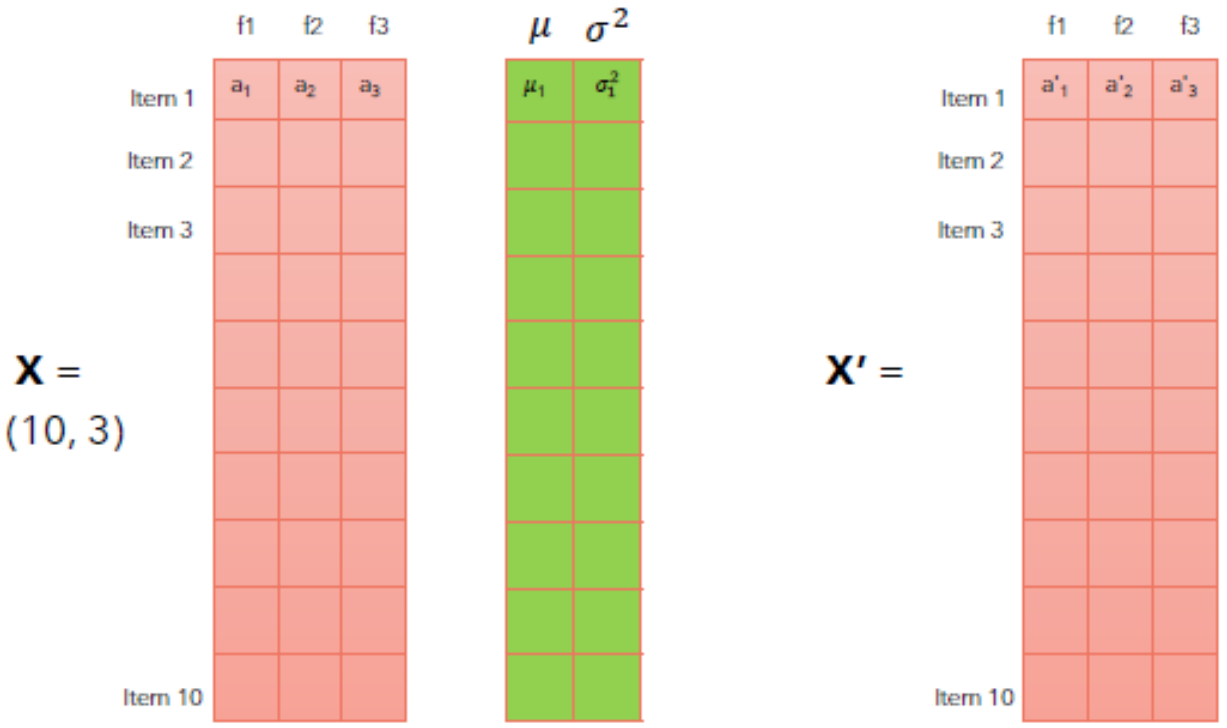
- La salida de una neurona para un ítem depende de las características de dicho ejemplo y de los parámetros de la neurona.

- Podemos pensar en la entrada a una neurona como la salida de una capa lineal anterior.

- Si la capa anterior, después de que sus pesos se actualizan debido al descenso de gradiente, cambia drásticamente su salida, la siguiente capa tendrá su entrada cambiada drásticamente, por lo que se verá forzada a reajustar sus pesos drásticamente en el siguiente paso del descenso de gradiente.

- El fenómeno por el cual las distribuciones de los nodos internos (neuronas) de una red neuronal cambian se conoce como **Cambio Interno de Covariantes** (Internal Covariate Shift). Queremos evitarlo porque hace que el entrenamiento de la red sea más lento, ya que las neuronas se ven forzadas a reajustar drásticamente sus pesos en una dirección u otra debido a los cambios drásticos en las salidas de las capas anteriores.

¡Una solución a las activaciones saltarinas: normalización de capas!



$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

- Cada ítem se actualiza con su valor normalizado, lo que lo convierte en una distribución normal con media 0 y varianza de 1.
- Los dos parámetros **gamma** y **beta** son parámetros entrenables que permiten al modelo "amplificar" la escala de cada característica o aplicar una traducción a la característica de acuerdo con las necesidades de la función de pérdida.

- Con la normalización por lotes (batch normalization), normalizamos por **columnas (características)**.
- Con la normalización por capas (layer normalization), normalizamos por **filas (ítems de datos)**.

Normalización de Raíz Cuadrada Media (Root Mean Square Normalization)

Root Mean Square Layer Normalization

Biao Zhang[1] **Rico Sennrich**[2,1]
[1]School of Informatics, University of Edinburgh
[2]Institute of Computational Linguistics, University of Zurich
B.Zhang@ed.ac.uk, sennrich@cl.uzh.ch

4 RMSNorm

A well-known explanation of the success of LayerNorm is its re-centering and re-scaling invariance property. The former enables the model to be insensitive to shift noises on both inputs and weights, and the latter keeps the output representations intact when both inputs and weights are randomly scaled. In this paper, we hypothesize that the re-scaling invariance is the reason for success of LayerNorm, rather than re-centering invariance.

We propose RMSNorm which only focuses on re-scaling invariance and regularizes the summed inputs simply according to the root mean square (RMS) statistic:

$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where} \ \ \text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}. \tag{4}$$

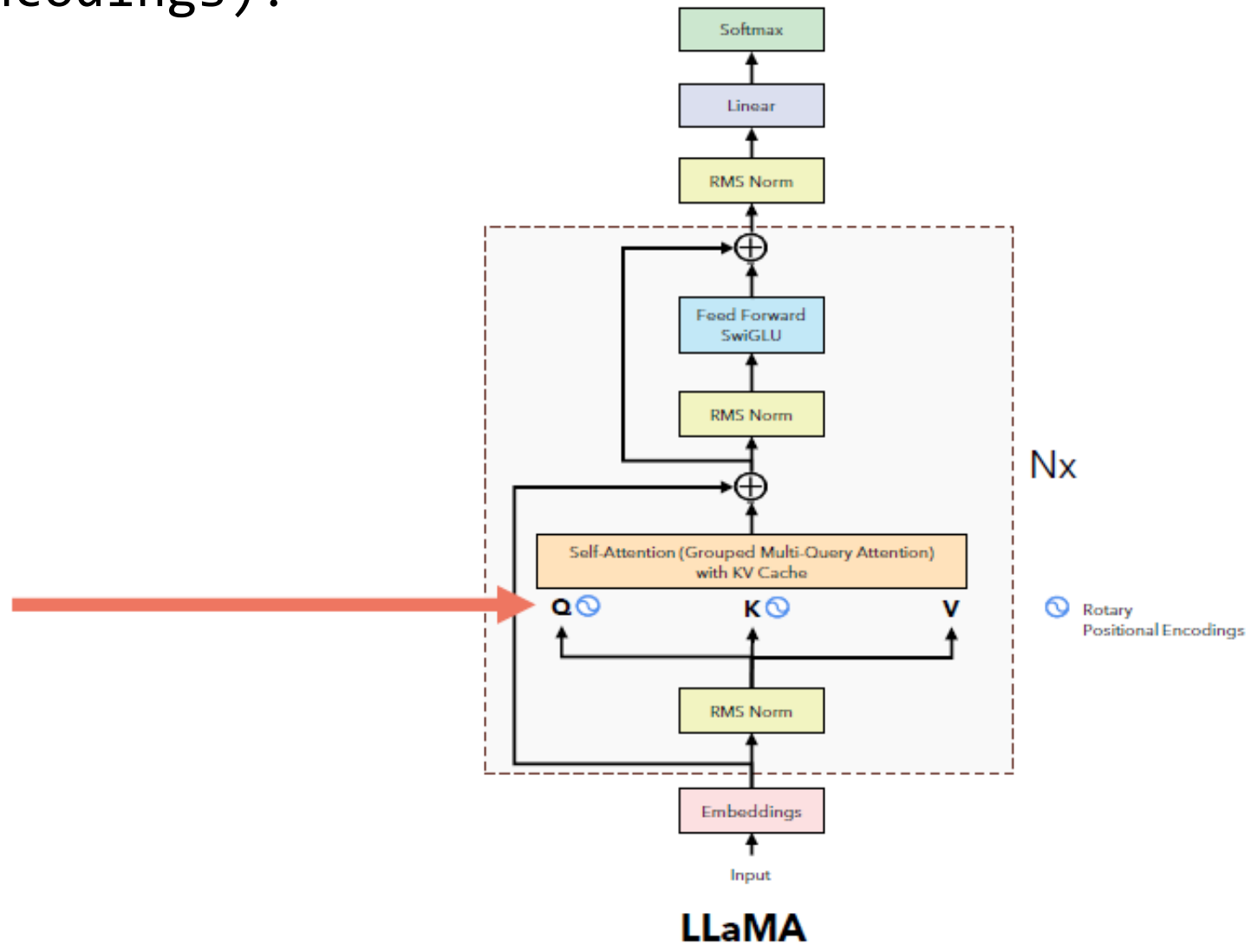
Intuitively, RMSNorm simplifies LayerNorm by totally removing the mean statistic in Eq. (3) at the cost of sacrificing the invariance that mean normalization affords. When the mean of summed inputs is zero, RMSNorm is exactly equal to LayerNorm. Although RMSNorm does not re-center

Al igual que con Layer Normalization, también tenemos un parámetro entrenable **gamma (g** en la fórmula a la izquierda) que se multiplica por los valores normalizados.

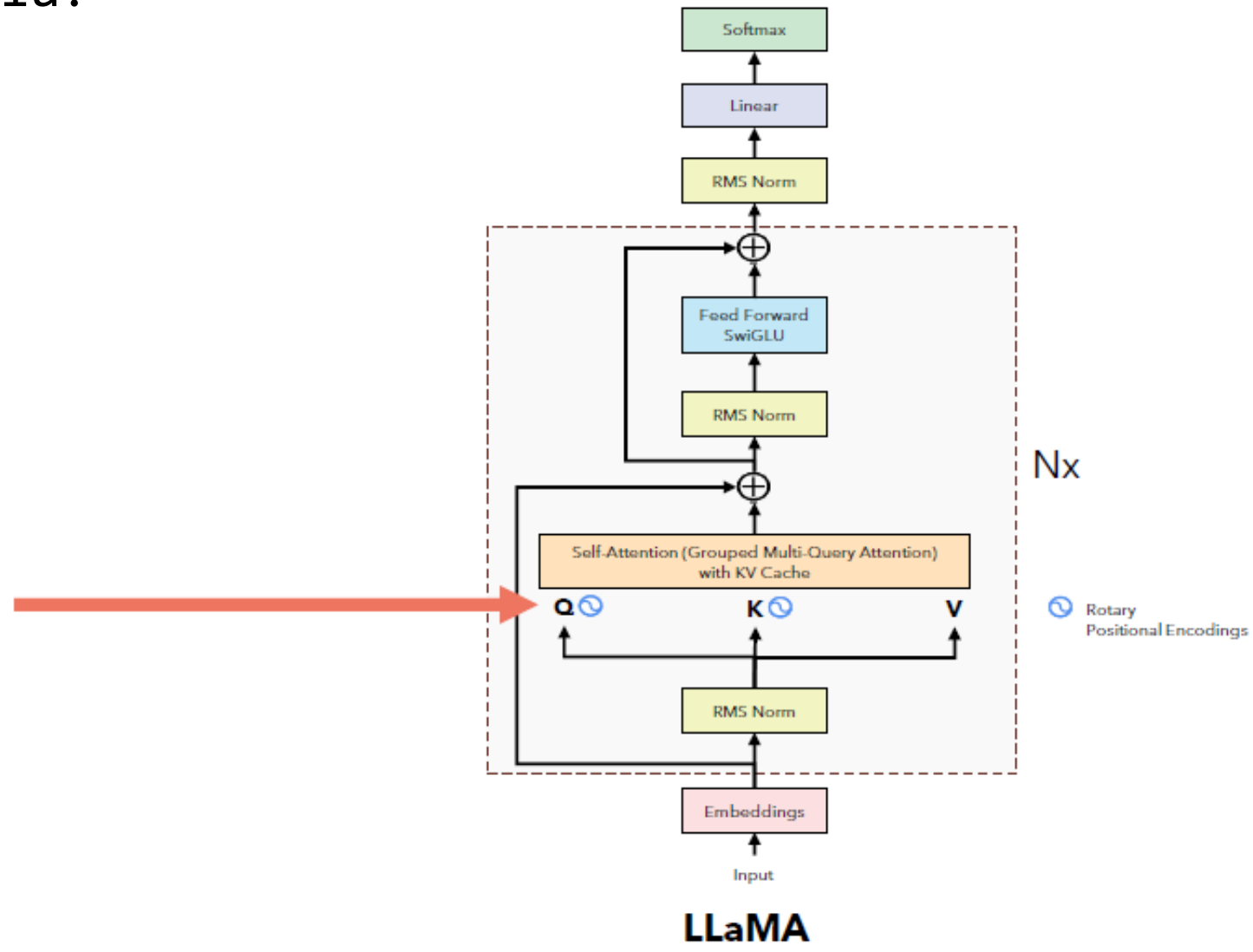
¿Por qué RMSNorm?

- Requiere menos cálculos en comparación con la Normalización por Capas (Layer Normalization).
- Funciona bien en la práctica.

¡Revisemos los Codificadores Posicionales (Positional Encodings)!



¿Qué es la Codificación Posicional Rotatoria?



¿Cuál es la diferencia entre las codificaciones posicionales absolutas y las relativas?

- Las codificaciones posicionales absolutas son vectores fijos que se añaden a la incrustación de un token para representar su posición absoluta en la oración. Por lo tanto, trabajan con **un token a la vez**. Puedes pensar en esto como el par (latitud, longitud) en un mapa: cada punto en la Tierra tendrá un par único.
- Las codificaciones posicionales relativas, por otro lado, trabajan con dos tokens a la vez y están involucradas cuando calculamos la atención: dado que el mecanismo de atención captura la "intensidad" de cuánto están relacionadas dos palabras entre sí, las codificaciones posicionales relativas le indican al mecanismo de atención la **distancia** entre las dos palabras involucradas. Entonces, dado **dos tokens**, creamos un vector que representa su distancia.
- Las codificaciones posicionales relativas fueron introducidas en el siguiente artículo:

Self-Attention with Relative Position Representations

Peter Shaw
Google
petershaw@google.com

Jakob Uszkoreit
Google Brain
usz@google.com

Ashish Vaswani
Google Brain
avaswani@google.com

Codificaciones Posicionales **Absolutas** De "Attention is all you need"

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}}$$

Codificaciones Posicionales **Relativas** De "Self-Attention with relative positional

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

Codificaciones Posicionales Rotatorias

RoFormer: Enhanced Transformer with Rotary Position Embedding

Jianlin Su
Zhuiyi Technology Co., Ltd.
Shenzhen
bojonesu@wezhuiyi.com

Yu Lu
Zhuiyi Technology Co., Ltd.
Shenzhen
julianlu@wezhuiyi.com

Shengfeng Pan
Zhuiyi Technology Co., Ltd.
Shenzhen
nickpan@wezhuiyi.com

Ahmed Murtadha
Zhuiyi Technology Co., Ltd.
Shenzhen
mengjiayi@wezhuiyi.com

Bo Wen
Zhuiyi Technology Co., Ltd.
Shenzhen
brucewen@wezhuiyi.com

Yunfeng Liu
Zhuiyi Technology Co., Ltd.
Shenzhen
glenliu@wezhuiyi.com

Codificaciones Posicionales Rotatorias: el producto interno

- El **producto punto** utilizado en el mecanismo de atención es un tipo de producto interno, que puede considerarse como una generalización del producto punto.

- ¿Podemos encontrar un producto interno sobre los dos vectores **q** (consulta) y **k** (clave) utilizados en el mecanismo de atención que solo dependa de los dos vectores y de la distancia relativa del token que representan?

Under the case of $d = 2$, we consider two-word embedding vectors $\boldsymbol{x}_q$, $\boldsymbol{x}_k$ corresponds to query and key and their position m and n, respectively. According to eq. (1), their position-encoded counterparts are:

$$\begin{aligned} \boldsymbol{q}_m &= f_q(\boldsymbol{x}_q, m), \\ \boldsymbol{k}_n &= f_k(\boldsymbol{x}_k, n), \end{aligned} \tag{20}$$

where the subscripts of $\boldsymbol{q}_m$ and $\boldsymbol{k}_n$ indicate the encoded positions information. Assume that there exists a function g that defines the inner product between vectors produced by $f_{\{q,k\}}$:

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n) \rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, n - m), \tag{21}$$

Codificaciones Posicionales Rotatorias: el producto interno

- Podemos definir una función **g** como la siguiente, que solo depende de los dos vectores de embeddings **q** y **k** y de su distancia relativa:

$$f_q(x_m, m) = (W_q x_m) e^{im\theta}$$

$$f_k(x_n, n) = (W_k x_n) e^{in\theta}$$

$$g(x_m, x_n, m-n) = \mathrm{Re}\left[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta}\right]$$

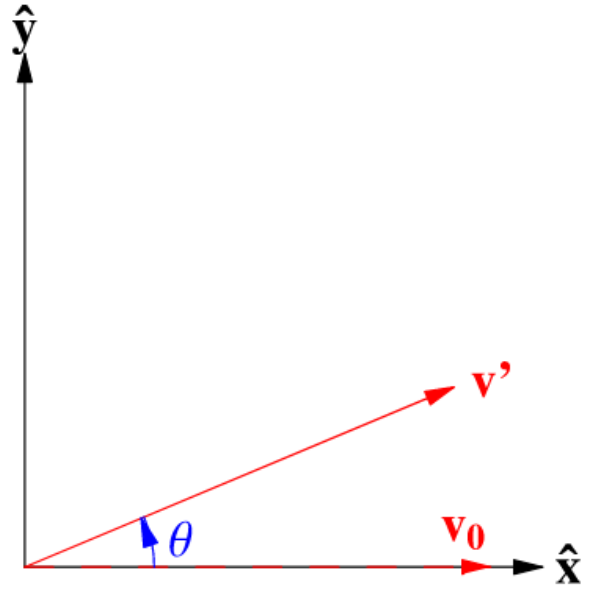
* **Conjugado** del número complejo

- Usando la **fórmula de Euler**, podemos escribirlo en su forma matricial:

$$f_{(q,k)}(x_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{q,k}^{(11)} & W_{q,k}^{(12)} \\ W_{q,k}^{(21)} & W_{q,k}^{(22)} \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix}$$

- Matriz de rotación en un espacio 2D, de ahí el nombre de codificaciones posicionales **rotatorias.**

Codificaciones Posicionales Rotatorias: la matriz de rotación



In $\mathbb{R}^2$, consider the matrix that rotates a given vector $\mathbf{v}_0$ by a counterclockwise angle θ in a fixed coordinate system. Then

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{1}$$

so

$$\mathbf{v}' = R_\theta \mathbf{v}_0. \tag{2}$$

Codificaciones Posicionales Rotatorias: la forma general

Dado que la matriz es **dispersa**, no es conveniente usarla para calcular las codificaciones posicionales.

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m}\boldsymbol{W}_{\{q,k\}}\boldsymbol{x}_m \tag{14}$$

where

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

is the rotary matrix with pre-defined parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$. A graphic illustration of RoPE is shown in Figure (1). Applying our RoPE to self-attention in Equation (2), we obtain:

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m}\boldsymbol{W}_q\boldsymbol{x}_m)^\intercal(\boldsymbol{R}^d_{\Theta,n}\boldsymbol{W}_k\boldsymbol{x}_n) = \boldsymbol{x}^\intercal\boldsymbol{W}_q R^d_{\Theta,n-m}\boldsymbol{W}_k\boldsymbol{x}_n \tag{16}$$

where $\boldsymbol{R}^d_{\Theta,n-m} = (\boldsymbol{R}^d_{\Theta,m})^\intercal\boldsymbol{R}^d_{\Theta,n}$. Note that $\boldsymbol{R}^d_\Theta$ is an orthogonal matrix, which ensures stability during the process of encoding position information. In addition, due to the sparsity of R^d_Θ, applying matrix multiplication directly as in Equation (16) is not computationally efficient; we provide another realization in theoretical explanation.

Codificaciones Posicionales Rotatorias: la forma computacionalmente eficiente

- Dado un token con el vector de embedding x, y la posición m del token dentro de la oración, así es como calculamos las codificaciones posicionales para el token.

3.4.2 Computational efficient realization of rotary matrix multiplication

Taking the advantage of the sparsity of $\boldsymbol{R}^d_{\Theta,m}$ in Equation (15), a more computational efficient realization of a multiplication of R^d_Θ and $\boldsymbol{x} \in \mathbb{R}^d$ is:

$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix} \tag{34}$$

Codificaciones Posicionales Rotatorias: decaimiento a largo plazo

Los autores calcularon un **límite superior** para el producto interno variando la distancia entre dos tokens y demostraron que este decae a medida que crece la distancia relativa. Esto significa que la "intensidad" de la relación entre dos tokens codificados con Codificaciones Posicionales Rotatorias será numéricamente más pequeña a medida que la distancia entre ellos aumente.

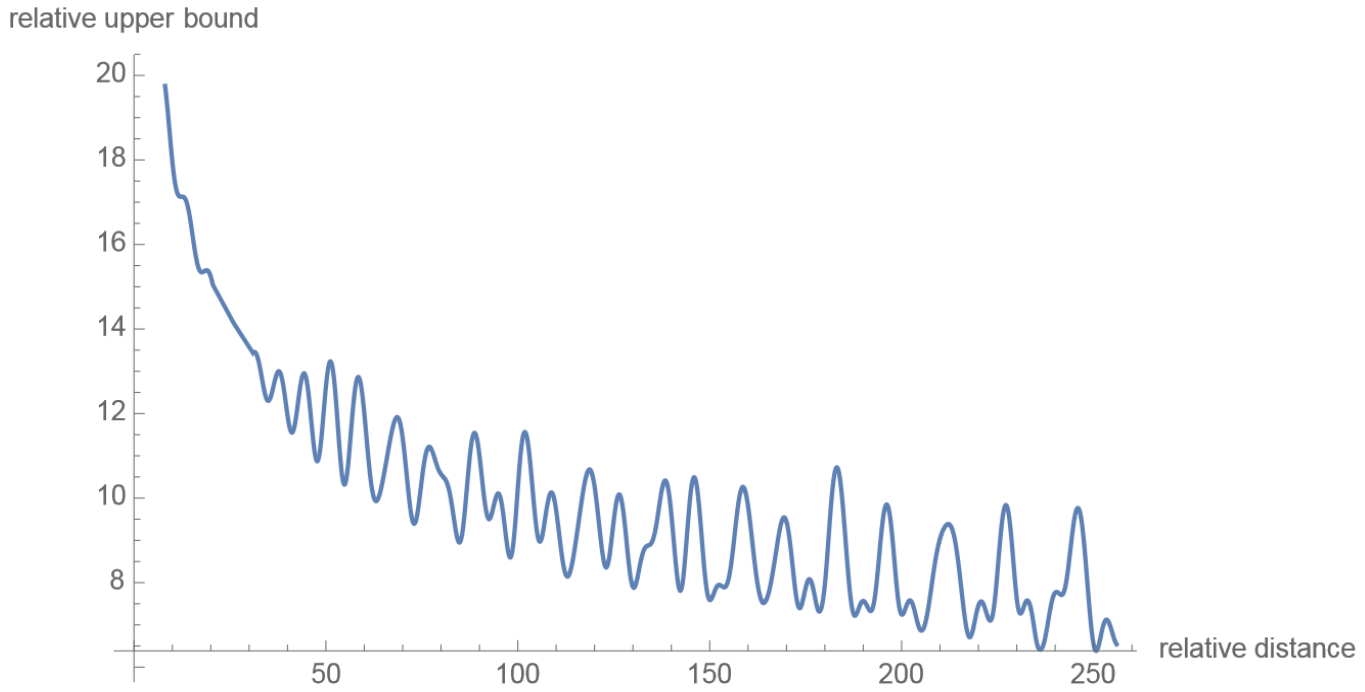
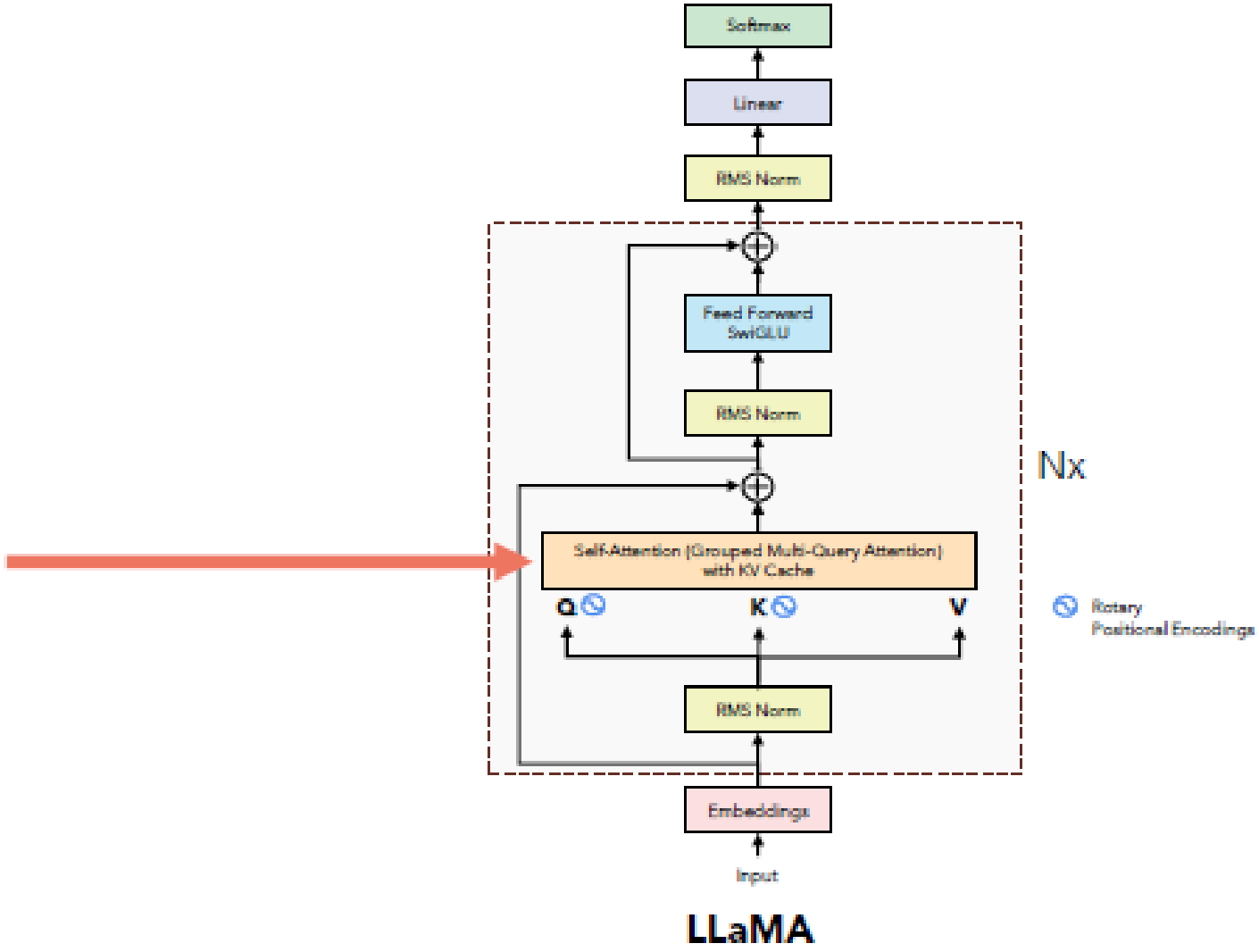


Figure 2: Long-term decay of RoPE.

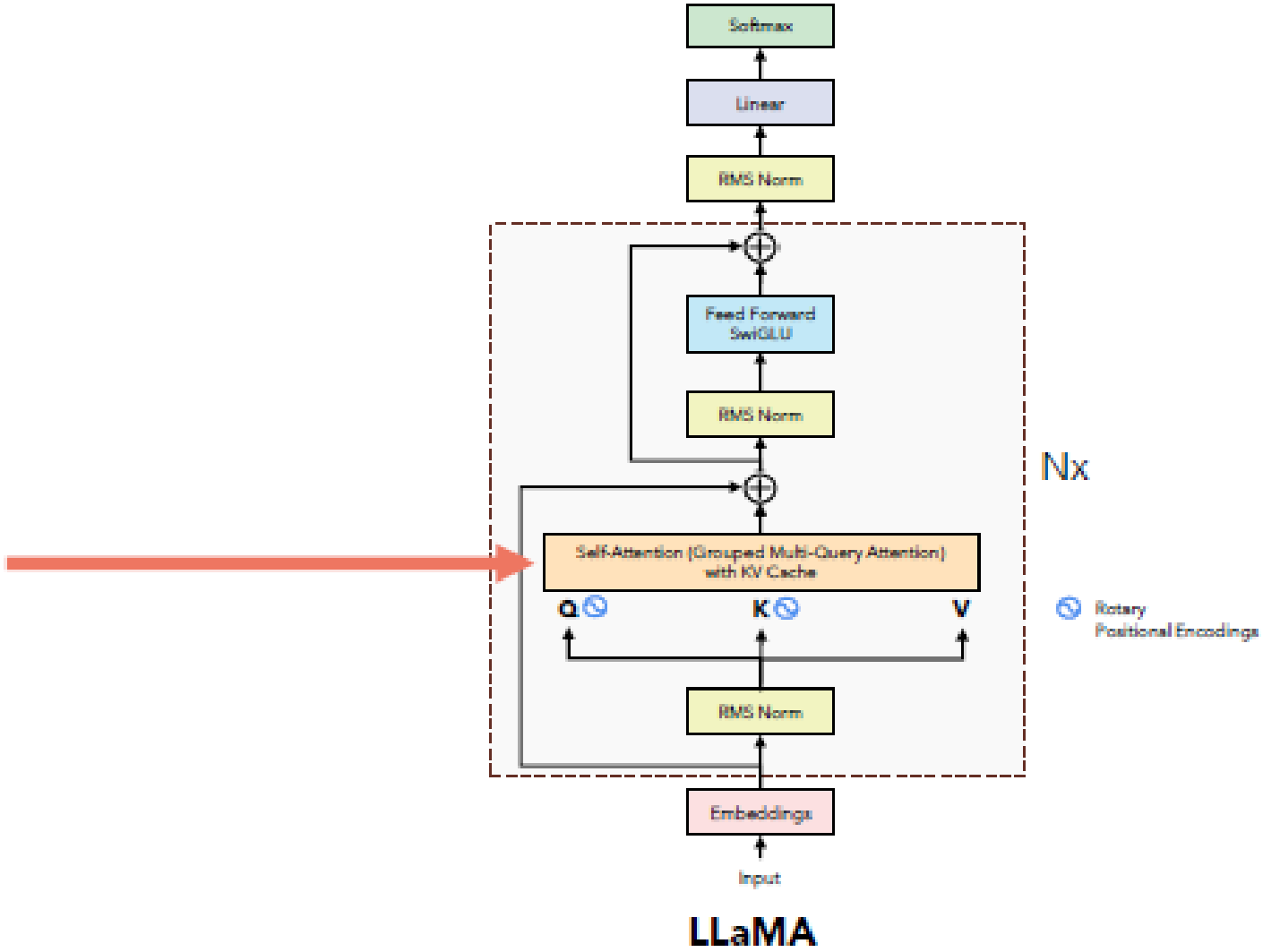
Codificaciones Posicionales Rotatorias: consideraciones prácticas

- Las codificaciones posicionales rotatorias **solo se aplican a la consulta (query) y a las claves (keys)**, pero no a los valores (values).
- Las codificaciones posicionales rotatorias se aplican después de que los vectores **q** y **k** han sido multiplicados por la matriz **W** en el mecanismo de atención, mientras que en el transformador clásico (vanilla transformer) se aplican antes.

¡Revisemos la Auto-Atención!



¿Qué es la Caché KV?

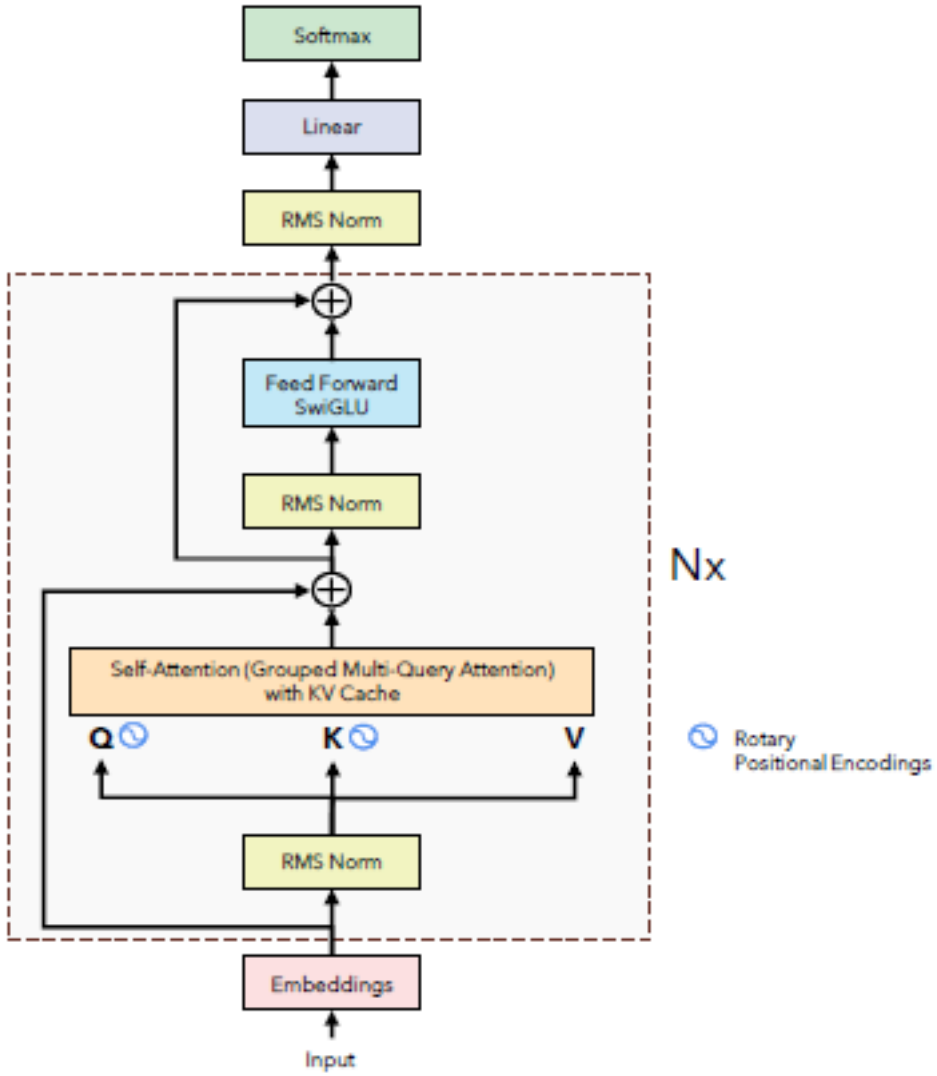


Tarea de Predicción del Siguiente Token

Imagina que queremos entrenar un modelo para escribir el 5.º Canto de la Divina Comedia de Dante Alighieri, del Infierno.

Amor, ch'al cor gentil ratto s'apprende,prese costui de la bella personache mi fu tolta; e 'l modo ancor m'offende.
Amor, ch'a nullo amato amar perdona,mi prese del costui piacer sì forte,che, come vedi, ancor non m'abbandona.
Amor condusse noi ad una morte:Caina attende chi a vita ci spense.

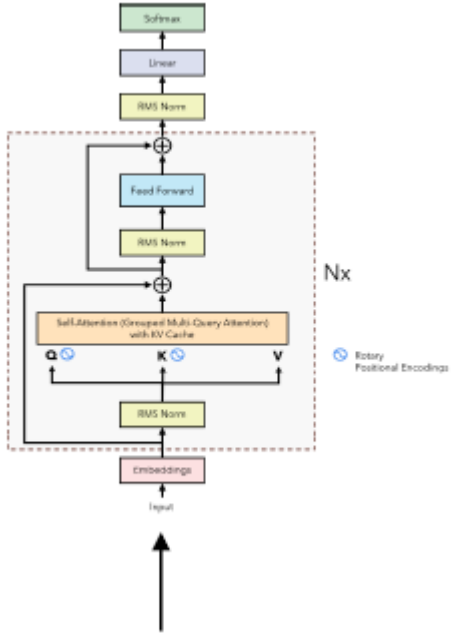
Amor, que rápidamente toma el gentil corazón,se apoderó de él a causa del bello cuerpoque me fue arrebatado; y aún me ofende la manera.
Amor, que a ninguno amado permite amar,me atrapó con tal fuerza en el placer de este hombre,que, como ves, aún no me abandona.
El amor nos llevó a una sola muerte:Caína espera a quien nos quitó la vida.



Tarea de Predicción del Siguiente Token

Objetivo: Amor que puede rápidamente apoderarse del gentil corazón [EOS]

Entrenamiento

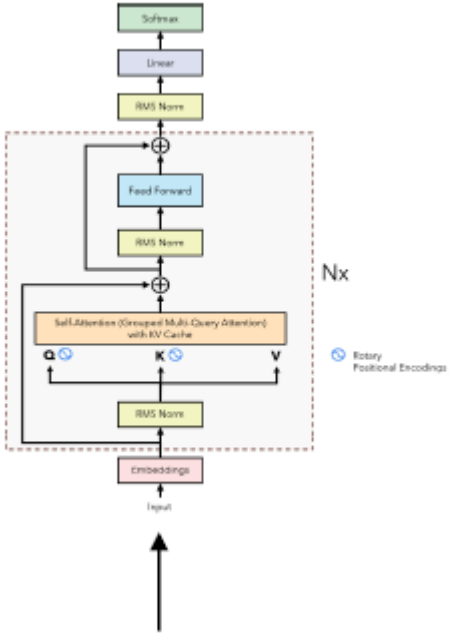


Entrada: [SOS] Amor que puede rápidamente apoderarse del gentil corazón

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor

Inferencia
T = 1

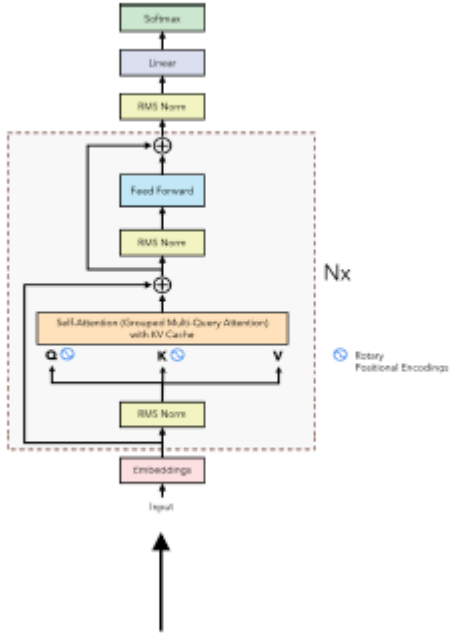


Entrada: [SOS]

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor que

Inferencia
T = 2

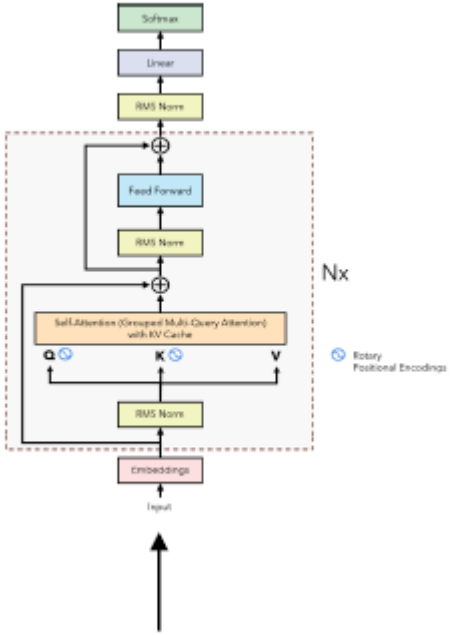


Entrada: [SOS] Amor

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor que puede

Inferencia
T = 3

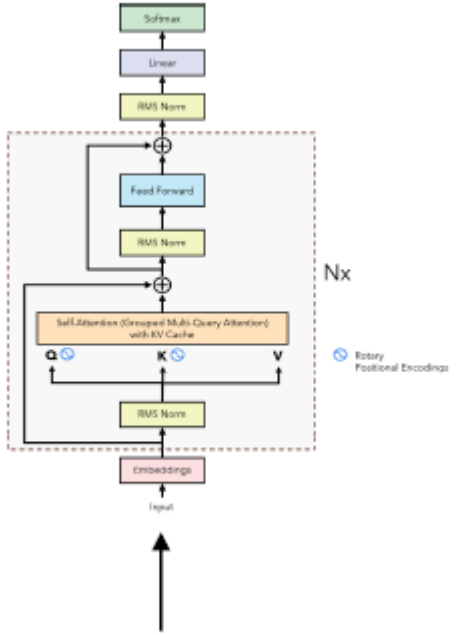


Entrada: [SOS] Amor que

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor que puede rápidamente

Inferencia
T = 4

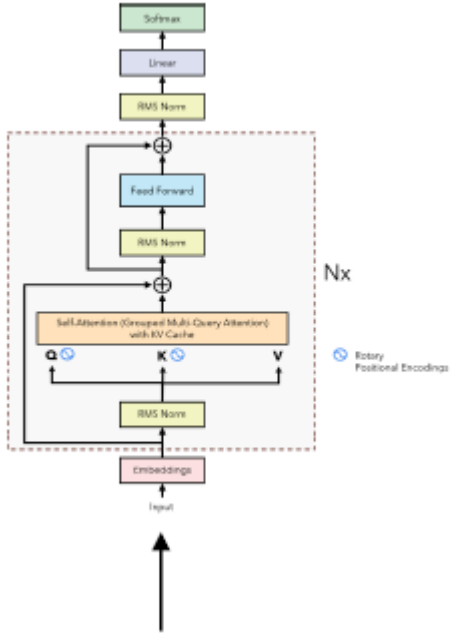


Entrada: [SOS] Amor que puede

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor que puede rápidamente apoderarse

Inferencia
T = 5

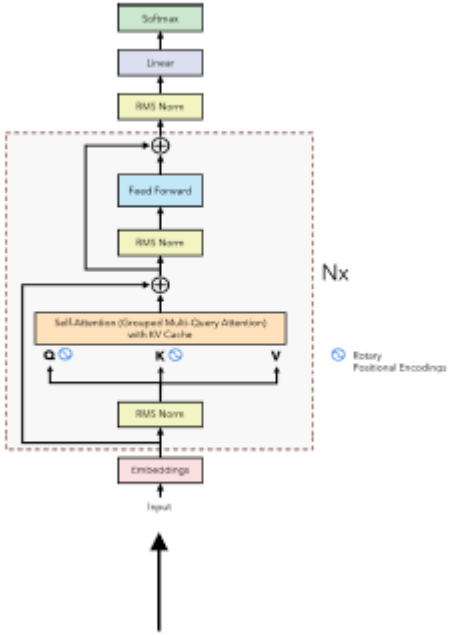


Entrada: [SOS] Amor que puede rápidamente

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor que puede rápidamente apoderarse del

Inferencia
T = 6

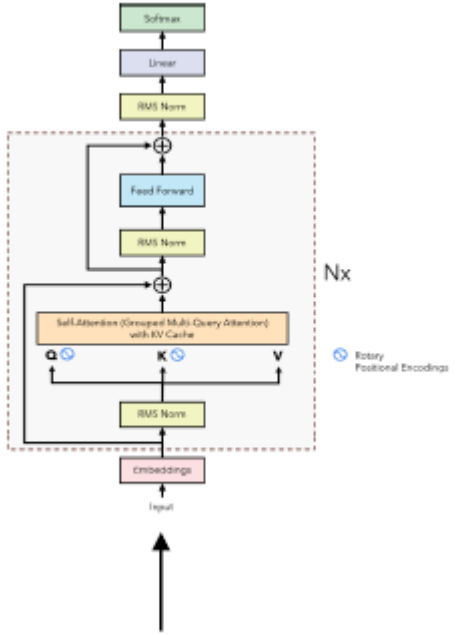


Entrada: [SOS] Amor que puede rápidamente apoderarse

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor que puede rápidamente apoderarse del gentil

Inferencia
T = 7

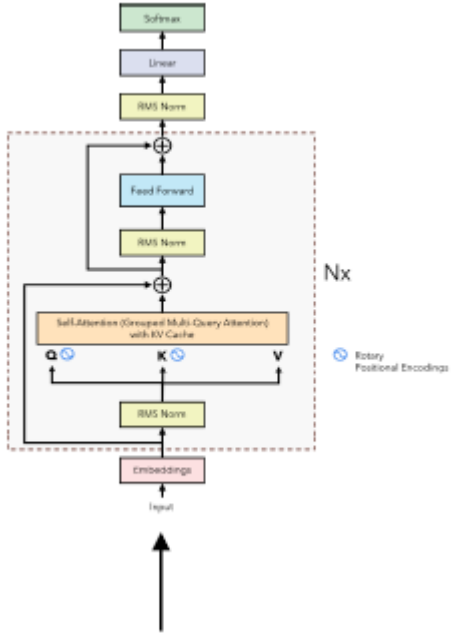


Entrada: [SOS] Amor que puede rápidamente apoderarse del

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor que puede rápidamente apoderarse del gentil corazón

Inferencia
T = 8

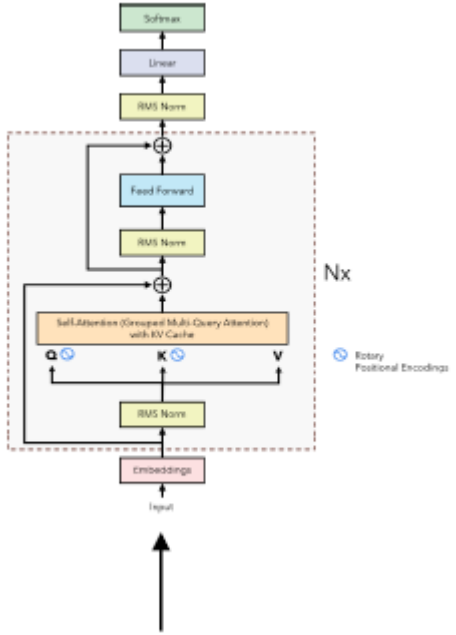


Entrada: [SOS] Amor que puede rápidamente apoderarse del gentil

Tarea de Predicción del Siguiente Token: Inferencia

Salida: Amor que puede rápidamente apoderarse del gentil corazón [EOS]

Inferencia
T = 9



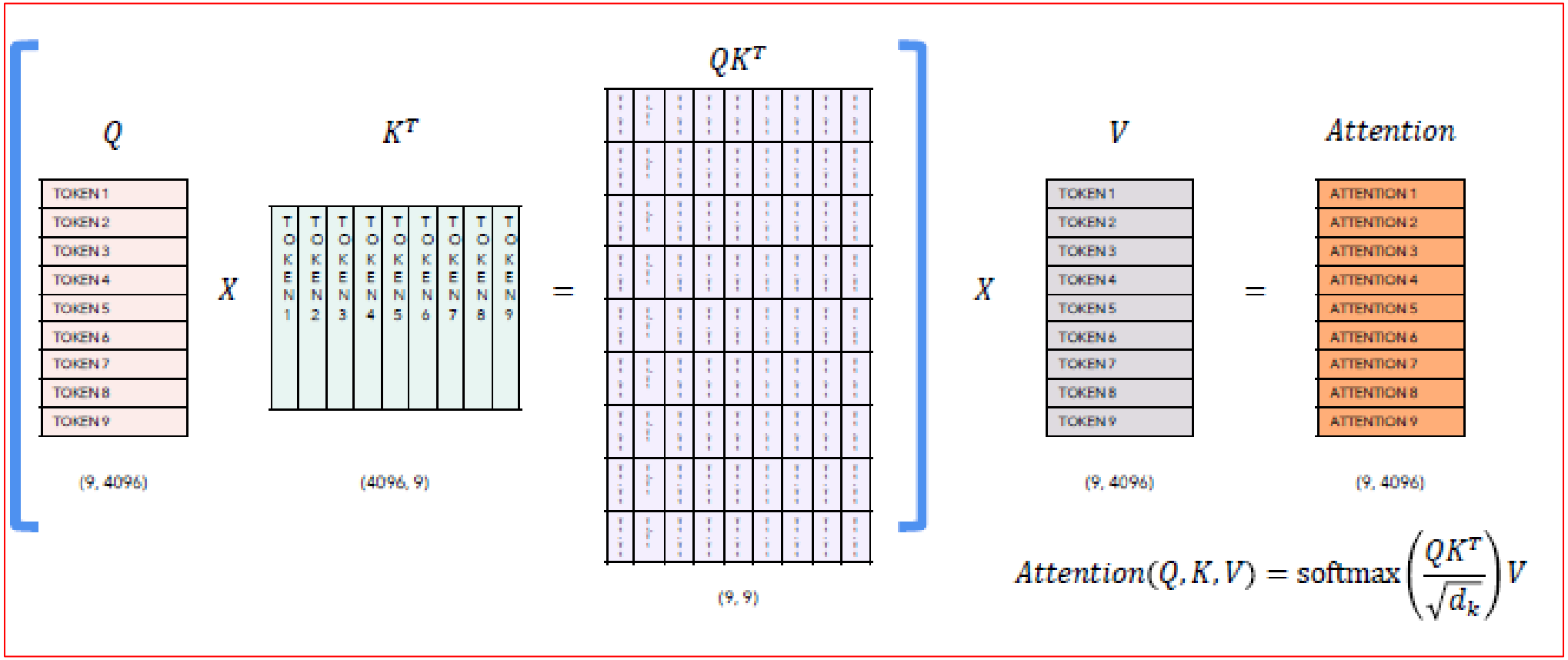
Entrada: [SOS] Amor que puede rápidamente apoderarse del gentil corazón

Tarea de Predicción del Siguiente Token: la motivación detrás de la caché KV

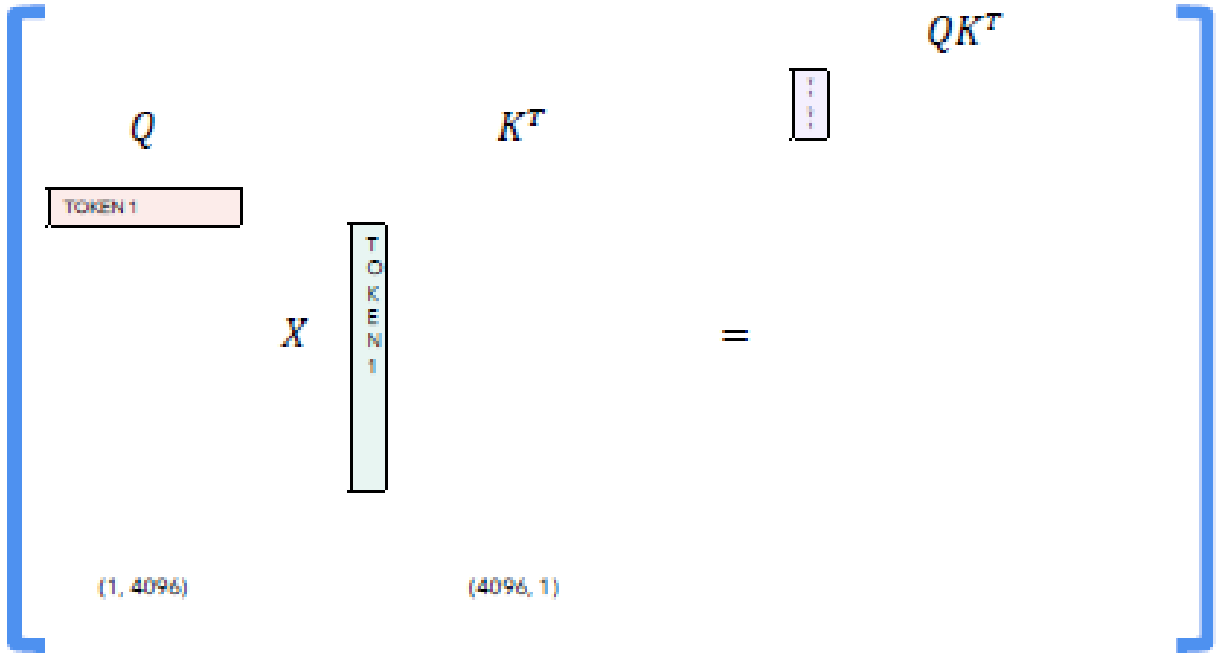
- En cada paso de la inferencia, solo nos interesa el **último token** generado por el modelo, porque ya tenemos los anteriores. Sin embargo, el modelo necesita acceso a todos los tokens anteriores para decidir qué token generar, ya que constituyen su contexto (o el "prompt").

- ¿Existe una manera de hacer que el modelo realice menos cálculos sobre los tokens que ya ha visto **durante la inferencia**? ¡SÍ! La solución es la **caché KV**.

Auto-Atención durante la Tarea de Predicción del Siguiente Token



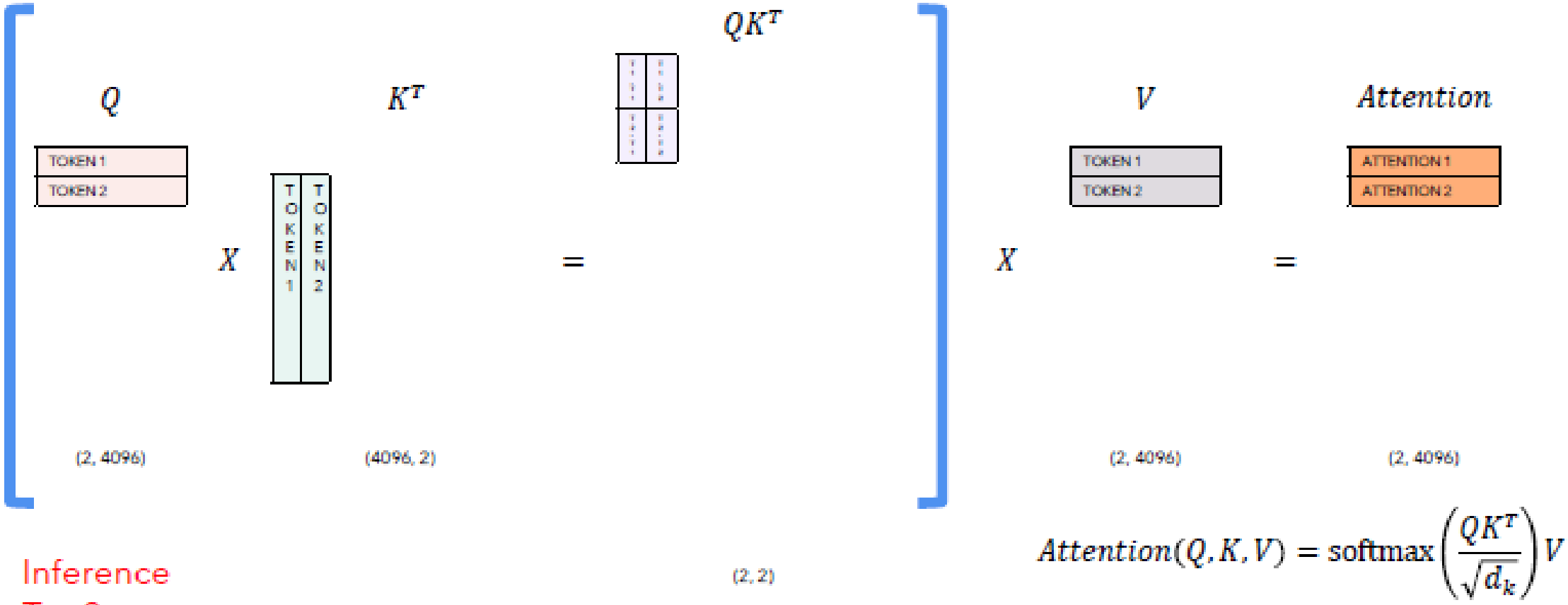
Auto-Atención durante la Tarea de Predicción del Siguiente Token



Inference
T = 1

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

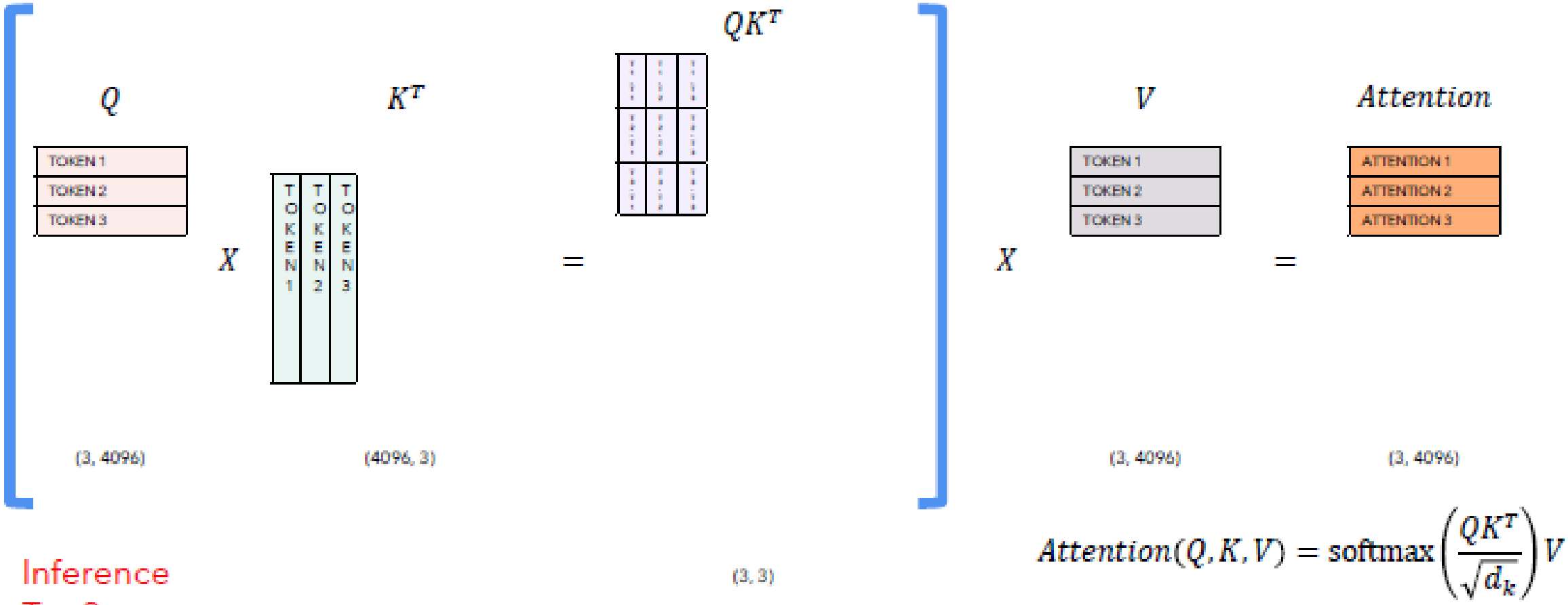
Auto-Atención durante la Tarea de Predicción del Siguiente Token



Inference
T = 2

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

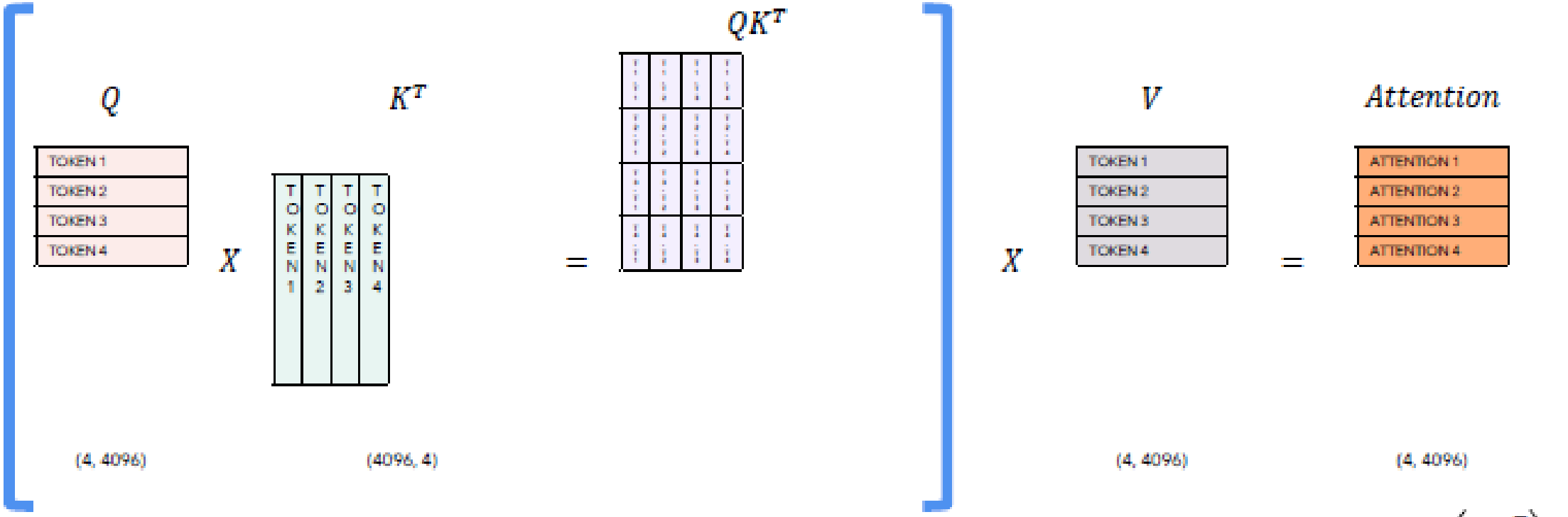
Auto-Atención durante la Tarea de Predicción del Siguiente Token



Inference
T = 3

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Auto-Atención durante la Tarea de Predicción del Siguiente Token



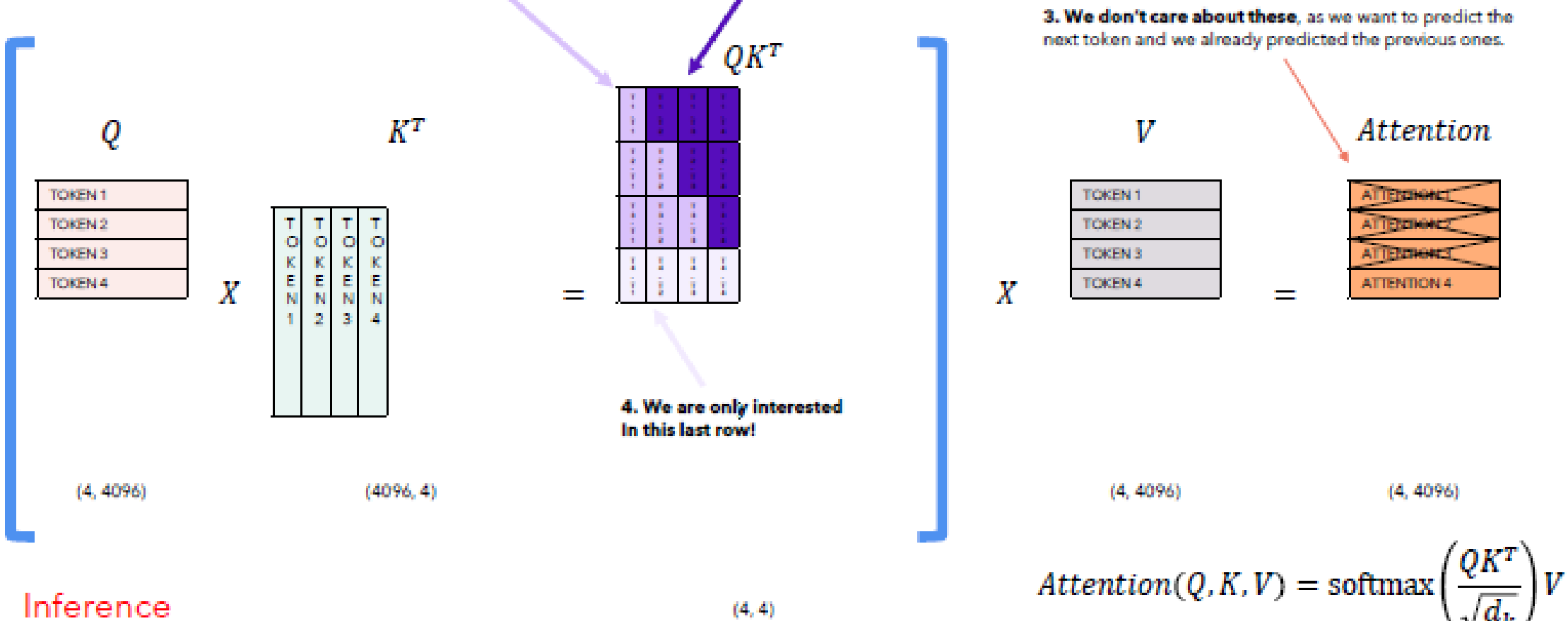
Inference
T = 4

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

1. We already computed these dot products In the previous steps. **Can we cache them**?

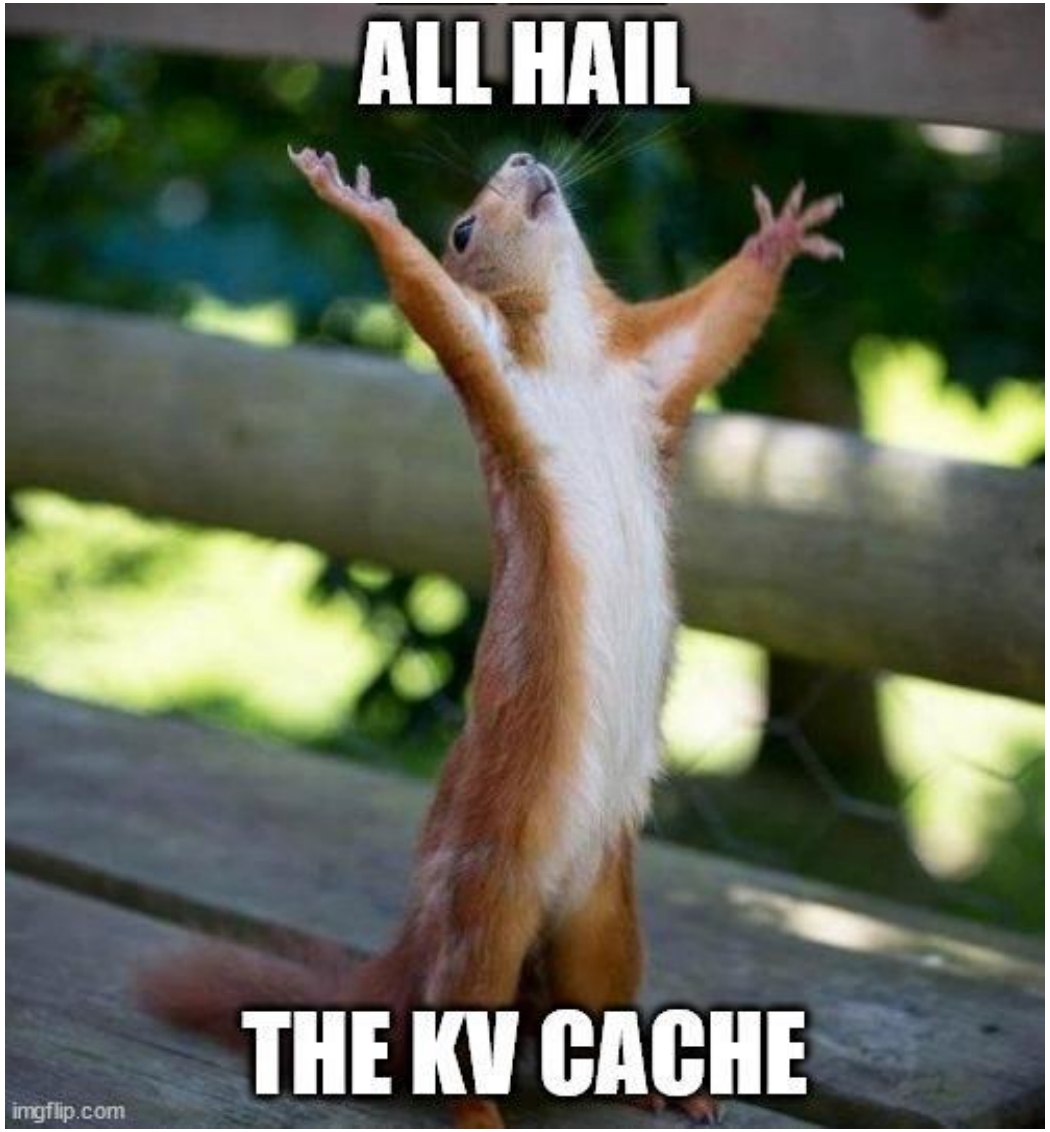
2. Since the model is causal, **we don't care about the attention of a token with its successors**, but only with the tokens before it.

3. **We don't care about these**, as we want to predict the next token and we already predicted the previous ones.

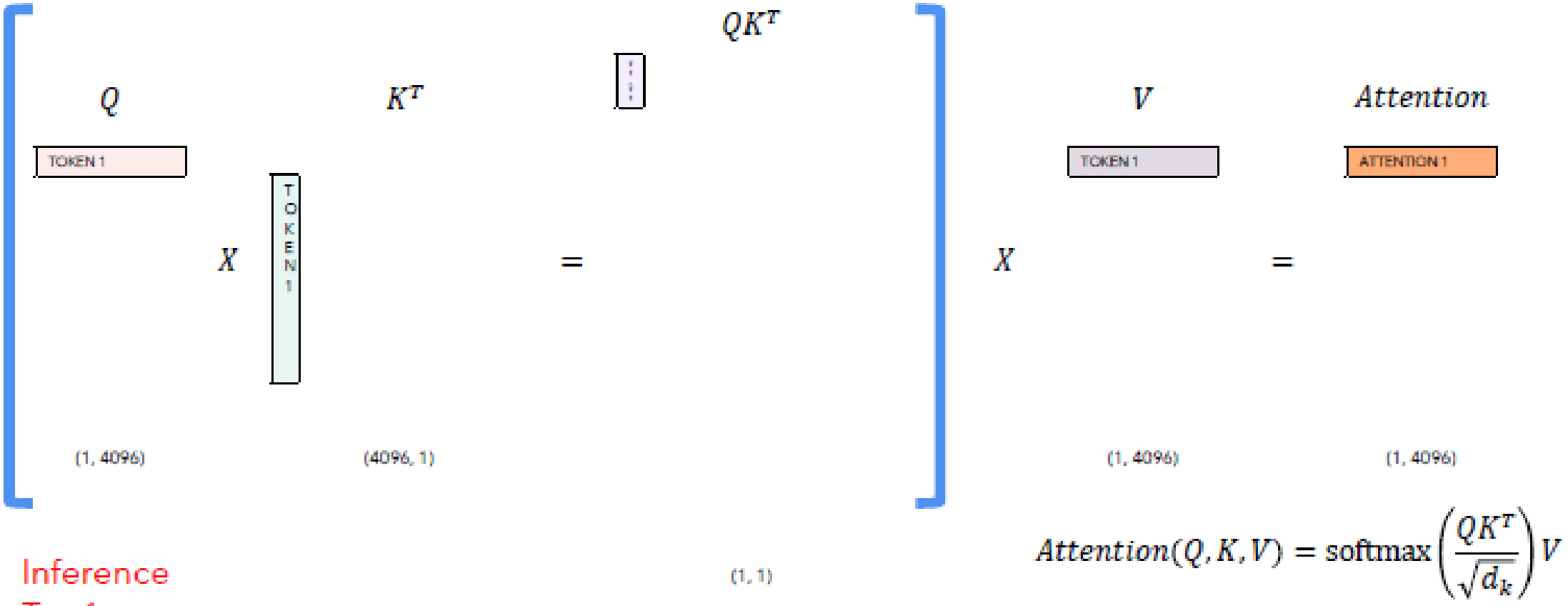


$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Inference
T = 4


ALL HAIL
THE KV CACHE
imgflip.com

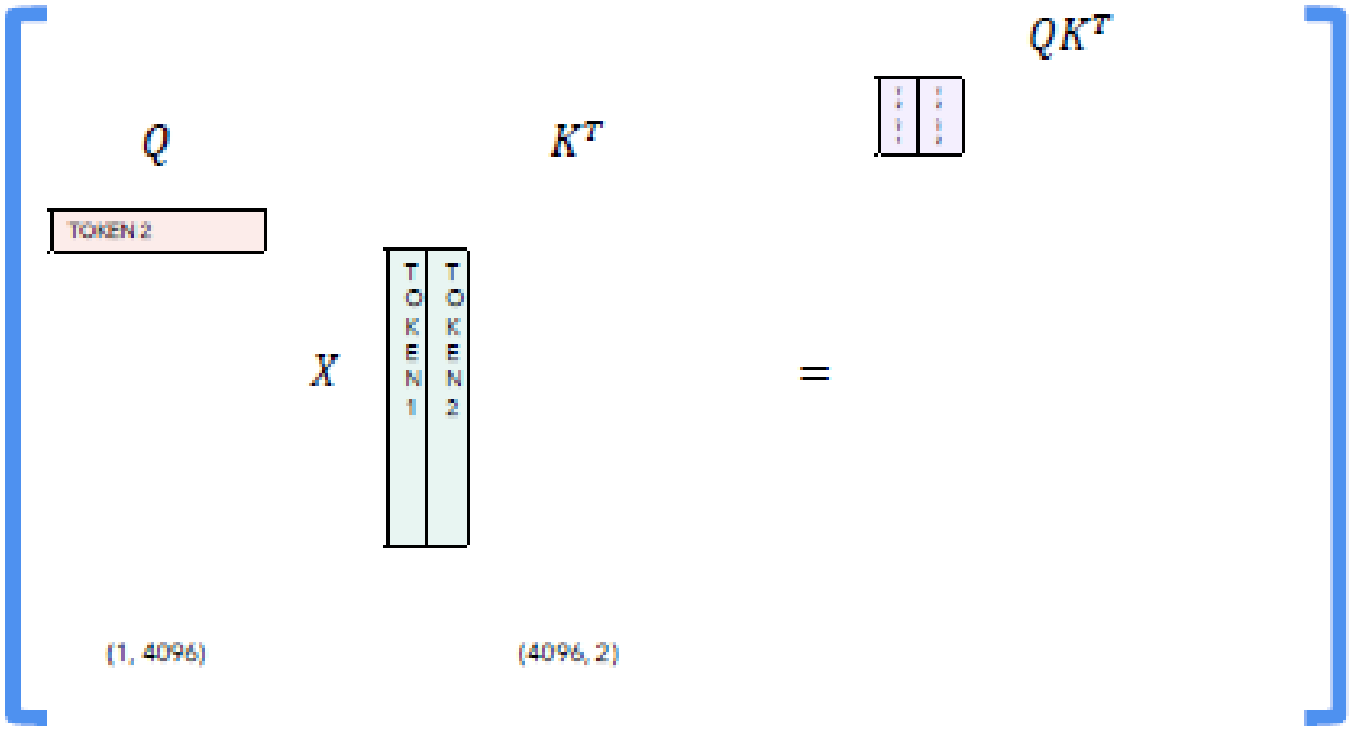
Auto-Atención con Caché KV



Inference
T = 1

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

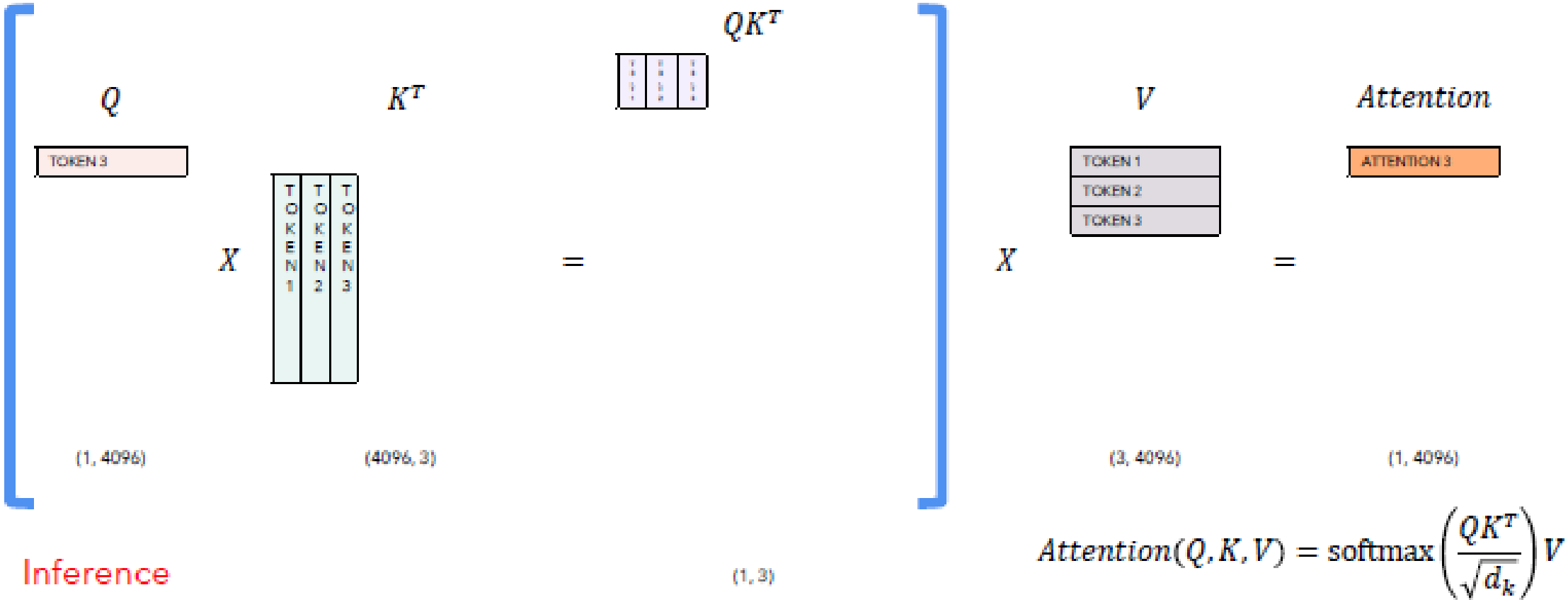
Auto-Atención con Caché KV



$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Inference
T = 2

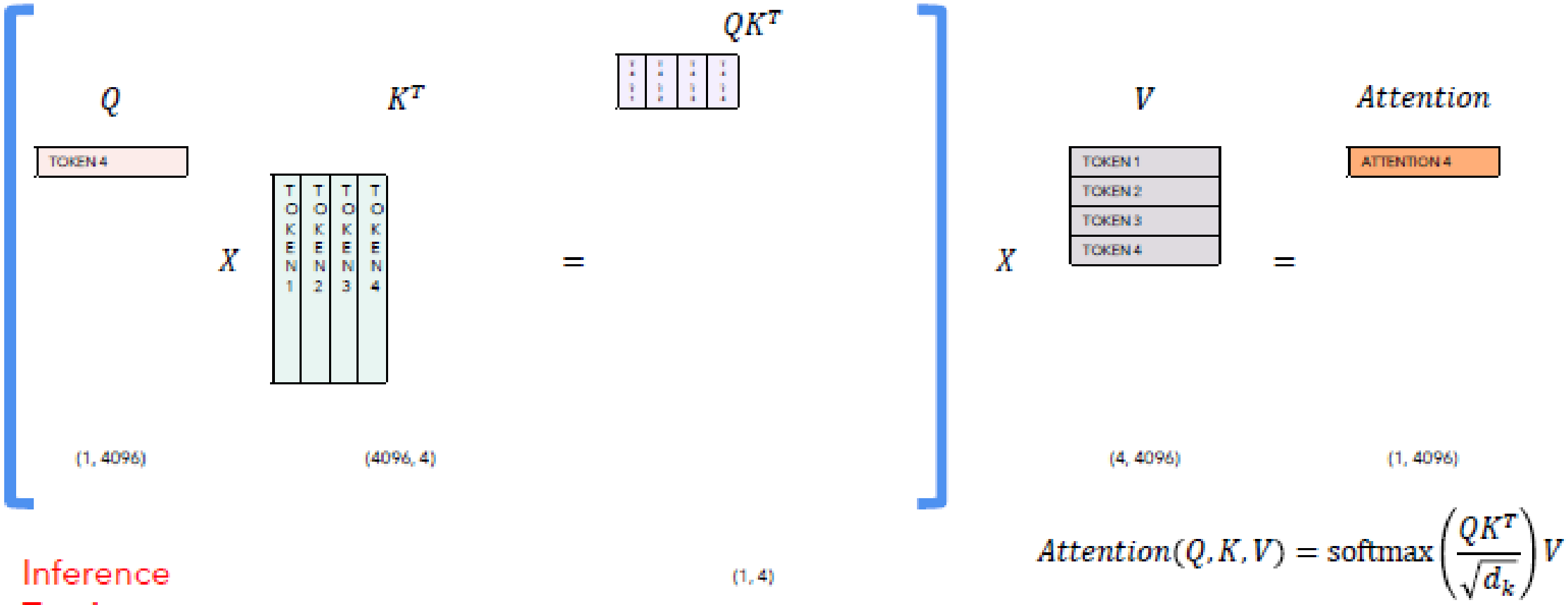
Auto-Atención con Caché KV



Inference
T = 3

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Auto-Atención con Caché KV

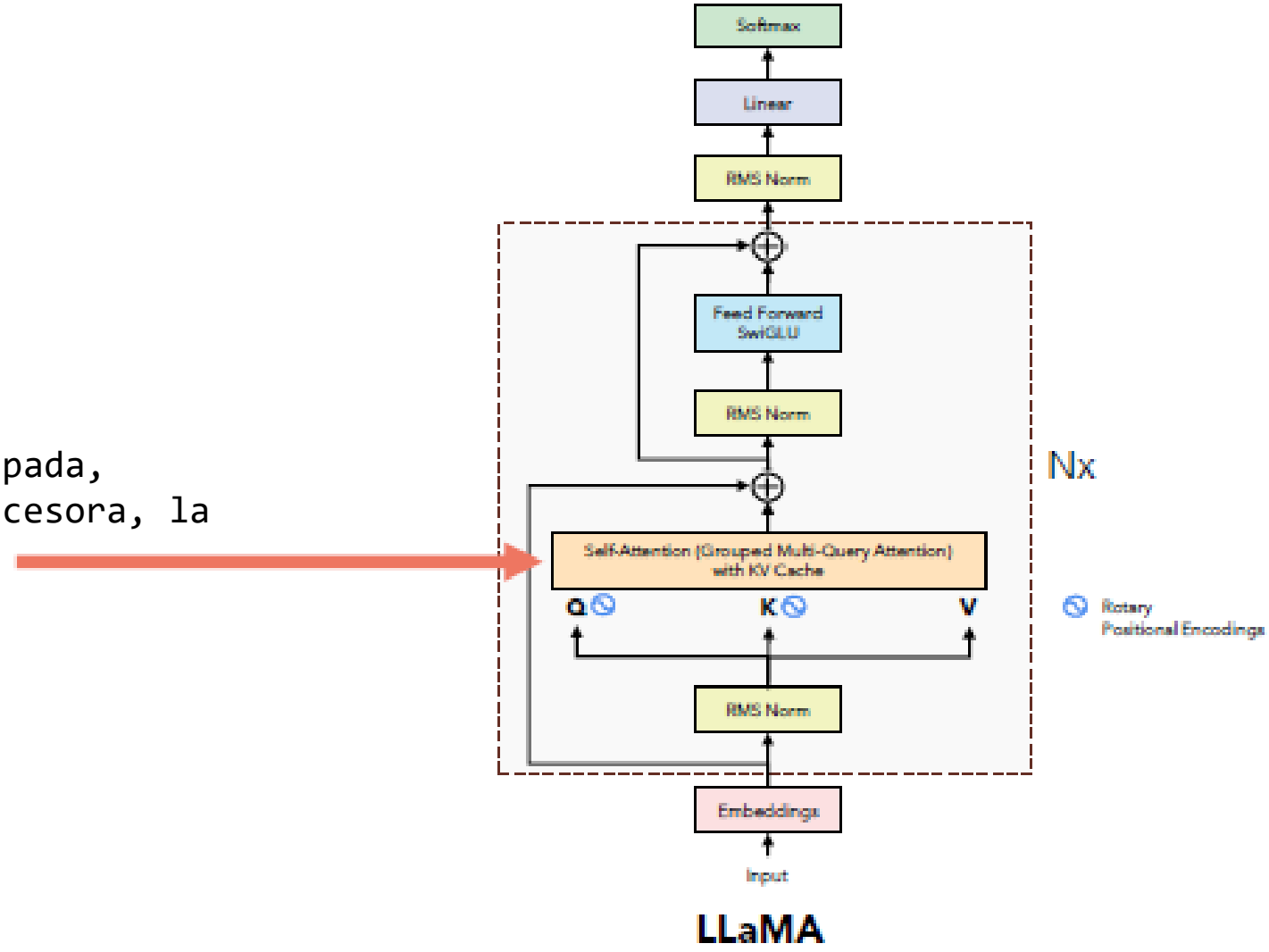


$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Inference
T = 4

¿Qué es la Atención Agrupada Multi-Consulta?

Antes de hablar sobre la MQA Agrupada, necesitamos introducir a su predecesora, la Atención Multi-Consulta (MQA).



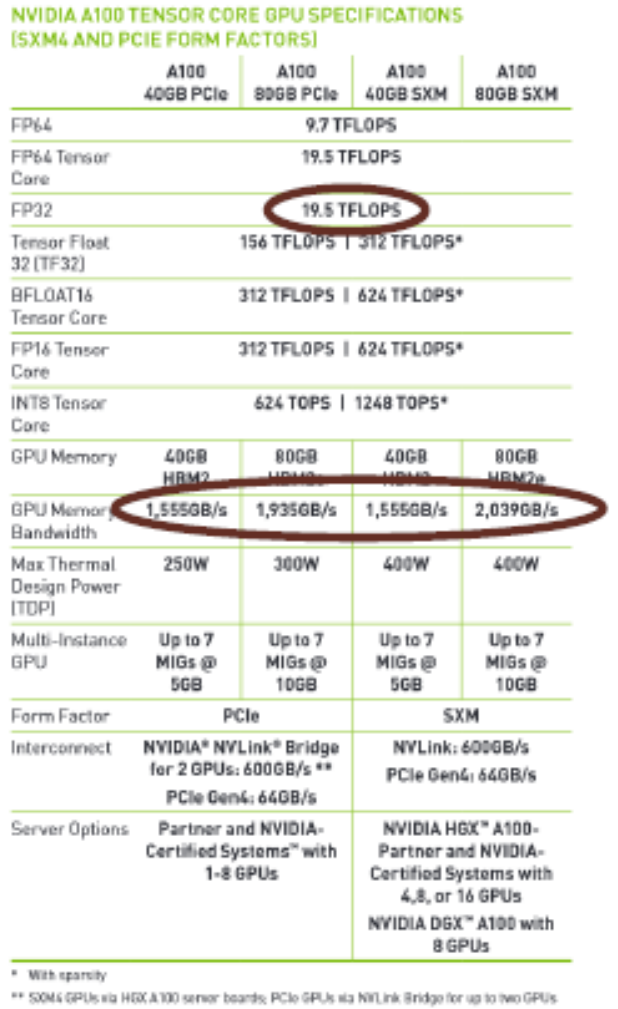
Las GPUs tienen un "problema": son demasiado rápidas.

- En los últimos años, las GPUs se han vuelto muy rápidas realizando cálculos, tanto que la velocidad de cálculo (FLOPs) es mucho mayor que el ancho de banda de la memoria (GB/s) o la velocidad de transferencia de datos entre áreas de memoria. Por ejemplo, una NVIDIA A100 puede realizar 19.5 TFLOPs mientras tiene un ancho de banda de memoria de 2.7 TB/s.

- Esto significa que a veces el cuello de botella no es cuántas operaciones realizamos, sino cuánta transferencia de datos o accesos a la memoria necesitamos, lo cual depende del tamaño y la cantidad de tensores involucrados en nuestros cálculos.

- Por ejemplo, realizar la misma operación sobre los mismos tensores N veces puede ser más rápido que realizar la misma operación sobre N tensores diferentes, incluso si tienen el mismo tamaño, esto se debe a que la GPU puede necesitar mover los tensores.

- **Esto significa que nuestro objetivo no solo debe ser optimizar el número de operaciones que realizamos, sino también minimizar los accesos/transferencias de memoria que realizamos.**



NVIDIA A100 TENSOR CORE GPU SPECIFICATIONS (SXM4 AND PCIE FORM FACTORS)

	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			
GPU Memory	40GB HBM2	80GB HBM2e	40GB HBM2	80GB HBM2e
GPU Memory Bandwidth	1,555GB/s	1,935GB/s	1,555GB/s	2,039GB/s
Max Thermal Design Power (TDP)	250W	300W	400W	400W
Multi-Instance GPU	Up to 7 MIGs @ 5GB	Up to 7 MIGs @ 10GB	Up to 7 MIGs @ 5GB	Up to 7 MIGs @ 10GB
Form Factor	PCIe		SXM	
Interconnect	NVIDIA® NVLink® Bridge for 2 GPUs: 600GB/s ** PCIe Gen4: 64GB/s		NVLink: 600GB/s PCIe Gen4: 64GB/s	
Server Options	Partner and NVIDIA-Certified Systems™ with 1-8 GPUs		NVIDIA HGX™ A100-Partner and NVIDIA-Certified Systems with 4,8, or 16 GPUs NVIDIA DGX™ A100 with 8 GPUs	

* With sparsity

** SXM4 GPUs via HGX A100 server boards; PCIe GPUs via NVLink Bridge for up to two GPUs

Introducción a la Atención Multi-Consulta

Fast Transformer Decoding: One Write-Head is All You Need

Noam Shazeer
Google
noam@google.com

November 7, 2019

Comparando diferentes algoritmos de atención: atención multi-cabeza agrupada en lotes estándar

- La Atención Multi-cabeza como se presentó en el artículo original "Attention is all you need".
- Al establecer $m = n$ (longitud de secuencia de consulta = longitud de secuencia de claves y valores)
- El número de operaciones aritméticas realizadas es $O(bnd^2)$
- La memoria total involucrada en las operaciones, dada por la suma de todos los tensores involucrados en los cálculos (incluyendo los derivados una vez), es $O(bnd + bhn^2 + d^2)$
- La relación entre la memoria total y el número de operaciones aritméticas es $O(\frac{1}{k} + \frac{1}{bn})$
- En este caso, la proporción es mucho menor que 1, lo que significa que el número de accesos a la memoria que estamos realizando es mucho menor que el número de operaciones aritméticas, por lo que el acceso a la memoria **no** es el cuello de botella aquí.

```python
def MultiheadAttentionBatched():
    d, m, n, b, h, k, v = 512, 10, 10, 32, 8, (512 // 8), (512 // 8)

    X = torch.rand(b, n, d)  # Query
    M = torch.rand(b, m, d)  # Key and Value
    mask = torch.rand(b, h, n, m)
    P_q = torch.rand(h, d, k)  # W_q
    P_k = torch.rand(h, d, k)  # W_k
    P_v = torch.rand(h, d, v)  # W_v
    P_o = torch.rand(h, d, v)  # W_o

    Q = torch.einsum("bnd,hdk->bhnk ", X, P_q)
    K = torch.einsum("bmd,hdk->bhmk", M, P_k)
    V = torch.einsum("bmd,hdv->bhmv", M, P_v)

    logits = torch.einsum("bhnk,bhmk->bhnm", Q, K)
    weights = torch.softmax(logits + mask, dim=-1)

    O = torch.einsum("bhnm,bhmv->bhnv ", weights, V)
    Y = torch.einsum("bhnv,hdv->bnd ", O, P_o)
    return Y
```

Comparando diferentes algoritmos de atención: atención multi-cabeza agrupada en lotes con KV cache

- Utiliza el KV cache para reducir el número de operaciones realizadas.
- Al establecer $m = n$ longitud de la secuencia de consulta = longitud de la secuencia de claves y valores)
- El número de operaciones aritméticas realizadas es $O(bnd^2)$
- La memoria total involucrada en las operaciones, dada por la suma de todos los tensores involucrados en los cálculos (incluyendo los derivados una vez), es $O(bn^2d + nd^2)$
- La relación entre la memoria total y el número de operaciones aritméticas es $O(\frac{n}{d} + \frac{1}{b})$
- Cuando $n \approx d$ (la longitud de la secuencia es cercana al tamaño del vector de codificación) o $b \approx 1$ (el tamaño del lote es 1), el ratio se convierte en 1 y el acceso a la memoria se convierte ahora en el cuello de botella del algoritmo. Para el caso de lote, esto no es un problema; el límite generalmente es mucho más alto que 1, mientras que para el caso de $\frac{n}{d}$ termina , necesitamos reducir la longitud de la secuencia. **Pero hay una mejor manera...**

```python
def MultiheadSelfAttentionIncremental():
    d, b, h, k, v = 512, 32, 8, (512 // 8), (512 // 8)

    m = 5  # Suppose we have already cached "m" tokens
    prev_K = torch.rand(b, h, m, k)
    prev_V = torch.rand(b, h, m, v)

    X = torch.rand(b, d)  # Query
    M = torch.rand(b, d)  # Key and Value
    P_q = torch.rand(h, d, k)  # W_q
    P_k = torch.rand(h, d, k)  # W_k
    P_v = torch.rand(h, d, v)  # W_v
    P_o = torch.rand(h, d, v)  # W_o

    q = torch.einsum("bd,hdk->bhk", X, P_q)
    new_K = torch.concat(
        [prev_K, torch.einsum("bd,hdk->bhk", M, P_k).unsqueeze(2)], axis=2
    )
    new_V = torch.concat(
        [prev_V, torch.einsum("bd,hdv->bhv", M, P_v).unsqueeze(2)], axis=2
    )
    logits = torch.einsum("bhk,bhmk->bhm", q, new_K)
    weights = torch.softmax(logits, dim=-1)
    O = torch.einsum("bhm,bhmv->bhv", weights, new_V)
    y = torch.einsum("bhv,hdv->bd", O, P_o)
    return y, new_K, new_V
```

Comparando diferentes algoritmos de atención: atención multi-consulta con KV cache

- Eliminamos la dimensión h del K y V, mientras la mantenemos para Q. Esto significa que todas las diferentes cabezas de consulta compartirán las mismas claves y valores.El número de operaciones aritméticas realizadas es $O(bnd^2)$
- La memoria total involucrada en las operaciones, dada por la suma de todos los tensores involucrados en los cálculos (incluyendo los derivados), es $O(bnd + bn^2k + nd^2)$
- La relación entre la memoria total y el número de operaciones aritméticas es $O(\frac{1}{d}+\frac{n}{dh}+\frac{1}{b})$
- Comparado con el enfoque anterior, hemos reducido el término expansivo $\frac{n}{d}$ por un factor de h.
- Las ganancias de rendimiento son importantes, mientras que la degradación del modelo es mínima.

```python
def MultiquerySelfAttentionIncremental():
    d, b, h, k, v = 512, 32, 8, (512 // 8), (512 // 8)

    m = 5  # Suppose we have already cached "m" tokens
    prev_K = torch.rand(b, m, k)
    prev_V = torch.rand(b, m, v)

    X = torch.rand(b, d)  # Query
    M = torch.rand(b, d)  # Key and Value
    P_q = torch.rand(h, d, k)  # W_q
    P_k = torch.rand(d, k)  # W_k
    P_v = torch.rand(d, v)  # W_v
    P_o = torch.rand(h, d, v)  # W_o

    q = torch.einsum("bd,hdk->bhk", X, P_q)
    K = torch.concat([prev_K, torch.einsum("bd,dk->bk", M, P_k).unsqueeze(1)], axis=1)
    V = torch.concat([prev_V, torch.einsum("bd,dv->bv", M, P_v).unsqueeze(1)], axis=1)
    logits = torch.einsum("bhk,bmk->bhm", q, K)
    weights = torch.softmax(logits, dim=-1)
    O = torch.einsum("bhm,bmv->bhv", weights, V)
    y = torch.einsum("bhv,hdv->bd", O, P_o)
    return y, K, V
```

Comparaciones de velocidad y calidad

Table 1: WMT14 EN-DE Results.

Attention Type	h	d_k, d_v	d_{ff}	ln(PPL) (dev)	BLEU (dev)	BLEU (test) beam 1 / 4
multi-head	8	128	4096	**1.424**	**26.7**	27.7 / 28.4
multi-query	8	128	5440	1.439	26.5	27.5 / **28.5**
multi-head local	8	128	4096	1.427	26.6	27.5 / 28.3
multi-query local	8	128	5440	1.437	26.5	27.6 / 28.2
multi-head	1	128	6784	1.518	25.8	
multi-head	2	64	6784	1.480	26.2	26.8 / 27.9
multi-head	4	32	6784	1.488	26.1	
multi-head	8	16	6784	1.513	25.8	

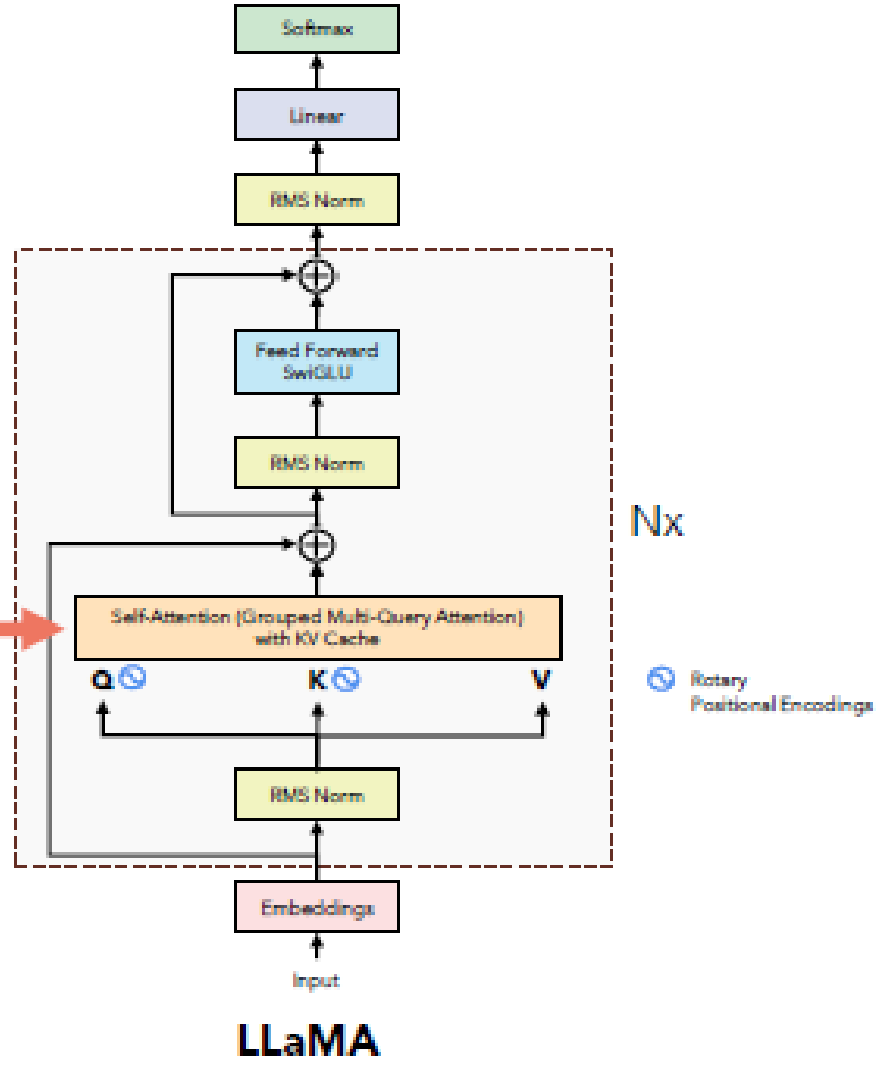
Table 2: Amortized training and inference costs for WMT14 EN-DE Translation Task with sequence length 128. Values listed are in TPUv2-microseconds per output token.

Attention Type	Training	Inference enc. + dec.	Beam-4 Search enc. + dec.
multi-head	13.2	1.7 + 46	2.0 + 203
multi-query	**13.0**	1.5 + 3.8	1.6 + 32
multi-head local	13.2	1.7 + 23	1.9 + 47
multi-query local	**13.0**	**1.5 + 3.3**	**1.6 + 16**

Para demostrar que la atención local y la atención multi-consulta son ortogonales, también entrenamos versiones "locales“ del modelo base y modelos de atención multi-consulta, donde las capas de auto-atención del decodificador (pero no las otras capas de atención) restringen la atención a la posición actual y a las 31 posiciones anteriores.

¿Qué es la Atención Multi-Consulta Agrupada?

Ahora, hablemos sobre MQA Agrupada (Multi-Query Attention Agrupada).



Atención Multi-Consulta Agrupada: un compromiso entre dos extremos.

Atención Multi-Cabeza:

- Alta calidad
- Computacionalmente lenta

Atención Multi-Consulta Agrupada:

- Un buen compromiso entre calidad y velocidad

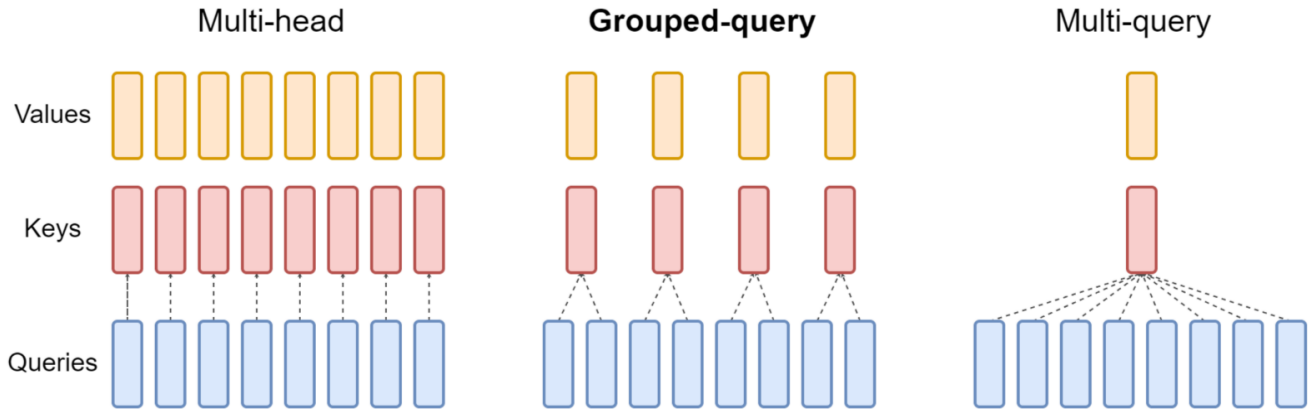
Atención Multi-Consulta:

- Pérdida de calidad
- Computacionalmente rápida

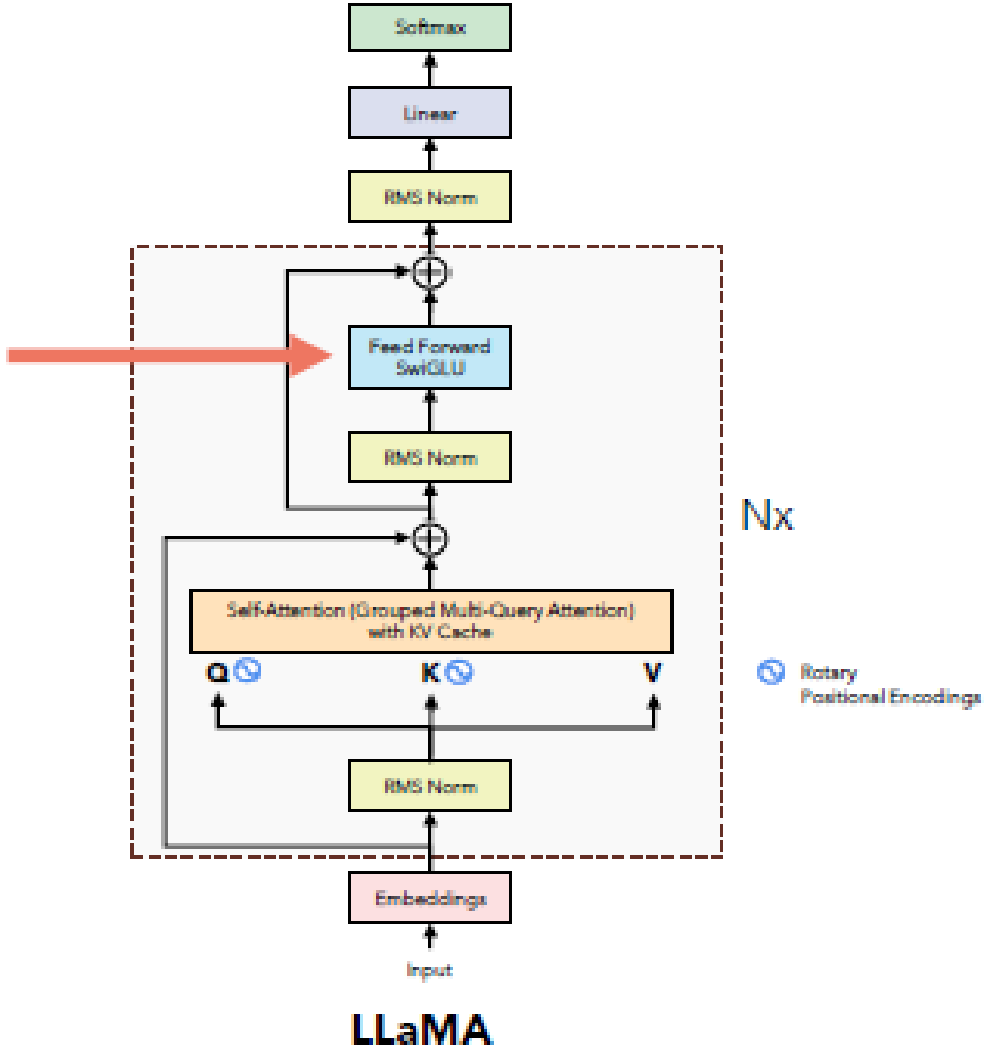
GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Joshua Ainslie, James Lee-Thorp, Michiel de Jong
Yury Zemlyanskiy, Federico Lebrón, Sumit Sanghai

Google Research



¿Qué es la función de activación SwiGLU?



Función de Activación SwiGLU

GLU Variants Improve Transformer

Noam Shazeer
Google
noam@google.com

February 14, 2020

Función de Activación SwiGLU

- El autor comparó el rendimiento de un modelo Transformer utilizando diferentes funciones de activación en la capa Feed-Forward de la arquitectura Transformer.

$$
\begin{aligned}
\mathrm{ReGLU}(x, W, V, b, c) &= \max(0, xW + b) \otimes (xV + c) \\
\mathrm{GEGLU}(x, W, V, b, c) &= \mathrm{GELU}(xW + b) \otimes (xV + c) \\
\mathrm{SwiGLU}(x, W, V, b, c, \beta) &= \mathrm{Swish}_\beta(xW + b) \otimes (xV + c)
\end{aligned} \tag{5}
$$

In this paper, we propose additional variations on the Transformer FFN layer which use GLU or one of its variants in place of the first linear transformation and the activation function. Again, we omit the bias terms.

$$
\begin{aligned}
\mathrm{FFN}_{\mathrm{GLU}}(x, W, V, W_2) &= (\sigma(xW) \otimes xV)W_2 \\
\mathrm{FFN}_{\mathrm{Bilinear}}(x, W, V, W_2) &= (xW \otimes xV)W_2 \\
\mathrm{FFN}_{\mathrm{ReGLU}}(x, W, V, W_2) &= (\max(0, xW) \otimes xV)W_2 \\
\mathrm{FFN}_{\mathrm{GEGLU}}(x, W, V, W_2) &= (\mathrm{GELU}(xW) \otimes xV)W_2 \\
\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) &= (\mathrm{Swish}_1(xW) \otimes xV)W_2
\end{aligned} \tag{6}
$$

All of these layers have three weight matrices, as opposed to two for the original FFN. To keep the number of parameters and the amount of computation constant, we reduce the number of hidden units d_{ff} (the second dimension of W and V and the first dimension of W_2) by a factor of $\frac{2}{3}$ when comparing these layers to the original two-matrix version.

Función de Activación SwiGLU

Transformer ("Attention is all you need"):

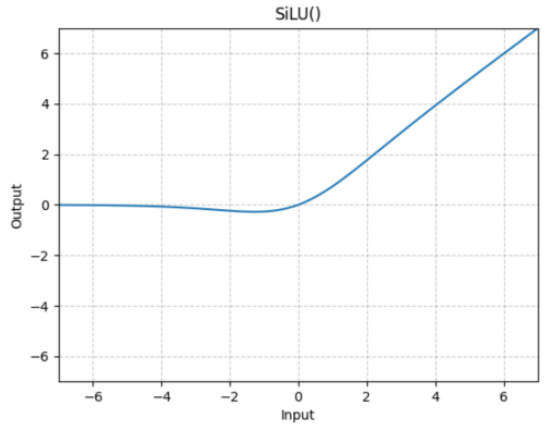
$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

LlaMA

$$FFN_{\text{SwiGLU}}(x, W, V, W_2) = (\text{Swish}(xW) \odot xV)W_2$$

Usamos la función Swish con $\beta = 1.$ En este caso, se llama la función **Sigmoid Linear Unit (SiLU).**

$$\text{swish}(x) = \frac{x}{1 + e^{-x}}$$




```python
class FeedForward(nn.Module):
    def __init__(
        self,
        dim: int,
        hidden_dim: int,
        multiple_of: int,
        ffn_dim_multiplier: Optional[float],
    ):
        super().__init__()
        hidden_dim = int(2 * hidden_dim / 3)
        # custom dim factor multiplier
        if ffn_dim_multiplier is not None:
            hidden_dim = int(ffn_dim_multiplier * hidden_dim)
        hidden_dim = multiple_of * ((hidden_dim + multiple_of - 1) // multiple_of)

        self.w1 = ColumnParallelLinear(
            dim, hidden_dim, bias=False, gather_output=False, init_method=lambda x: x
        )
        self.w2 = RowParallelLinear(
            hidden_dim, dim, bias=False, input_is_parallel=True, init_method=lambda x: x
        )
        self.w3 = ColumnParallelLinear(
            dim, hidden_dim, bias=False, gather_output=False, init_method=lambda x: x
        )

    def forward(self, x):
        return self.w2(F.silu(self.w1(x)) * self.w3(x))
```

Función de Activación SwiGLU

Table 1: Heldout-set log-perplexity for Transformer models on the segment-filling task from [Raffel et al., 2019]. All models are matched for parameters and computation.

Training Steps	65,536	524,288
FFN_{ReLU} (*baseline*)	1.997 (0.005)	1.677
FFN_{GELU}	1.983 (0.005)	1.679
FFN_{Swish}	1.994 (0.003)	1.683
FFN_{GLU}	1.982 (0.006)	1.663
$FFN_{Bilinear}$	1.960 (0.005)	1.648
FFN_{GEGLU}	**1.942** (0.004)	**1.633**
FFN_{SwiGLU}	**1.944** (0.010)	**1.636**
FFN_{ReGLU}	1.953 (0.003)	1.645

Table 2: GLUE Language-Understanding Benchmark [Wang et al., 2018] (dev).

	Score Average	CoLA MCC	SST-2 Acc	MRPC F1	MRPC Acc	STSB PCC	STSB SCC	QQP F1	QQP Acc	MNLIm Acc	MNLImm Acc	QNLI Acc	RTE Acc
FFN_{ReLU}	83.80	51.32	94.04	**93.08**	**90.20**	89.64	89.42	89.01	91.75	85.83	86.42	92.81	80.14
FFN_{GELU}	83.86	53.48	94.04	92.81	**90.20**	89.69	89.49	88.63	91.62	85.89	86.13	92.39	80.51
FFN_{Swish}	83.60	49.79	93.69	92.31	89.46	89.20	88.98	88.84	91.67	85.22	85.02	92.33	81.23
FFN_{GLU}	84.20	49.16	94.27	92.39	89.46	89.46	89.35	88.79	91.62	86.36	86.18	92.92	**84.12**
FFN_{GEGLU}	84.12	53.65	93.92	92.68	89.71	90.26	90.13	89.11	91.85	86.15	86.17	92.81	79.42
$FFN_{Bilinear}$	83.79	51.02	**94.38**	92.28	89.46	90.06	89.84	88.95	91.69	**86.90**	**87.08**	92.92	81.95
FFN_{SwiGLU}	84.36	51.59	93.92	92.23	88.97	**90.32**	**90.13**	**89.14**	**91.87**	86.45	86.47	**92.93**	83.39
FFN_{ReGLU}	**84.67**	**56.16**	**94.38**	92.06	89.22	89.97	89.85	88.86	91.72	86.20	86.40	92.68	81.59
[Raffel et al., 2019]	83.28	53.84	92.68	92.07	88.92	88.02	87.94	88.67	91.56	84.24	84.57	90.48	76.28
ibid. stddev.	0.235	1.111	0.569	0.729	1.019	0.374	0.418	0.108	0.070	0.291	0.231	0.361	1.393

¿Por qué funciona tan bien SwiGLU?

Conclusiones

Hemos extendido la familia GLU de capas y propuesto su uso en Transformer. En un entorno de aprendizaje por transferencia, las nuevas variantes parecen producir mejores perplexities para el objetivo de eliminación de ruido utilizado en el preentrenamiento, así como mejores resultados en muchas tareas de comprensión del lenguaje. Estas arquitecturas son simples de implementar y no presentan aparentes inconvenientes computacionales. No ofrecemos ninguna explicación de por qué estas arquitecturas parecen funcionar; atribuimos su éxito, como todo lo demás, a la benevolencia divina.