CSCE 689-609 Fundamentals of Large Language Models (LLMs)

Jeff Huang
jeff@cse.tamu.edu
o2lab.github.io

The Transformer

https://arxiv.org/pdf/1706.03762

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

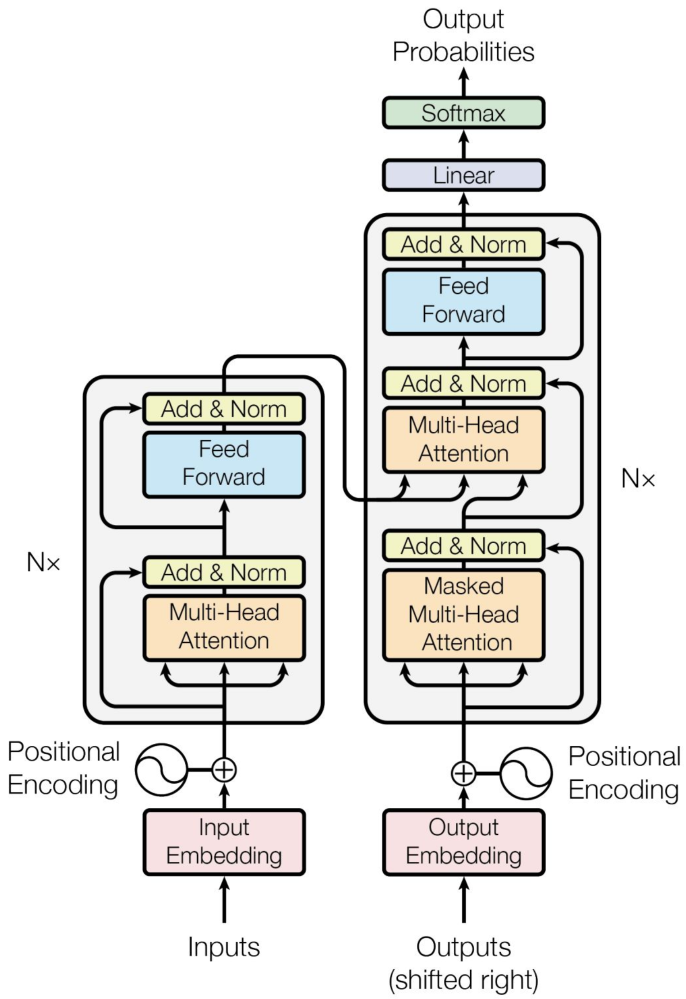


Figure 1: The Transformer - model architecture.

Attention is all your need

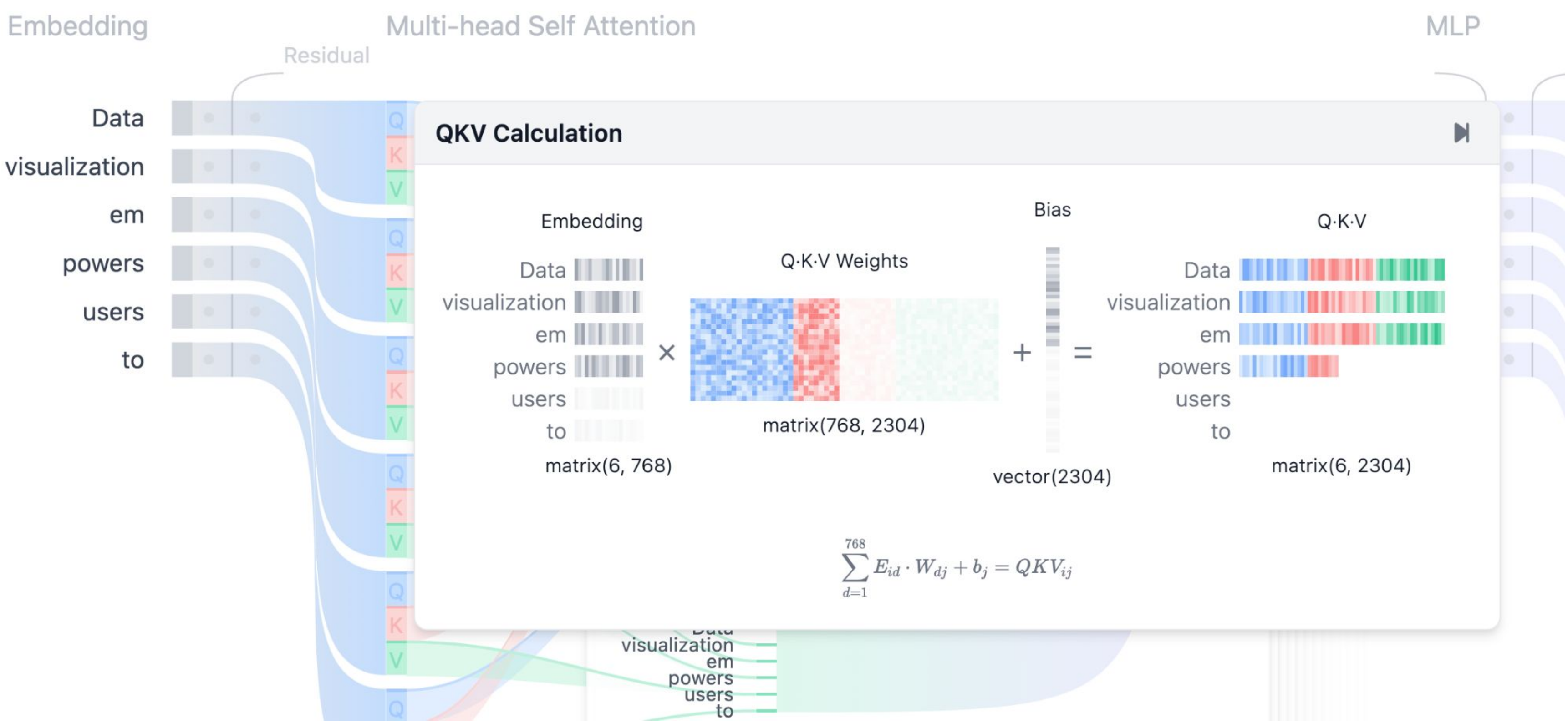
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where}\ \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

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

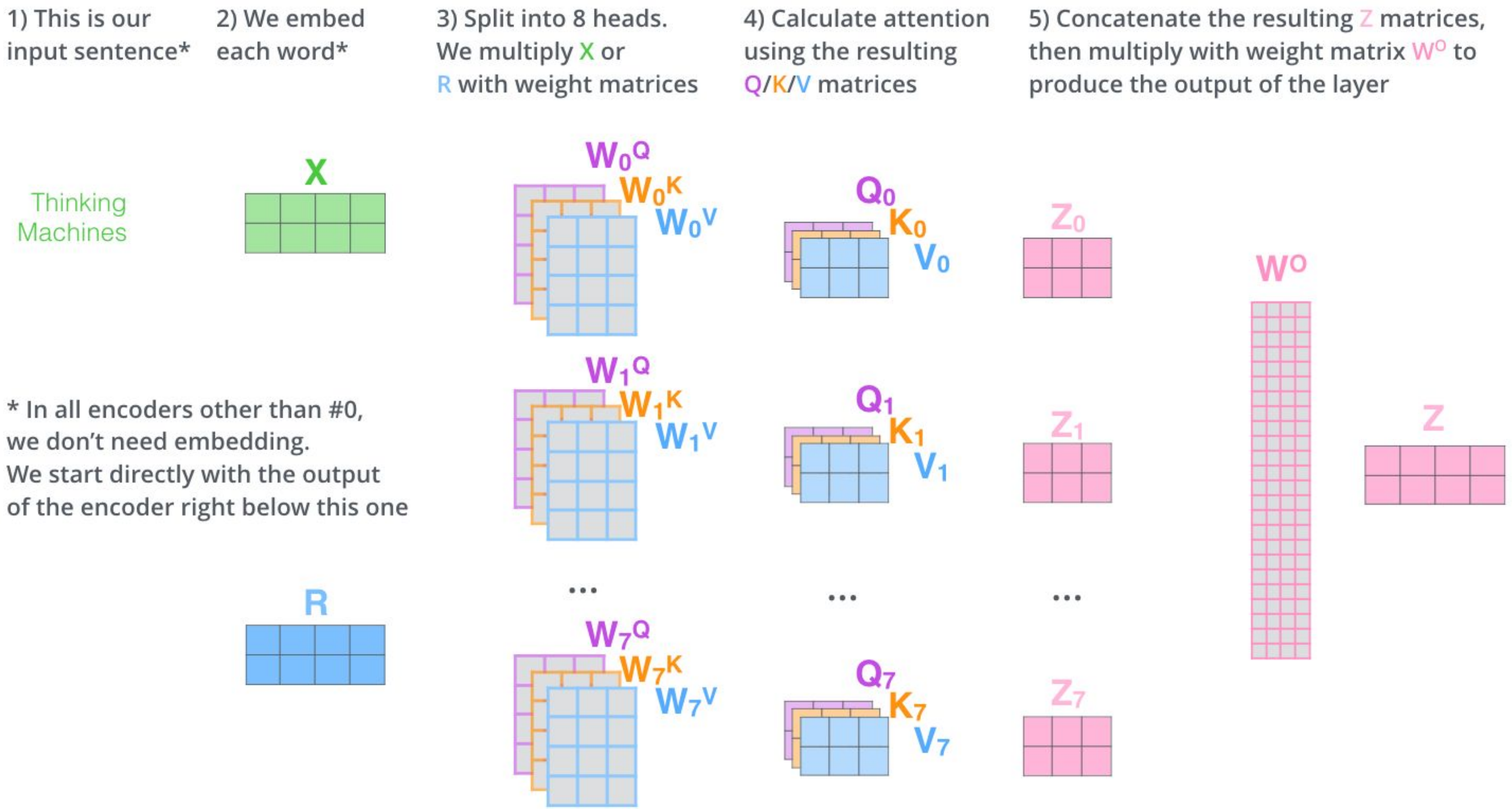
Transformer Explainer

https://poloclub.github.io/transformer-explainer/

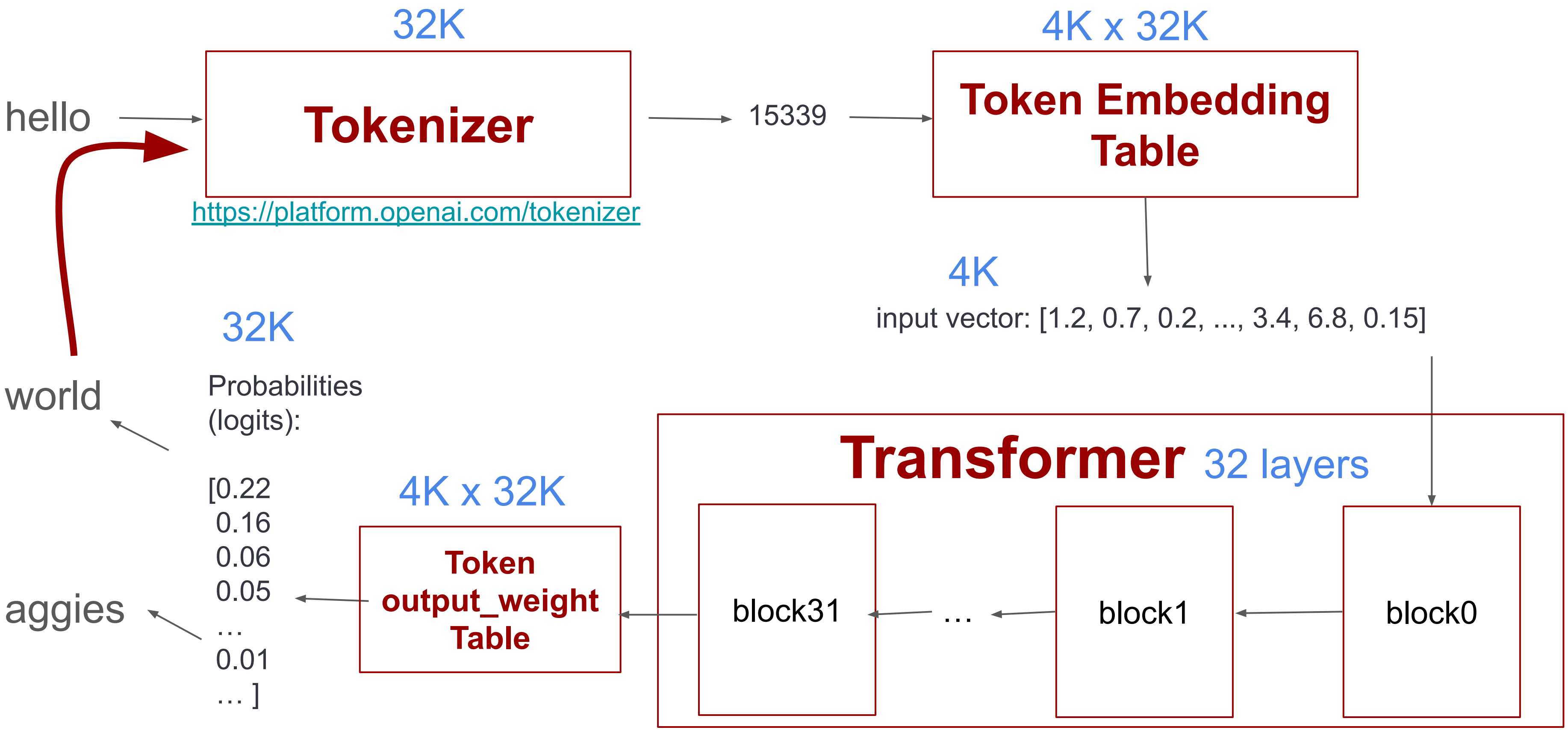


Illustrated Transformer

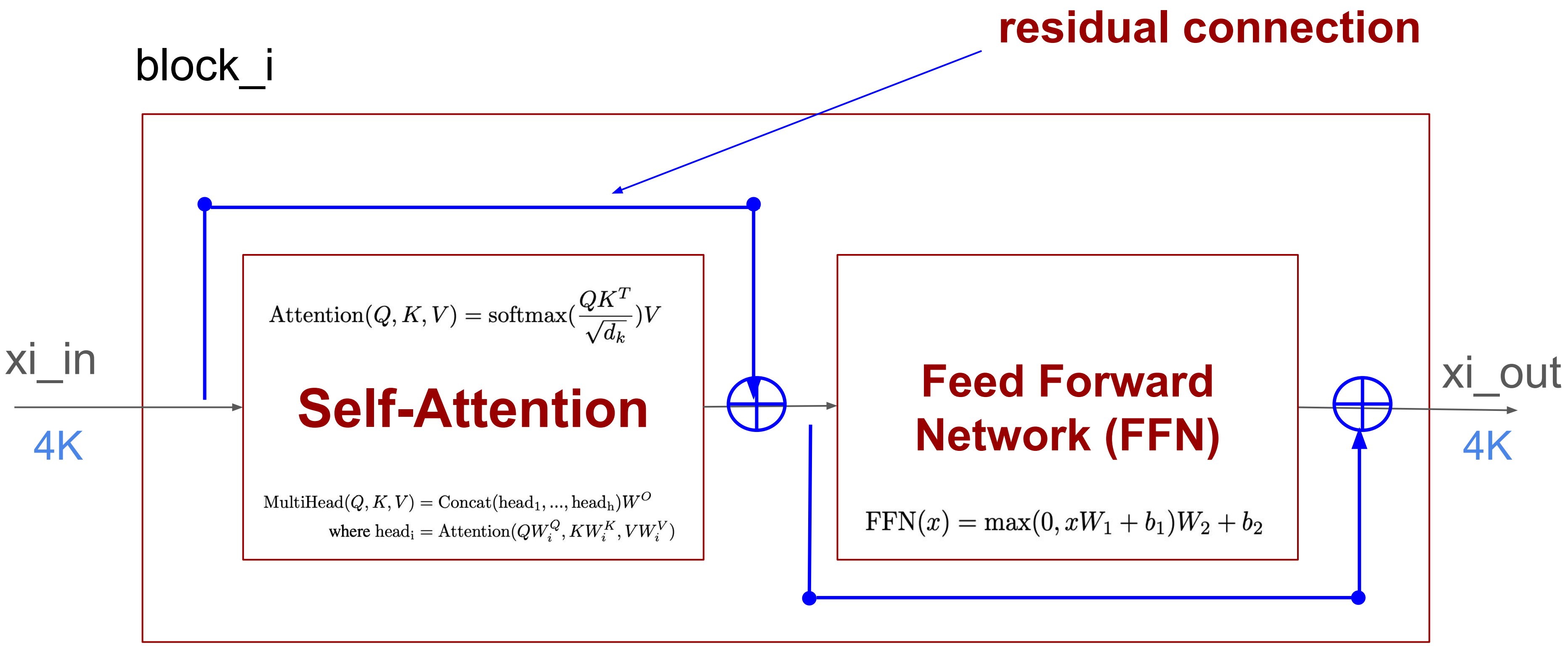
https://jalammar.github.io/illustrated-transformer/



A bird's view (llama2_7b)



A bird's view (llama2_7b)



A bird's view (llama2_7b)

Self-Attention

xi_in

4K

Positional Encoding

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

Q: 4K x 4K
K: 4K x 4K
V: 4K x 4K

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

W^o: 4K x 4K
h:32

4K

A bird's view (llama2_7b)

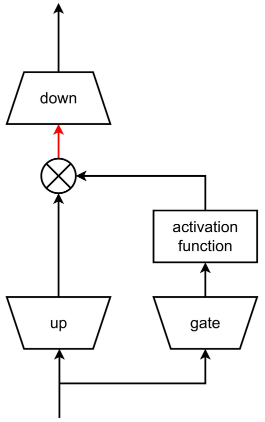
FFN

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

W2(silu(W1(x)) * W3(x))

4K

W1(ffn_gate): 4K x 11008
W2(ffn_down): 11008 x 4K
W3(ffn_up): 4K x 11008



4K

Residual Connection

During backpropagation, gradients can become very small (vanishing gradients) or very large (exploding gradients)

- Residual connections mitigate these issues

Layer Normalization (RMSNorm: Root Mean Square)

$$\mathrm{RMSNorm}(x) = \frac{x}{\mathrm{RMS}(x)} \cdot \gamma$$

where RMS is calculated as:

$$\mathrm{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$$

Here, d is the dimensionality of the input, and γ is a learnable scaling parameter.

Question: Size of each component in llama2_7b?

Question: Size of each component in llama2_7b?

Total: 25.1G (type f32: 32-bit floating point)
Token embedding: 4K x 32K x 4 bytes = 0.5 GB
Each block:
- K/Q/V: 4K x 4K x 4 bytes x 3 = 192 MB
- attn_out: 4K x 4K x 4 bytes = 64 MB
- attn_norm: 4K x 4 bytes = 16 KB
- ffn_gate: 4K x 11008 x 4 bytes = 172 MB
- ffn_down: 11008 x 4K x 4 bytes = 172 MB
- ffn_up: 4K x 11008 x 4 bytes = 172 MB
- ffn_norm: 4K x 4 bytes = 16KB

Block x 32 = 772 MB x 32 = 24.1 GB
Output_weight: 4K x 32K x 4 bytes = 0.5 GB

Question: Why QKV?

Question: Why QKV?

https://towardsdatascience.com/all-you-need-to-know-about-attention-and-transformers-in-depth-understanding-part-1-552f0b41d021

Retrieval

$$attention(q, k, v) = \sum_i similarity(q, k_i) * v_i$$

Query

Database

Key 1	Value 1
Key 2	Value 2
Key 3	Value 3
Key 4	Value 4
Key 5	Value 5

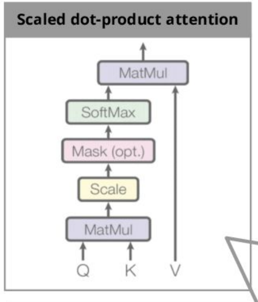
Value 5

Output

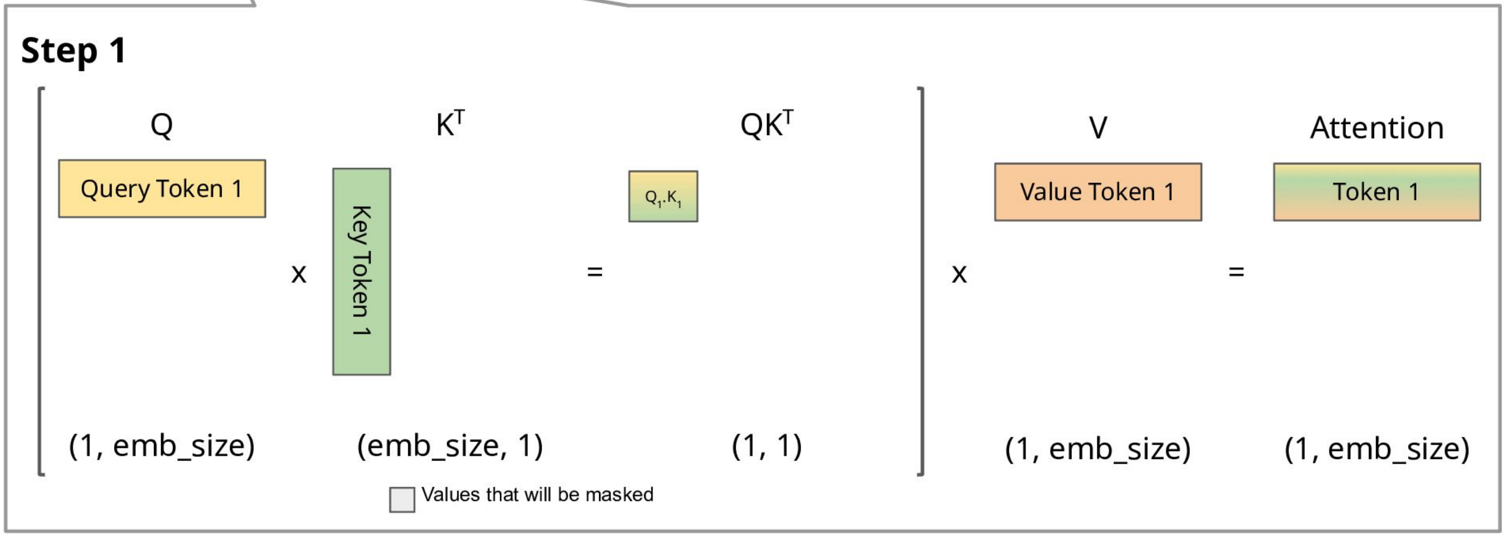
1. It measures the similarity between the query and each key
2. This similarity returns a weight for each key
3. Output is the weighted combination of all values

QKV

https://medium.com/@joaolages/kv-caching-explained-276520203249



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

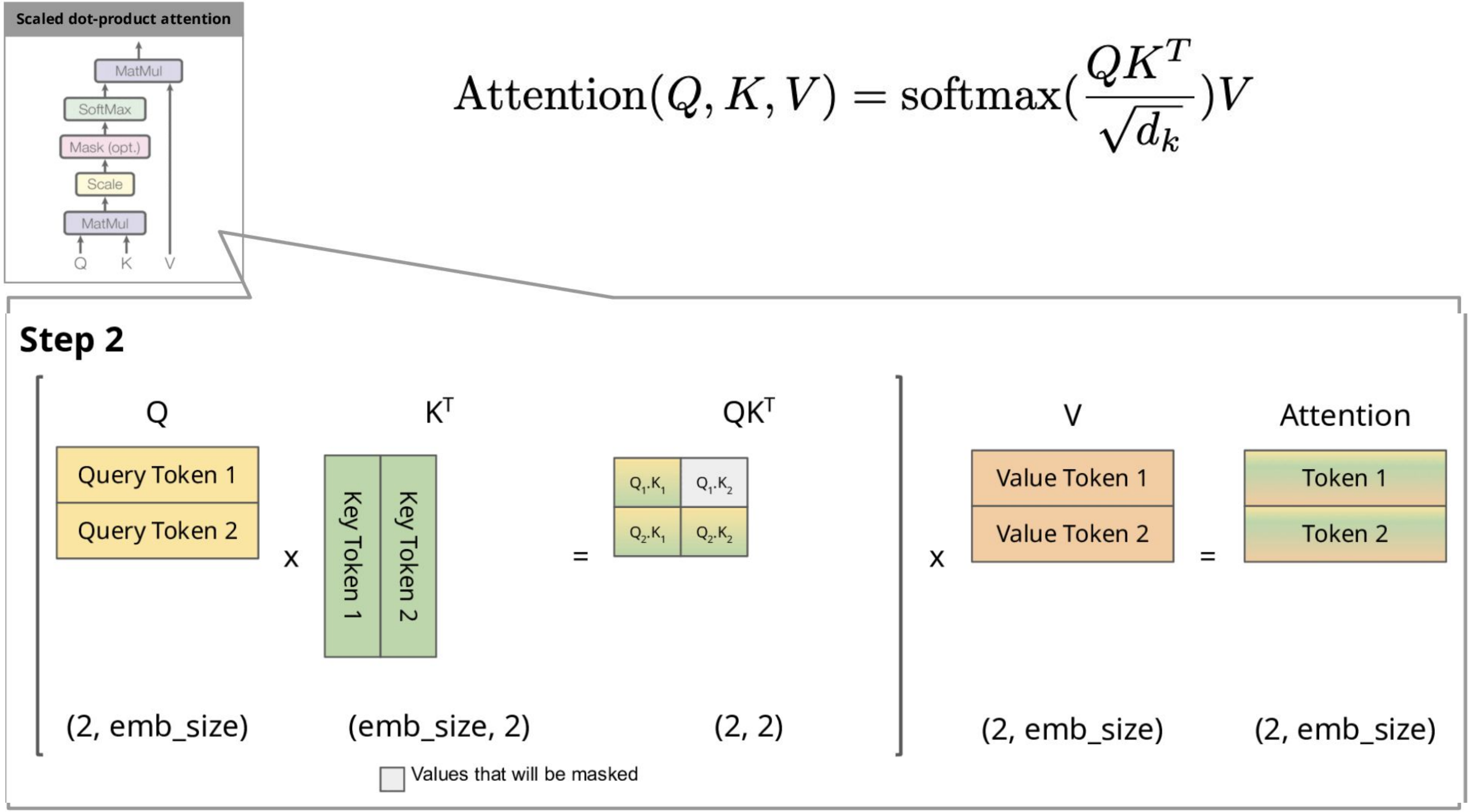


Zoom-in! (simplified without Scale and Softmax)

QKV

https://medium.com/@joaolages/kv-caching-explained-276520203249

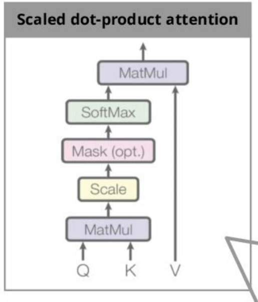
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



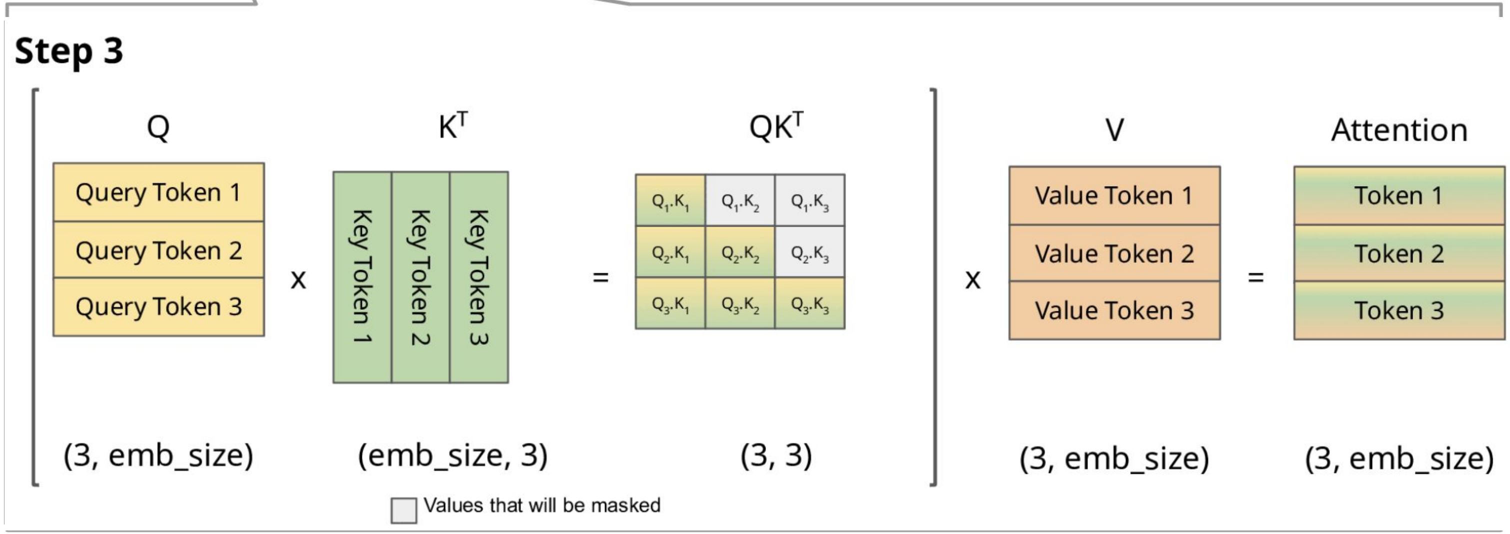
Zoom-in! (simplified without Scale and Softmax)

QKV

https://medium.com/@joaolages/kv-caching-explained-276520203249



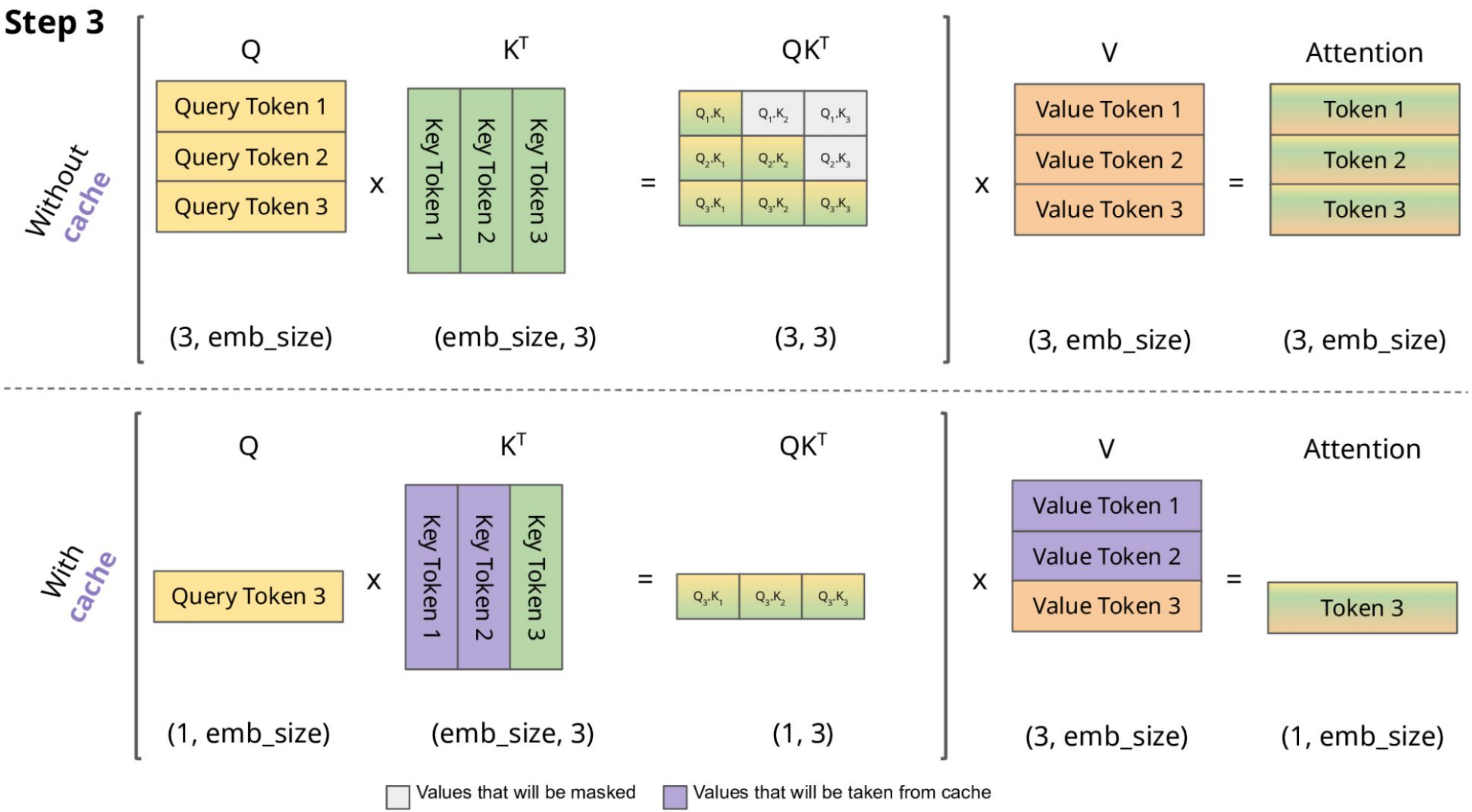
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Zoom-in! (simplified without Scale and Softmax)

KV Cache

https://medium.com/@joaolages/kv-caching-explained-276520203249



Multihead Attention

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

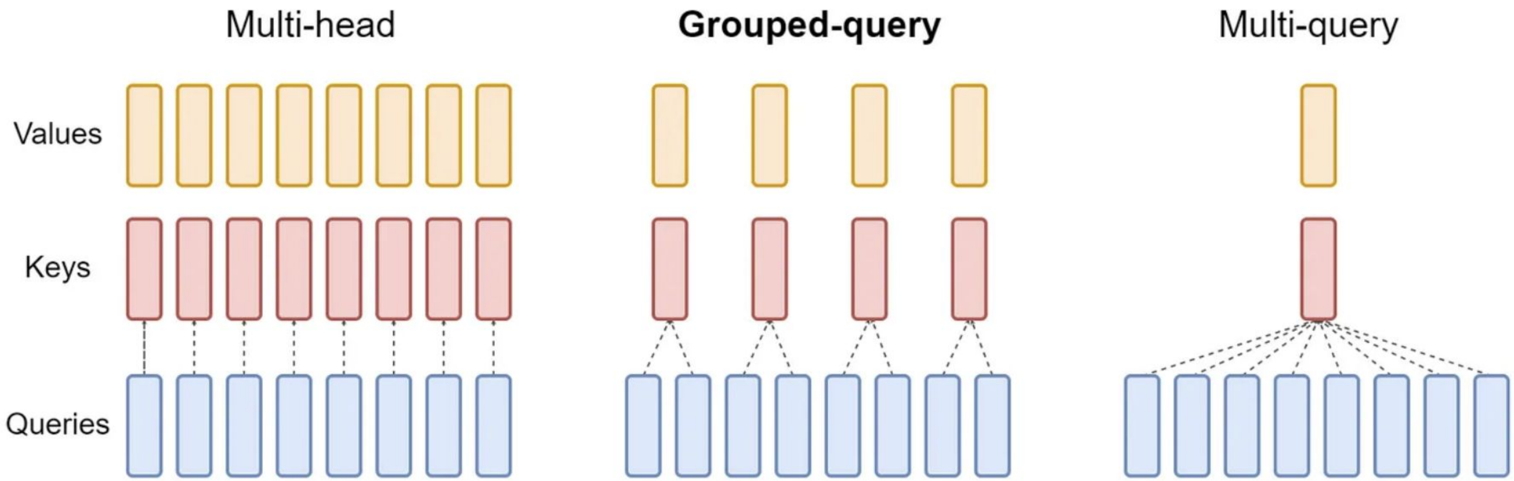
Multihead Attention (Reduced KV Heads)

Llama 2:
llama.attention.head_count = 32
llama.attention.head_count_kv = 32

Llama 3.1:
llama.attention.head_count = 32
llama.attention.head_count_kv **= 8**
grouped query attention (GQA)

Grouped Query Attention (GQA)

Paper: https://arxiv.org/pdf/2305.13245



Difference between MHA, GQA, and MQA (Source — https://arxiv.org/pdf/2305.13245.pdf)

Blog: https://towardsdatascience.com/demystifying-gqa-grouped-query-attention-3fb97b678e4a

softmax

$$\operatorname{softmax}(x_i) = \frac{\exp(x_i - \max(x))}{\sum_{j=0}^{\text{size}-1} \exp(x_j - \max(x))}$$

Positional Encoding (RoPE: rotary positional encoding)

https://pub.towardsai.net/the-quest-to-have-endless-conversations-with-llama-and-chatgpt-%EF%B8%8F-81360b9b34b2

$$\mathrm{RoPE}(x_{2i}, x_{2i+1}, p) = \begin{pmatrix} x_{2i} \cdot \cos(p \cdot \omega_i) - x_{2i+1} \cdot \sin(p \cdot \omega_i) \\ x_{2i} \cdot \sin(p \cdot \omega_i) + x_{2i+1} \cdot \cos(p \cdot \omega_i) \end{pmatrix}$$

The frequency for the i-th dimension is calculated as:

$$\omega_i = \frac{1}{10000^{\frac{2i}{d}}}$$

where d is the dimension of the embedding vector.

Paper: Extending Context Window of Large Language Models via Positional Interpolation

Let’s write one!

https://github.com/karpathy/llama2.c

https://github.com/karpathy/llama2.c/blob/master/run.c#L249-L354

```c
    // forward all the layers
    for(unsigned long long l = 0; l < p->n_layers; l++) {

        // attention rmsnorm
        rmsnorm(s->xb, x, w->rms_att_weight + l*dim, dim);

        // key and value point to the kv cache
        int loff = l * p->seq_len * kv_dim; // kv cache layer offset for convenience
        s->k = s->key_cache + loff + pos * kv_dim;
        s->v = s->value_cache + loff + pos * kv_dim;

        // qkv matmuls for this position
        matmul(s->q, s->xb, w->wq + l*dim*dim, dim, dim);
        matmul(s->k, s->xb, w->wk + l*dim*kv_dim, dim, kv_dim);
        matmul(s->v, s->xb, w->wv + l*dim*kv_dim, dim, kv_dim);

        // RoPE relative positional encoding: complex-valued rotate q and k in each head
        for (int i = 0; i < dim; i+=2) {
```

Meta's Llama 2 7B model as an example

tensor	0 :	token_embd.weight	q4_K	[4096, 32000, 1, 1]
tensor	1 :	blk.0.attn_norm.weight	f32	[4096, 1, 1, 1]
tensor	2 :	blk.0.ffn_down.weight	q6_K	[11008, 4096, 1, 1]
tensor	3 :	blk.0.ffn_gate.weight	q4_K	[4096, 11008, 1, 1]
tensor	4 :	blk.0.ffn_up.weight	q4_K	[4096, 11008, 1, 1]
tensor	5 :	blk.0.ffn_norm.weight	f32	[4096, 1, 1, 1]
tensor	6 :	blk.0.attn_k.weight	q4_K	[4096, 4096, 1, 1]
tensor	7 :	blk.0.attn_output.weight	q4_K	[4096, 4096, 1, 1]
tensor	8 :	blk.0.attn_q.weight	q4_K	[4096, 4096, 1, 1]
tensor	9 :	blk.0.attn_v.weight	q6_K	[4096, 4096, 1, 1]
…				
tensor	281 :	blk.31.attn_norm.weight	f32	[4096, 1, 1, 1]
tensor	282 :	blk.31.ffn_down.weight	q6_K	[11008, 4096, 1, 1]
tensor	283 :	blk.31.ffn_gate.weight	q4_K	[4096, 11008, 1, 1]
tensor	284 :	blk.31.ffn_up.weight	q4_K	[4096, 11008, 1, 1]
tensor	285 :	blk.31.ffn_norm.weight	f32	[4096, 1, 1, 1]
tensor	286 :	blk.31.attn_k.weight	q4_K	[4096, 4096, 1, 1]
tensor	287 :	blk.31.attn_output.weight	q4_K	[4096, 4096, 1, 1]
tensor	288 :	blk.31.attn_q.weight	q4_K	[4096, 4096, 1, 1]
tensor	289 :	blk.31.attn_v.weight	q6_K	[4096, 4096, 1, 1]
tensor	290 :	output_norm.weight	f32	[4096, 1, 1, 1]

Question: How many matmuls to generate a token?

Important Notes

- Read:
  - llama2.c: https://github.com/karpathy/llama2.c
  - SGLang https://github.com/sgl-project/sglang
- Due
  - HW0 (Saturday)
    - Don Knuth: https://cs.stanford.edu/~knuth/chatGPT20.txt