



Language Model Architecture

COMP4901Y

Binhang Yuan

Overview

- What is a language model?
- Tokenization:
 - How do we represent language to machines?
- Model categorization:
 - Encoder-only, decoder-only, encoder-decoder.
- Training objectives:
 - How are large language models (LLM) trained?
- Transformer architecture:
 - The main innovation that enabled large language models.

Language Model

What Is a Language Model?

- The classic definition of a *language model (LM)* is a probability distribution over sequences of tokens.
- Suppose we have a vocabulary \mathcal{V} of a set of tokens.
- A language model P assigns each sequence of tokens $x_1, x_2, \dots, x_L \in \mathcal{V}$ to a probability (a number between 0 and 1): $p(x_1, x_2, \dots, x_L) \in [0,1]$.
- The probability intuitively tells us how “good” a sequence of tokens is.
 - For example, if the vocabulary is $\mathcal{V} = \{\text{ate, ball, cheese, mouse, the}\}$, the language model might assign:
$$p(\text{the, mouse, ate, the, cheese}) = 0.02$$
$$p(\text{the, cheese, ate, the, mouse}) = 0.01$$
$$p(\text{mouse, the, the, chesse, ate}) = 0.0001$$

Language Model Generation

- A language model P takes a sequence and returns a probability to assess its goodness.
- We can also generate a sequence given a language model.
- The purest way to do this is to sample a sequence $x_{1:L}$ from the language model P with probability equal to $p(x_{1:L})$ denoted:

$$x_{1:L} \sim P$$

Autoregressive Language Models

- A common way to write the joint distribution $p(x_{1:L})$ of a sequence to $x_{1:L}$ is using the chain rule of probability:

$$p(x_{1:L}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \dots p(x_L|x_{1:L-1}) = \prod_{i=1}^L p(x_i|x_{1:i-1})$$

- In particular, $p(x_i|x_{1:i-1})$ is a conditional probability distribution of the next token x_i given the previous tokens $x_{1:i-1}$.
- An autoregressive language model is one where each conditional distribution $p(x_i|x_{1:i-1})$ can be computed efficiently (e.g., using a feedforward neural network).

Tokenization

Tokenization

- Recall: language model P is a probability distribution over a sequence of tokens where each token comes from some vocabulary \mathcal{V} , e.g.,:

[I, love, cats, and, dogs]

- Natural language doesn't come as a sequence of tokens, but as just a string (concretely, sequence of Unicode characters):

I love cats and dogs

- A tokenizer converts any string into a sequence of tokens:

I love cats and dogs \Rightarrow [I, love, cats, and, dogs]

Split by Space

- The simplest solution is to do: `text.split(' ')`
- This doesn't work for languages such as Chinese, where sentences are written without spaces between words:
 - 我今天去了商店: [I went to the store today.]
- Then there are languages like German that have long compound words:
 - Abwasserbehandlungsanlage: [Wastewater treatment plant]
- Even in English, there are hyphenated words (e.g., father-in-law) and contractions (e.g., don't), which should get split up.

What Makes a Good Tokenization?

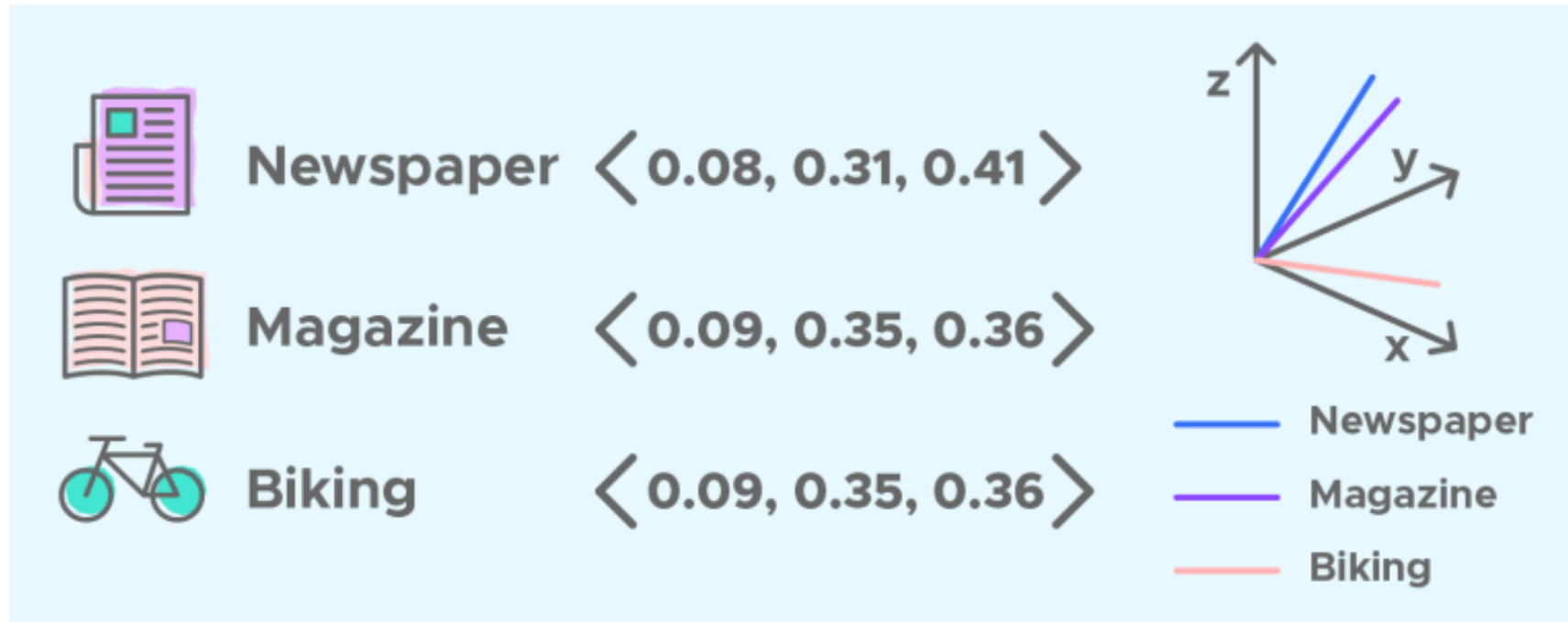
- We don't want too many tokens:
 - The extreme case characters or bytes;
 - The sequence becomes difficult to model.
- We don't want too few tokens:
 - There won't be parameter sharing between words (e.g., should mother-in-law and father-in-law be completely different)?
 - This is especially problematic for morphologically rich languages (e.g., Arabic, Turkish, etc.).
- Each token should be a linguistically or statistically meaningful unit.

Some Encoding Methods

- Byte pair encoding (BPE)
 - Start with each character as its own token and combine tokens that co-occur a lot.
 - <https://arxiv.org/pdf/1508.07909.pdf>
- Unigram model (SentencePiece):
 - Rather than just splitting by frequency, a more “principled” approach is to define an objective function that captures what a good tokenization looks like.
 - <https://arxiv.org/pdf/1804.10959.pdf>

Representation: Word as Vectors

- Tokens can be represented as number index:
[I, love, cats, and, dogs] \Rightarrow [328, 793, 3989, 537, 3255, 269]
- But indices are also meaningless.
- Represent words in a vector space
 - Vector distance \Rightarrow similarity.



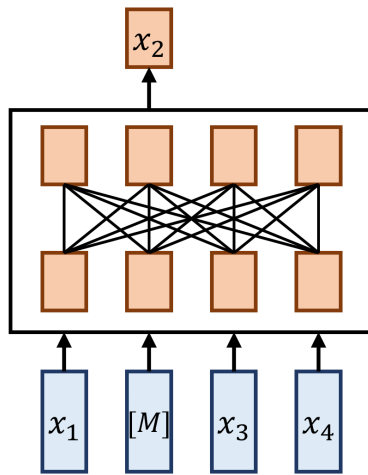
LLM Categorization

Contextual Embeddings

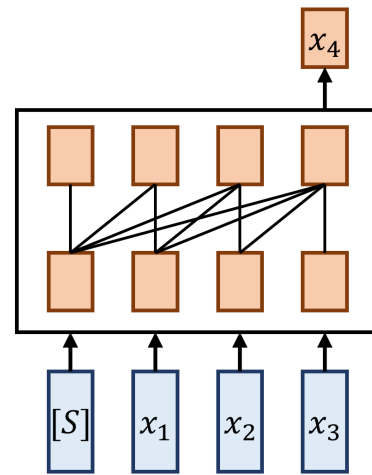
- Language model:
 - Associate a sequence of tokens with a corresponding sequence of contextual embeddings.
- Embedding function (analogous to a feature map for sequences):
 - $\emptyset: \mathcal{V}^L \rightarrow \mathbb{R}^{L \times D}$
 - A token sequence $x_{1:L}[x_1, x_2, \dots, x_L] \in \mathcal{V}^L$
 - Map to $\emptyset(x_{1:L}) \in \mathbb{R}^{L \times D}$
- For example, if $D = 2$:
 - $[\text{I, love, cats, and, dogs}] \Rightarrow [328, 793, 3989, 537, 3255, 269] \Rightarrow \begin{bmatrix} (0.2, 0.3) \\ (0.8, 0.7) \\ (0.2, 0.1) \\ (0.0, 0.7) \\ (0.1, 0.0) \\ (0.1, 0.4) \end{bmatrix}$

Types of language models

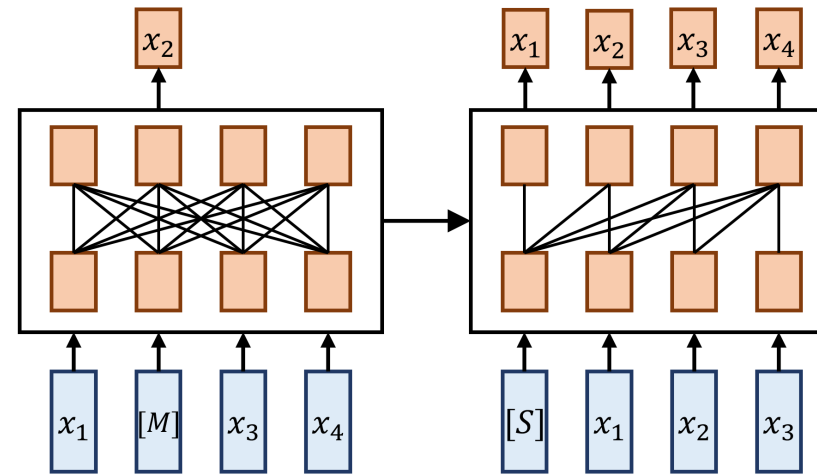
- Encoder-only models (BERT, RoBERTa, etc.)
- Encoder-decoder models (BART, T5, etc.)
- **Decoder-only models** (GPT-3, Llama-3, Deepseek-V3, etc.)



Encoder-only



Decoder-only



Encoder-decoder

Encoder-only Models

- Encoder-only models produce contextual embeddings but cannot be used directly to generate text:

$$x_{1:L} \Rightarrow \emptyset(x_{1:L})$$

- These contextual embeddings are generally used for classification tasks (sometimes boldly called natural language understanding tasks).
 - Example: sentiment classification: `[[CLS],the,movie,was,great]` \Rightarrow positive.
- Pros:
 - Contextual embedding for x_i can depend bidirectionally on both the left context ($x_{1:i-1}$) and the right context ($x_{i+1:L}$).
- Cons:
 - Cannot naturally generate completions.
 - Requires more ad-hoc training objectives (masked language modeling).

Decoder-only Models

- Decoder-only models are our standard autoregressive language models.
- Given a prompt $x_{1:i}$ produces both contextual embeddings and a distribution over next tokens x_{i+1} , and recursively, over the entire completion $x_{i+1:L}$:
$$x_{1:i} \Rightarrow \phi(x_{1:i}), p(x_{i+1}|x_{1:i})$$
- Example: text autocomplete
 - `[[CLS],the,movie,was]⇒great`
- Pro:
 - Can naturally generate completions.
 - Simple training objective (maximum likelihood).
- Con:
 - Contextual embedding for x_i can only depend **unidirectionally** on both the left context ($x_{1:i-1}$).

Encoder-decoder Models

- Encoder-decoder models can be the best of both worlds: they can use bidirectional contextual embeddings for the input $x_{1:L}$ and can generate the output $y_{1:L}$:

$$x_{1:L} \Rightarrow \phi(x_{1:L}), p(y_{1:L} | \phi(x_{1:L}))$$

- Example: table-to-text generation
 - [name,:,Clowns,|,eatType,:,coffee,shop] \Rightarrow [Clowns,is,a,coffee,shop].
- Pro:
 - Can naturally generate outputs.
- Con:
 - Requires more ad-hoc training objectives.

LLM Training Objectives

Decoder-only Model Training Objectives

- Recall that an autoregressive language model defines a conditional distribution:
 $p(x_i | x_{1:i-1})$
- Define it as follows:
 - Map $x_{1:i-1}$ to contextual embedding $\phi(x_{1:i-1})$;
 - Apply an embedding matrix $E \in \mathbb{R}^{D \times |\mathcal{V}|}$ to obtain scores for each token $\phi(x_{1:i-1})_i E$;
 - Exponentiate and normalize it to produce the distribution over x_i .
- Put them together:

$$p(x_{i+1} | x_{1:i}) = \text{softmax}(\phi(x_{1:i})_i E)$$

Decoder-only Model Training Objectives

- Maximum likelihood. Let θ be all the parameters of large language models.
- Let \mathcal{D} be the training data consisting of a set of sequences. We can then follow the maximum likelihood principle and define the following negative log-likelihood objective function:

$$\mathcal{O}(\theta) = \sum_{x_{1:L} \in \mathcal{D}} -\log p_{\theta}(x_{1:L}) = \sum_{x_{1:L} \in \mathcal{D}} \sum_{i=1}^L -\log p_{\theta}(x_i | x_{1:i-1})$$

- Then we can use the SGD optimizers we have talked to optimize this loss function.

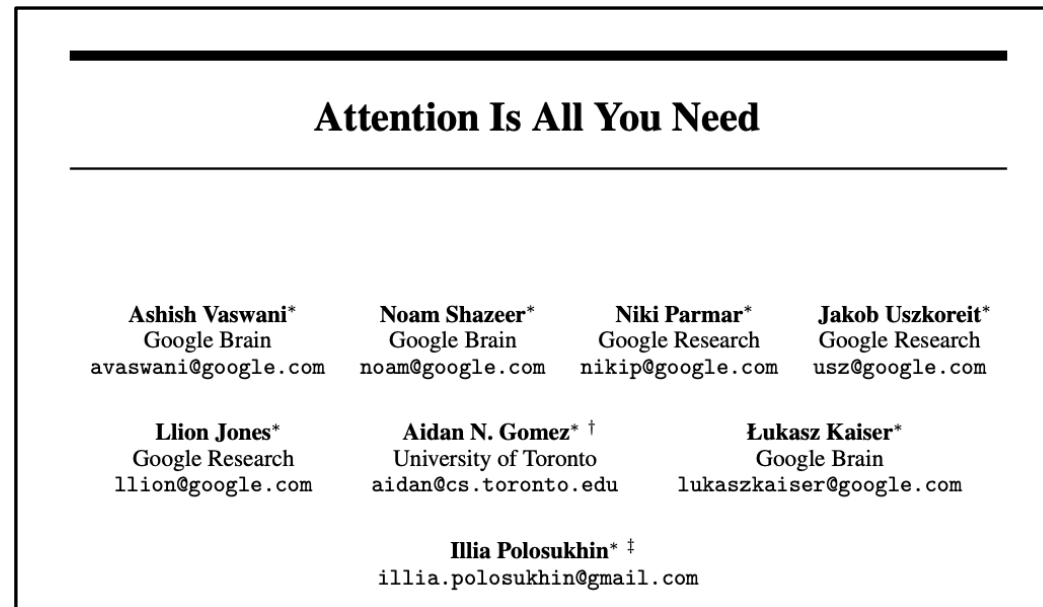
LLM Architecture Details

EmbedToken

- Convert sequences of tokens into sequences of vectors.
- **EmbedToken** does exactly this by looking up each token in an embedding matrix $E \in \mathbb{R}^{|\mathcal{V}| \times D}$, a parameter that will be learned from data
- $\text{EmbedToken}(x_{1:L}: \mathcal{V}^L) \rightarrow \mathbb{R}^{L \times D}$:
 - Turns each token x_i in the sequence $x_{1:L}$ into a vector $E_{x_i} \in \mathbb{R}^D$;
 - Return $[E_{x_1}, E_{x_2}, \dots, E_{x_L}]$.
- These are context-independent word embeddings.
- Next the **TransformerBlock**(s) takes these context-independent embeddings and maps them into contextual embeddings.

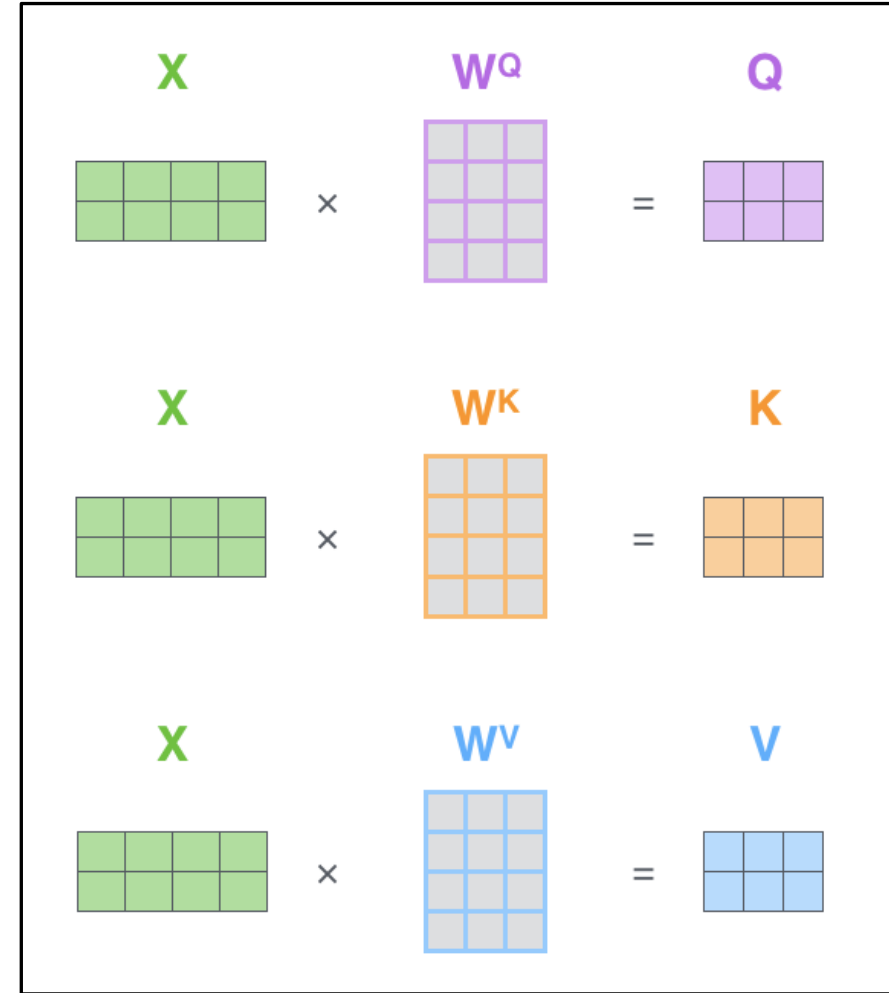
TransformerBlock

- **TransformerBlock**(s) takes these context-independent embeddings and maps them into contextual embeddings.
- $\text{TransformerBlocks}(x_{1:L}: \mathbb{R}^{L \times D}) \rightarrow \mathbb{R}^{L \times D}$:
 - Process each element x_i in the sequence $x_{1:L}$ with respect to other elements.
- **TransformerBlock**(s) are the building blocks of decoder-only (GPT-2, GPT-3), encoder-only (BERT, RoBERTa), and decoder-encoder (BART, T5) models.



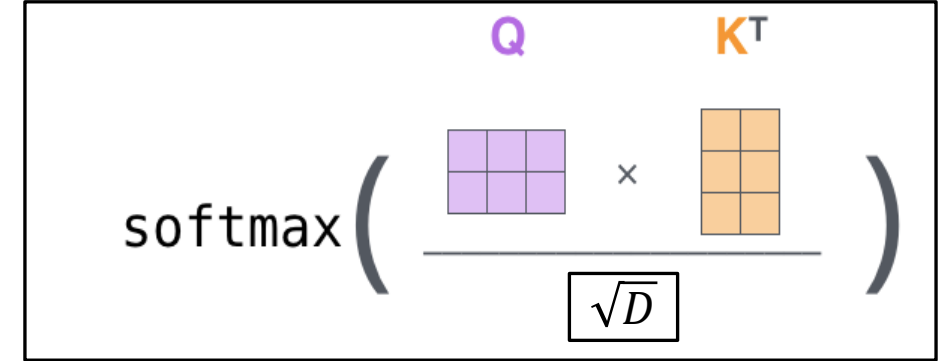
Attention Mechanism-1

- **First step:** in each transformer block, for each token, we create a query vector, a key vector, and a value vector by multiplying the embedding by three weight matrices.
- Formally, for each token $x_i \in \mathbb{R}^D$:
 - Query: $q_i = x_i \times W^Q$
 - key: $k_i = x_i \times W^K$
 - Value: $v_i = x_i \times W^V$
- In the tensor representation for sequence $x_{1:L} \in \mathbb{R}^{L \times D}$:
 - Query: $Q = q_{1:L} = x_{1:L} \times W^Q$
 - key: $K = k_{1:L} = x_{1:L} \times W^K$
 - Value: $V = v_{1:L} = x_{1:L} \times W^V$



Attention Mechanism-2

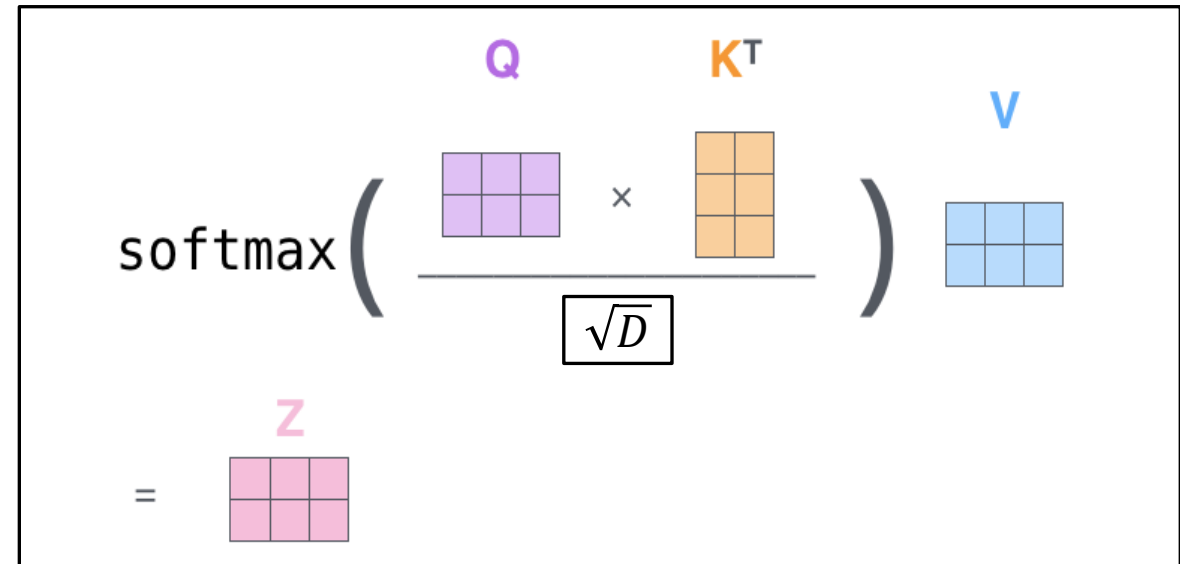
- **Second step:** Calculate a score determining how much focus to place on other parts of the input sentence as we encode a token at a certain position.
- Calculated by:
 - Taking the dot product of the query vector with the key vector of the respective word we're scoring;
 - Divide the scores by the square root of the dimension of the key vectors;
 - Conduct a softmax operation.
- $\text{score} = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)$



The diagram illustrates the calculation of attention scores. It shows the formula $\text{softmax}\left(\frac{Q \times K^T}{\sqrt{D}}\right)$ enclosed in a large black rectangular box. Above the purple 2x3 grid representing matrix Q is a purple 'Q'. Above the orange 3x2 grid representing matrix K^T is an orange 'K^T'. A multiplication symbol '×' is placed between the two grids. A horizontal line is drawn below the grids, with a box containing \sqrt{D} centered underneath it. The entire expression is enclosed in large parentheses.

Attention Mechanism-3

- **Third step:** combine the value and the score.
 - $Z = \text{att} = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right) V$
- **Multi-head Attention:** there can be multiple aspects (e.g., syntax, semantics) we would want to match on.
- To accommodate this, we can simultaneously have **multiple attention heads (e.g. H heads)** and simply combine their outputs, e.g:
 - $Z = [\text{att}_1, \text{att}_2, \dots, \text{att}_H]$
- The attention output will be:
 - $\text{Out} = ZW^O$



Feedforward Layer

- After the attention layer, the output is put to a feed-forward neural network, then sends out the output upwards to the next encoder.
 - $x'_{1:L} = \text{relu}(\text{Out}W^1)W^2$
 - W^1, W^2 are two weight matrices;
 - $x'_{1:L}$ is the output embedding for the current layer and the input of the next layer.
- Summarize a common weight dimension in one **TransformerBlock**:
 - Attention layer: $W^Q, W^K, W^V, W^O \in \mathbb{R}^{D \times D}$
 - Feedforward layer: $W^1 \in \mathbb{R}^{D \times 4D}, W^2 \in \mathbb{R}^{4D \times D}$

TransformerBlocks($x \in \mathbb{R}^{L \times D}$) $\rightarrow x' \in \mathbb{R}^{L \times D}$

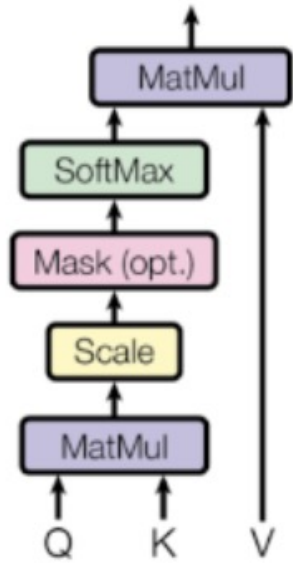
- L is the sequence length;
- D is the model dimension;
- Multi-head attention:
 $D = n_h \times H$
- H is the head dimension;
- n_h is the number of heads.

Computation	Input	Output
$Q = xW^Q$	$x \in \mathbb{R}^{L \times D}, W^Q \in \mathbb{R}^{D \times D}$	$Q \in \mathbb{R}^{L \times D}$
$K = xW^K$	$x \in \mathbb{R}^{L \times D}, W^K \in \mathbb{R}^{D \times D}$	$K \in \mathbb{R}^{L \times D}$
$V = xW^V$	$x \in \mathbb{R}^{L \times D}, W^V \in \mathbb{R}^{D \times D}$	$V \in \mathbb{R}^{L \times D}$
$[Q_1, Q_2 \dots, Q_{n_h}] = \text{Partion}_{-1}(Q)$	$Q \in \mathbb{R}^{L \times D}$	$Q_i \in \mathbb{R}^{L \times H}, i = 1, \dots n_h$
$[K_1, K_2 \dots, K_{n_h}] = \text{Partion}_{-1}(K)$	$K \in \mathbb{R}^{L \times D}$	$K_i \in \mathbb{R}^{L \times H}, i = 1, \dots n_h$
$[V_1, V_2 \dots, V_{n_h}] = \text{Partion}_{-1}(V)$	$V \in \mathbb{R}^{L \times D}$	$V_i \in \mathbb{R}^{L \times H}, i = 1, \dots n_h$
$\text{Score}_i = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \dots n_h$	$Q_i, K_i \in \mathbb{R}^{L \times H}$	$\text{score}_i \in \mathbb{R}^{L \times L}$
$Z_i = \text{score}_i V_i, i = 1, \dots n_h$	$\text{score}_i \in \mathbb{R}^{L \times L}, V_i \in \mathbb{R}^{L \times H}$	$Z_i \in \mathbb{R}^{L \times H}$
$Z = \text{Merge}_{-1}([Z_1, Z_2 \dots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{L \times H}, i = 1, \dots n_h$	$Z \in \mathbb{R}^{L \times D}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{L \times D}, W^O \in \mathbb{R}^{D \times D}$	$\text{Out} \in \mathbb{R}^{L \times D}$
$A = \text{Out} W^1$	$\text{Out} \in \mathbb{R}^{L \times D}, W^1 \in \mathbb{R}^{D \times 4D}$	$A \in \mathbb{R}^{L \times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{L \times 4D}$	$A' \in \mathbb{R}^{L \times 4D}$
$x' = A'W^2$	$A' \in \mathbb{R}^{L \times 4D}, W^2 \in \mathbb{R}^{4D \times D}$	$x' \in \mathbb{R}^{L \times D}$

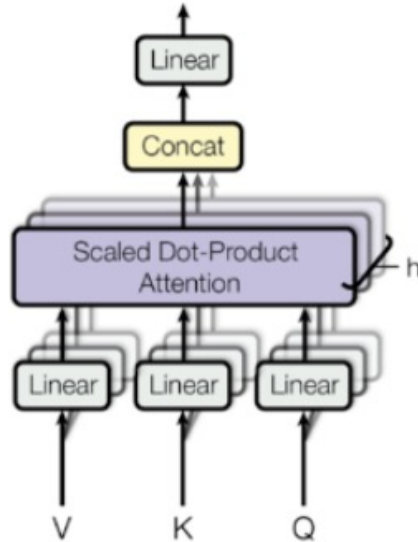
Other Components

- Residual connections:
 - Instead of simply return $\text{TransformerBlock}(x_{1:L})$
 - Return: $x_{1:L} + \text{TransformerBlock}(x_{1:L})$
- Layer normalization:
 - $\text{LayerNorm}(x_{1:L}) = \alpha \frac{x_{1:L} - \mu}{\sigma} + \beta$
 - μ is the mean; σ is the standard deviation.
- Positional embeddings:
 - So far, the embedding of a token doesn't depend on where it occurs in the sequence, which is not sensible.
 - $$\begin{cases} \text{PosEmb}(i, 2j) = \sin(\frac{i}{10000^{2j/D}}) \\ \text{PosEmb}(i, 2j + 1) = \cos(\frac{i}{10000^{2j/D}}) \end{cases}$$

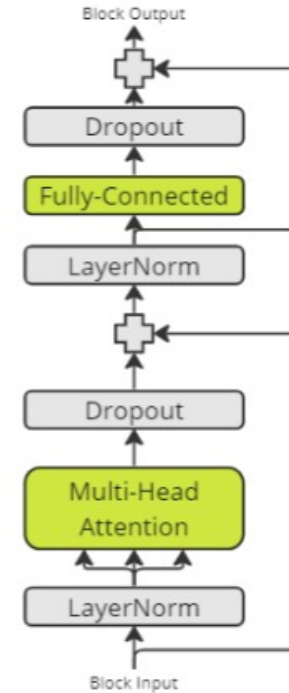
Put Them Together



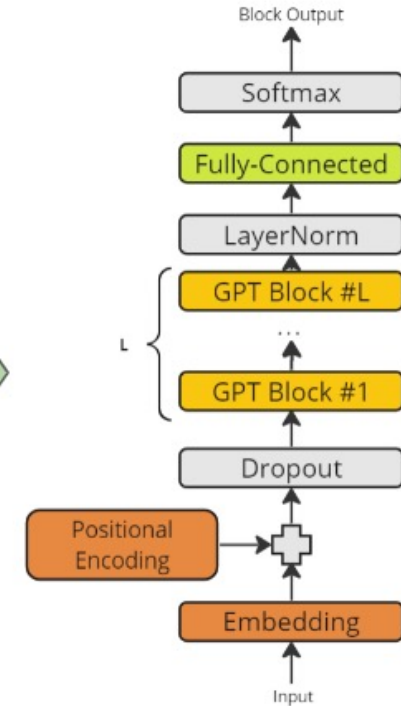
Scale Causal Attention



Multi-Head Attention

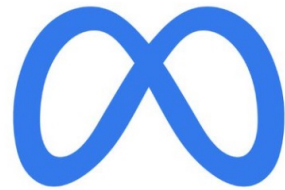


Transformer Block



GPT Model

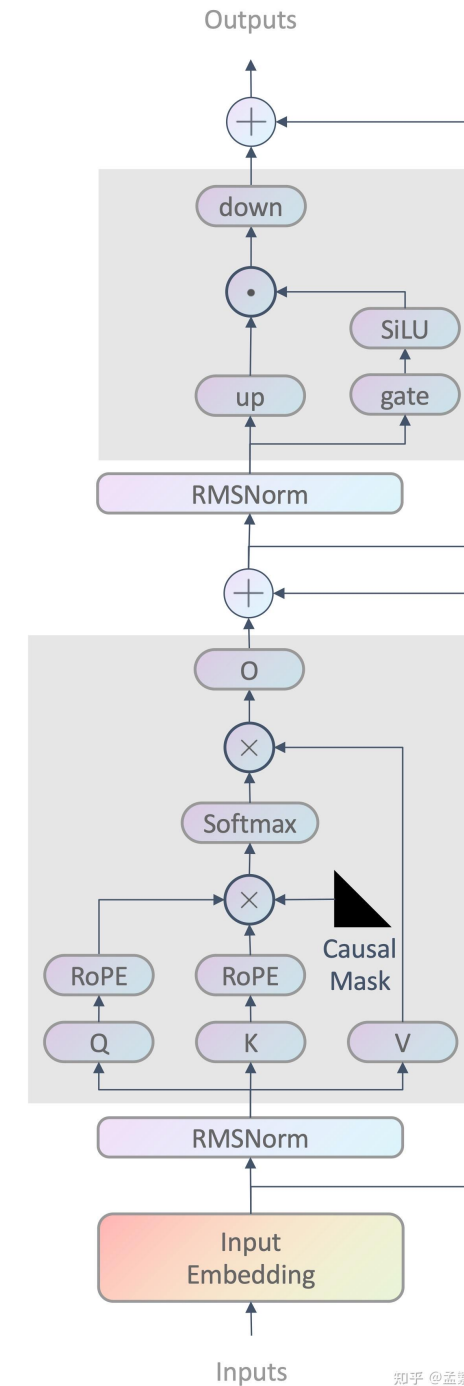
LLM Architecture Case Study



LLaMa 3

Llama-3 Block Overview

- RMSNorm (Root Mean Square Norm);
- RoPE (Rotary Position Embedding);
- GQA (Group Query Attention);
- SiLU activation function in MLP.



Llama-3 RMSNorm

- RMSNorm: $\text{RMSNorm}(x_{1:L}) = \alpha \frac{x_{1:L}}{\sqrt{\mu^2 + \epsilon}}$
 - μ is the mean.
 - 40% Speed-up compared with LayerNorm.

RMSNorm

```
CLASS torch.nn.RMSNorm(normalized_shape, eps=None, elementwise_affine=True, device=None, dtype=None) [SOURCE]
```

Applies Root Mean Square Layer Normalization over a mini-batch of inputs.

This layer implements the operation as described in the paper [Root Mean Square Layer Normalization](#)

$$y_i = \frac{x_i}{\text{RMS}(x)} * \gamma_i, \quad \text{where} \quad \text{RMS}(x) = \sqrt{\epsilon + \frac{1}{n} \sum_{i=1}^n x_i^2}$$

The RMS is taken over the last D dimensions, where D is the dimension of `normalized_shape`. For example, if `normalized_shape` is `(3, 5)` (a 2-dimensional shape), the RMS is computed over the last 2 dimensions of the input.

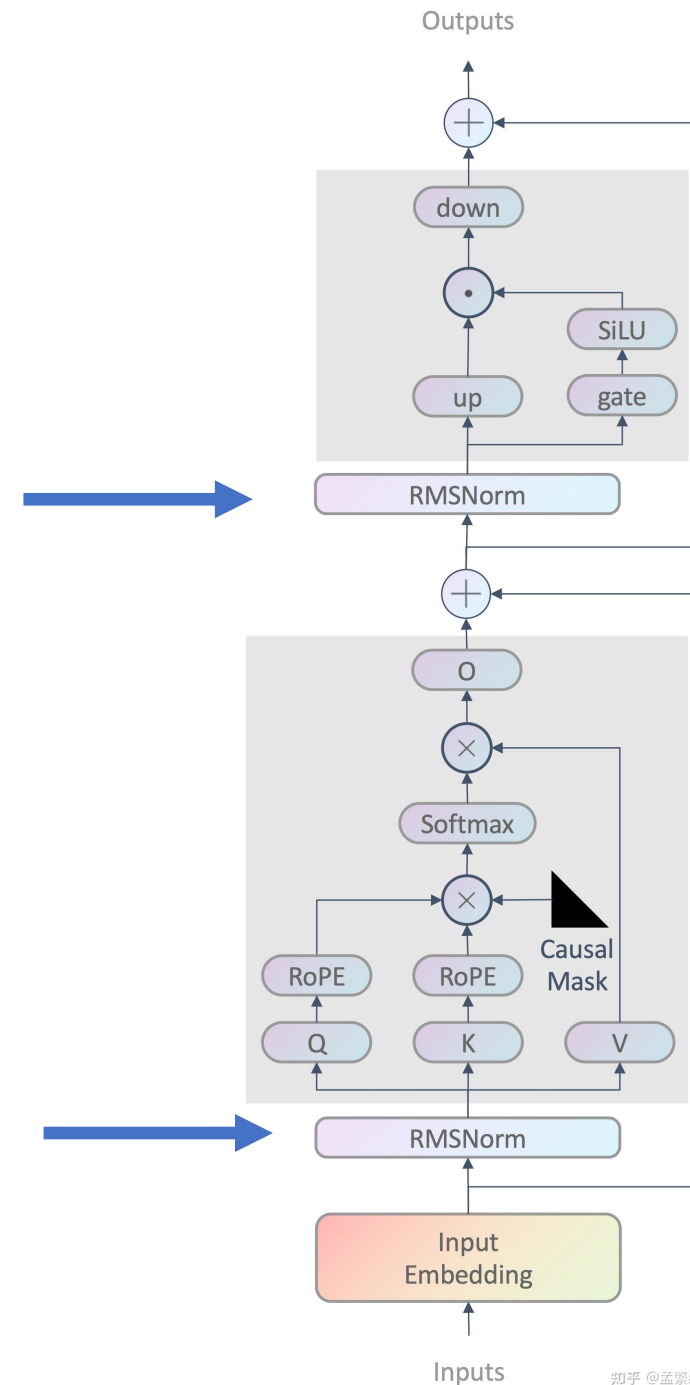
Parameters

- normalized_shape** (*int or list or torch.Size*) – input shape from an expected input of size

`[* × normalized_shape[0] × normalized_shape[1] × ... × normalized_shape[-1]`

If a single integer is used, it is treated as a singleton list, and this module will normalize over the last dimension which is expected to be of that specific size.

- eps** (*Optional[float]*) – a value added to the denominator for numerical stability. Default: `torch.finfo(x.dtype).eps()`
- elementwise_affine** (*bool*) – a boolean value that when set to `True`, this module has learnable per-element affine parameters initialized to ones (for weights). Default: `True`.



Llama-3 RoPE

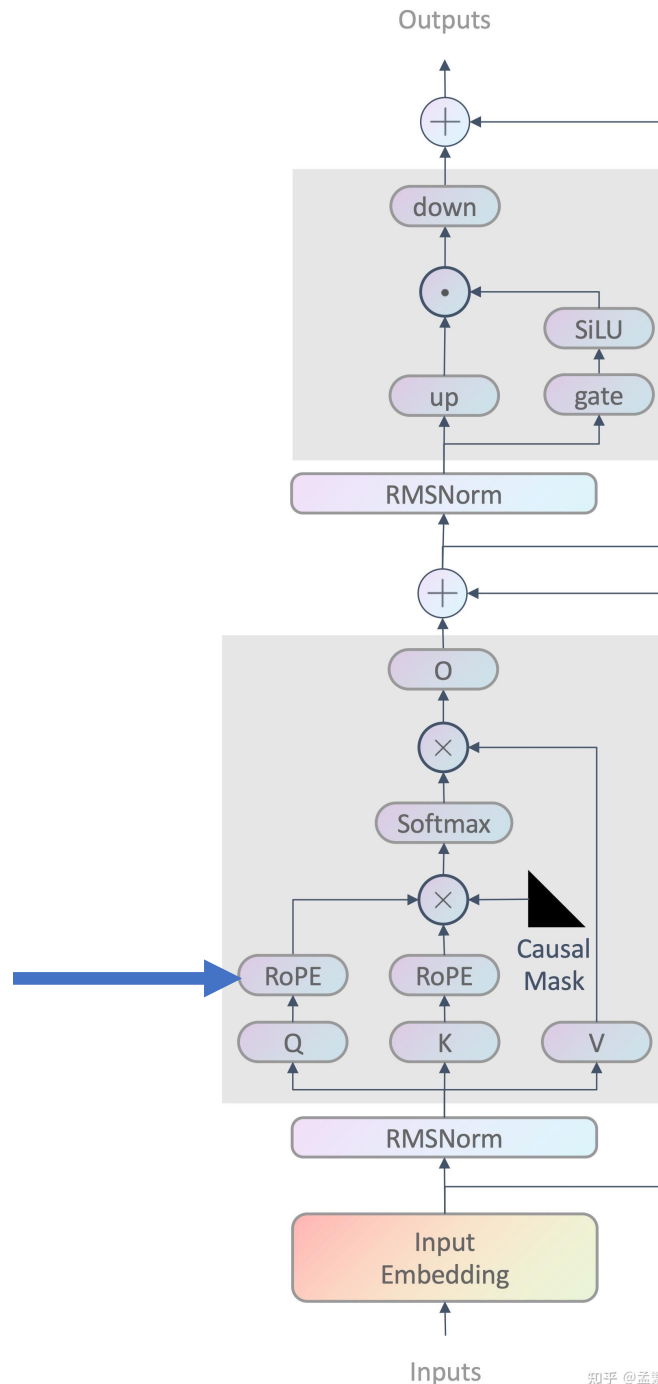
- RoPE incorporates both absolute and relative positional information.
- Computation efficient implementation transform a position m :

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

- \otimes indicates element-wise multiplication;

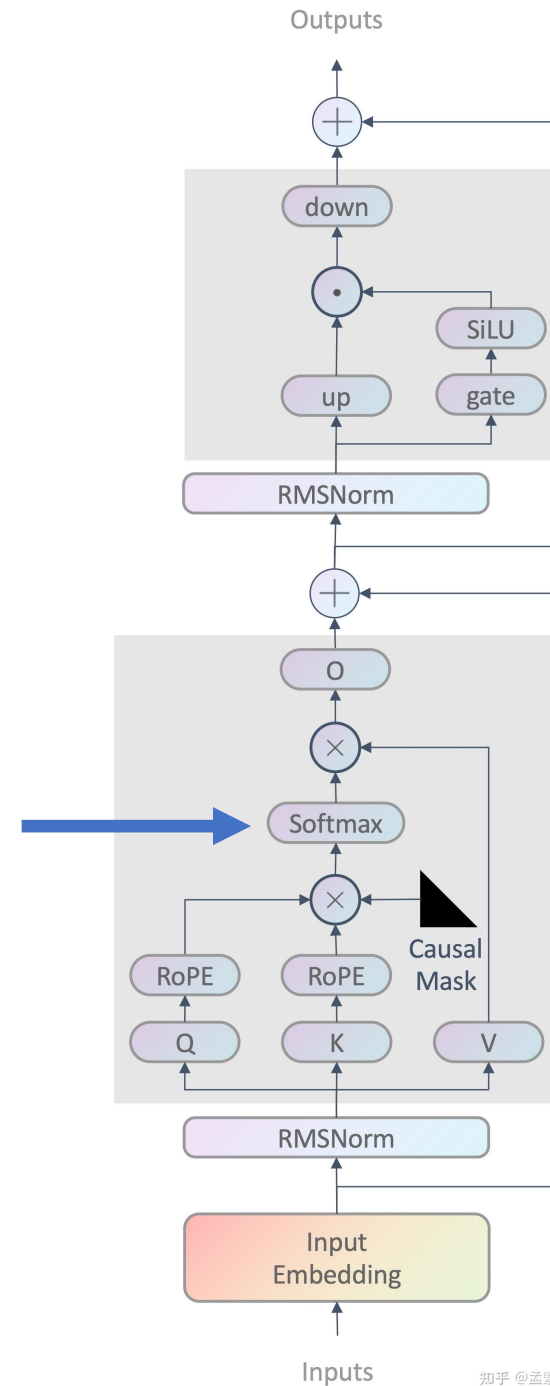
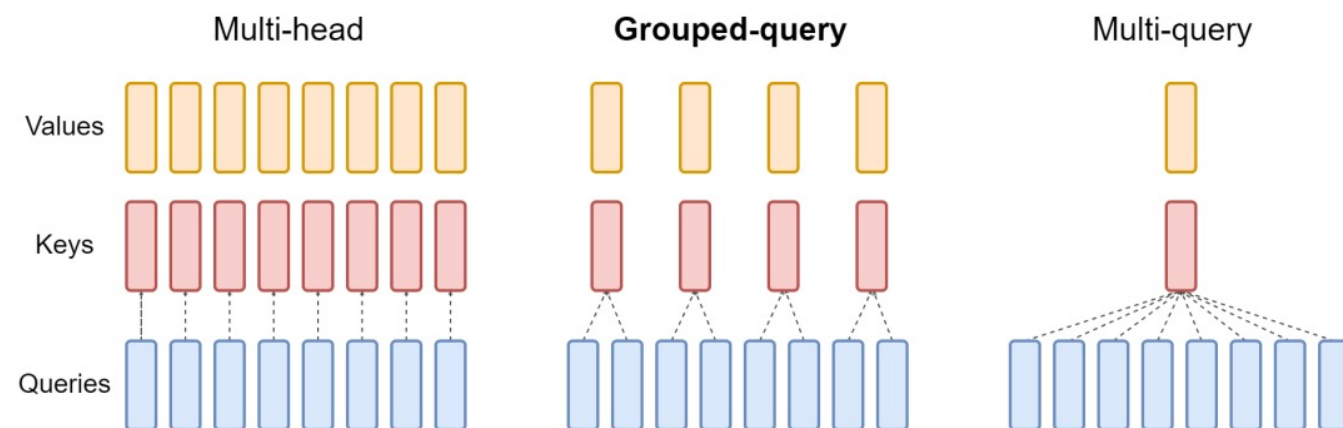
- $\theta_i = 10000^{-\frac{2i}{L}}$

- <https://arxiv.org/pdf/2104.09864>



Llama-3 GQA

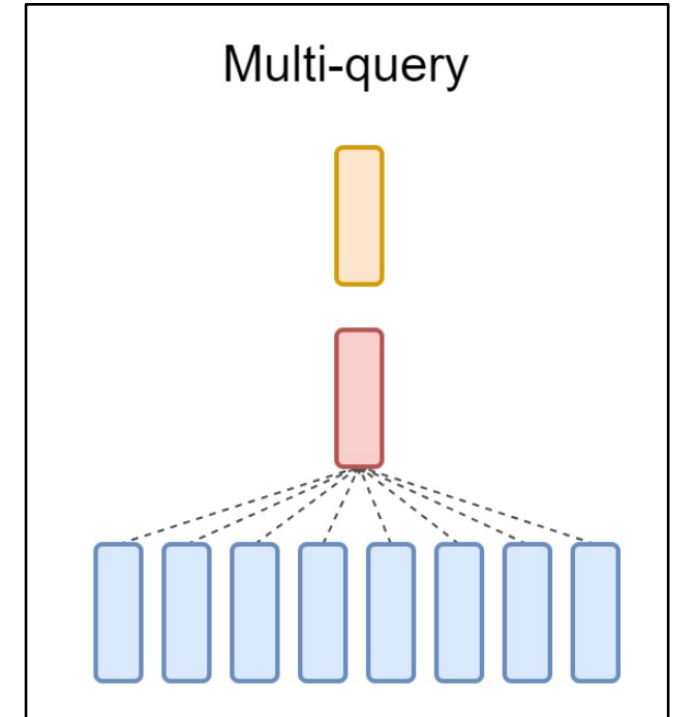
- Replace multi-head attention with grouped-query attention.



Multi-Query Attention (MQA)

- The idea is simple yet effective:
 - Use multiple query heads but only **a single** key and value head.

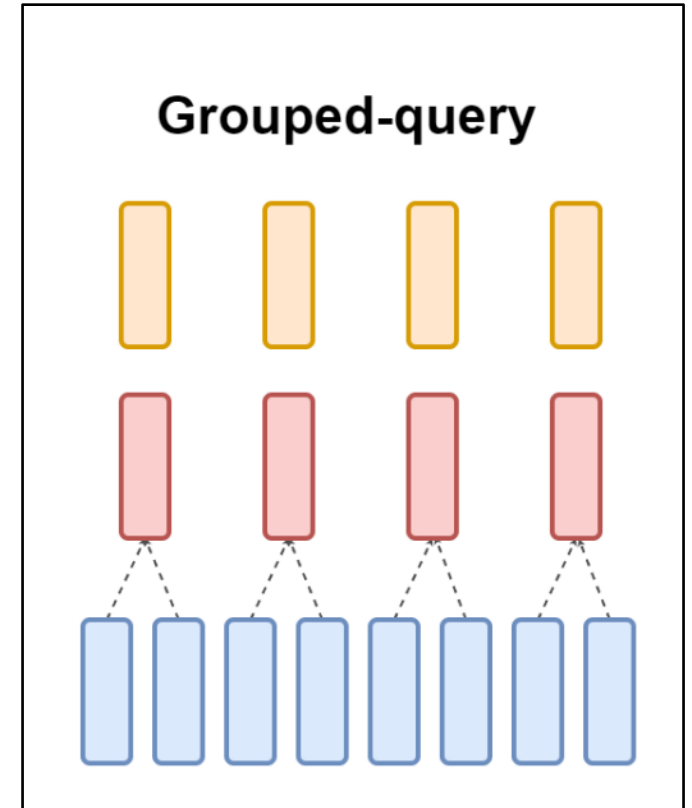
Computation	Input	Output
$Q = xW^Q$	$x \in \mathbb{R}^{L \times D}, W^Q \in \mathbb{R}^{D \times D}$	$Q \in \mathbb{R}^{L \times D}$
$K = xW^K$	$x \in \mathbb{R}^{L \times D}, W^K \in \mathbb{R}^{D \times H}$	$K \in \mathbb{R}^{L \times H}$
$V = xW^V$	$x \in \mathbb{R}^{L \times D}, W^V \in \mathbb{R}^{D \times H}$	$V \in \mathbb{R}^{L \times H}$
$[Q_1, Q_2, \dots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L \times D}$	$Q_i \in \mathbb{R}^{L \times H}, i = 1, \dots, n_h$
$\text{Score}_i = \text{softmax}(\frac{Q_i K^T}{\sqrt{D}}), i = 1, \dots, n_h$	$Q_i, K \in \mathbb{R}^{L \times H}$	$\text{score}_i \in \mathbb{R}^{L \times L}$
$Z_i = \text{score}_i V, i = 1, \dots, n_h$	$\text{score}_i \in \mathbb{R}^{L \times L}, V \in \mathbb{R}^{L \times H}$	$Z_i \in \mathbb{R}^{L \times H}$



Group-Query Attention (GQA)

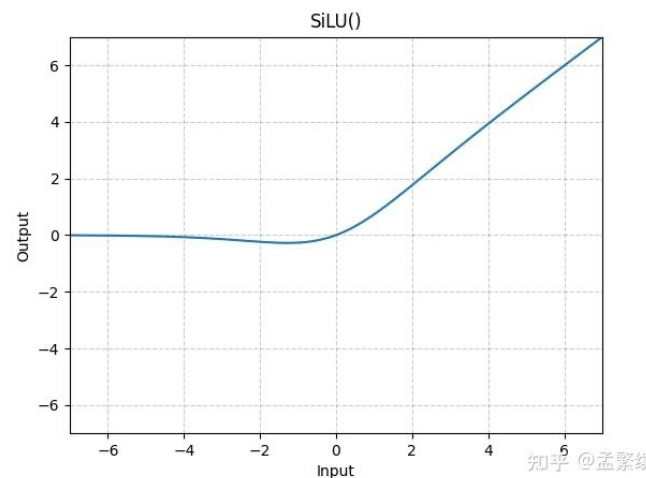
- The trade-off between MHA and MQA:
 - Divide query heads into g groups, each sharing a single key head and value head;
 - MHA: $g = n_H$; MQG: $g = 1$.

Computation	Input	Output
$Q = xW^Q$	$x \in \mathbb{R}^{L \times D}, W^Q \in \mathbb{R}^{D \times D}$	$Q \in \mathbb{R}^{L \times D}$
$K = xW^K$	$x \in \mathbb{R}^{L \times D}, W^K \in \mathbb{R}^{D \times gH}$	$K \in \mathbb{R}^{L \times gH}$
$V = xW^V$	$x \in \mathbb{R}^{L \times D}, W^V \in \mathbb{R}^{D \times gH}$	$V \in \mathbb{R}^{L \times gH}$
$[Q_1, Q_2, \dots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L \times D}$	$Q_i \in \mathbb{R}^{L \times H}, i = 1, \dots, n_h$
$[K_1, K_2, \dots, K_g] = \text{Partition}_{-1}(K)$	$K \in \mathbb{R}^{L \times gH}$	$K_i \in \mathbb{R}^{L \times H}, i = 1, \dots, g$
$[V_1, V_2, \dots, V_g] = \text{Partition}_{-1}(V)$	$V \in \mathbb{R}^{L \times gH}$	$V_i \in \mathbb{R}^{L \times H}, i = 1, \dots, g$
$\text{Score}_i = \text{softmax}(\frac{Q_i K_{[i/g]}^T}{\sqrt{D}}), i = 1, \dots, n_h$	$Q_i, K_{[i/g]} \in \mathbb{R}^{L \times H}$	$\text{score}_i \in \mathbb{R}^{L \times L}$
$Z_i = \text{score}_i V_{[i/g]}, i = 1, \dots, n_h$	$\text{score}_i \in \mathbb{R}^{L \times L}, V \in \mathbb{R}^{L \times H}$	$Z_i \in \mathbb{R}^{L \times H}$

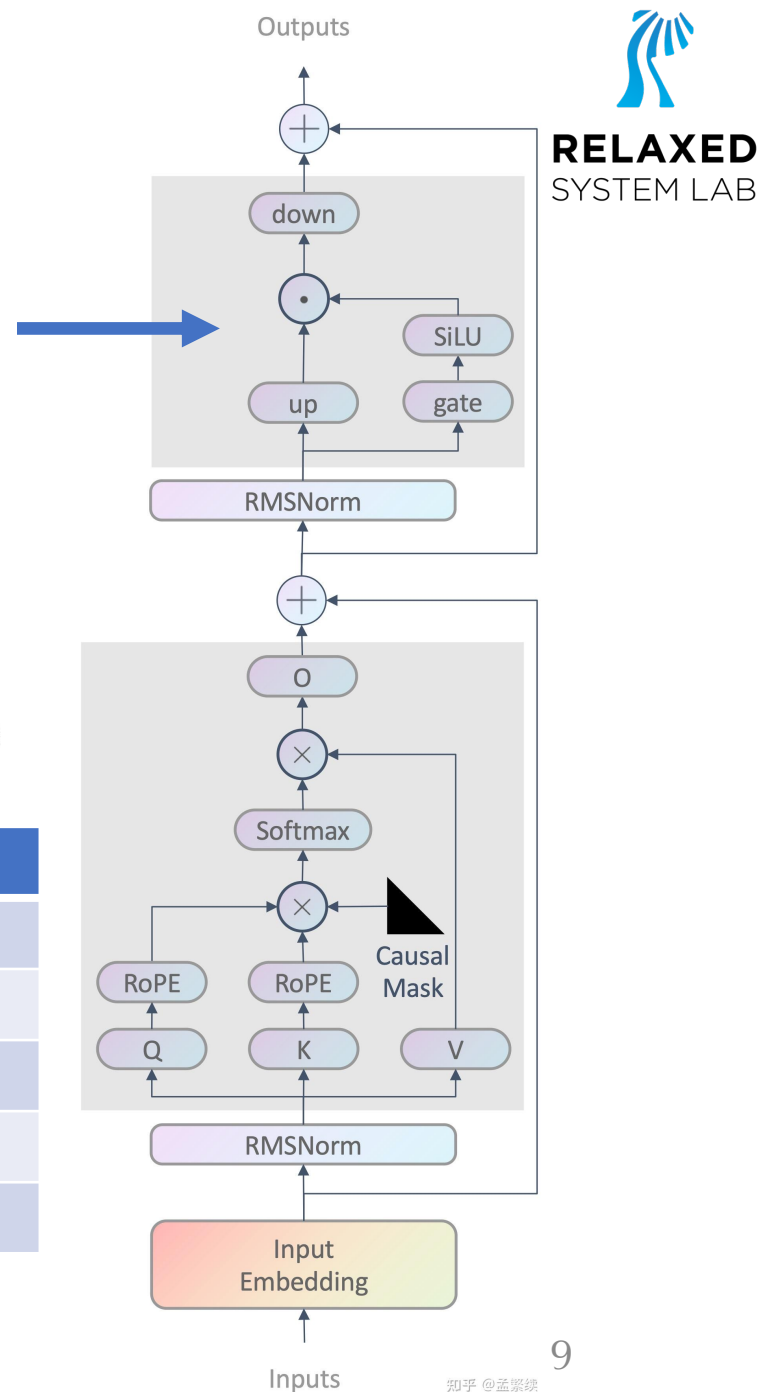


Llama-3 SiLU MLP

- Empirically shown to enhance model quality in various tasks.



Computation	Input	Output
$A = \text{Out} W^1$	$\text{Out} \in \mathbb{R}^{L \times D}, W^1 \in \mathbb{R}^{D \times 4D}$	$A \in \mathbb{R}^{L \times 4D}$
$B = \text{Out} W^2$	$\text{Out} \in \mathbb{R}^{L \times D}, W^2 \in \mathbb{R}^{D \times 4D}$	$B \in \mathbb{R}^{L \times 4D}$
$B' = \text{SwiGLU}(B)$	$B \in \mathbb{R}^{L \times 4D}$	$B' \in \mathbb{R}^{L \times 4D}$
$B'' = A \otimes B'$	$A \in \mathbb{R}^{L \times 4D}, B' \in \mathbb{R}^{L \times 4D}$	$B'' \in \mathbb{R}^{L \times 4D}$
$x' = B'' W^3$	$B'' \in \mathbb{R}^{L \times 4D}, W^3 \in \mathbb{R}^{4D \times D}$	$x' \in \mathbb{R}^{L \times D}$



References

- https://scholar.harvard.edu/sites/scholar.harvard.edu/files/binxuw/files/mlfs_tutorial_nlp_transformer_ssl_updated.pdf
- <https://jalammar.github.io/illustrated-transformer/>
- <https://stanford-cs324.github.io/winter2022/lectures/introduction/>
- <https://stanford-cs324.github.io/winter2022/lectures/modeling/>
- <https://stanford-cs324.github.io/winter2022/lectures/training/>
- <https://zhuanlan.zhihu.com/p/636784644>
- <https://arxiv.org/pdf/2104.09864>