https://hao-ai-lab.github.io/dsc204a-f25/

DSC 204A: Scalable Data Systems
Fall 2025

Staff

Instructor: Hao Zhang

TAs: Mingjia Huo, Yuxuan Zhang

@haozhangml @haoailab

haozhang@ucsd.edu

Logistics

- **Fall 2025 Student Evaluations of Teaching were sent**
  - **Again: if 80% of you finish the evaluation, all will get 2 bonus points.**

  - **Completion rate as of today: 58%**

- **Exam recitation session: next Monday evening (exact time TBD)**

High-level Picture

Data

Model

Compute

✅ $\{x_i\}^n{}_{i=1}$

✅ Math primitives (mostly matmul)

❓Make them run on (clusters of) different kinds of hardware

✅A repr that expresses the computation using primitives

Focus of the rest of lectures

Data

✅ $\{x_i\}^n{}_{i=1}$

Model

✅ Math primitives (mostly matmul)

✅ A repr that expresses the computation using primitives

Compute

❓Make **LLMs** run on (**large clusters** of) **GPUs**

Large Language Models

- **Transformers, Attentions**
- Serving and inference
- Parallelization
- Attention optimization

Next Token Prediction

$$P(next\,word \mid prefix)$$

	surfing	0.4
San Diego has very nice _	weather	0.5
	snow	0.01
San Francisco is a city of _	innovation	0.6
	homeless	0.3

Next Token Prediction

Probability("San Diego has very nice weather")
= P("San Diego") P("has" | "San Diego")P("very" | "San Diego has")P("city" | ...)...P("weather" | ...)

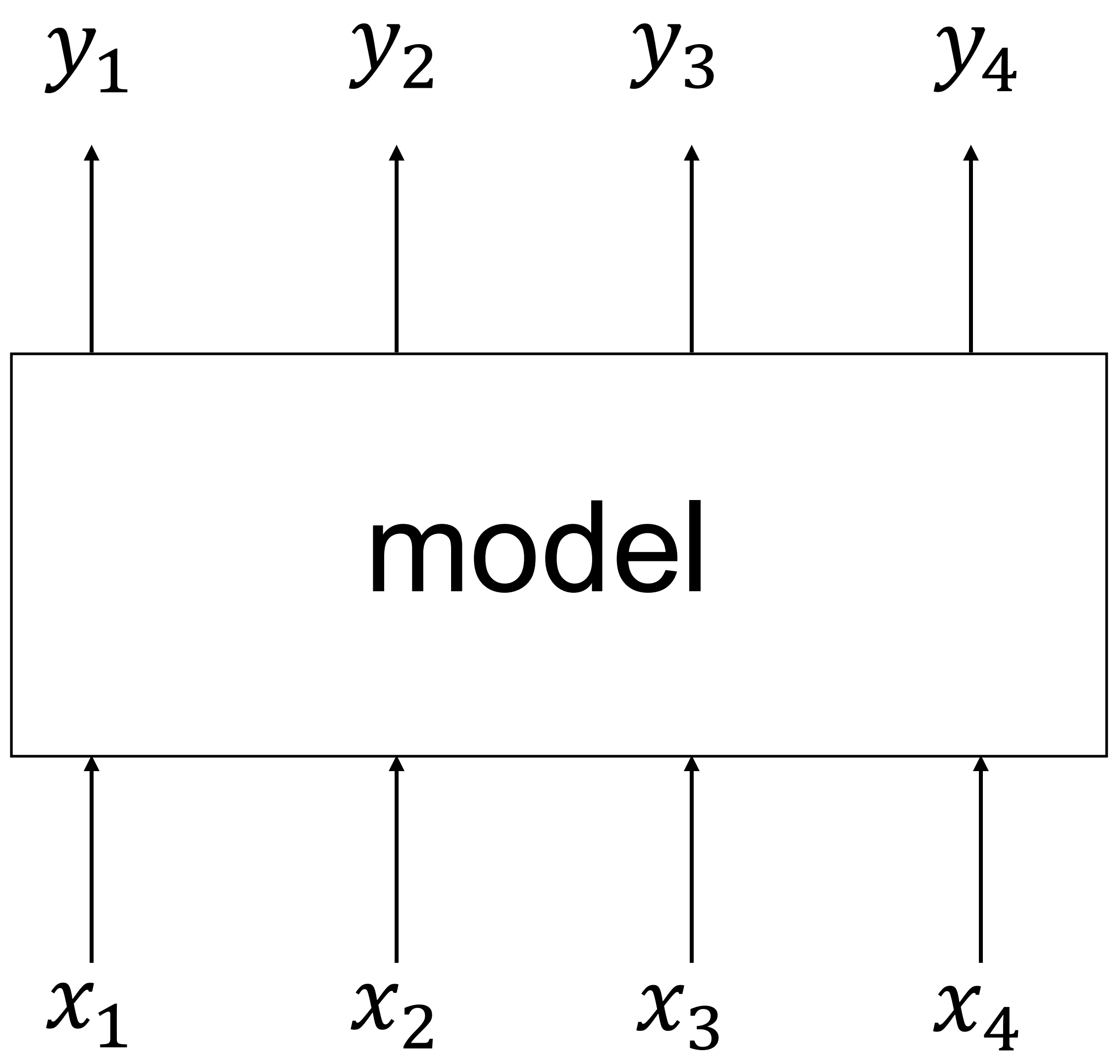
$$\mathrm{Max}\,Prob(x_{1:T}) = \prod_{t=1}^{T} P(x_{t+1}|x_{1\ldots t})$$

MLE on observed data $x_{1:T}$,

This is next token prediction.
Predicting using seq2seq NNs.

Sequence Prediction

Take a set of input sequence, predict the output sequence



….

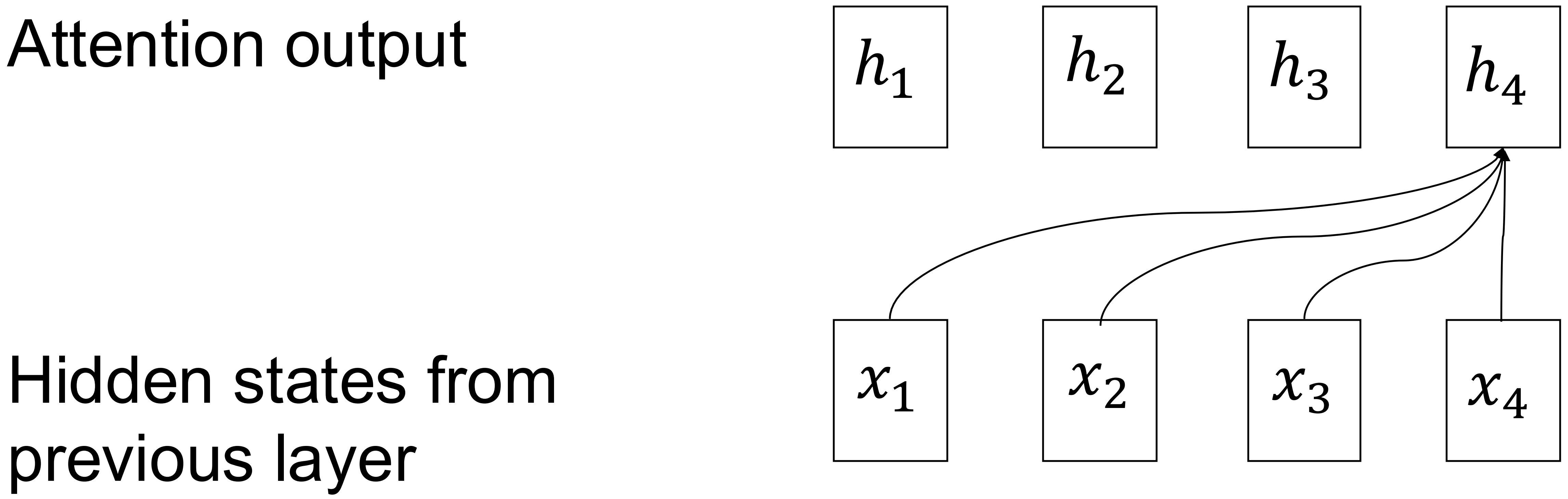
$$\prod_{t=1}^{T} P(x_{t+1} | x_{1...t})$$

Predict each output based on history $y_t = f_\theta(x_{1:t})$

There are many ways to build up the predictive model

“Attention” Mechanism

Generally refers to the approach that weighted combine individual states



$$h_t = \sum_{i=1}^{t} s_i x_t$$

Intuitively s_i is “attention score” that computes how relevant the position i’s input is to this current hidden output

There are different methods to decide how attention score is being computed

Self-Attention Operation

Self attention refers to a particular form of attention mechanism.

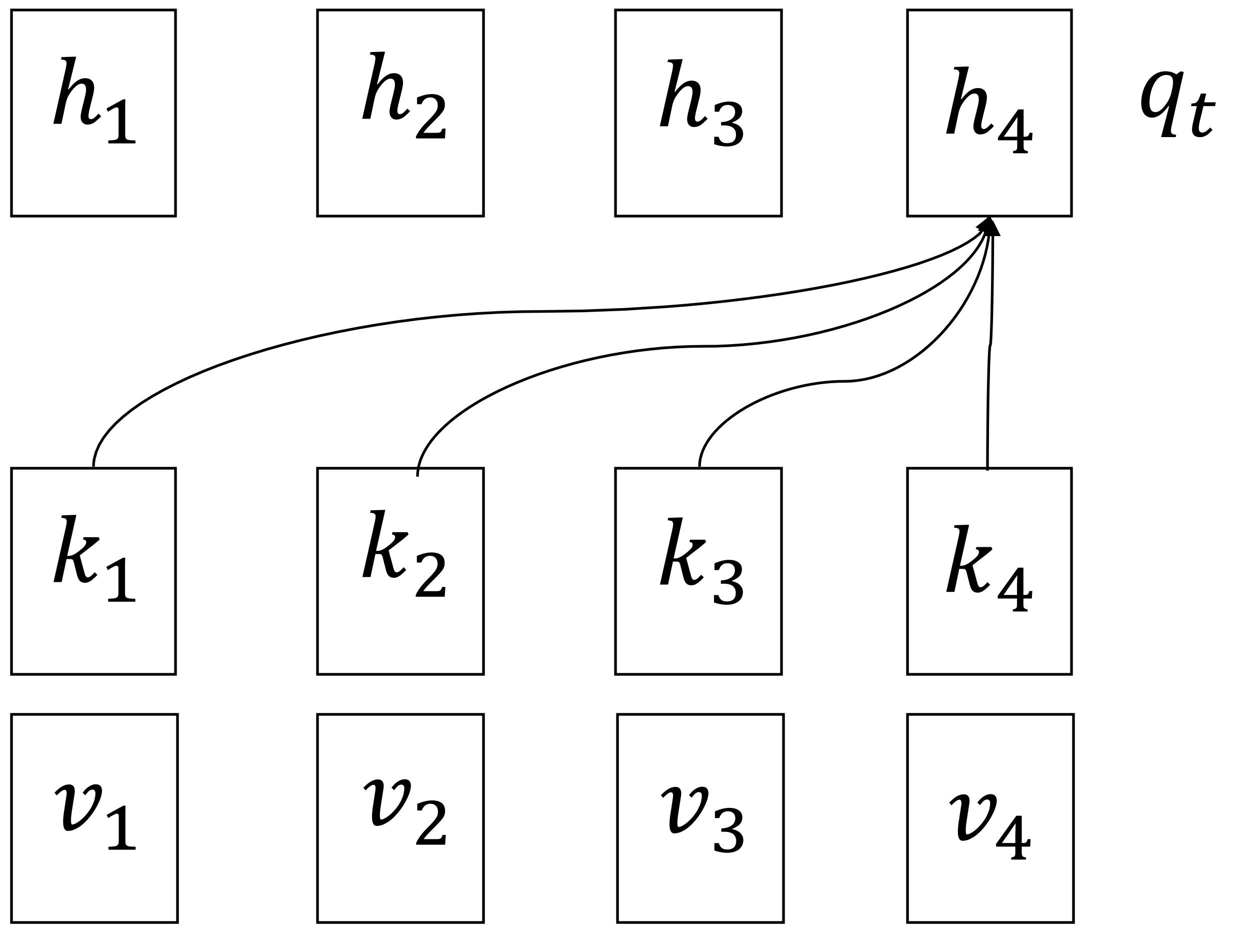
Given three inputs $Q, K, V \in \mathbb{R}^{T \times d}$ (“queries”, “keys”, “values”)

Define the self-attention as:

$$\mathrm{SelfAttention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}}\right)V$$

A Closer Look at Self-Attention

Use q_t, k_t, v_t to refers to row t of the K matrix



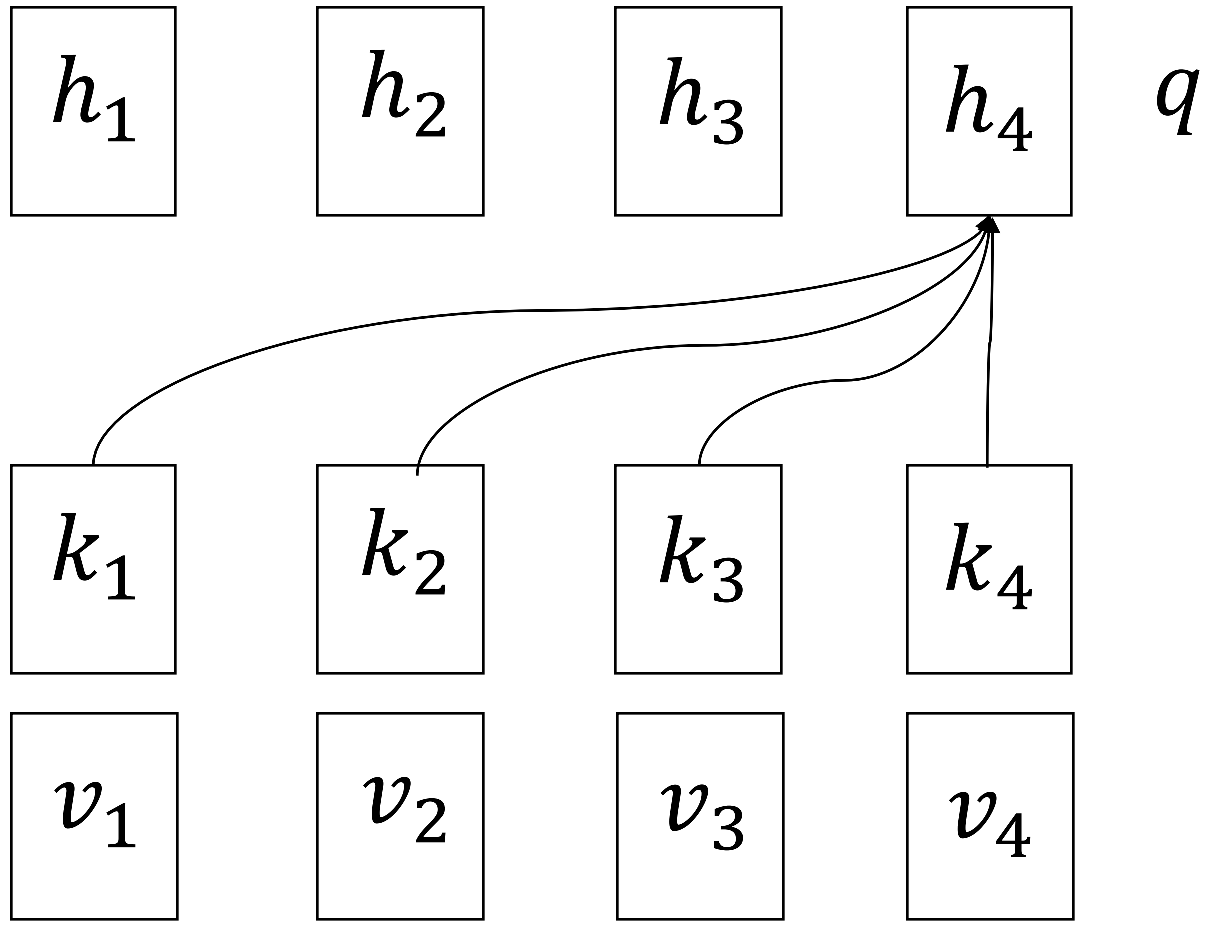
Ask the following question:

How to compute the output h_t, based on q_t, K, V one timestep t

To keep presentation simple, we will drop suffix t and just use q to refer to q_t in next few slide

A Closer Look at Self-Attention

Use q_t, k_t, v_t to refers to row t of the K matrix



Conceptually, we compute the output in the following two steps:

Pre-softmax “attention score”

$$s_i = \frac{1}{\sqrt{d}} q k_i^T$$

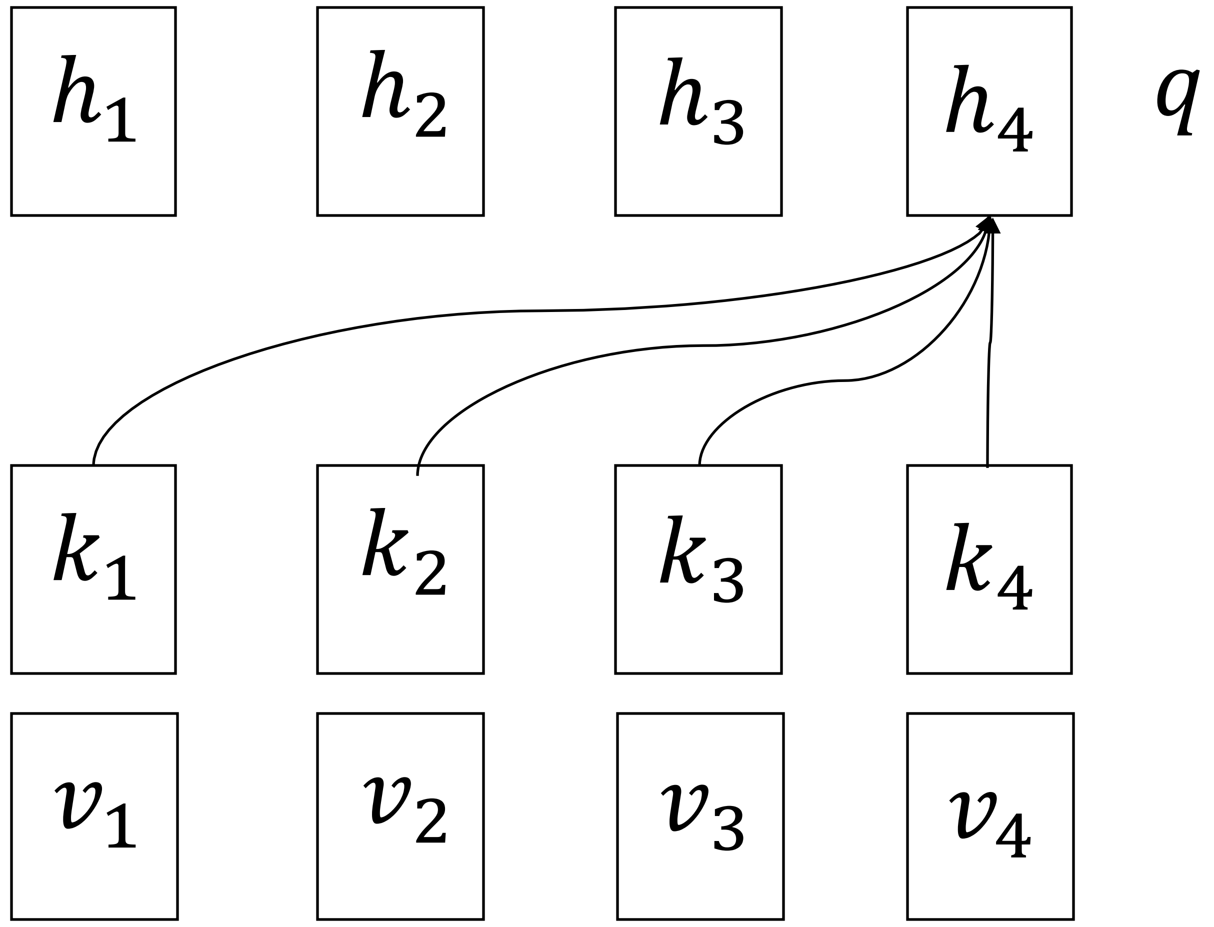
Weighed average via softmax

$$h = \sum_i \mathrm{softmax}(s)_i v_i = \frac{\sum_i \exp(s_i) v_i}{\sum_j \exp(s_j)}$$

Intuition: s_i computes the relevance of k_i to the query q,
then we do weighted sum of values proportional to their relevance

Comparing the Matrix Form and the Decomposed Form

Use q_t, k_t, v_t to refers to row t of the K matrix



$$\mathrm{SelfAttention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}}\right)V$$

Pre-softmax “attention score”

$$S_{ti} = \frac{1}{\sqrt{d}} q_t k_i^T$$

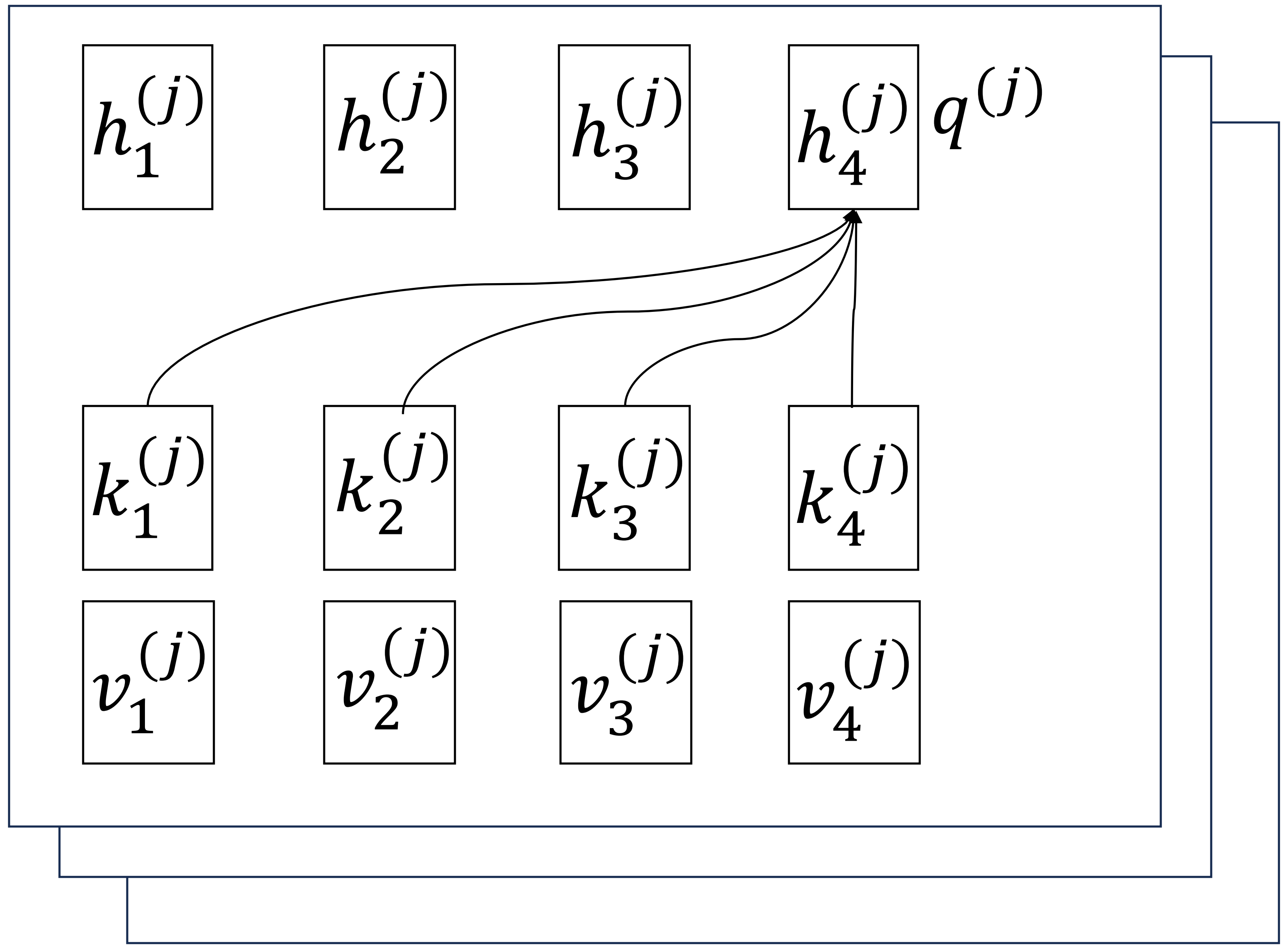
Weighed average via softmax

$$h_t = \sum_i \mathrm{softmax}\left(S_{t,:}\right)_i v_i = \mathrm{softmax}\left(S_{t,:}\right)V$$

Intuition: s_i computes the relevance of k_i to the query q,
then we do weighted sum of values proportional to their relevance

Multi-Head Attention

Have multiple "attention heads" $Q^{(j)}, K^{(j)}, V^{(j)}$ denotes j-th attention head



Apply self-attention in each attention head

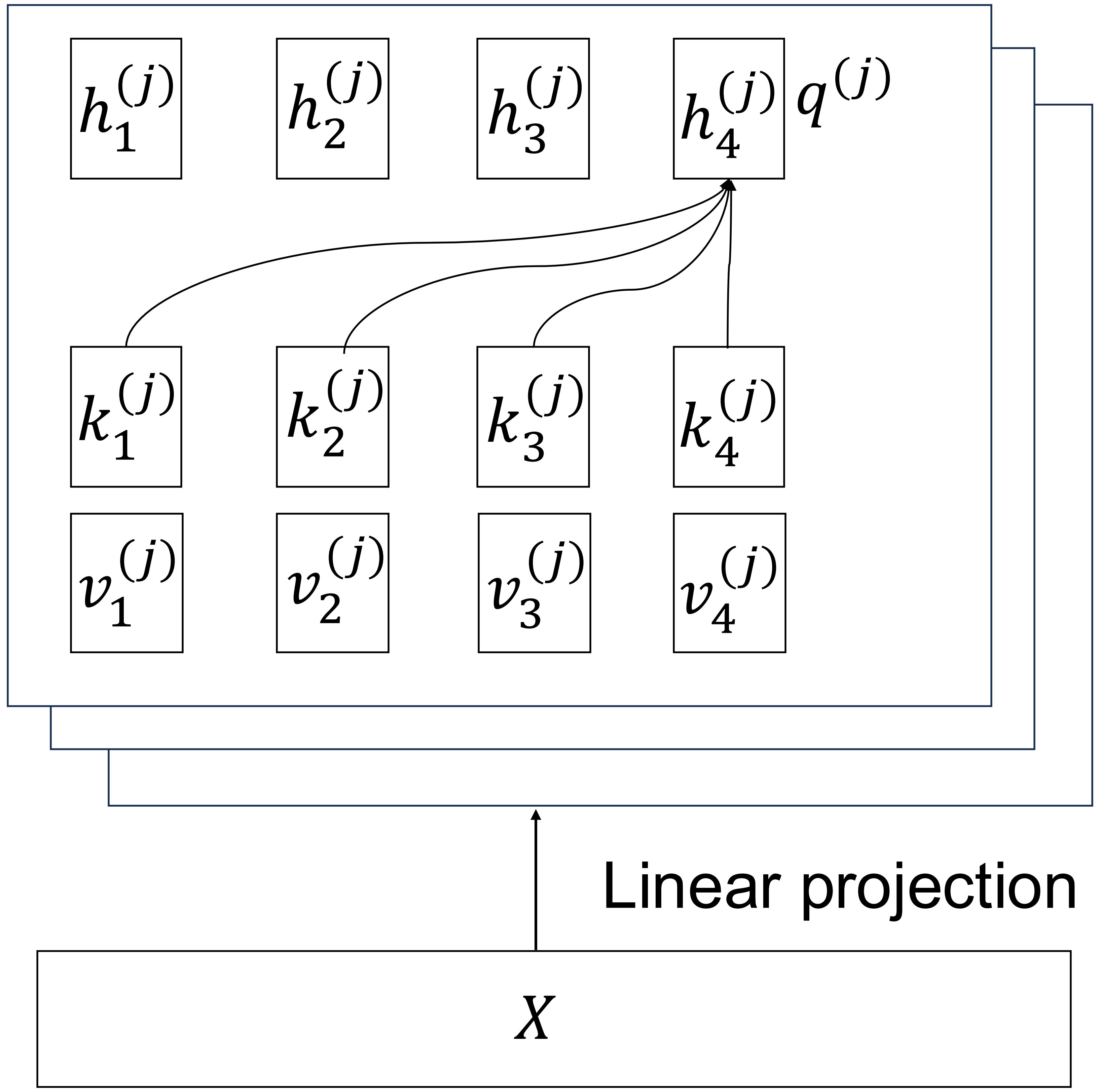
$$\mathrm{SelfAttention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}}\right)V$$

Concatenate all output heads together as output

Each head can correspond to different kind of information.
Sometimes we can share the heads: GQA(group query attention) all heads share K, V but have different Q

How to get Q K V?

Obtain Q, K, V from previous layer's hidden state X by linear projection



$$Q = XW_q$$
$$K = XW_K$$
$$V = XW_V$$

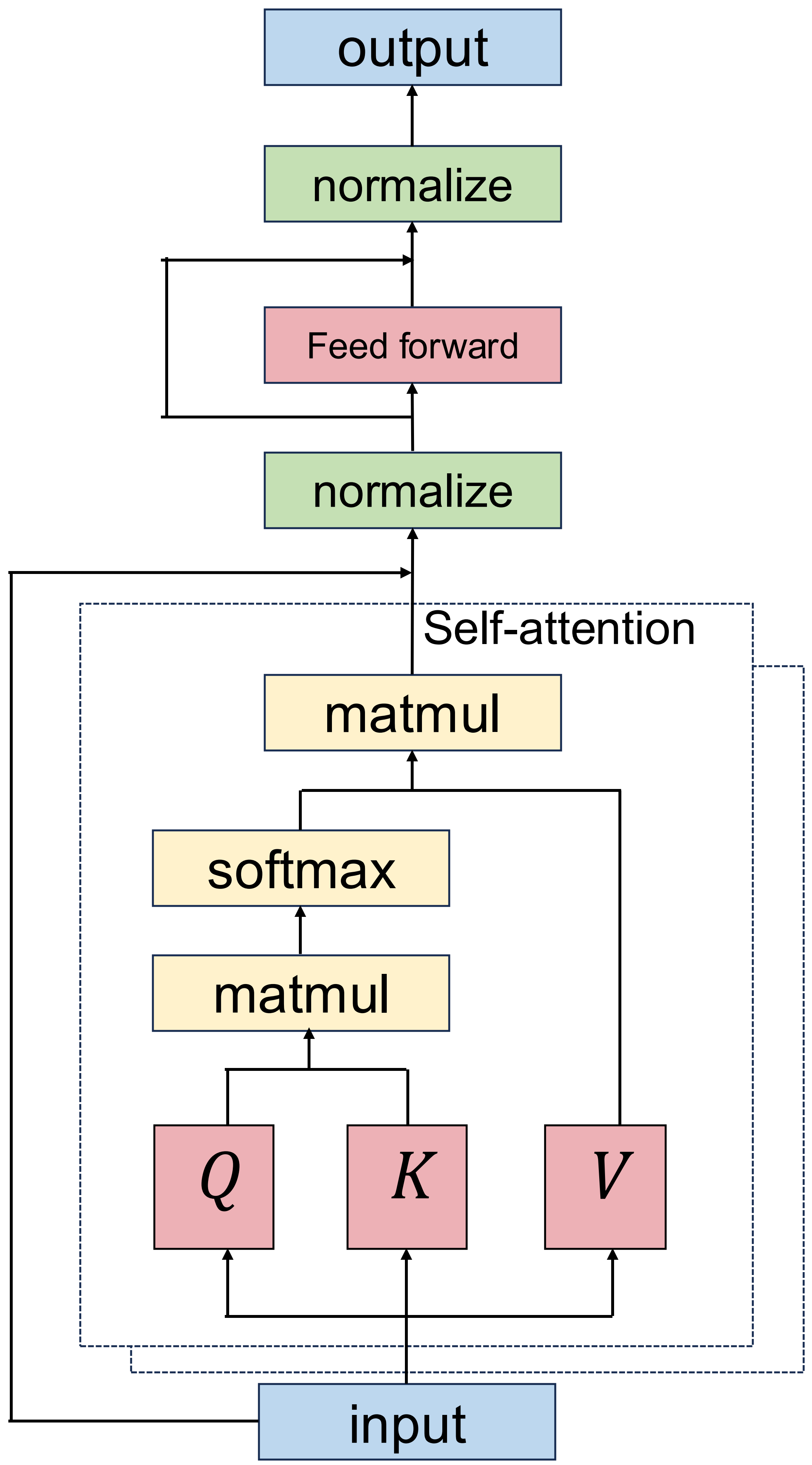
Can compute all heads and Q, K, V together then split/reshape out into individual Q, K, V with multiple heads

Transformer Block

A typical transformer block

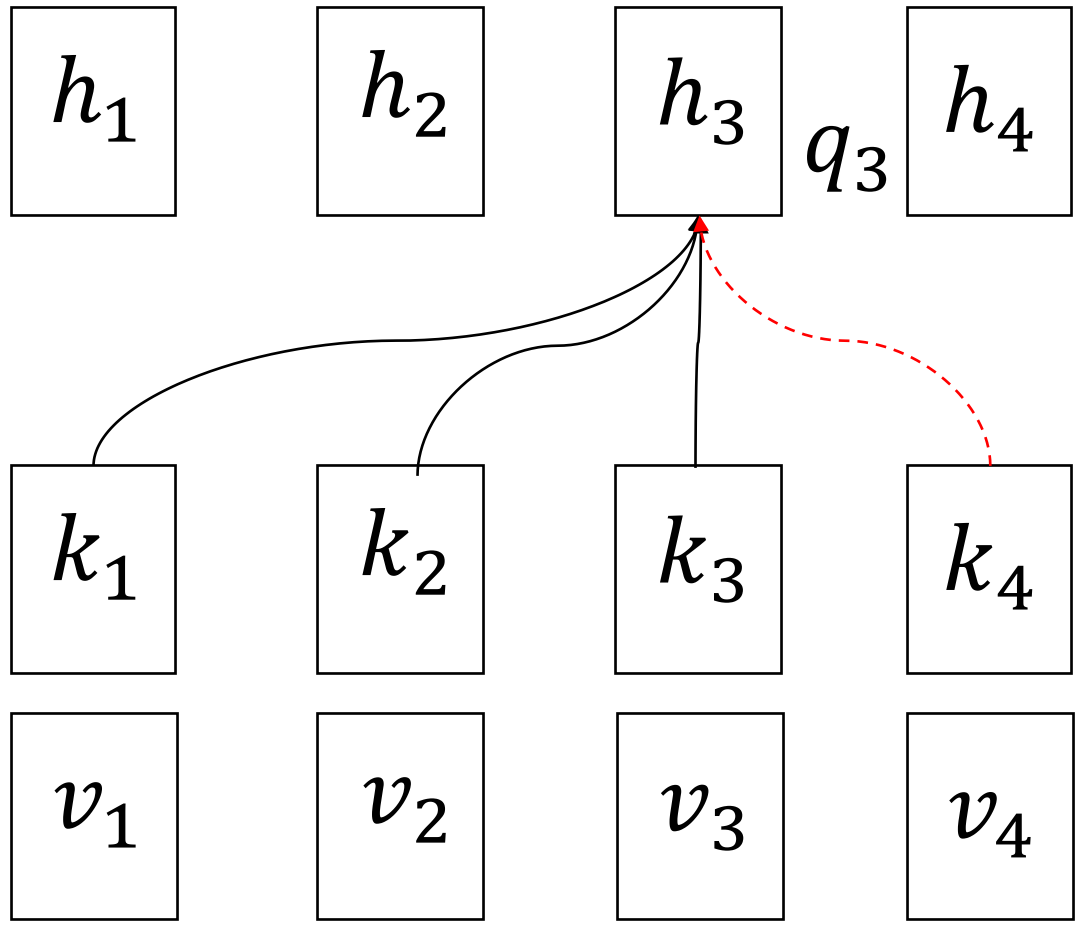
$$Z = \mathrm{SelfAttention}\left(XW_K, XW_Q, XW_V\right)$$
$$Z = \mathrm{LayerNorm}(X + Z)$$
$$H = \mathrm{LayerNorm}(\mathrm{ReLU}(ZW_1)W_2 + Z)$$

(multi-head) self-attention, followed by a linear layer and ReLU and some additional residual connections and normalization



Masked Self-Attention

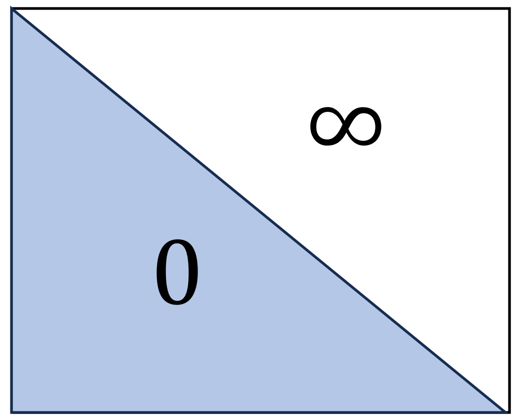
In the matrix form, we are computing weighted average over all inputs



In auto regressive models, usually it is good to maintain casual relation, and only attend to some of the inputs (e.g. skip the red dashed edge on the left). We can add "attention mask"

$$\mathrm{MaskedSelfAttention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}} - M\right)V$$

$$M_{ij} = \begin{cases} \infty, j > i \\ 0, j \leq i \end{cases}$$

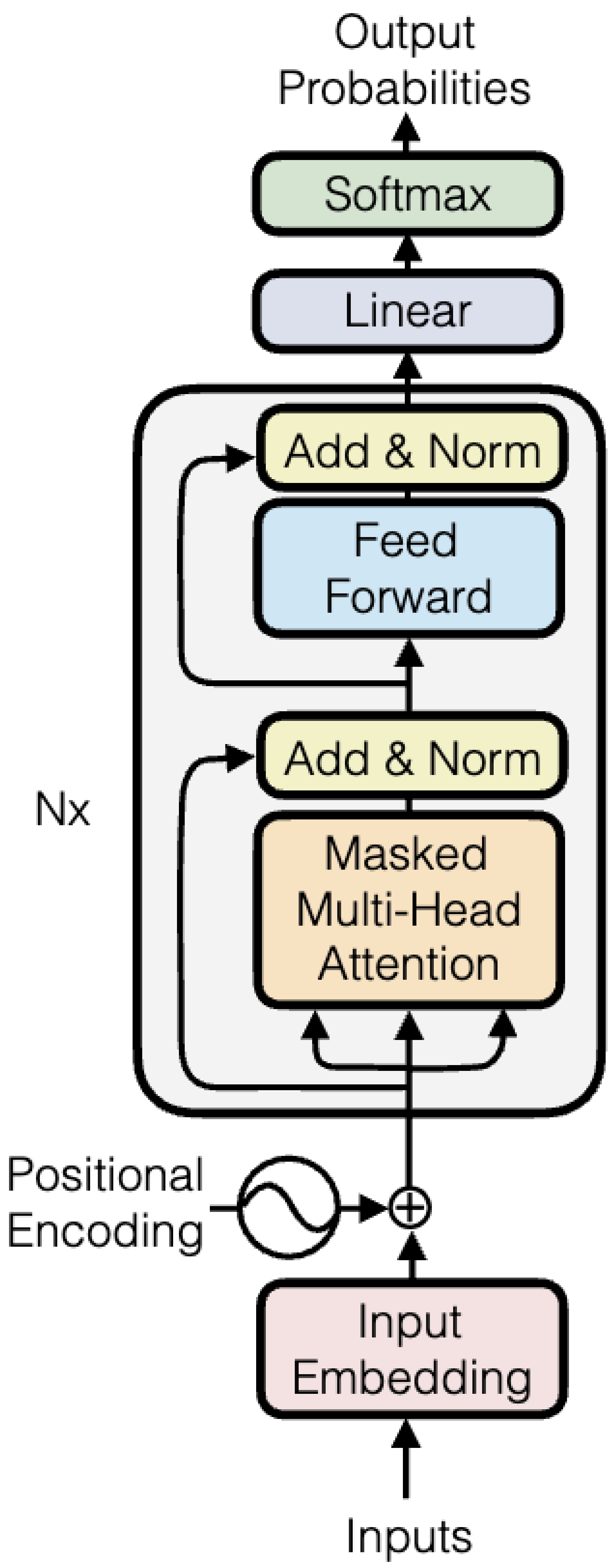


Only attend to previous inputs. Depending on input structure and model, attention mask can change.

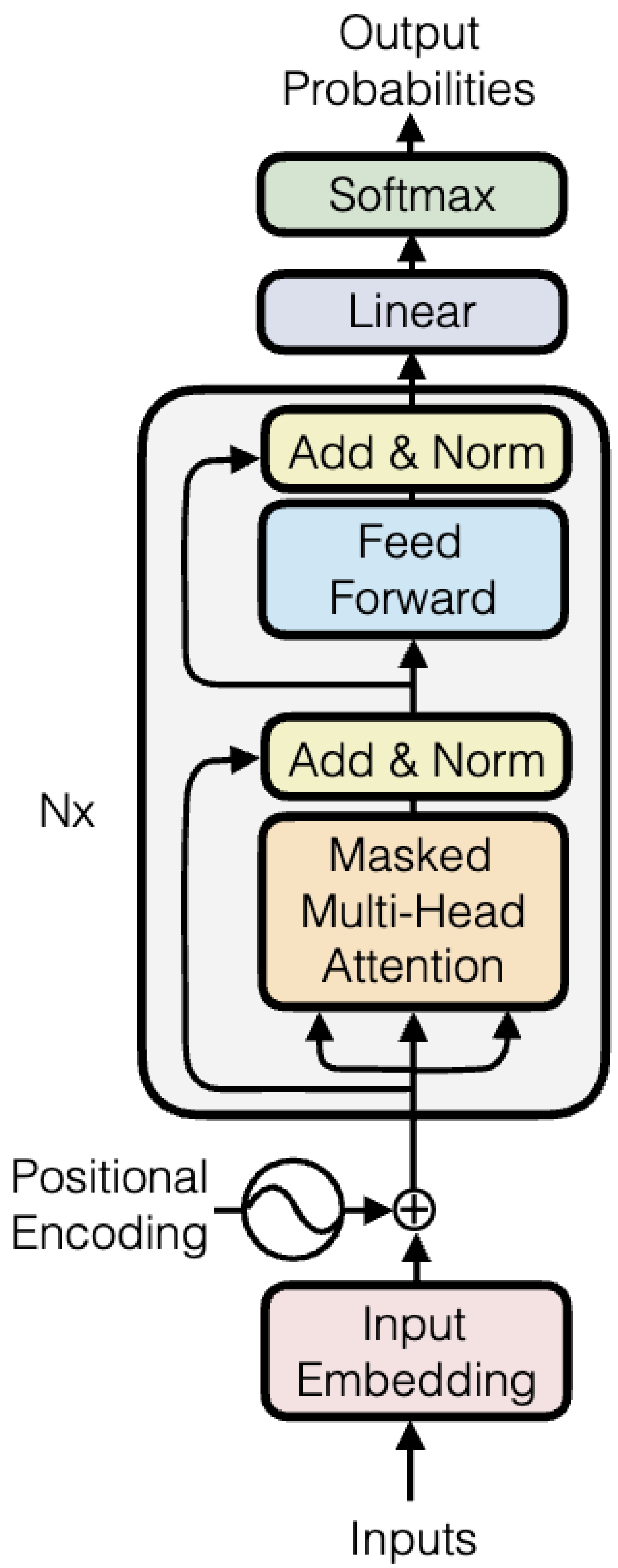
We can also simply skip the computation that are masked out if there is a special implementation to do so

Summary: Transformers

- Transformer decoders
  - Many of them
  - Really just: attentions + layernorm + MLPs + … dual
- Word embeddings
- Position embeddings
  - Rotary embedding
- Loss function: cross entropy loss over a seque

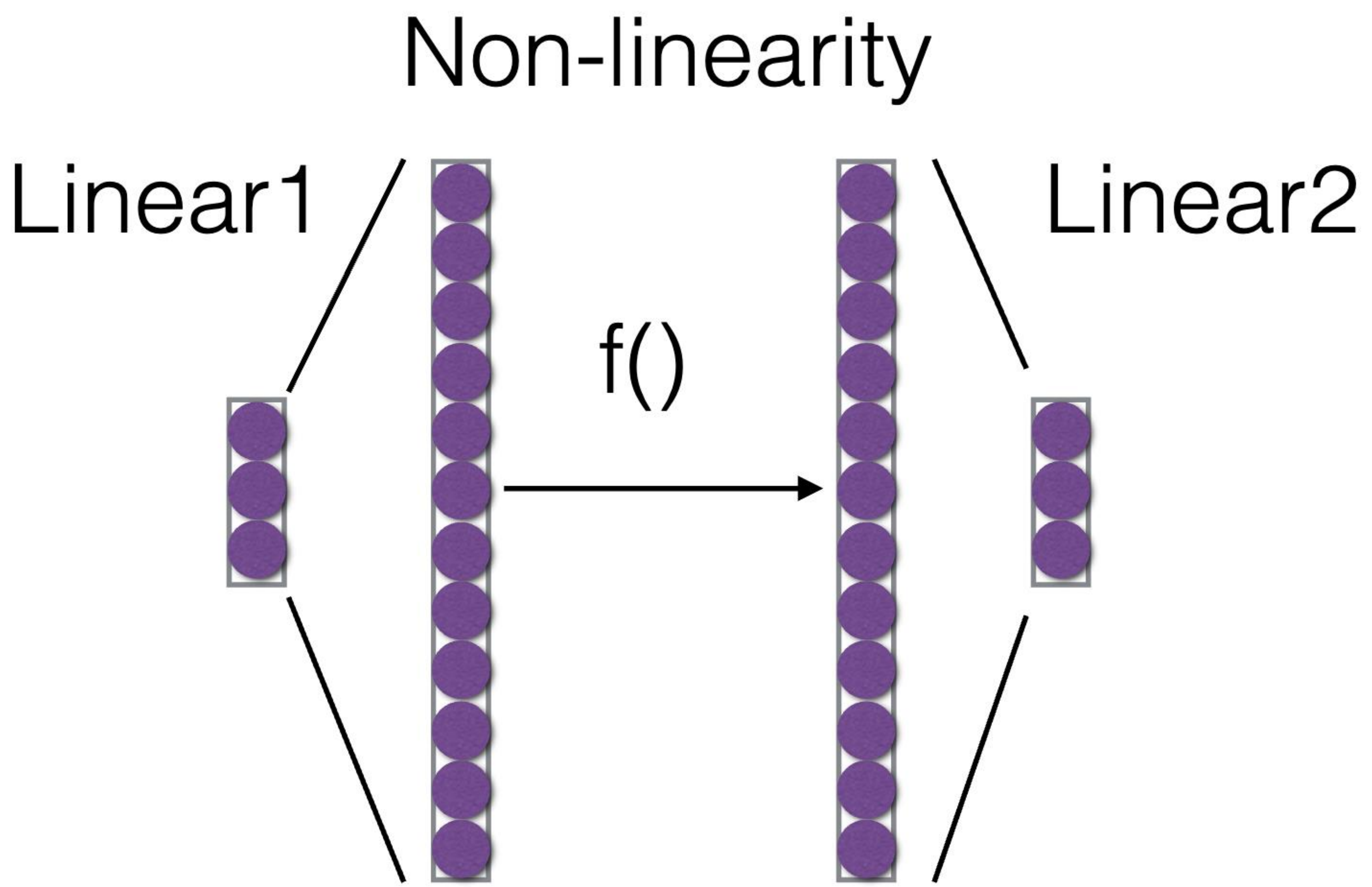


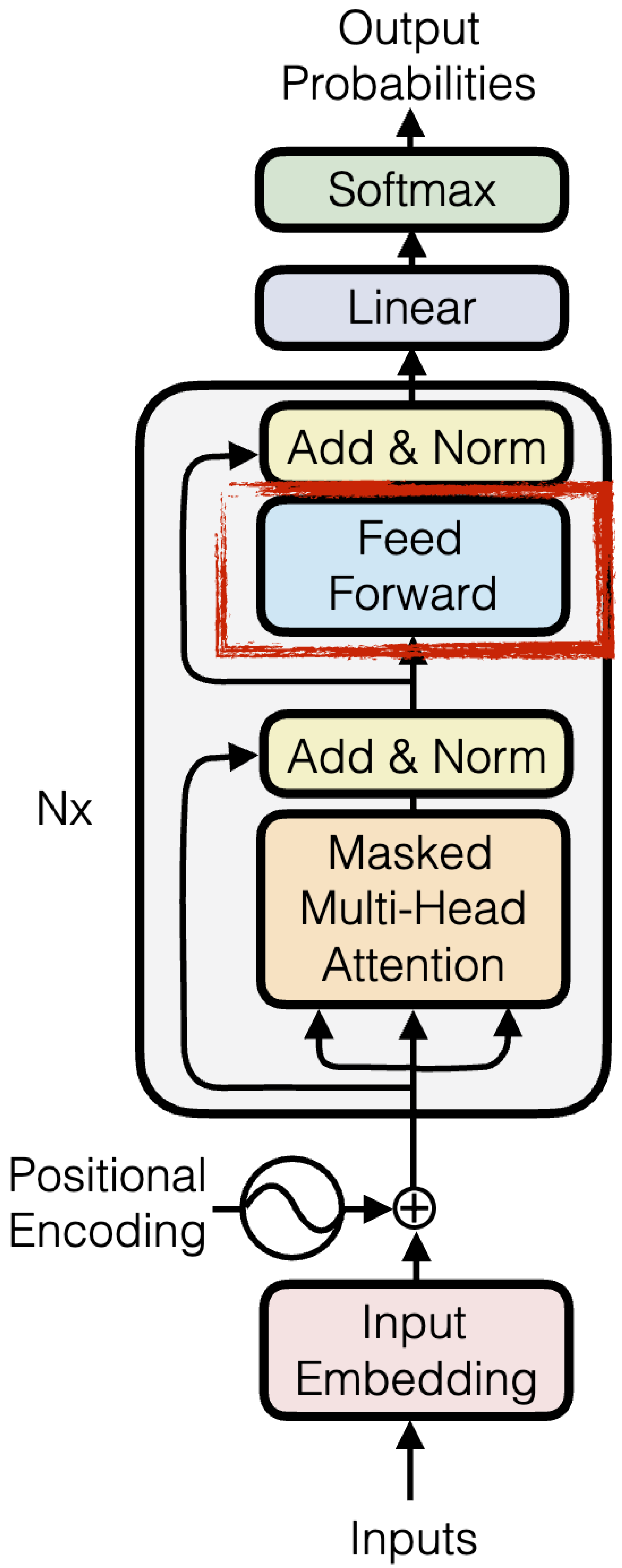
Transformers



Feedforward Layers

$$\mathrm{FFN}(x; W_1, \mathbf{b}_1, W_2, \mathbf{b}_2) = f(\mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2$$

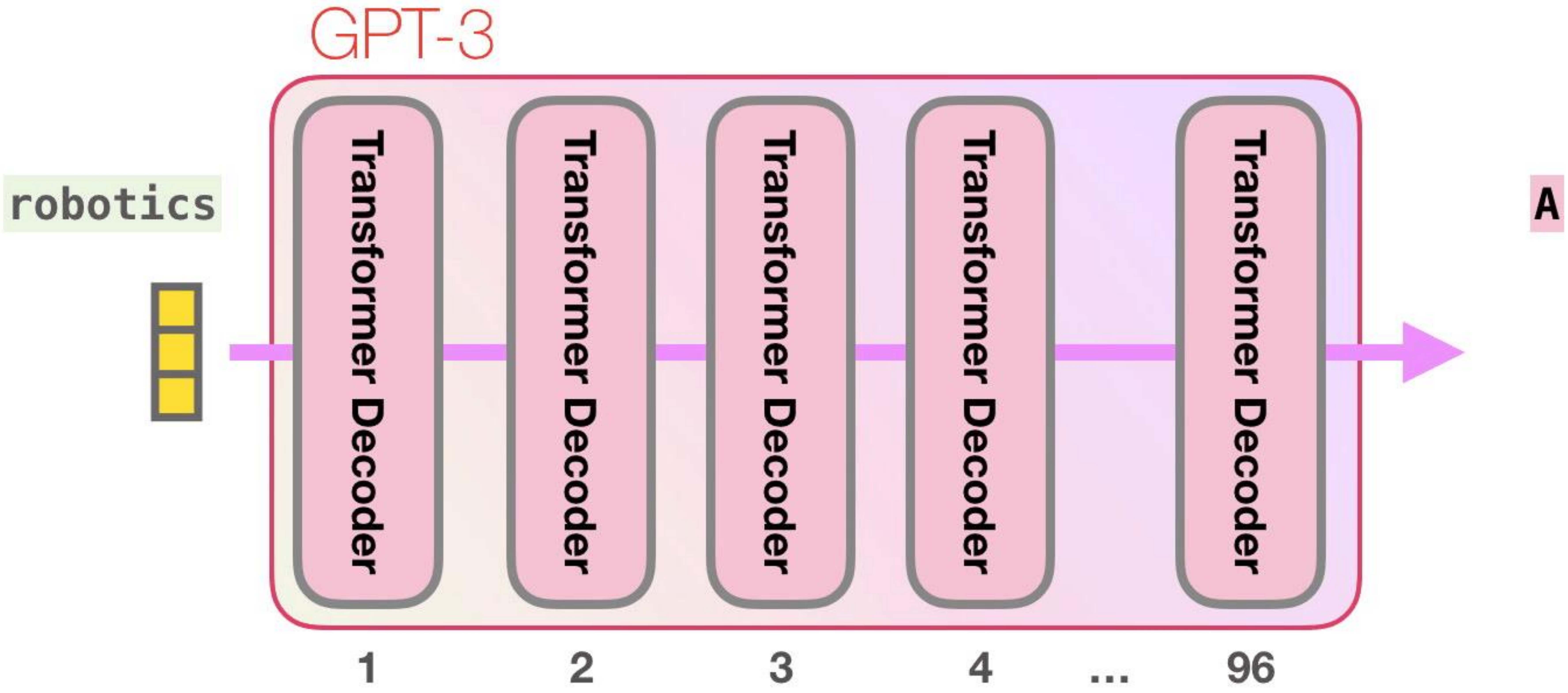




Computing Components in LLMs?

- Transformer decoders (many of them)
  - self-attentions (slow)
  - layernorm, residual (fast)
  - MLPs (slow)
  - Nonlinear (fast)
- Word embeddings (fast)
- Position embeddings (fast)
- Loss function: cross entropy loss over a sequence of words

LLMs

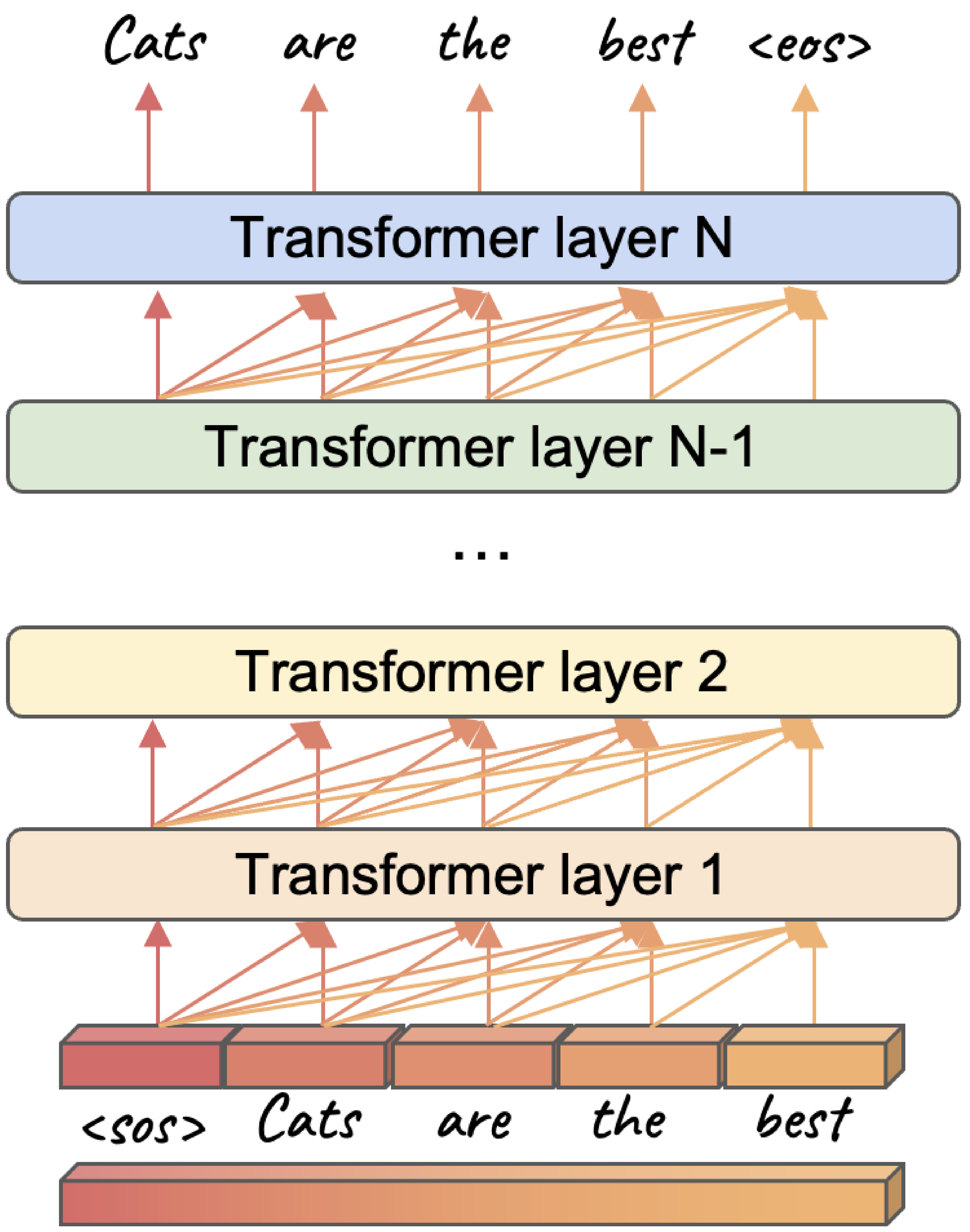

GPT-3
robotics
Transformer Decoder
Transformer Decoder
Transformer Decoder
Transformer Decoder
Transformer Decoder
A
1
2
3
4
...
96

Original Transformer vs. LLM today

	Vaswani et al.	LLaMA
Norm Position	Post	Pre
Norm Type	LayerNorm	RMSNorm
Non-linearity	ReLU	SiLU
Positional Encoding	Sinusoidal	RoPE

Training LLMs

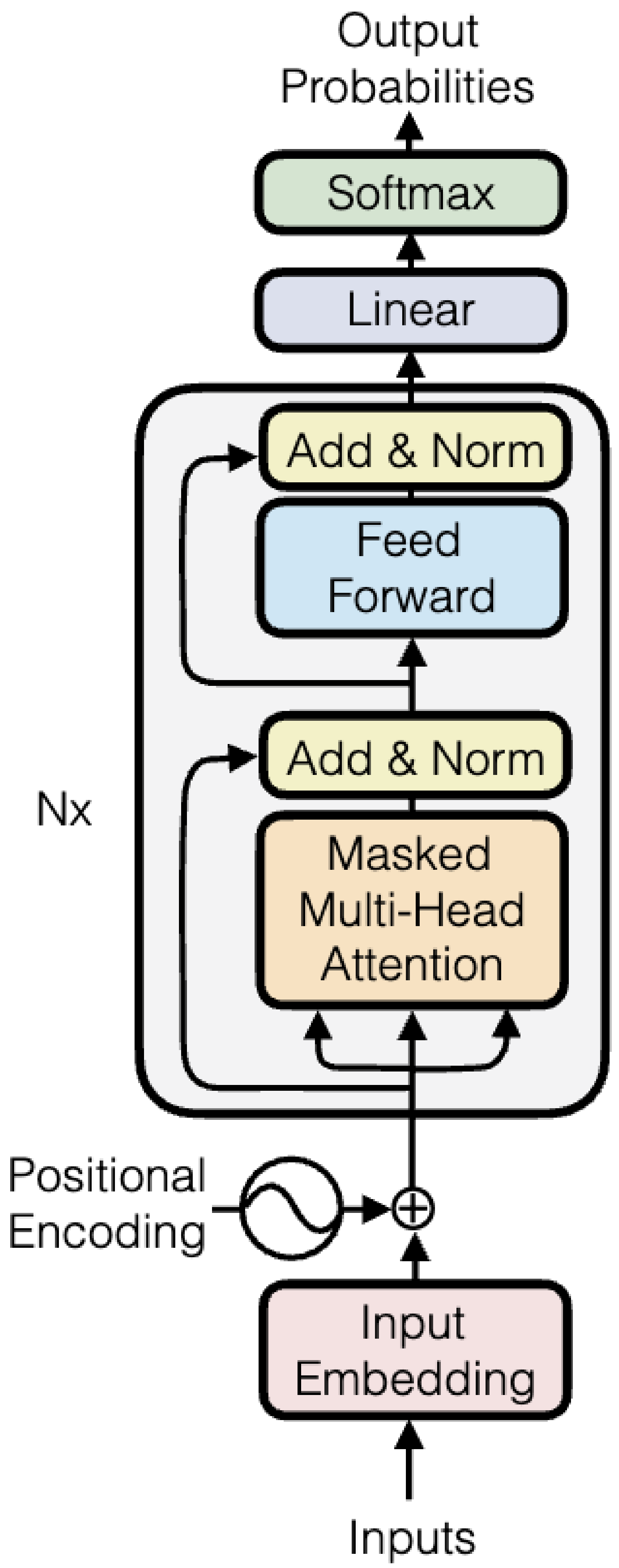
- Sequences are **known a priori**
- For each position, look at [1, 2, …, t-1] words to predict word t, and calculate the loss at t
- Parallelize the computation across all token positions, and then apply masking



Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

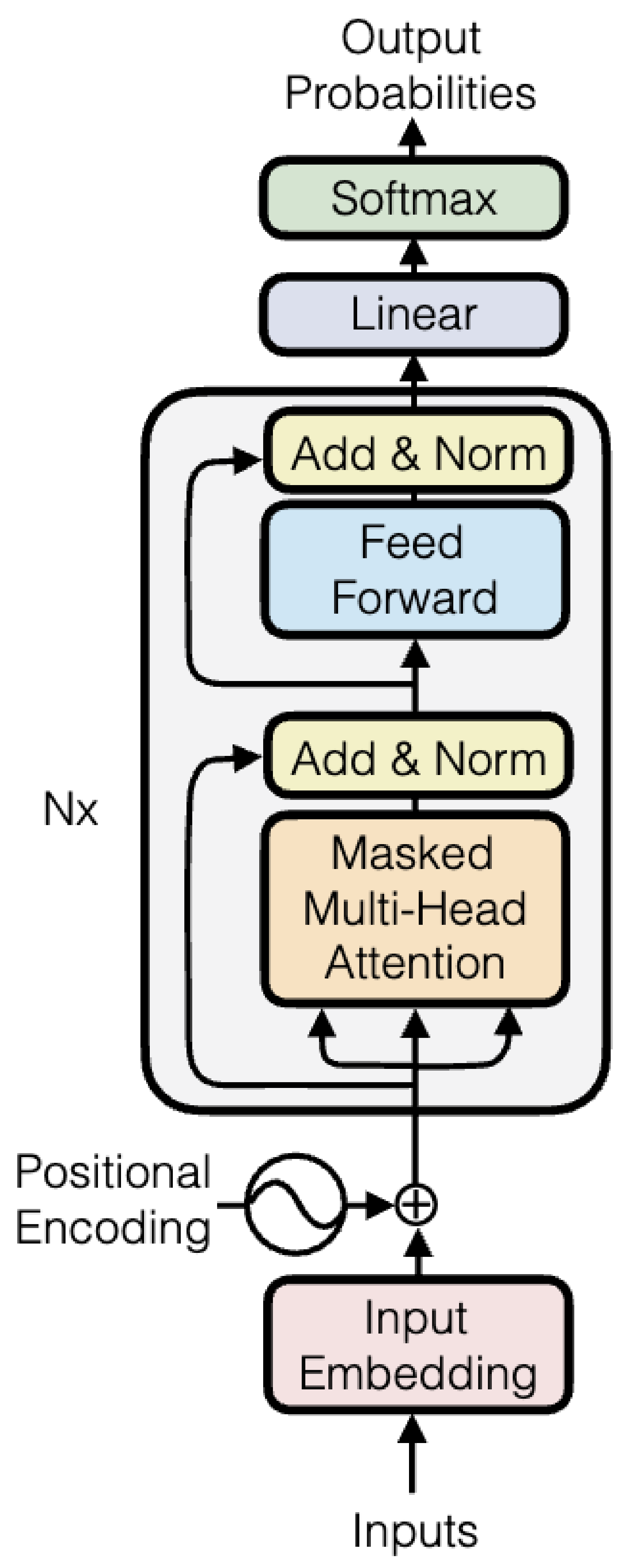
- calculate the number of parameters of an LLM?
    - memory, communication
- calculate the flops needed to train an LLM?
    - compute
- calculate the memory needed to train an LLM?
    - memory, communication

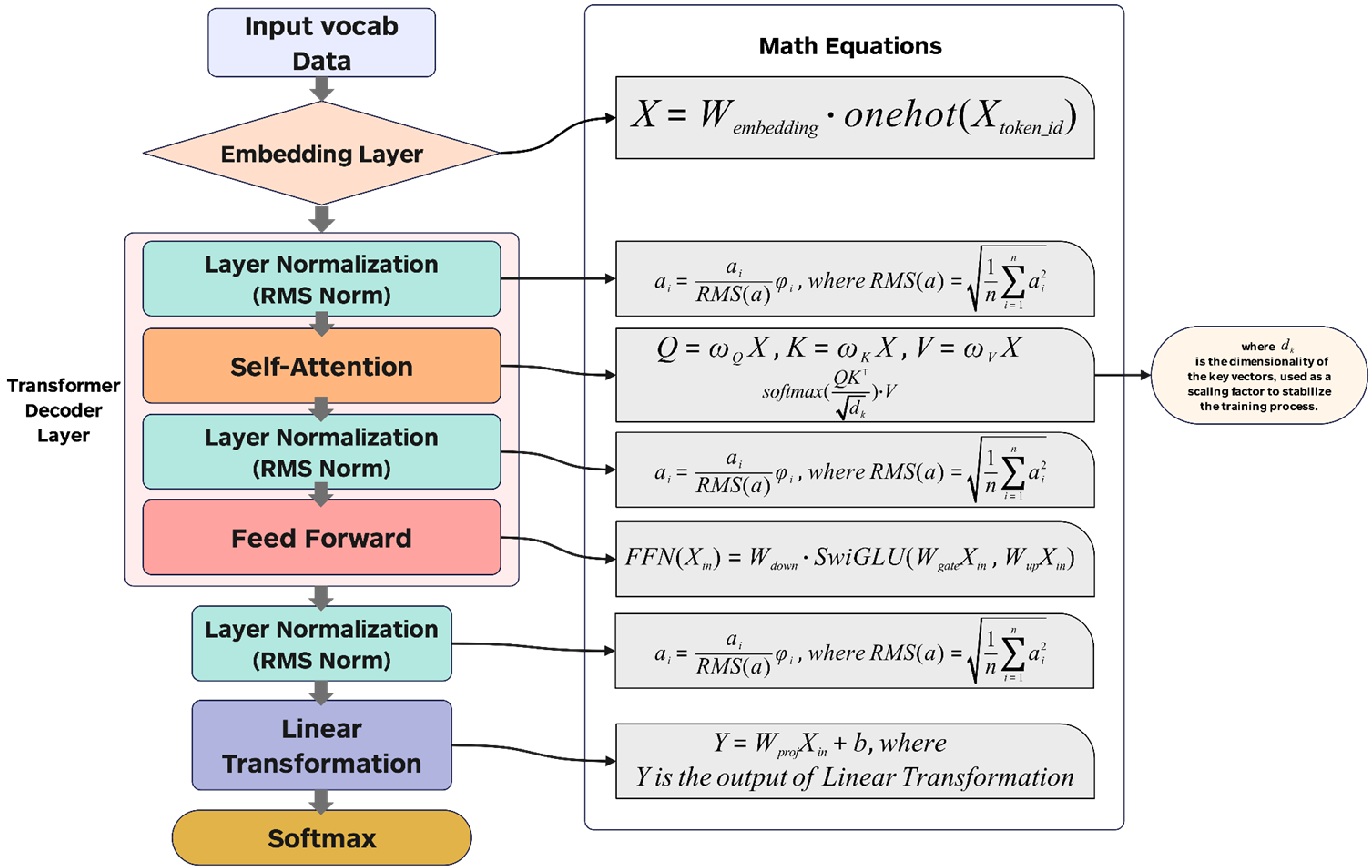


Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?





Input vocab Data
Embedding Layer
Transformer Decoder Layer
Layer Normalization (RMS Norm)
Self-Attention
Layer Normalization (RMS Norm)
Feed Forward
Layer Normalization (RMS Norm)
Linear Transformation
Softmax
Math Equations
$X = W_{embedding} \cdot onehot(X_{token\_id})$
$a_i = \frac{a_i}{RMS(a)}\varphi_i$, where $RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$
$Q = \omega_Q X, K = \omega_K X, V = \omega_V X$
$softmax(\frac{QK^\top}{\sqrt{d_k}}) \cdot V$
where d_k is the dimensionality of the key vectors, used as a scaling factor to stabilize the training process.
$a_i = \frac{a_i}{RMS(a)}\varphi_i$, where $RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$
$FFN(X_{in}) = W_{down} \cdot SwiGLU(W_{gate}X_{in}, W_{up}X_{in})$
$a_i = \frac{a_i}{RMS(a)}\varphi_i$, where $RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$
$Y = W_{proj}X_{in} + b$, where
Y is the output of Linear Transformation

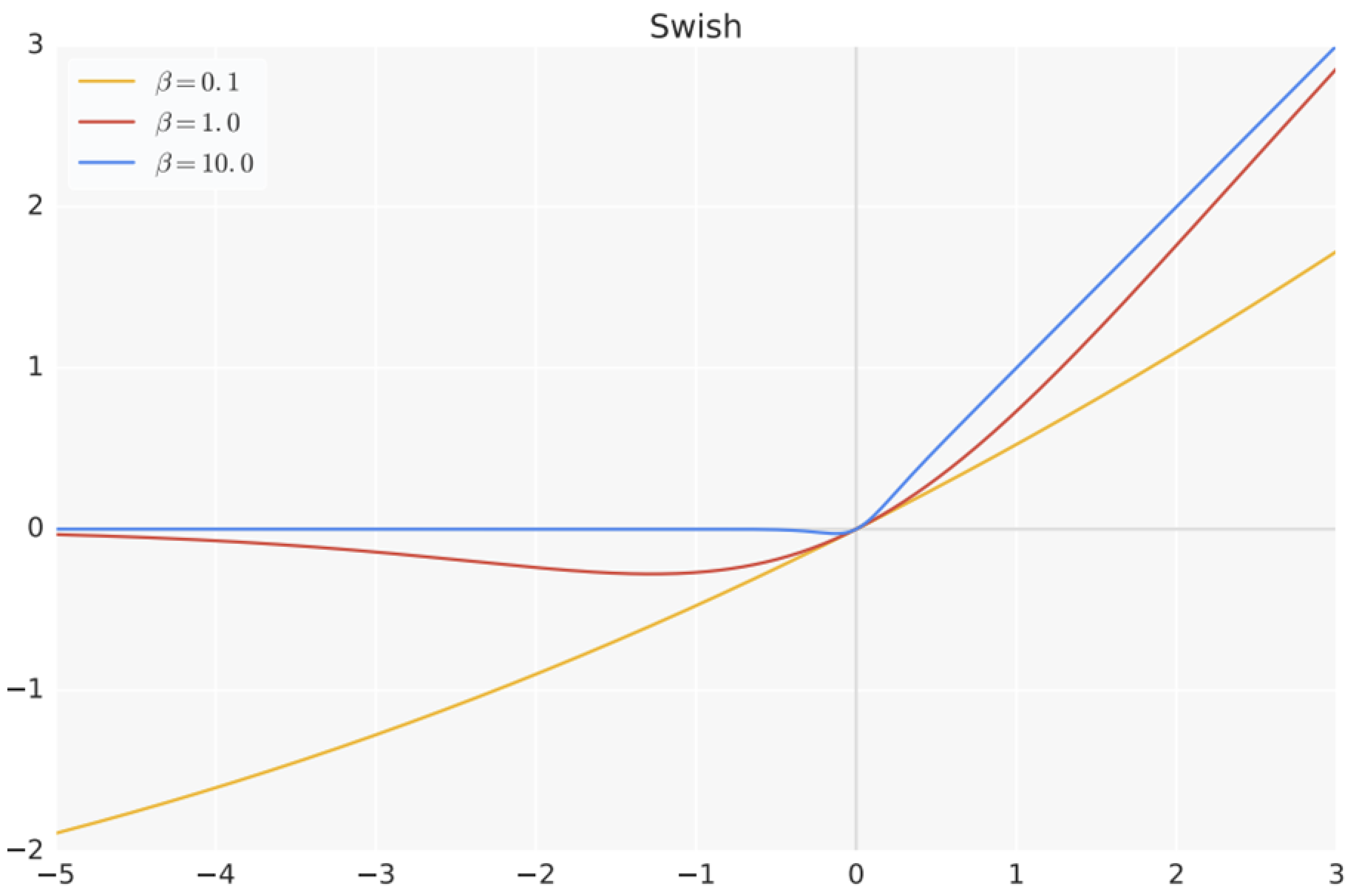
Feed Forward SwiGLU

The general formula for SwiGLU is:

$$\mathrm{SwiGLU}(x) = \mathrm{Swish}(xW_1 + b_1) \odot (xW_2 + b_2)$$

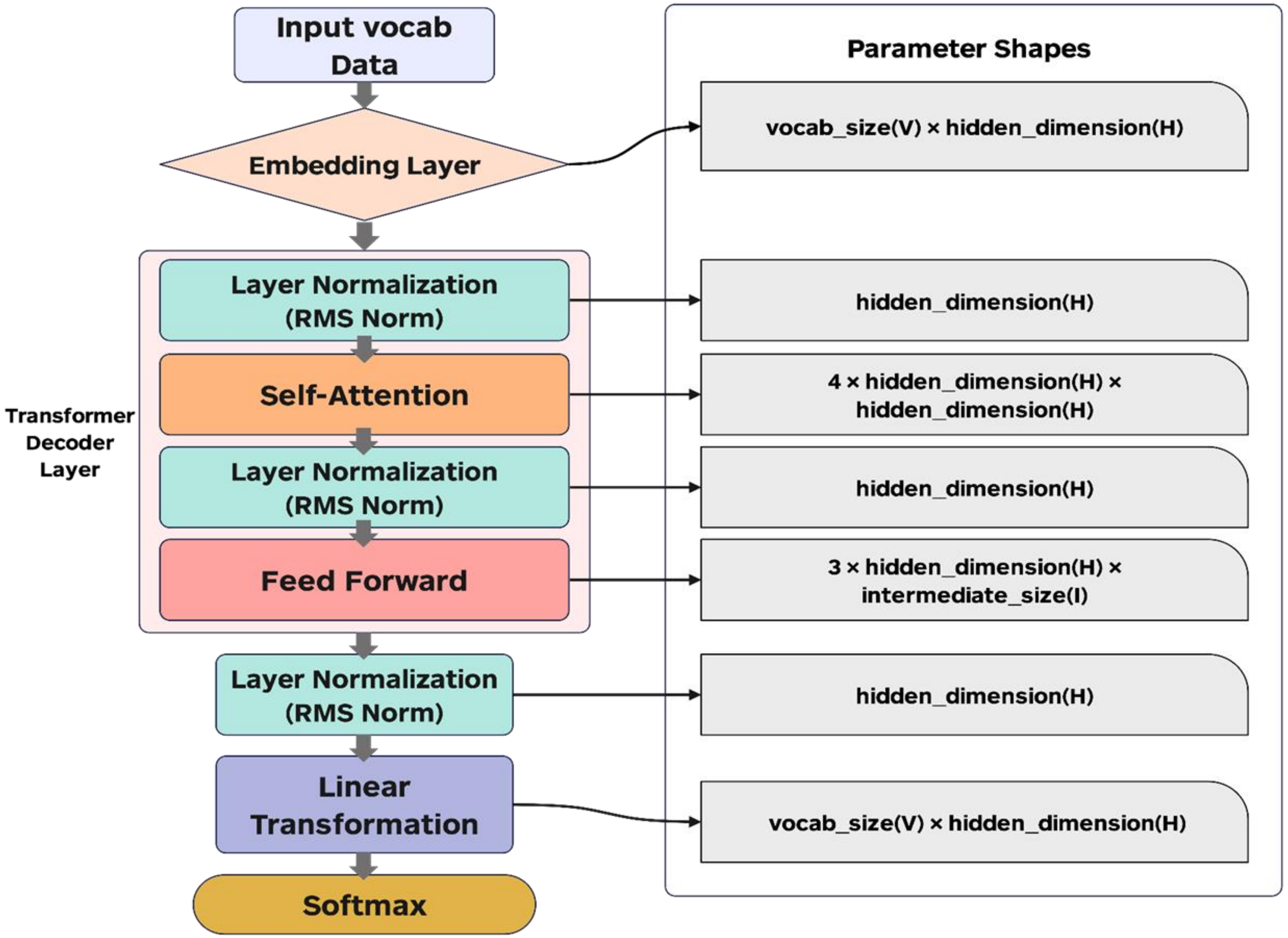
Swish is the activation function applied to one branch, defined as:

$$\mathrm{Swish}(z) = z \cdot \sigma(z)$$



- SwiGLU helps the model capture more complex patterns by selectively gating information
- Swish is smoother than traditional activations ReLU

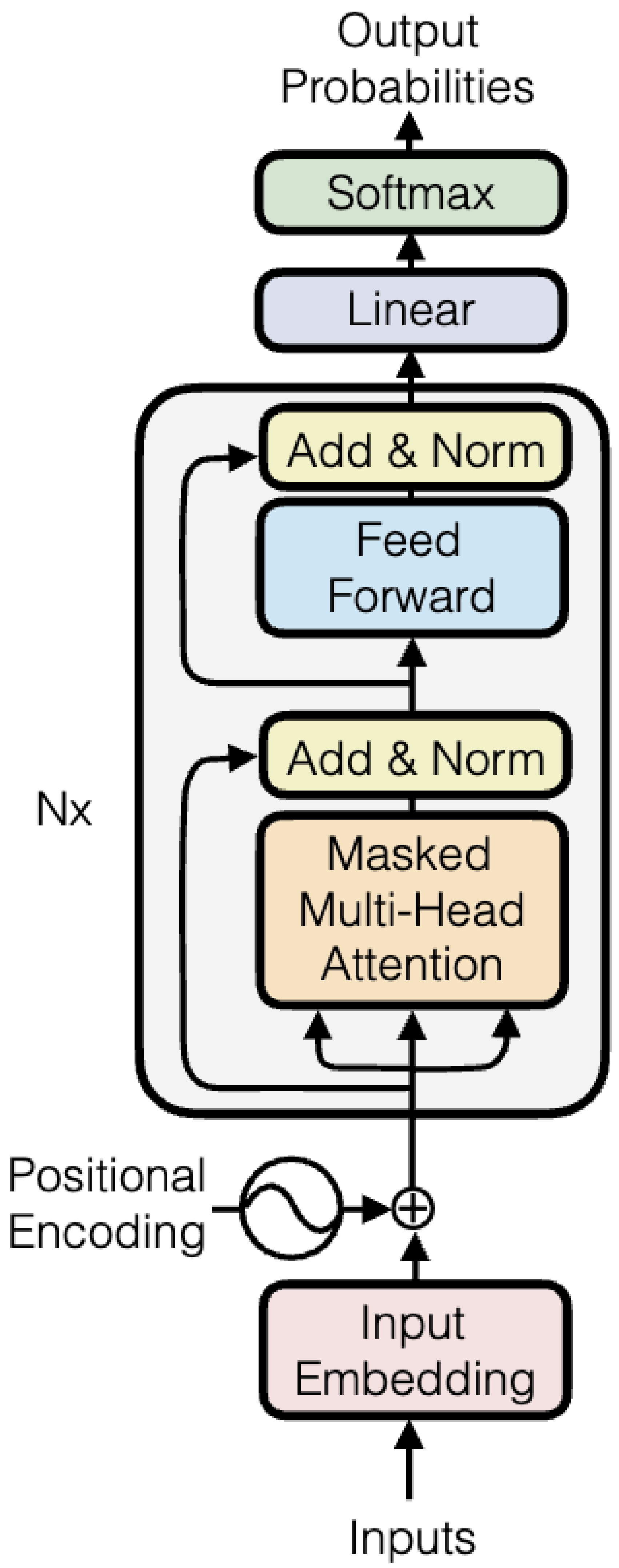
Summary


Input vocab Data
Embedding Layer
Layer Normalization (RMS Norm)
Self-Attention
Layer Normalization (RMS Norm)
Feed Forward
Transformer Decoder Layer
Layer Normalization (RMS Norm)
Linear Transformation
Softmax
Parameter Shapes
vocab_size(V) × hidden_dimension(H)
hidden_dimension(H)
4 × hidden_dimension(H) × hidden_dimension(H)
hidden_dimension(H)
3 × hidden_dimension(H) × intermediate_size(I)
hidden_dimension(H)
vocab_size(V) × hidden_dimension(H)

Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?

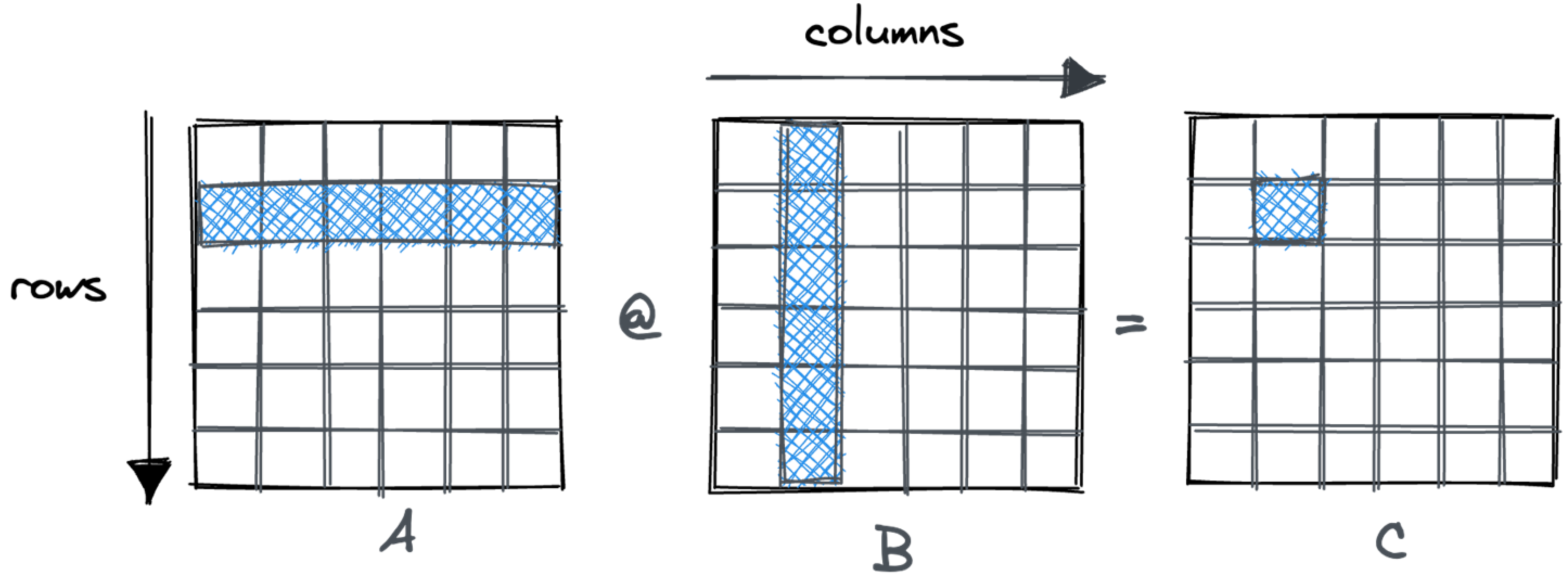


Estimate the Compute: FLOPs

The FLOPs for multiplying two matrices of dimensions m×n and n×h can be calculated as follows:

$$\text{FLOPs} = m \times h \times (2n - 1)$$

So the total number of FLOPs is roughly $\text{FLOPs} \approx 2m \times n \times h$



LLama 2 7B Flops Forward Calculation (Training)

Hyperparameters:

Batch size: b

Sequence length: s

The number of attention heads: n

Hidden state size of one head: d

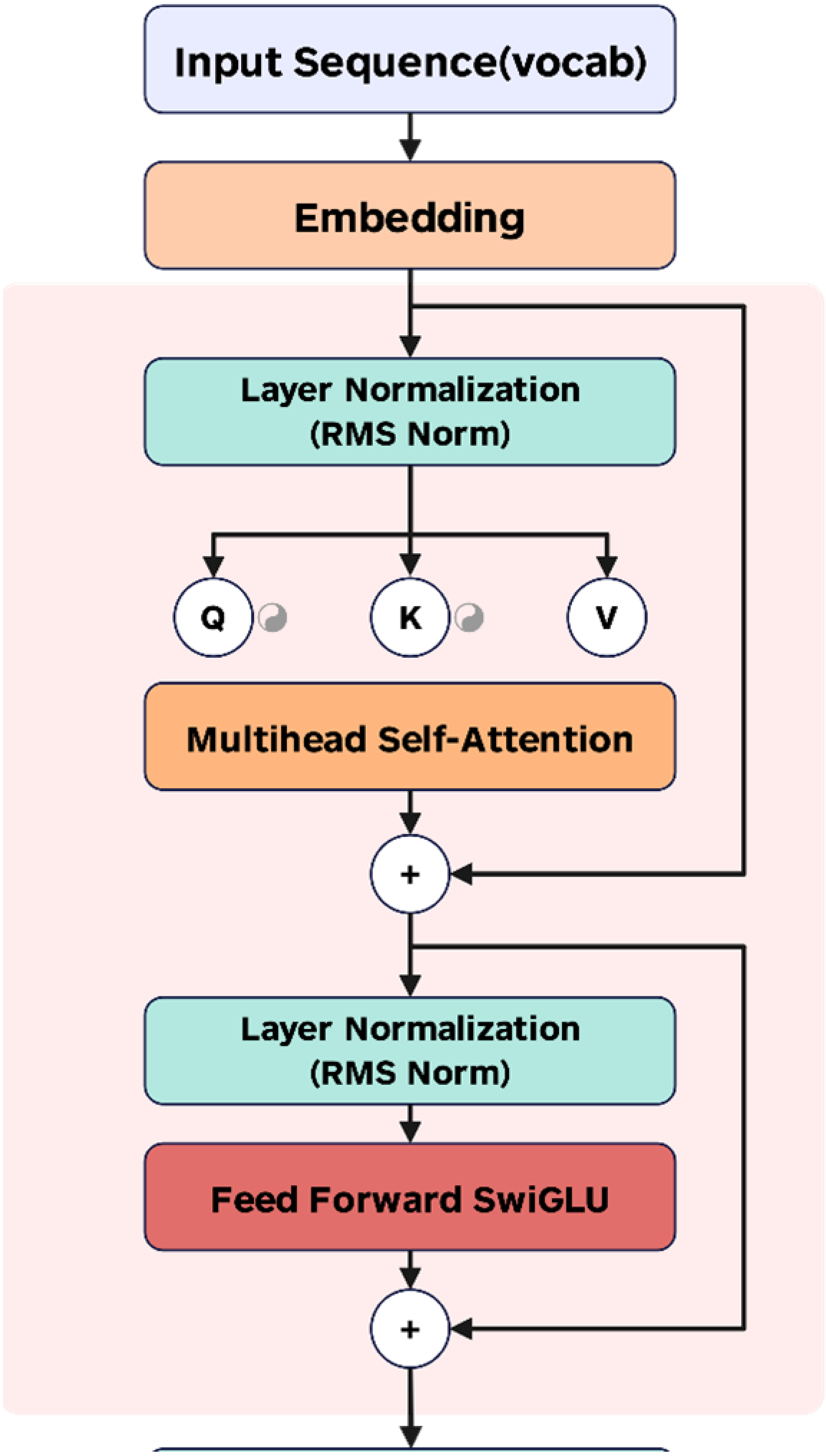
Hidden state size: h (h = n * d)

SwiGLU proj dim: i

Vocab size: v

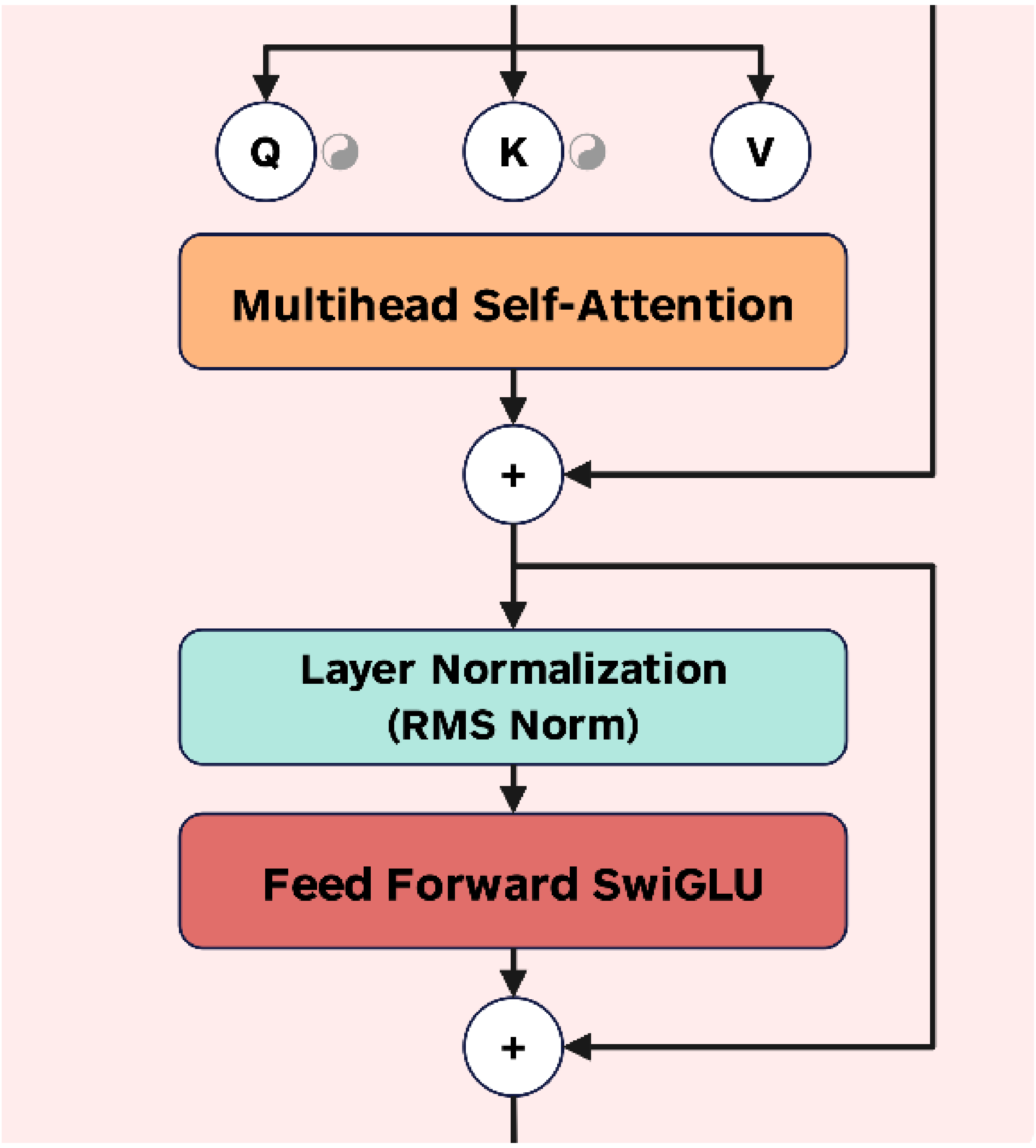
Batch size: b
Sequence length: s
# of attention heads: n
Hidden state dim of one head: d
Hidden state dim: h

	Output Shape:	FLOPs
Input:		
X	(b, s, h)	0
Self Attention:		
XW_Q, XW_K, XW_V	(b, s, h)	$3 * 2bsh^2$
RoPE	(b, n, s, d)	3bsnd
$P = \text{Softmax}(QK^T/\sqrt{d})$	(b, n, s, s)	$2bs^2nd + 3bs^2n$
PV	(b, n, s, d)	$2bs^2nd$
AW_O	(b, s, h)	$2bsh^2$
Residual Connection:	(b, s, h)	bsh



Batch size: b
Sequence length: s
Hidden state dim: h
SwiGLU proj dim: i

Output from Self Attn:	Output Shape:	FLOPs
X	(b, s, h)	0
Feed-Forward SwiGLU:		
XW_{gate}, XW_{up}	(b, s, i)	2 * 2bshi
Swish Activation	(b, s, i)	4bsi
Element-wise *	(b, s, i)	bsi
XW_{down}	(b, s, h)	2bshi
RMS Norm:		
	(b, s, h)	4bsh + 2bs



1. **Calculate Root Mean Square**:
   - $\mathrm{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$
2. **Normalize**:
   - $\mathrm{RMSNorm}(x) = \frac{x}{\mathrm{RMS}(x)+\epsilon} \cdot \gamma$

$$\mathrm{SwiGLU}(x) = \mathrm{Swish}(xW_1 + b_1) \odot (xW_2 + b_2)$$

LLama 2 7B Flops Forward (Training)

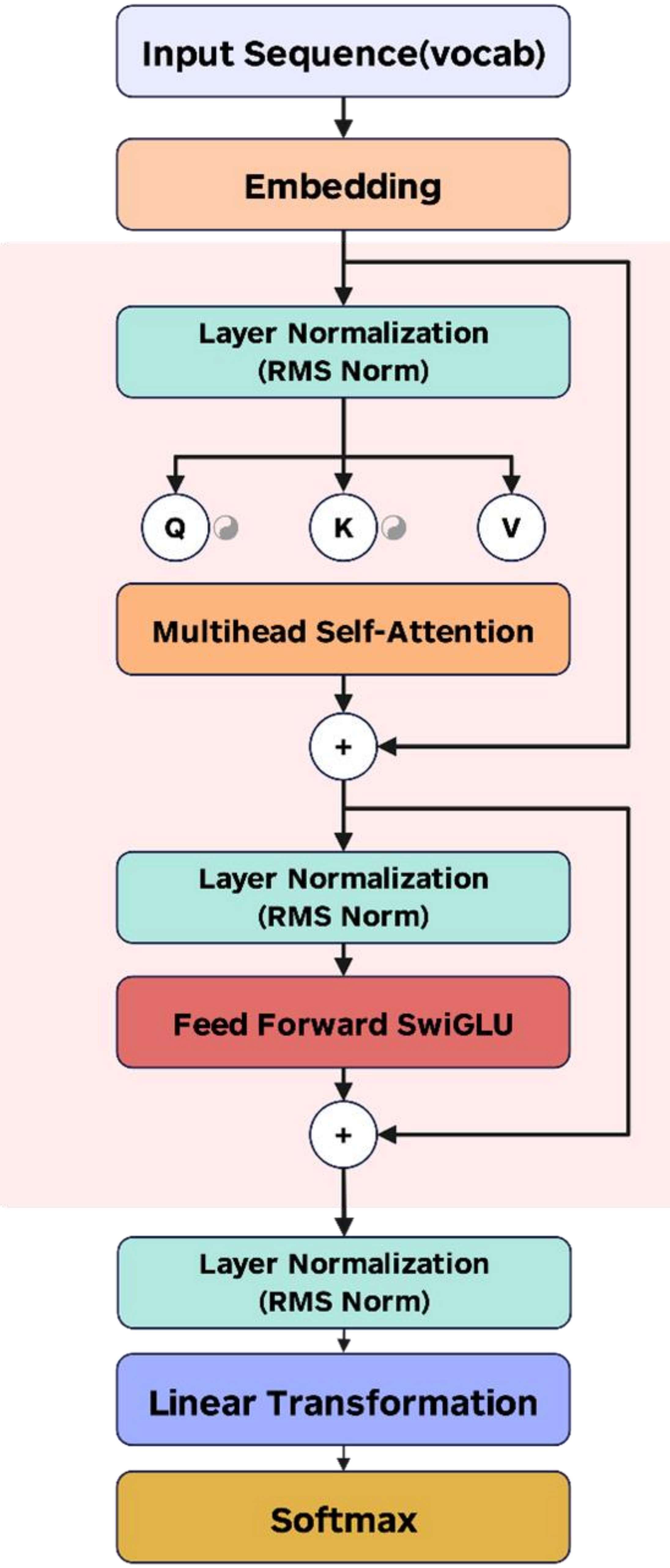
Total Flops ≈ #num_layers * (Attention block + SwiGLU block)

+ Prediction head

= #num_layers * ($6bsh^2 + 4bs^2h + 3bs^2n + 2bsh^2$)

+ #num_layers (6bshi)

+ 2 bshv



LLama 2 7B Flops Forward Calculation (Training)

Hyperparameters:

Batch size: b=1

Sequence length: s=4096

The number of attention heads: n=32

Hidden state size of one head: d=128

Hidden state size: h =4096

SwiGLU proj dim: i=11008

Vocab size: v=32000

The number of layers: N=32

Total Flops ≈ $N * (6bsh^2 + 4bs^2h + 3bs^2n + 2bsh^2)$

$+ N (6bshi)$

$+ 2 bshv$

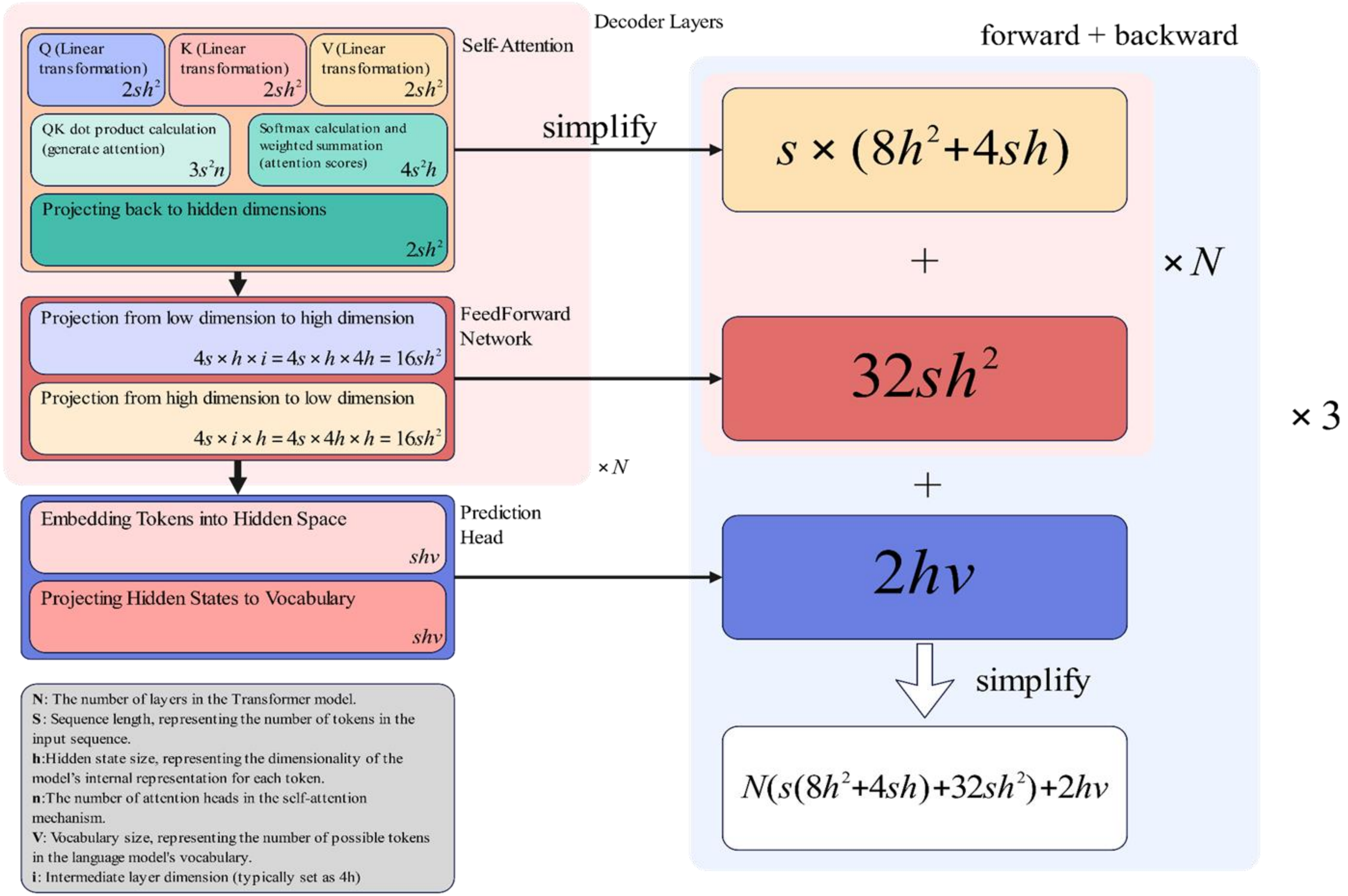
≈ 63 TFLOPs

Flops Distribution

Training Computational Costs Breakdown:

- **Total Training TeraFLOPs:** 192.17 TFLOPs
- **FLOP Distribution by Layer:**
    - **Embedding Layer:** 1.676%
    - **Normalization:** 0.007%
    - **Residual:** 0.003%
    - **Attention:** 41.276%
    - **MLP (Multi-Layer Perceptron):** 55.361%
    - **Linear:** 1.676%

Scaling Up: Where is the Potential Bottleneck?



Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?