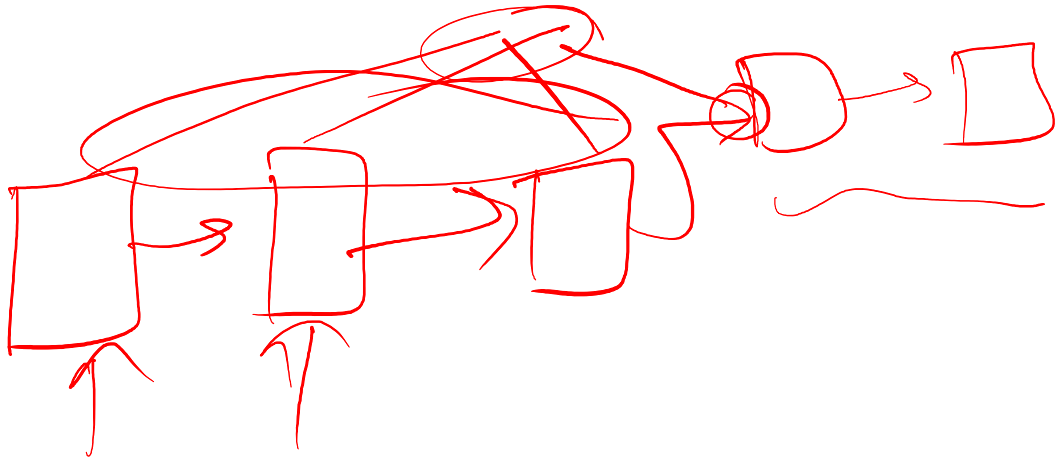
Deep Learning Frameworks

Attention (Self and Cross), Positional Embeddings, Encoder and Decoder in Transformers, Transfer Learning using T5

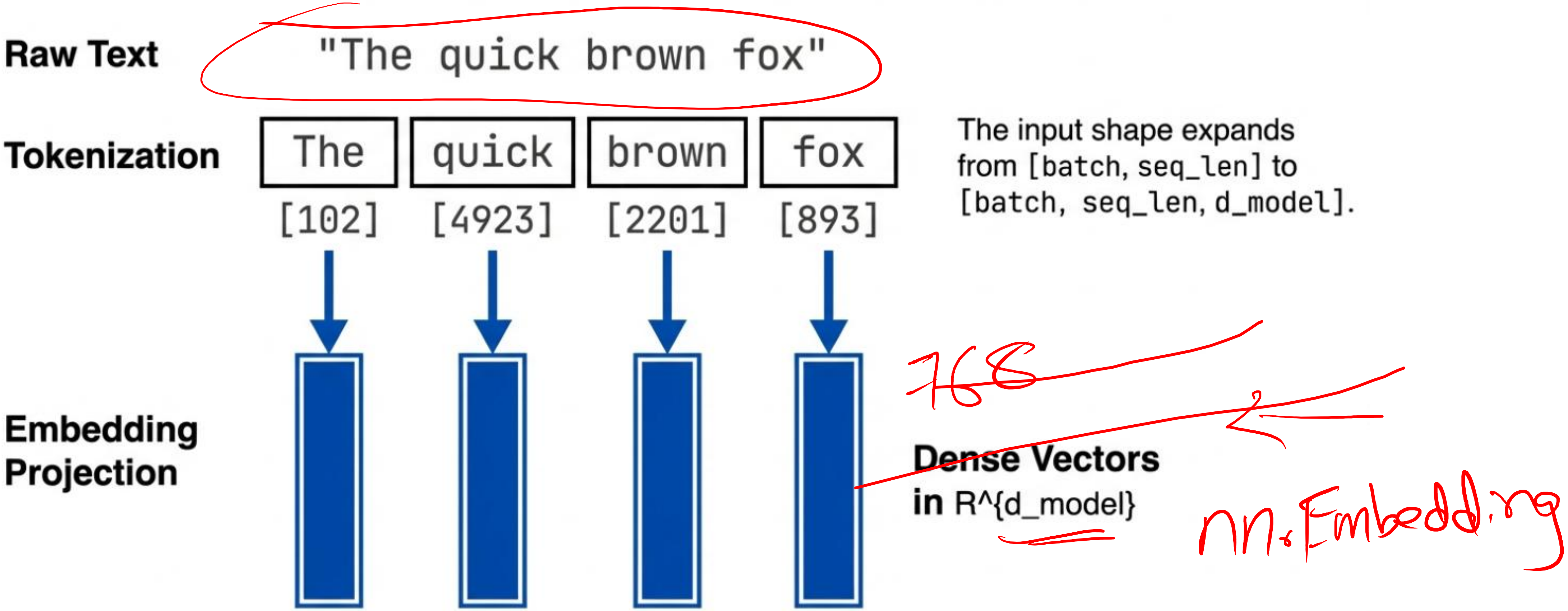
https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

Recap



- **RNN**
  - **Handles sequential data**
  - **Vanishing/exploding gradients** make learning long-range dependencies hard
  - Also: **sequential computation** → slow (can't parallelize across time steps)
- **GRU**
  - Helps with long-term dependencies **compared to vanilla RNN** (gates help preserve/forget info)
  - Still processes tokens **sequentially** (so still slow for long sequences)
  - Important nuance: GRU doesn't "solve" long-term dependency perfectly; it **mitigates** it.
- **Attention**
  - Can handle long-range dependencies **better** because any token can directly attend to any other token

Tokens processed Parallelly



The input shape expands from [batch, seq_len] to [batch, seq_len, d_model].

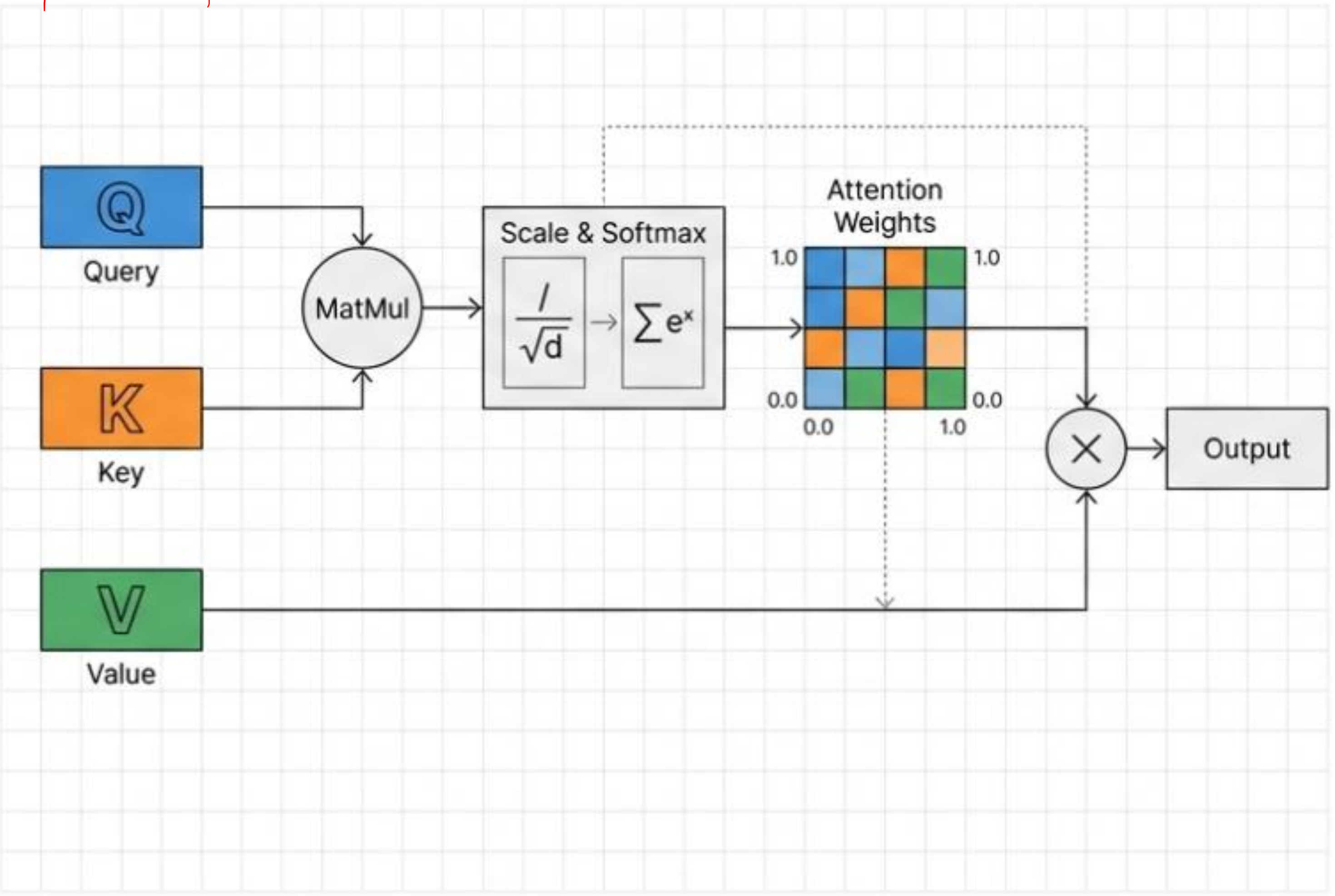
Attention

I went to shop and bought oranges

At its core, Attention is a database lookup. A **Query** searches a set of **Keys** to find compatibility.

The resulting “attention weights” determine how much of each corresponding **Value** is retrieved.

The output is simply a weighted sum of these Values.

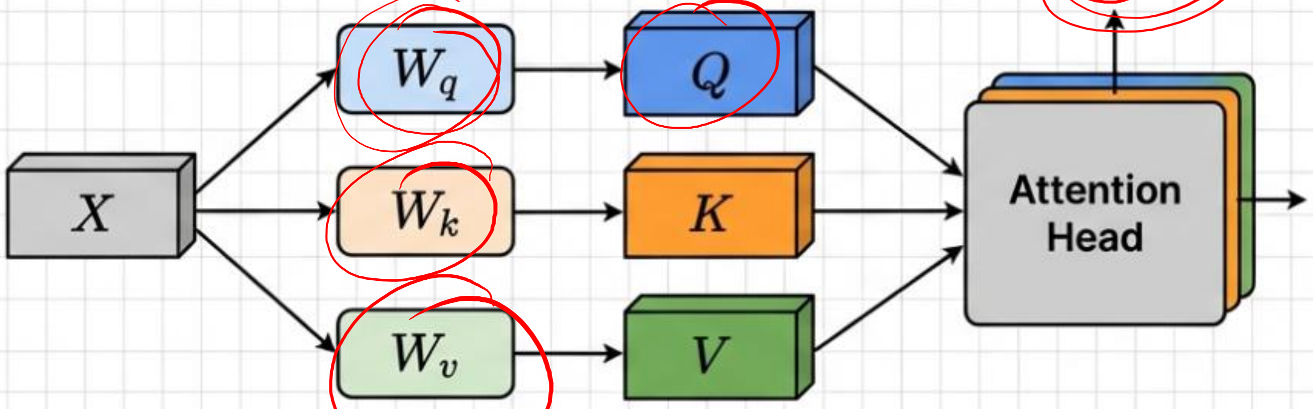


Self Attention

$e^{-\infty}$

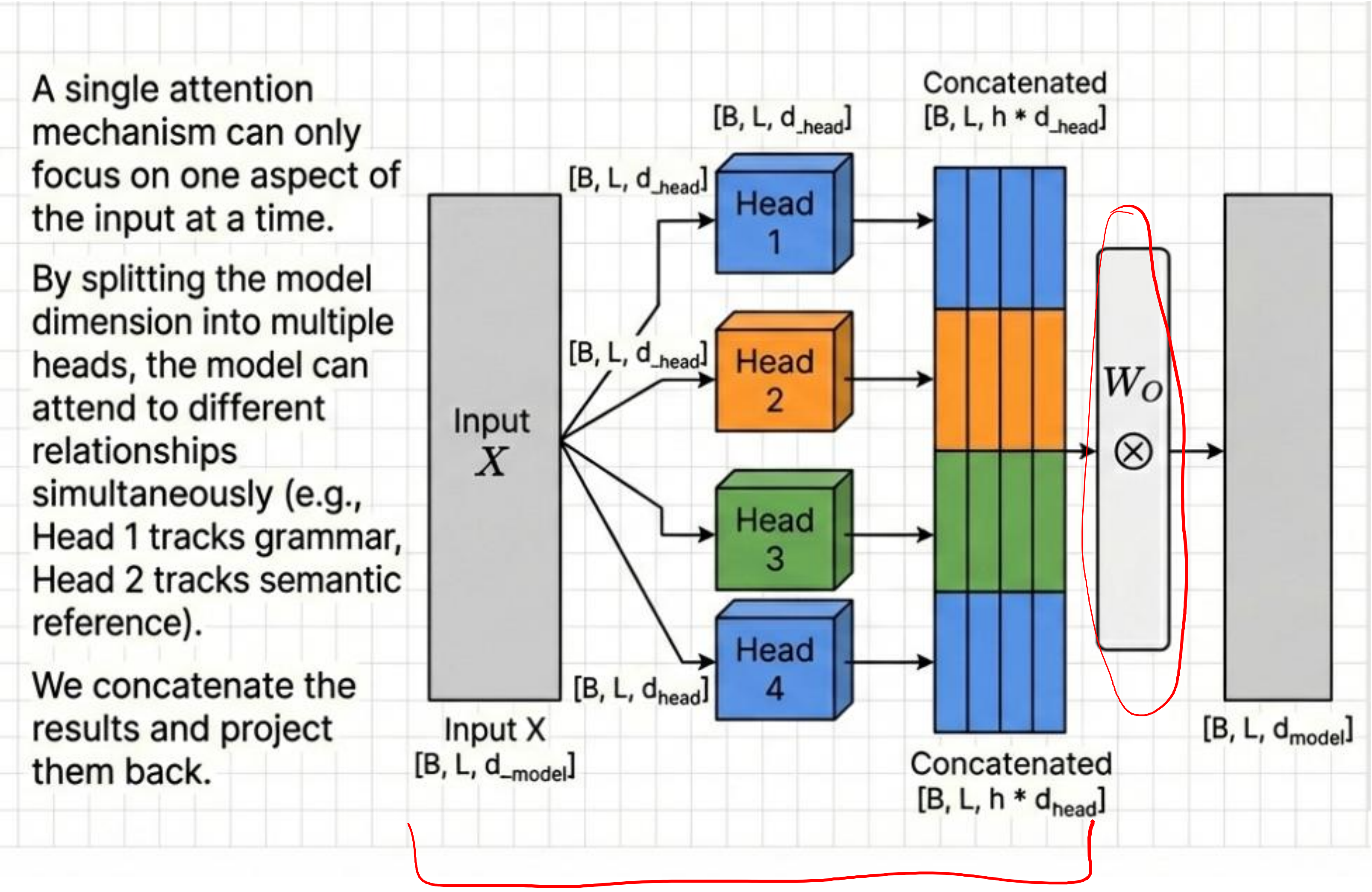
$\frac{e^{x}}{\sum e^{y}}$

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



We project the input X through three linear layers to create distinct semantic representations. We divide by $\sqrt{d_k}$ to prevent dot products from exploding in magnitude, which would push the Softmax into regions with vanishing gradients.

Multi Head Attention



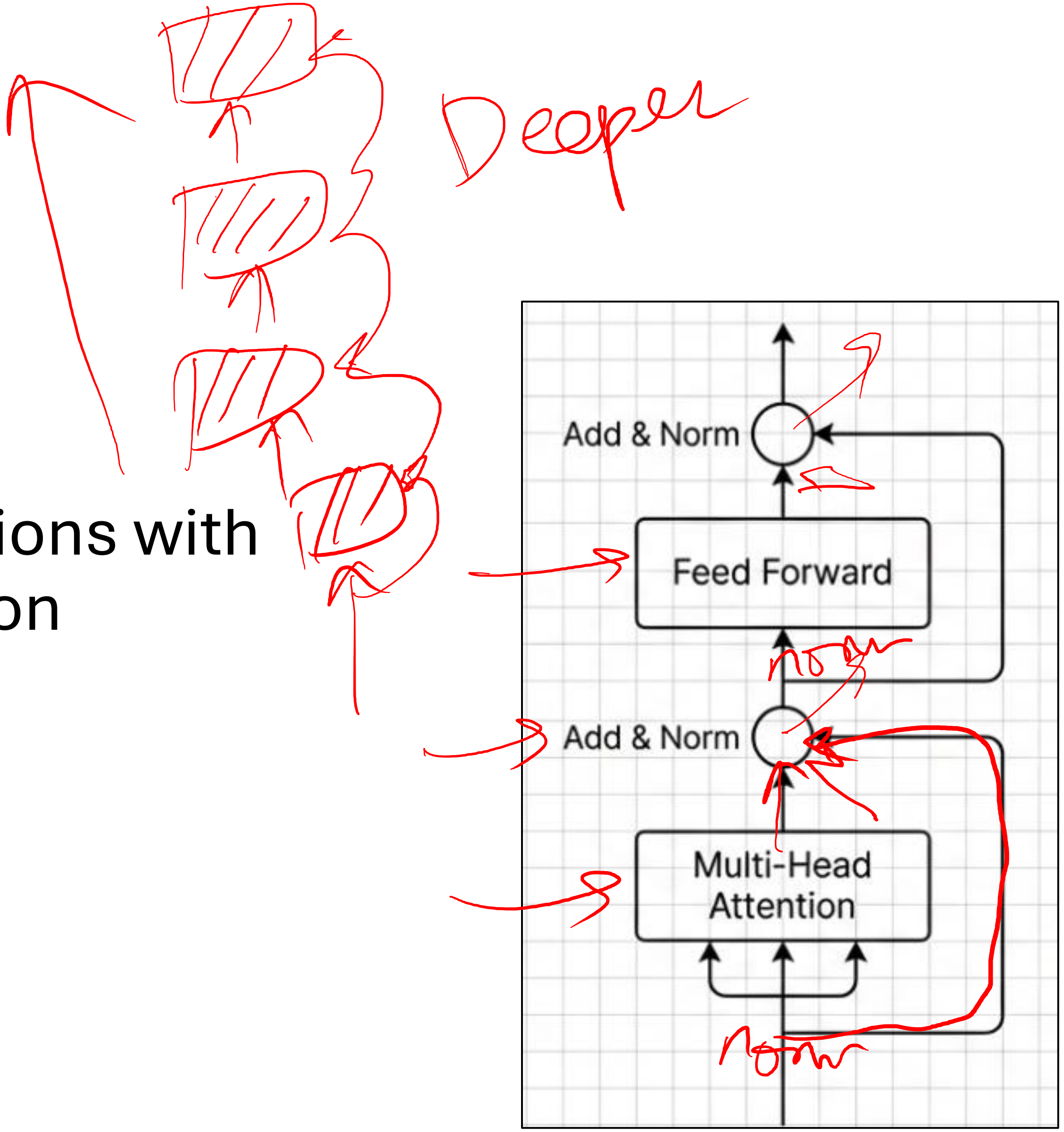
A single attention mechanism can only focus on one aspect of the input at a time.

By splitting the model dimension into multiple heads, the model can attend to different relationships simultaneously (e.g., Head 1 tracks grammar, Head 2 tracks semantic reference).

We concatenate the results and project them back.

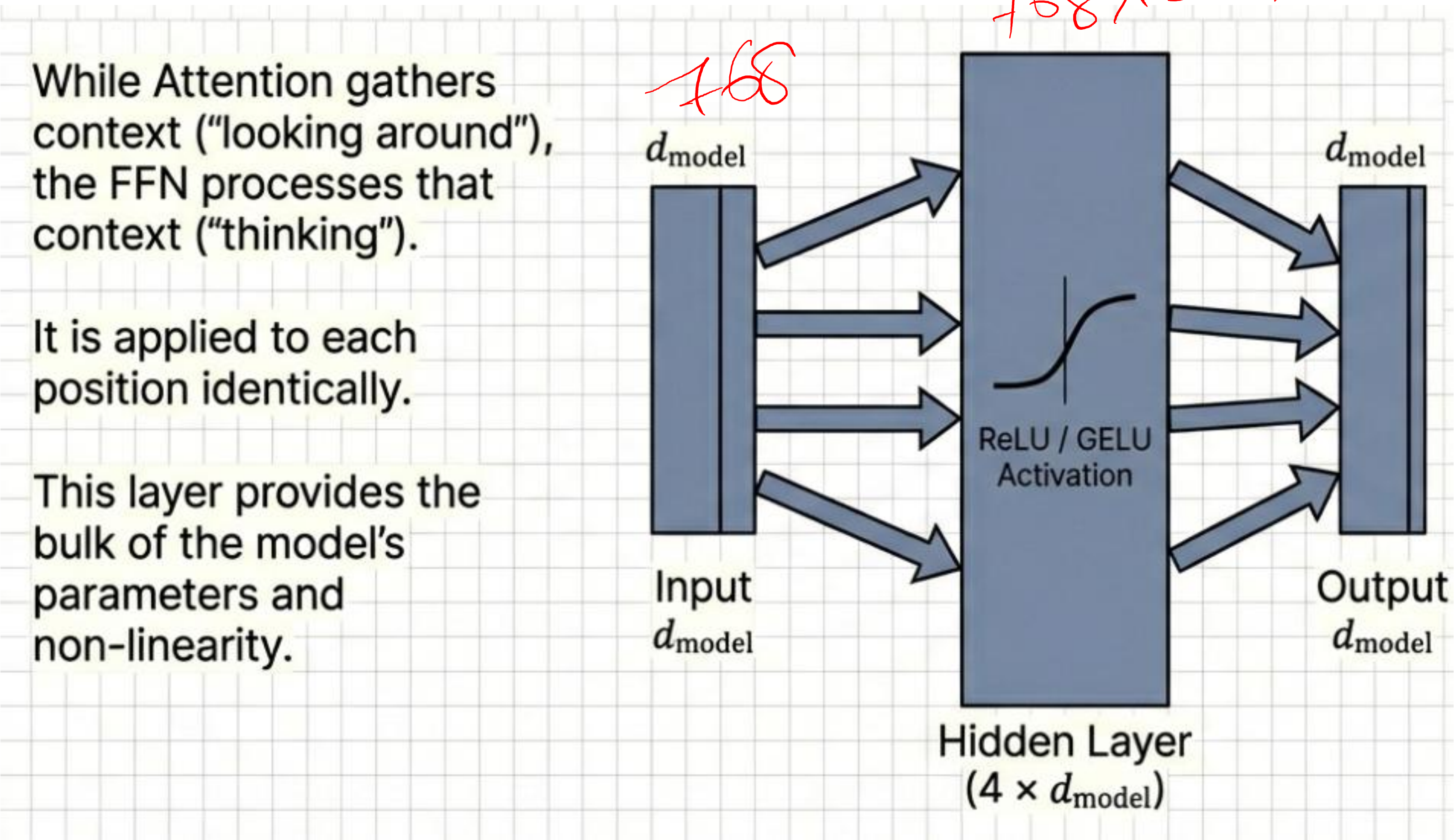
Encoder Block

- Multi Head Attention
- Feed Forward: 2 Layer MLP
- Add and Norm: Residual Connections with Layer Normalization



Feed Forward

nn.Linear (768, 768x4)
↓
ReLU
↓
nn.Linear (768x4, 768)

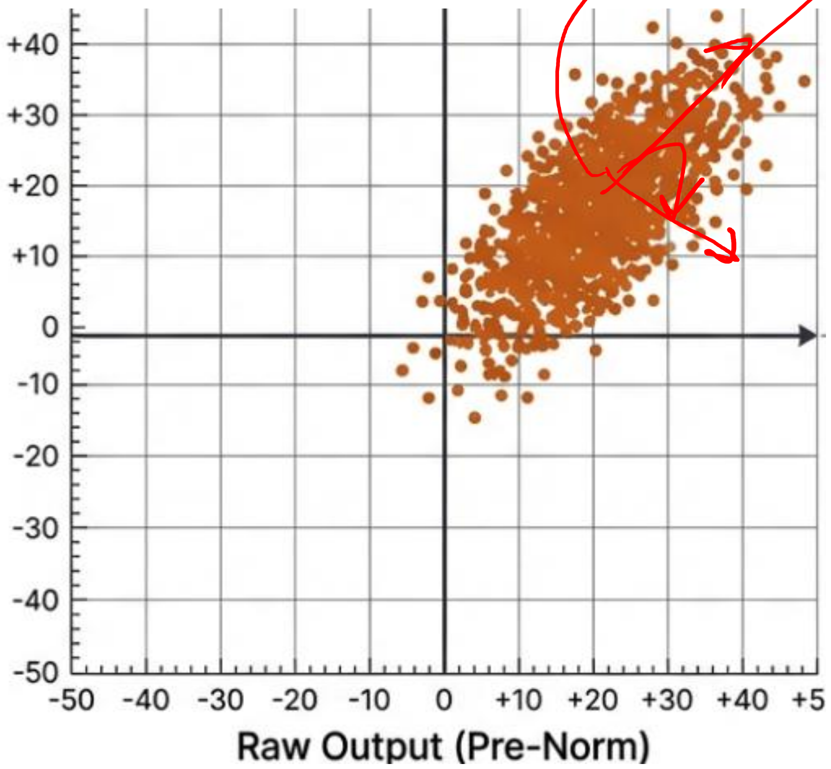


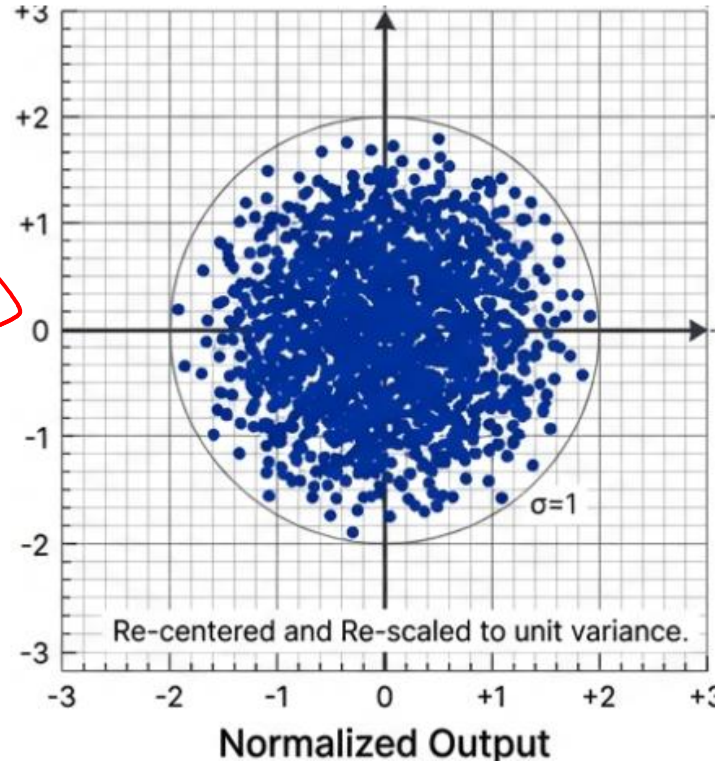
While Attention gathers context (“looking around”), the FFN processes that context (“thinking”).

It is applied to each position identically.

This layer provides the bulk of the model’s parameters and non-linearity.

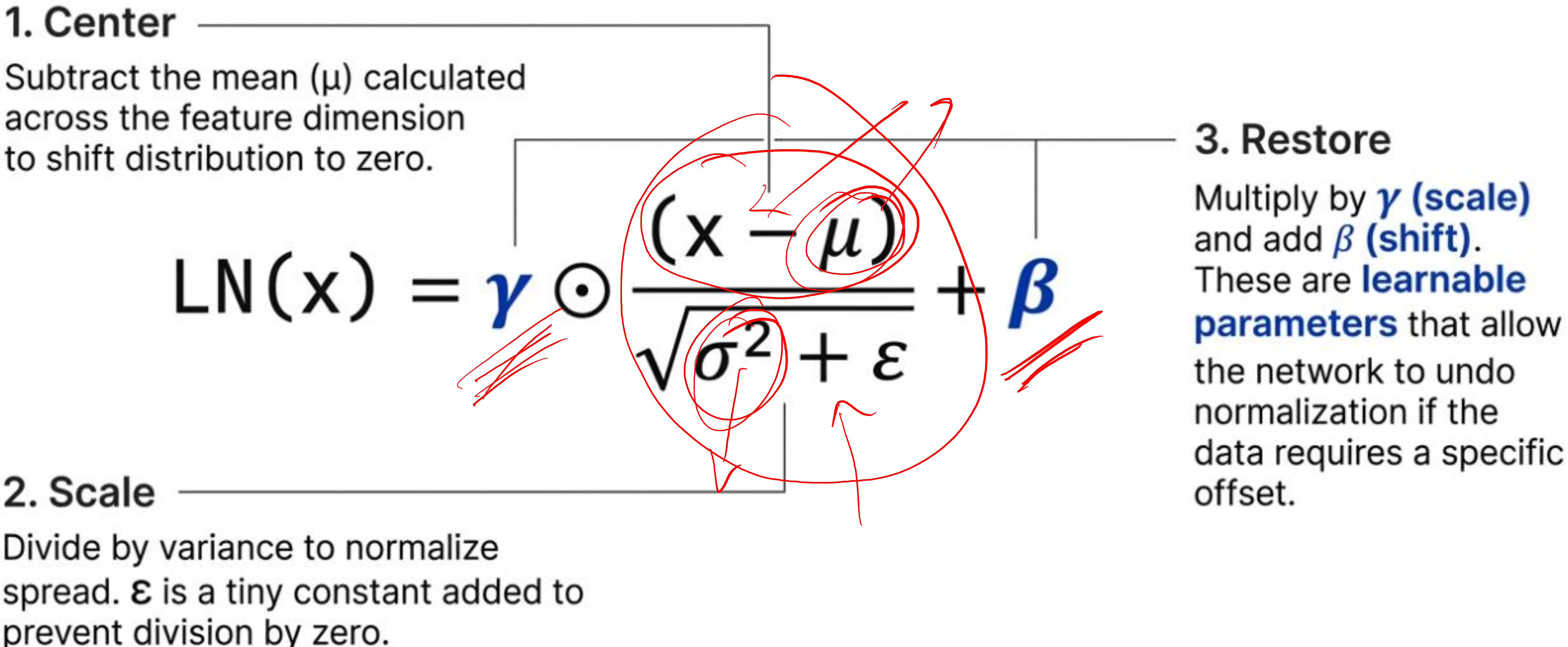
Recap: Normalization - Idea





Re-centered and Re-scaled to unit variance.

Recap: Normalization - Mathematics



1. Center

Subtract the mean (μ) calculated across the feature dimension to shift distribution to zero.

$$\mathrm{LN}(x) = \gamma \odot \frac{(x - \mu)}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

3. Restore

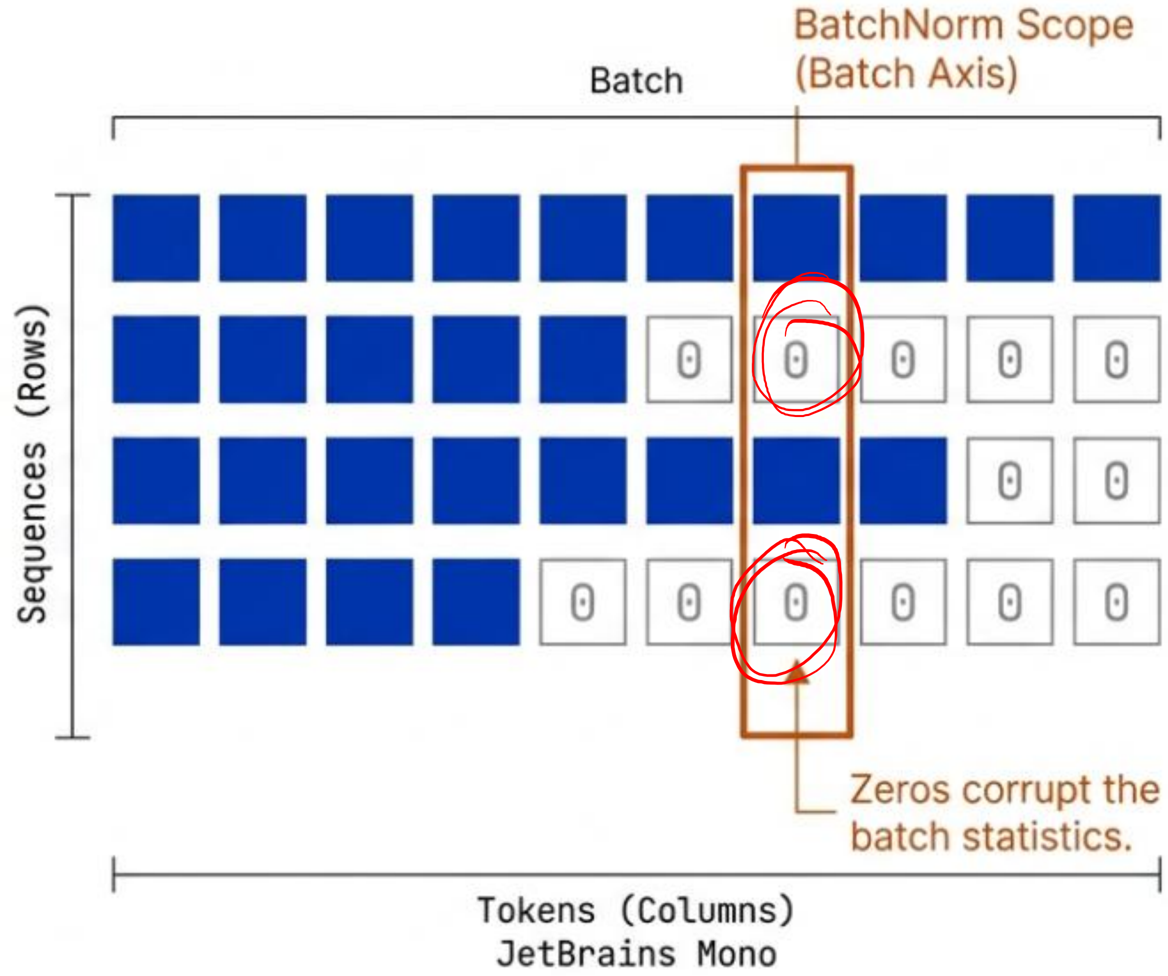
Multiply by γ **(scale)** and add β **(shift)**. These are **learnable parameters** that allow the network to undo normalization if the data requires a specific offset.

2. Scale

Divide by variance to normalize spread. **ε** is a tiny constant added to prevent division by zero.

Issue with Batch Normalization in Sequence

- Batch Size Small
- Variable Sequence Length



Batch

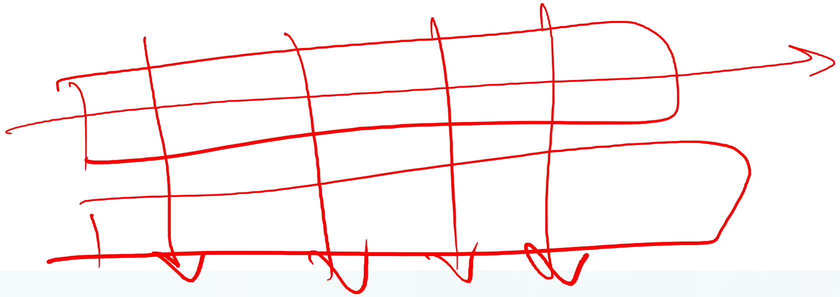
BatchNorm Scope
(Batch Axis)

Sequences (Rows)

Zeros corrupt the
batch statistics.

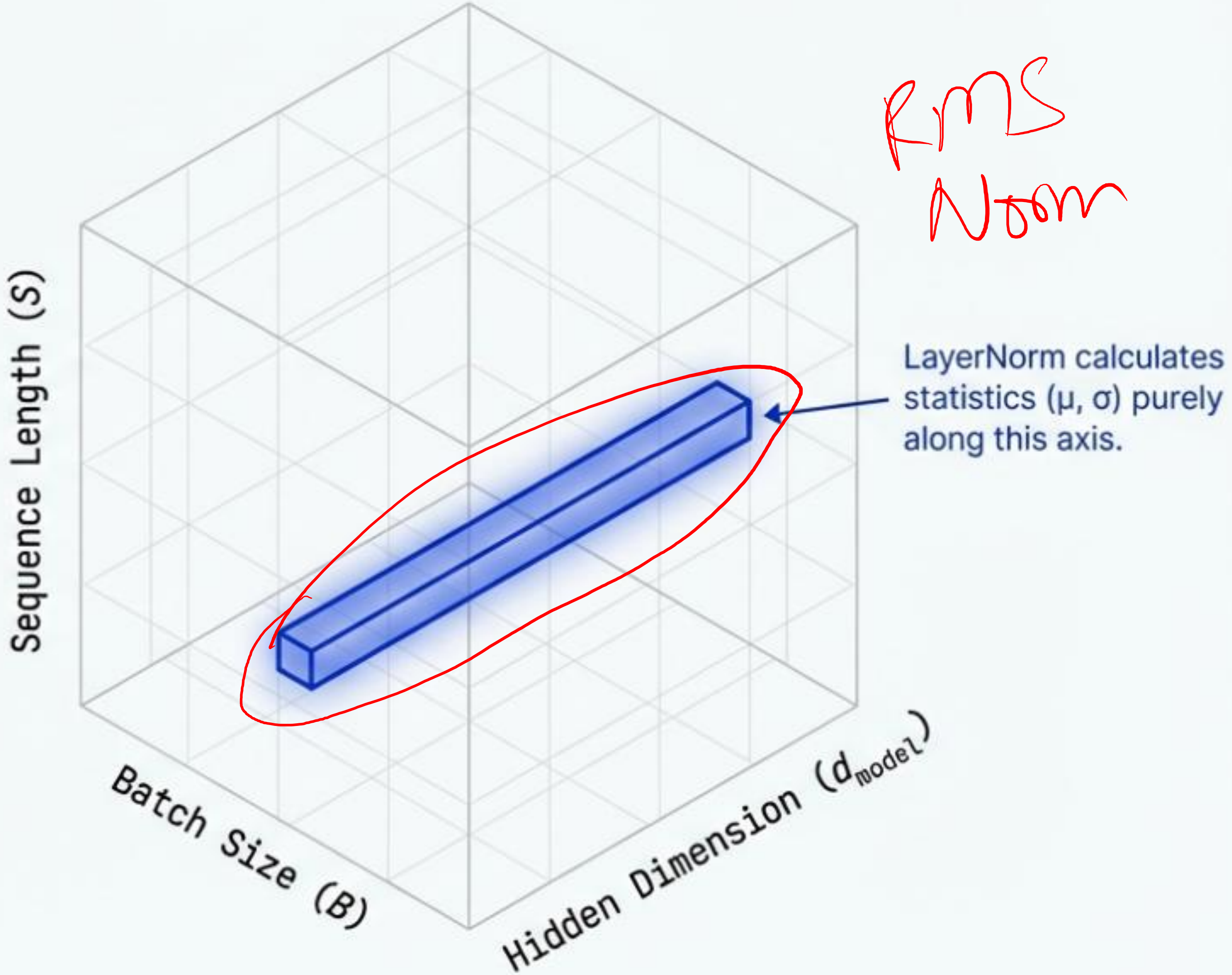
Tokens (Columns)
JetBrains Mono

LayerNorm



Key Concept: LayerNorm normalizes across the features of a single token. It is completely independent of other tokens in the sequence and other examples in the batch.

Takeaway: This independence is what makes LayerNorm robust for NLP tasks.

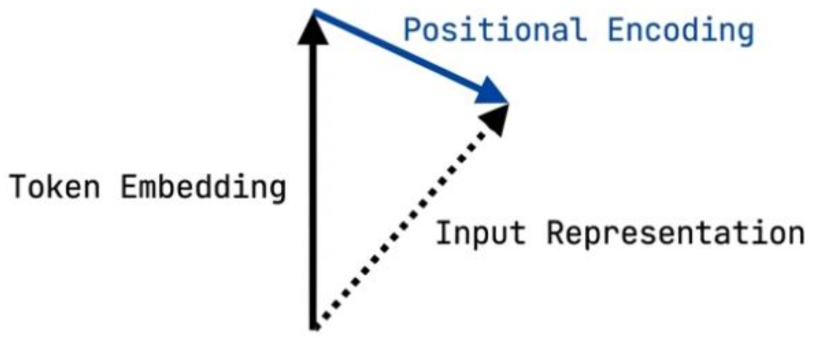


LayerNorm calculates statistics (μ, σ) purely along this axis.

RMS Norm

Positional Encoding

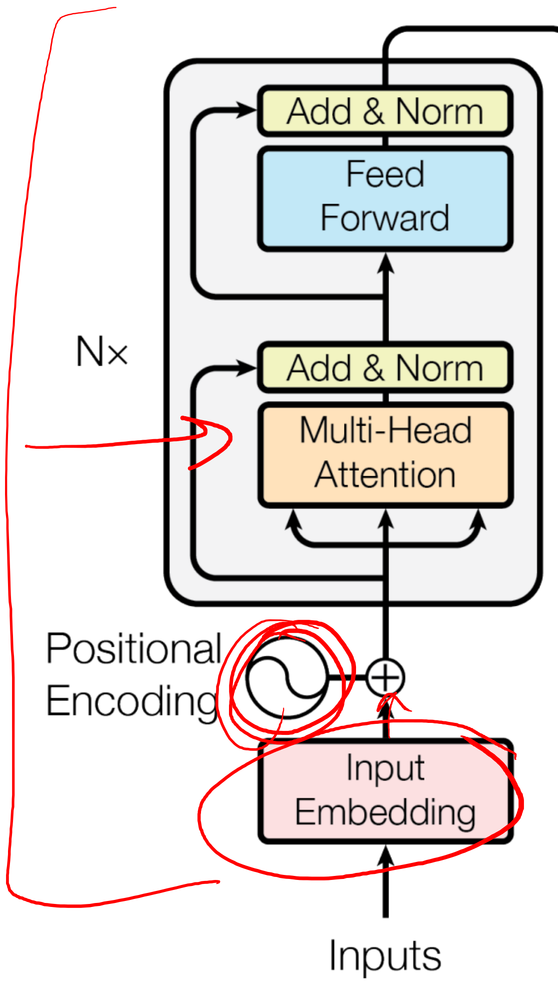
The Problem: Neural networks are permutation invariant. Without explicit signals, “The dog bit the man” looks identical to “The man bit the dog” to the attention mechanism.



```
X_0 = TokenEmbed(x) + PosEmbed(pos)
```

Encoding Strategies:

1. Absolute (Sinusoidal/Learned)
2. Relative (Distance-based bias)
3. RoPE (Rotary Position Embeddings - rotation of Q/K)



Positional Encoding: Sinusoidal

- A → 00
- Quick → 01
- Brown → 10
- Fox → 11

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

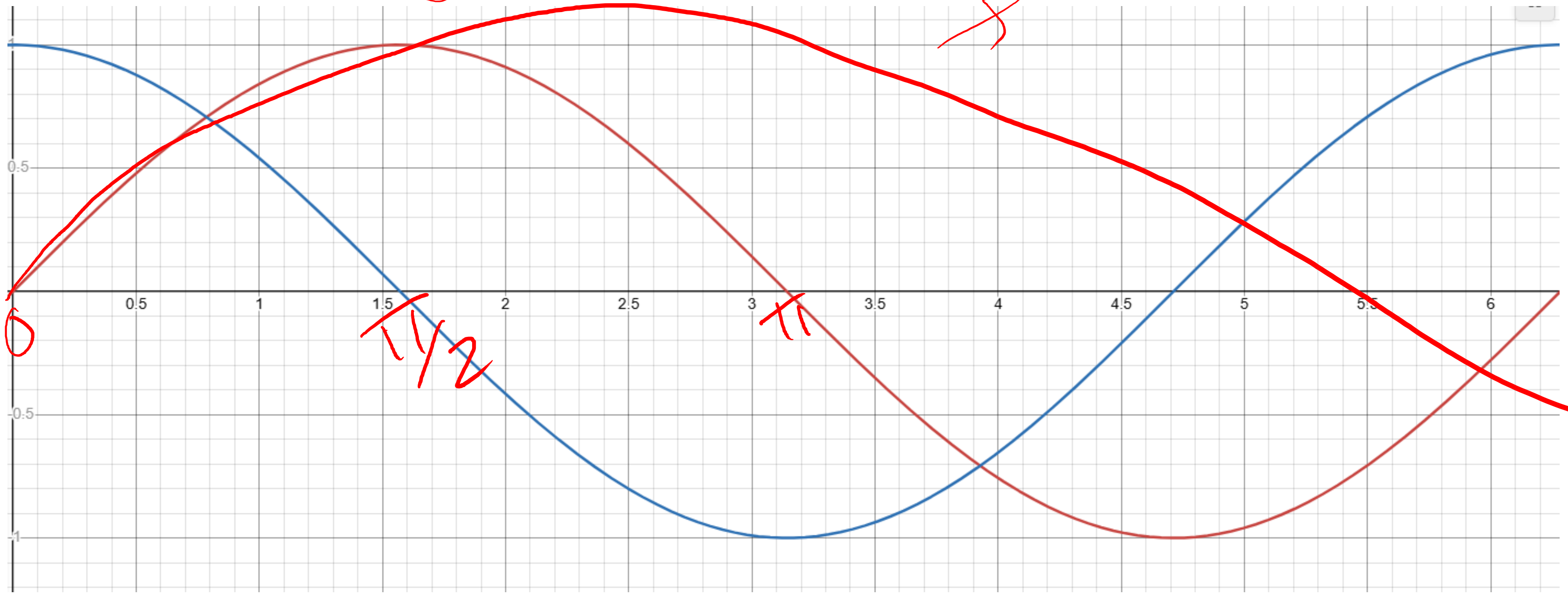
s | c

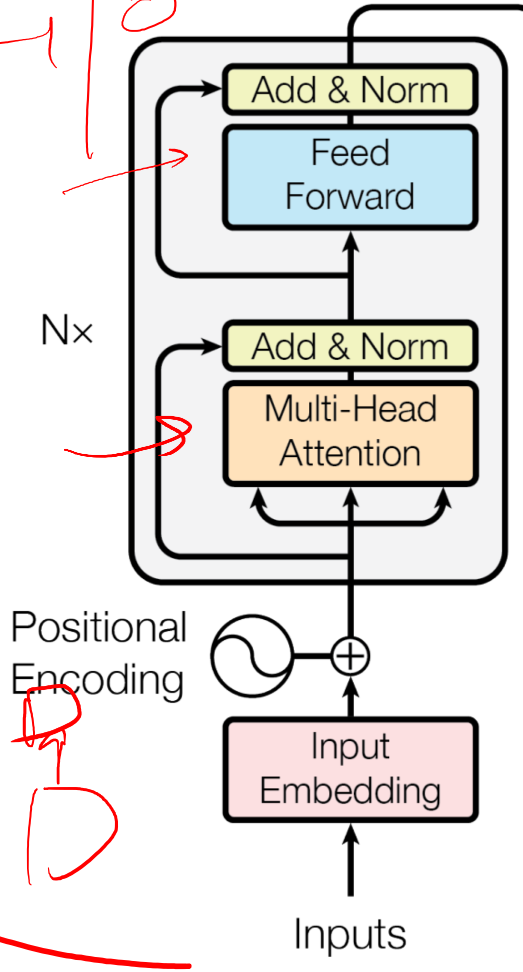
0 → 0 | 1 →

1 → 1 | 0

2 → 0 | -1

3 → -1 | 0





Lab

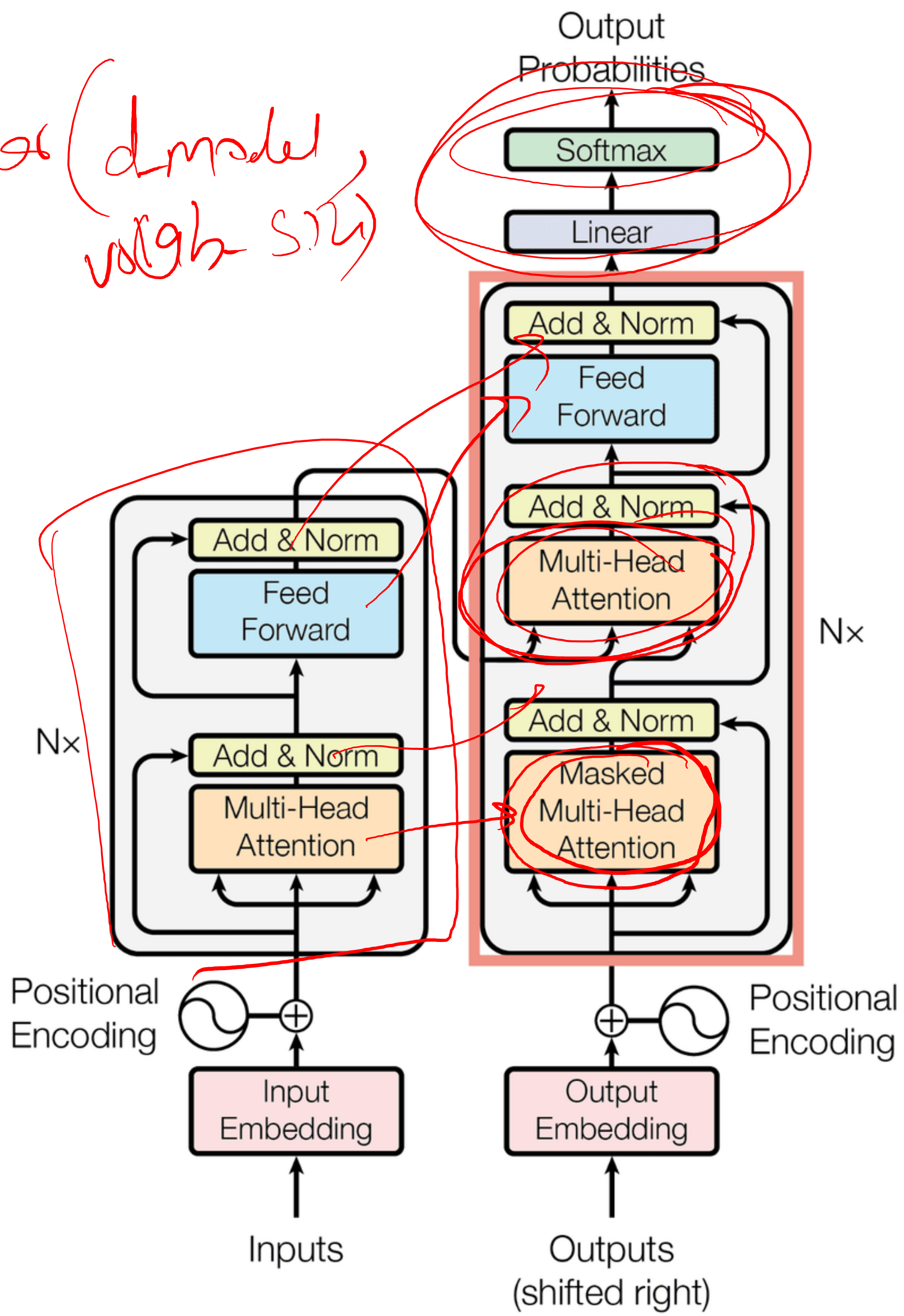
https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

Encoder - Decoder

nn.Linear(dmodel, vocab size)

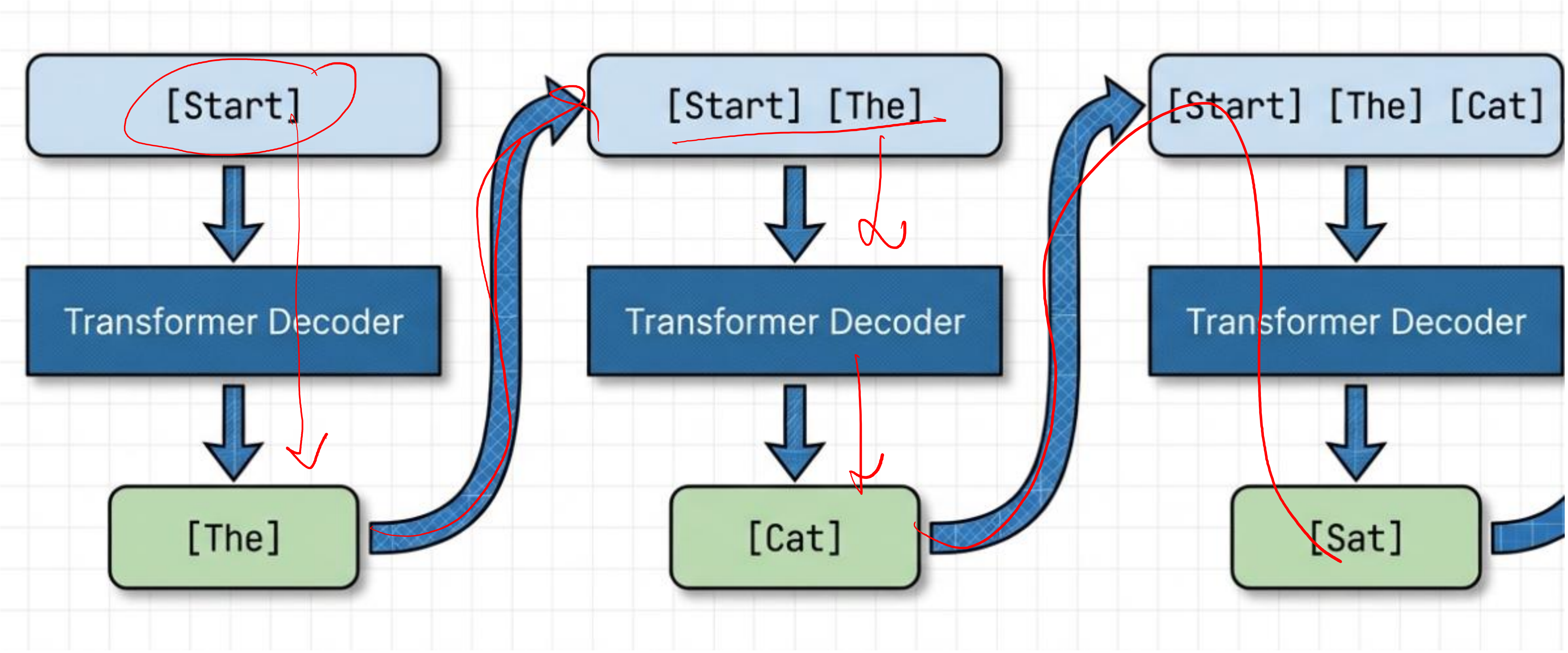
- While decoder inference is sequential, can we make decoder training parallel? - Masked Attention
- Passing Encoded Input to Decoder - Cross Attention
- Cross Entropy – nn.Linear(d_model, vocab_size)

$p(t_i \mid \theta, t_{<i})$

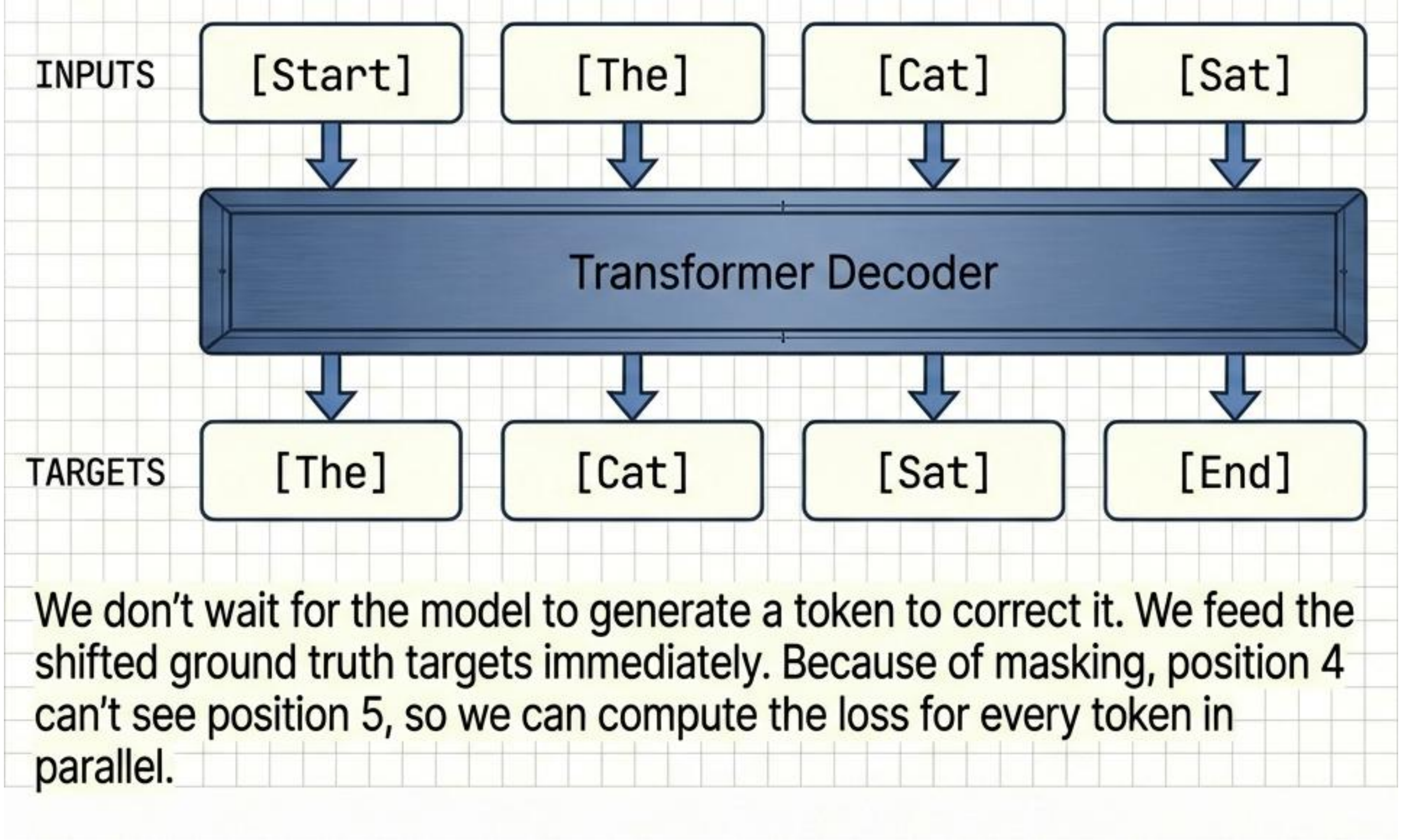


Inference - Auto Regressive

[End]



Training – Teacher Forcing



We don’t wait for the model to generate a token to correct it. We feed the shifted ground truth targets immediately. Because of masking, position 4 can’t see position 5, so we can compute the loss for every token in parallel.

Masked Self Attention

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

<SOS>

<SOS> The

<SOS> The Cat

<SOS> The Cat Sat

$$\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

0

Enforcing Causality (No Peeking)

	t1	t2	t3	t4	t5
t1	0.9	-inf	-inf	-inf	-inf
t2	0.4	0.5	-inf	-inf	-inf
t3	0.2	0.3	0.5	-inf	-inf
t4	0.1	0.1	0.3	0.5	-inf
t5	0.0	0.1	0.2	0.3	0.4

Causal Mask (Upper Triangle = -inf)

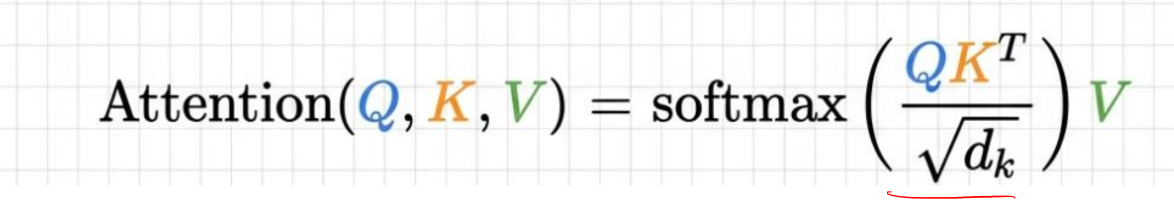
Logic: During training, the model must predict the next word based **ONLY** on the past.

Mechanism: The upper triangular mask sets future position scores to negative infinity before softmax, forcing their probability to zero.

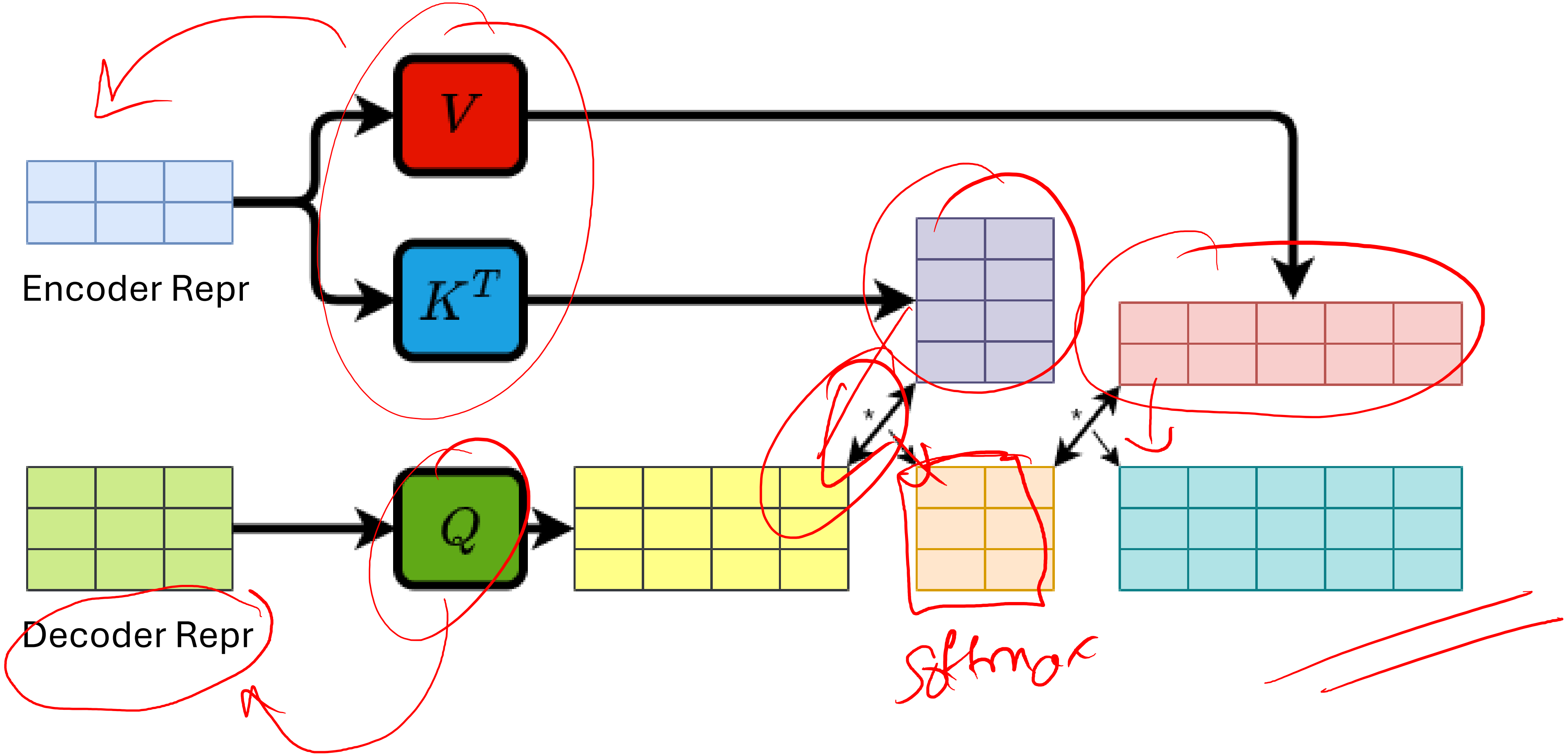
$$x \rightarrow \begin{cases} Q \rightarrow \\ K \rightarrow \\ V \rightarrow \end{cases} \text{Atn}(\ )$$

The decoder uses this mask to ensure autoregressive generation, processing one token at a time without access to future context.

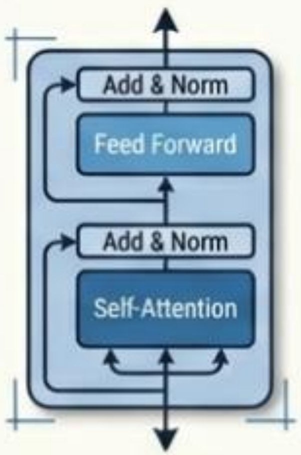
Cross Attention



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



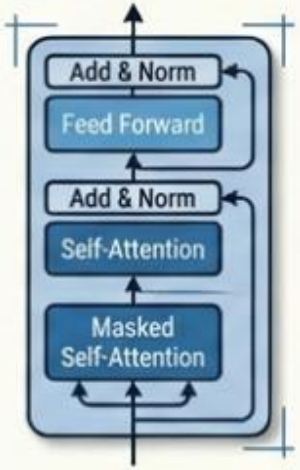
Encoder-Only



Example: BERT

- Use Case: Classification
- Sentiment
- Search

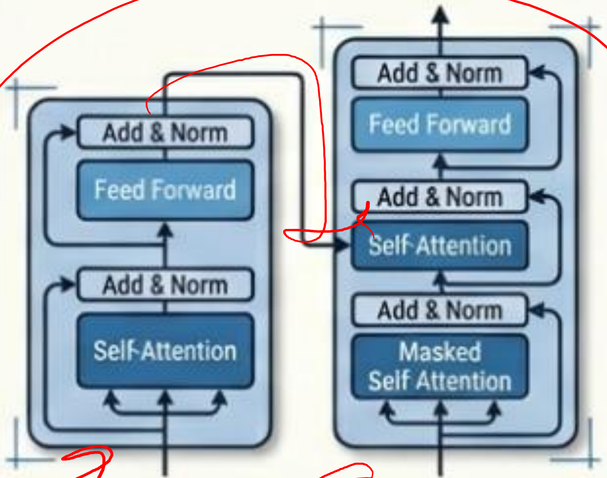
Decoder-Only



Example: GPT

- Use Case: Text Generation
- Zero-shot tasks

Encoder-Decoder



Example: T5, BART

- Use Case: Translation
- Summarization

The ‘LLM’ revolution is a story of specialization. BERT optimizes for understanding; GPT optimizes for creation; T5 optimizes for translation.

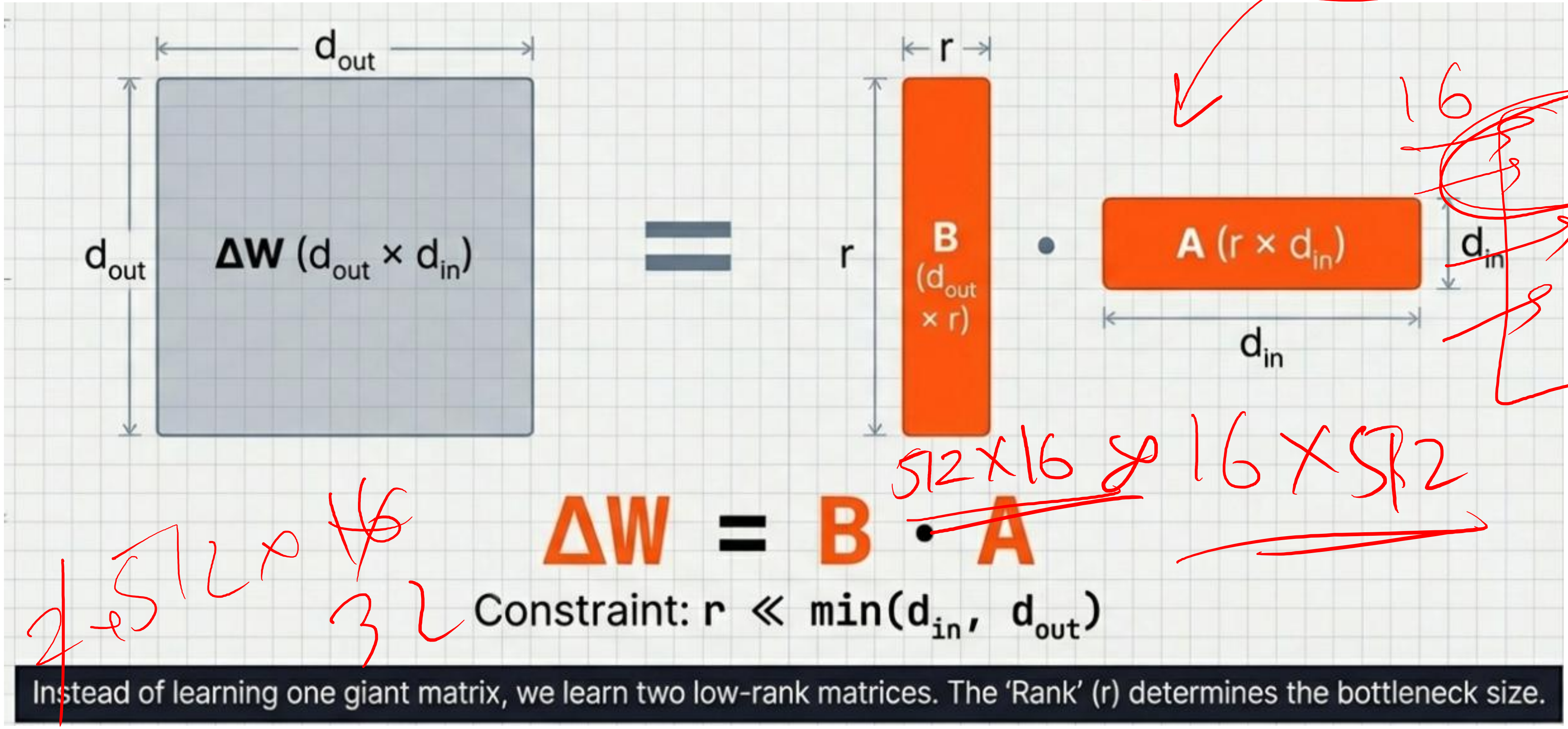
Lab

https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

Appendix

Parameter Efficient Finetuning: Low Rank Adaptation (LoRA)

512×512 $W_Q + \Delta W_Q$ W_K + Train

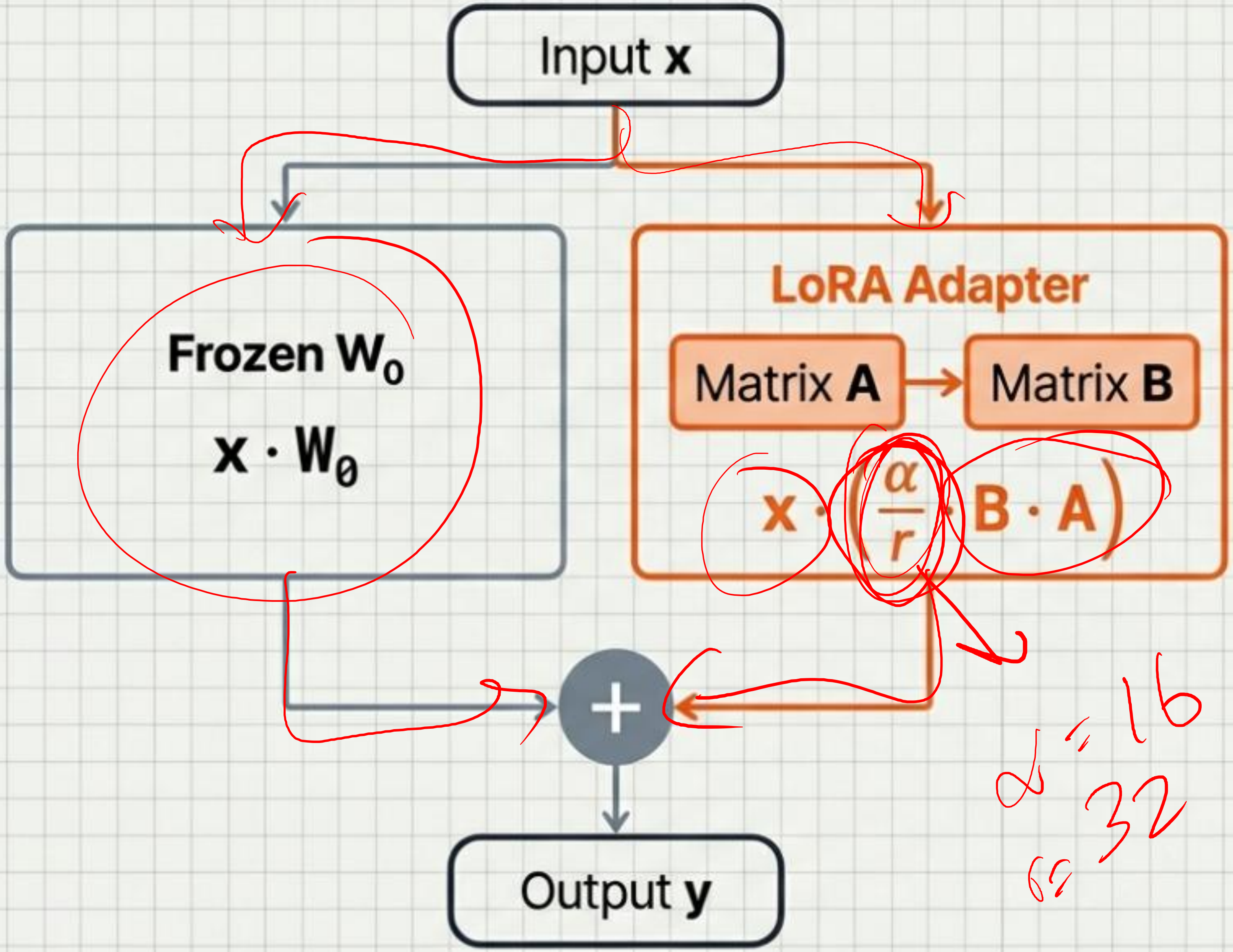


Instead of learning one giant matrix, we learn two low-rank matrices. The 'Rank' (r) determines the bottleneck size.

W_O

512×512

LoRA: Forward Pass



Initialization Strategy

- Matrix A Init: Random Gaussian Noise
- Matrix B Init: ZERO
- Result at Step 0: **B · A = 0**

Key Insight:

Because **B** starts at zero, the model behaves exactly like the pretrained model at the start of training. No instability.