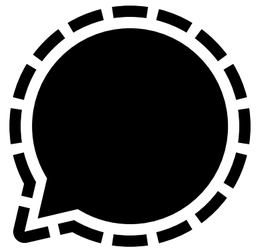
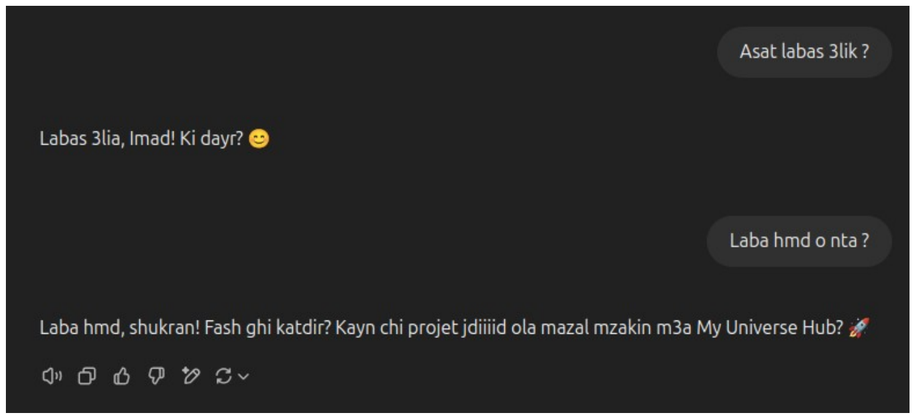
Teaching a language model to chat like you

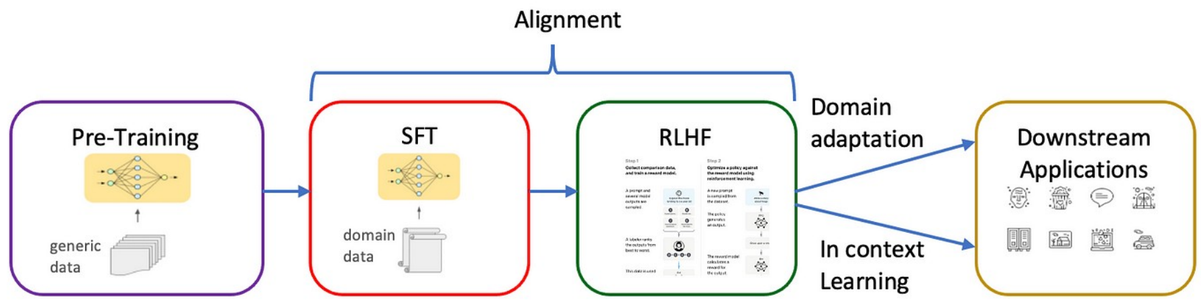
Taught by **Imad Saddik**


Why ?

- See if we can **mathematically simulate** how a person talks.
- *Bring someone who died back to life?* In the form of a language model.
- **Learn the full pipeline:** data extraction, processing, and model training.
- LLMs struggle with Moroccan Darija.



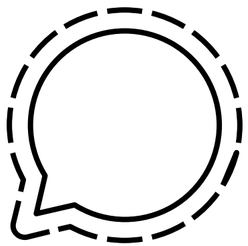


Medium article

How ?

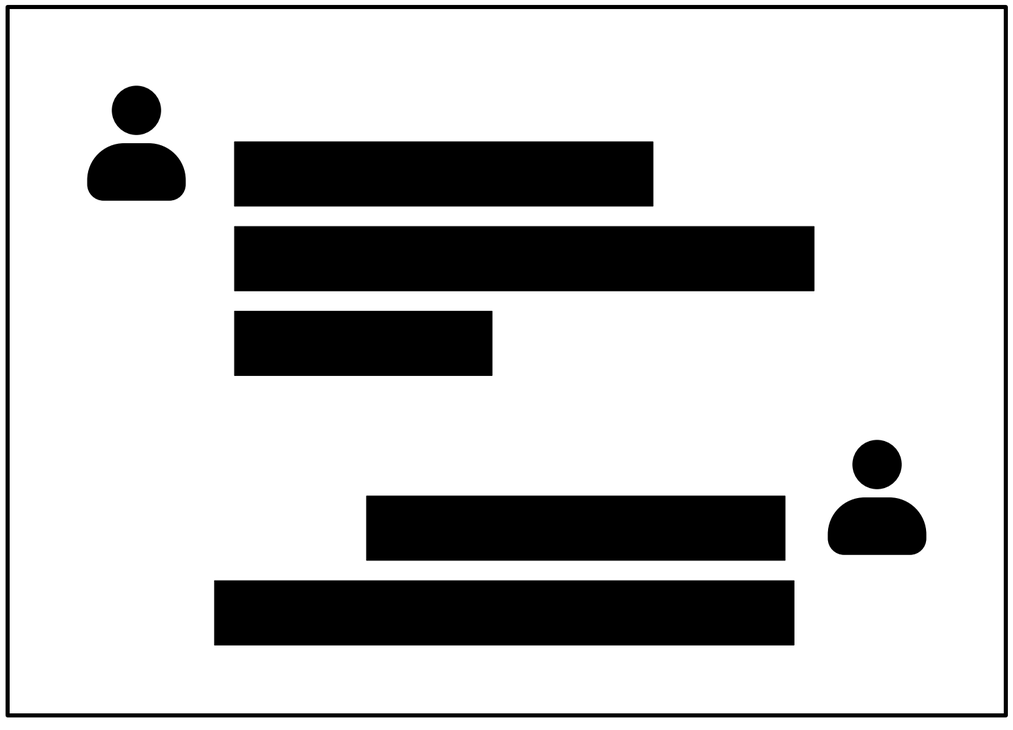
1) Data extraction

- Group chats.
- Personal conversations.

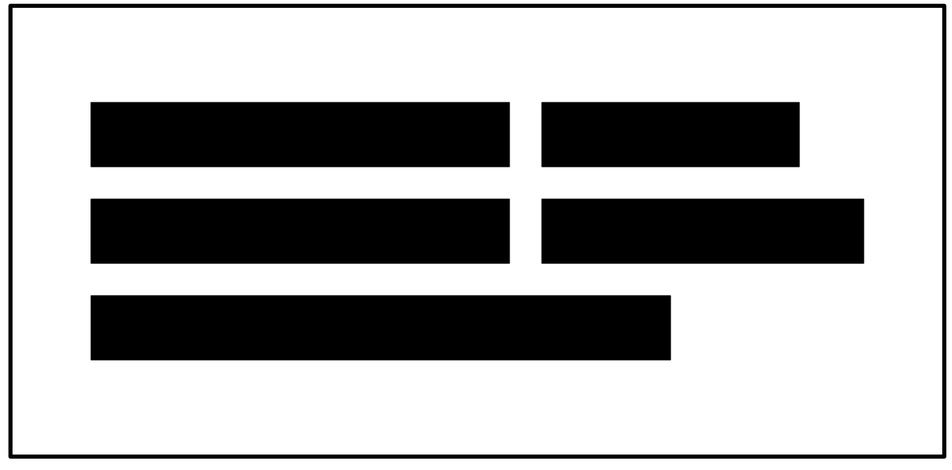


How ?

1) Data extraction
2) Combine the extracted text



Raw data



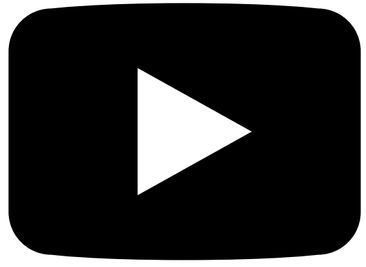
Sequence of text

How ?

1) Data extraction
2) Combine the extracted text
3) BPE algorithm & text encoding

Wikipedia article

Hugging Face course



Andrej Karpathy’s video

How ?

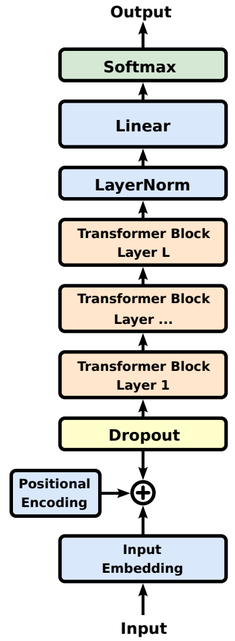
1) Data extraction
2) Combine the extracted text
3) BPE algorithm
4) Text encoding

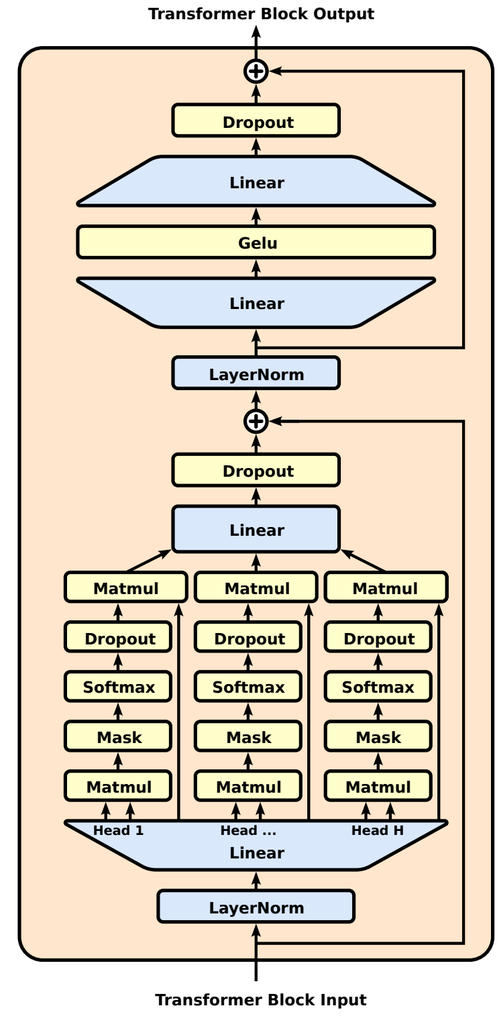
How ?

1) Data extraction
2) Combine the extracted text
3) BPE algorithm
4) Text encoding

5) Create the model

StatQuest video





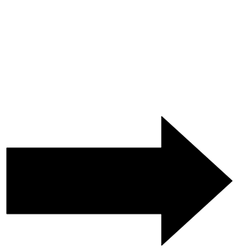
How ?

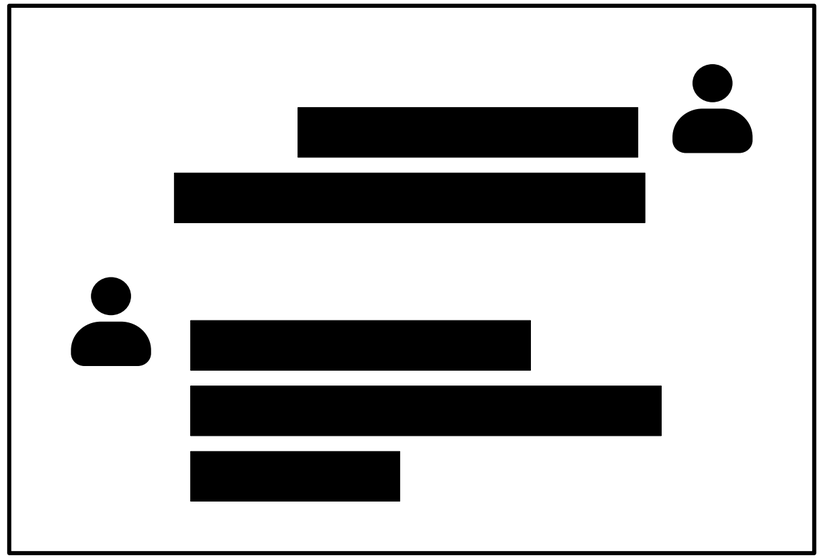
1) Data extraction
2) Combine the extracted text
3) BPE algorithm
4) Text encoding
5) Create the model
6) Data splitting & model training

How ?

1) Data extraction
2) Combine the extracted text
3) BPE algorithm
4) Text encoding
5) Create the model
6) Data splitting & model training
7) Prepare the fine-tuning dataset

Base model





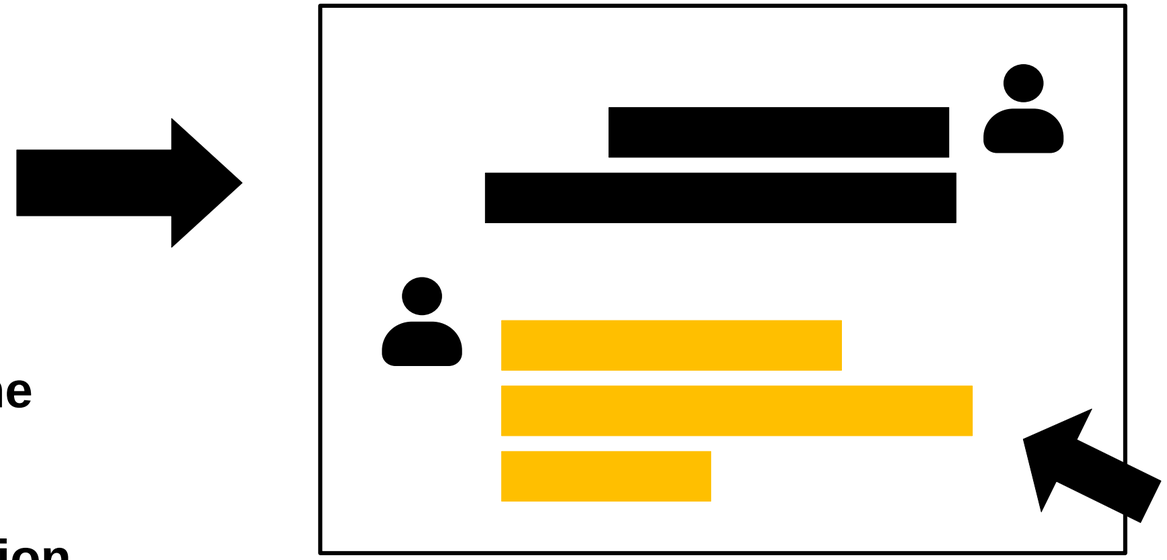
Fine-tuning dataset

- The model learned to **simulate the training data.**
- It can only do **next token prediction**.
- **Not useful**.

How ?

1) Data extraction
2) Combine the extracted text
3) BPE algorithm
4) Text encoding
5) Create the model
6) Data splitting & model training
7) Prepare the fine-tuning dataset

Base model



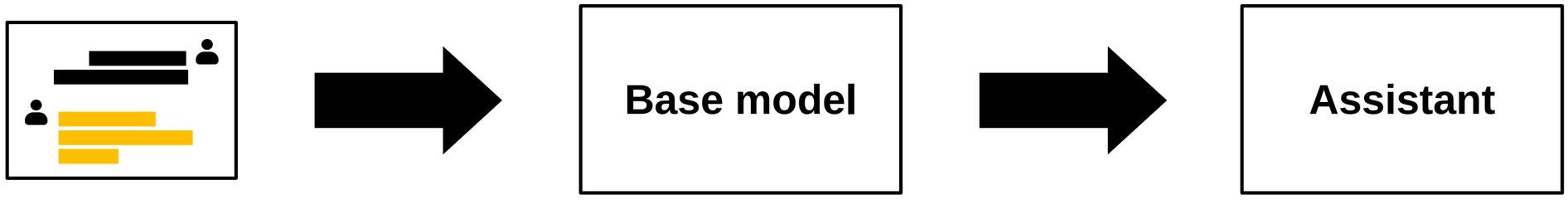
Fine-tuning dataset

The person you want to imitate

- The model learned to **simulate the training data.**
- It can only do **next token prediction**.
- **Not useful**.

How ?

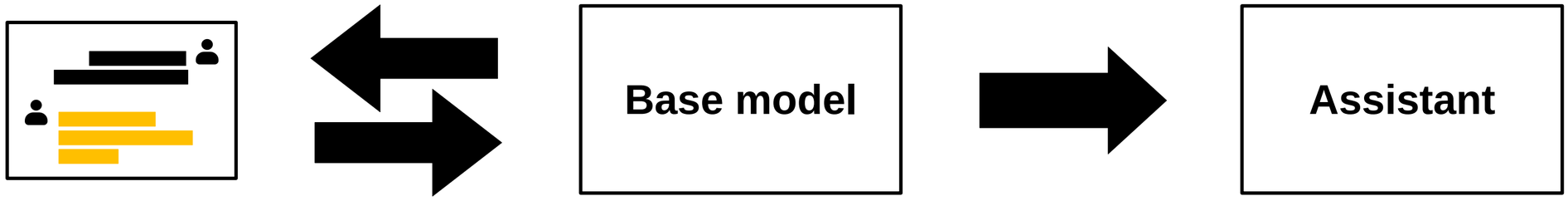
1) Data extraction
2) Combine the extracted text
3) BPE algorithm
4) Text encoding
5) Create the model
6) Data splitting & model training
7) Prepare the fine-tuning dataset
8) Fine-tune the base model



- The model now is able to **mimic the person** we chose in the dataset.
- The model is **useful**.

How ?

1) Data extraction
2) Combine the extracted text
3) BPE algorithm
4) Text encoding
5) Create the model
6) Data splitting & model training
7) Prepare the fine-tuning dataset
8) Fine-tune the base model

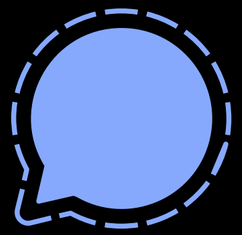


Finding the best dataset is a challenging task.

- The model now is able to **mimic the person** we chose in the dataset.
- The model is **useful**.

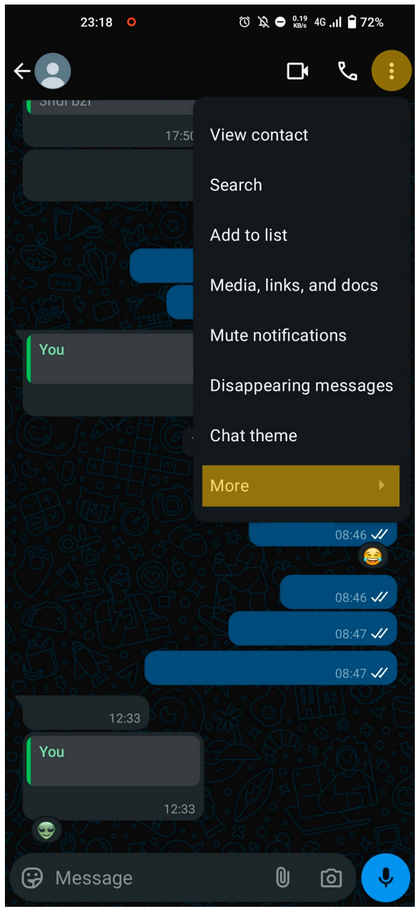
Let’s start

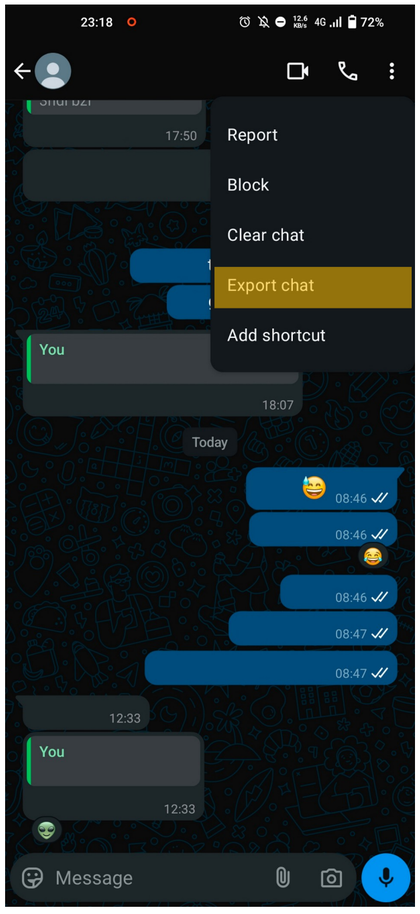
DATA EXTRACTION

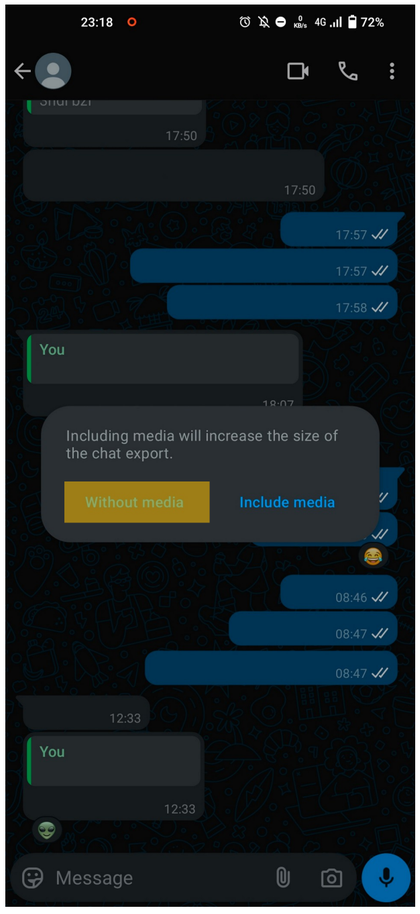


Data extraction

- Extracting the data from WhatsApp is a manual process.
- Follow these steps:





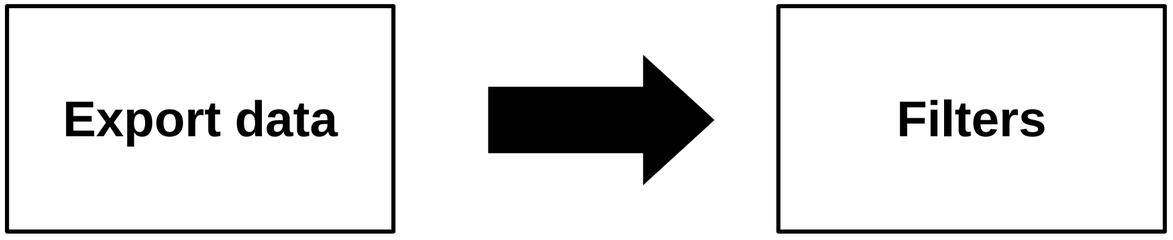


Data extraction

Export data

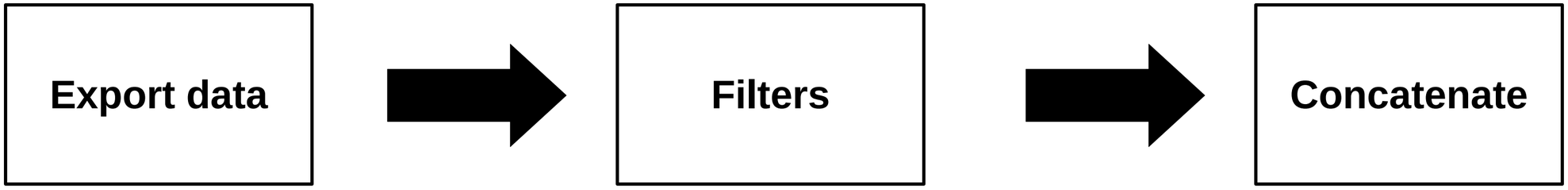
01/04/2069, 04:20 - You: Hello world!

Data extraction



- Remove **links**
- Remove special tags like **<Media omitted>**, **<You deleted this message>**
- Remove **email addresses**
- Remove **tagging**
- ...

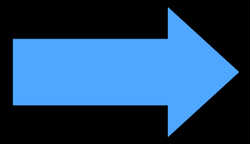
Data extraction



One big sequence of text

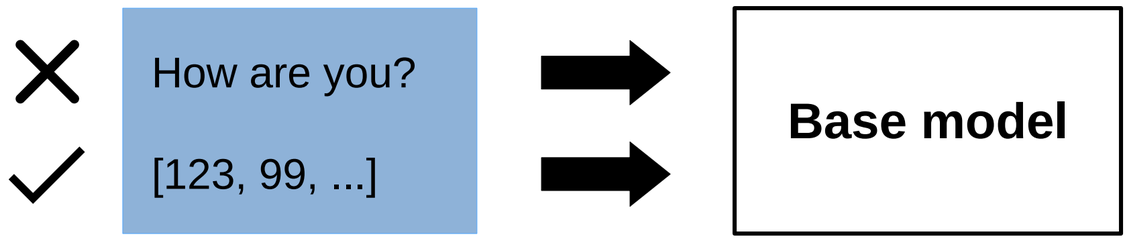
Let’s code

TEXT ENCODING

Hello world  256, 99, 543

Text encoding

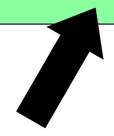
- Language models **don’t use text** as input.
- They work with **numbers**.
- **Encoding** is turning text into numbers.
- Methods for **text encoding**: Character level, Word level, BPE (Sub word level), ...



Text encoding

- Methods for **text encoding**: Character level, Word level, BPE.
- **Input example**: *Text encoding is an important step that affects later stages in the pipeline.*
- You should find a **compromise** between the **vocabulary size** and the **sequence length**.
  - ↑ Vocabulary size ↓ Sequence length
  - ↓ Vocabulary size ↑ Sequence length

Tokenization	Vocabulary size	Example tokens	Sequence length
Character level	~100-500	"T", "e", "x", "t"	78
BPE	~10,000-50,000	"Text", "enco", "ding"	14
Word level	~50,000+	"Text", "encoding", "is"	13



Computationally expensive

Text encoding

- Methods for **text encoding**: Character level, Word level, BPE.
- **Input example**: *Text encoding is an important step that affects later stages in the pipeline.*
- You should find a **compromise** between the **vocabulary size** and the **sequence length**.
  - ↑ Vocabulary size ↓ Sequence length
  - ↓ Vocabulary size ↑ Sequence length

Tokenization	Vocabulary size	Example tokens	Sequence length
Character level	~100-500	"T", "e", "x", "t"	78
BPE	~10,000-50,000	"Text", "encoding", "is"	14
Word level	~50,000+	"Text", "enco", "ding"	13

Text encoding - BPE

- BPE (Byte Pair Encoding)
- **Compresses** the text into a small sequence.

aaabdaaabac → [97, 97, 97, 98, 100, 97, 97, 97, 98, 97, 99]

→ **(97, 97)** : 4
(97, 98) : 2
...

→ [**666**, 97, 98, 100, **666**, 97, 98, 97, 99]

...

Merge #1

[666, 97, 98, 100, 666, 97, 98, 97, 99] →

(666, 97) : 2
(97, 98) : 2
...

→ [**777**, 98, 100, **777**, 98, 97, 99]

Merge #2

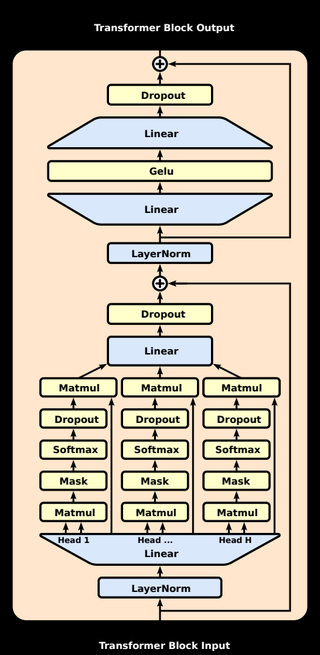
...

Text encoding – Useful links

- Wikipedia: https://en.wikipedia.org/wiki/Byte_pair_encoding
- YouTube: https://www.youtube.com/watch?v=zduSFxRajkE&t
- Medium: link
- Tool: https://tiktokenizer.vercel.app/
- minBPE: https://github.com/karpathy/minbpe

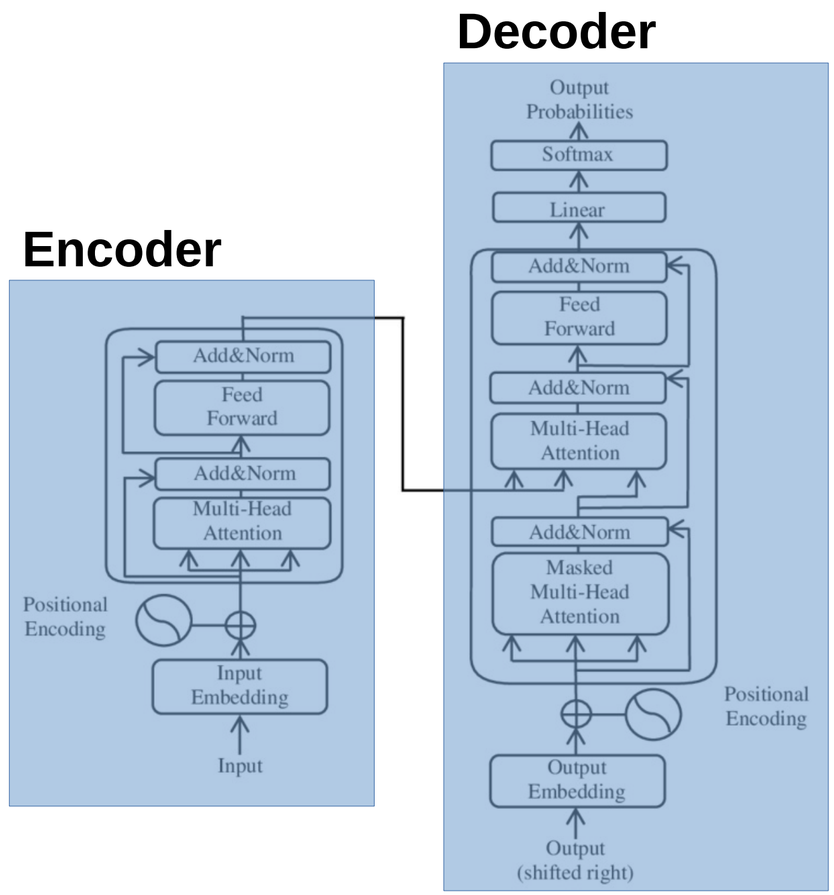
Let’s code

THE TRANSFORMER MODEL



The transformer

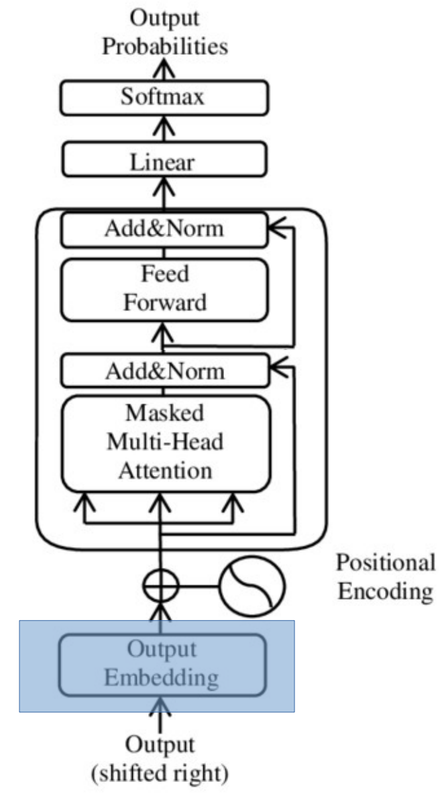
- The **transformer** architecture is used to train language models.
- It is divided into 2 parts: **The encoder and the decoder**.
- For tasks like machine translation you use both parts.
- To imitate the training data, you can use just the decoder.



The transformer - Decoder

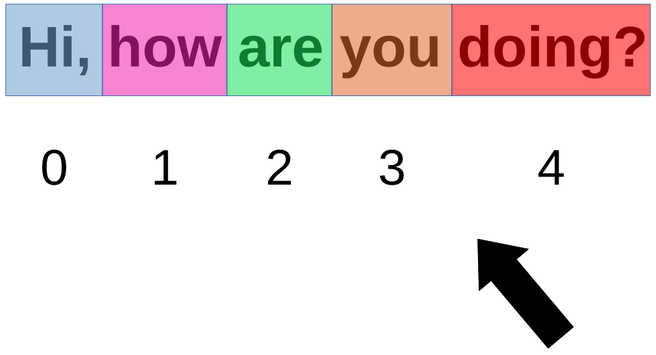
- The decoder is composed of these components:
  - **Token embedding** (Represents a token with a vector).

Embedding dimensions	0	1	2	...	n-1
hel	0.21	-2.1	0.62	...	-0.33
imad	-2.81	0.1	0.23	...	0.55
you	0.71	0.11	1.77	...	-0.23

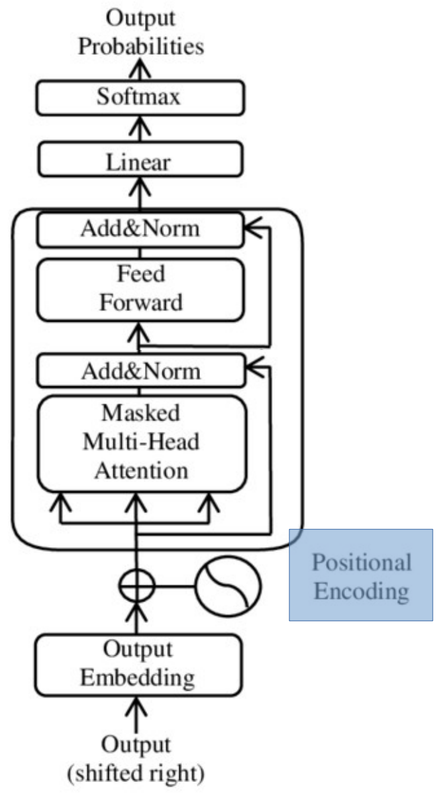


The transformer - Decoder

- The decoder is composed of these components:
  - **Token embedding** (Represent a token with a vector).
  - **Positional encoding** (Preserves the token order).

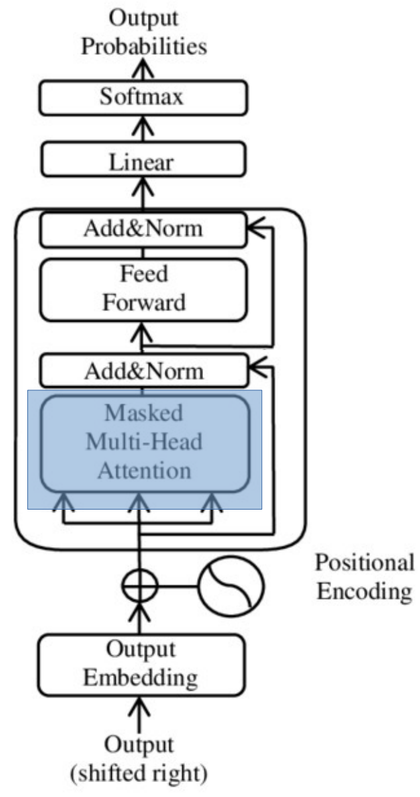


*The position encoding is a vector not a single value.



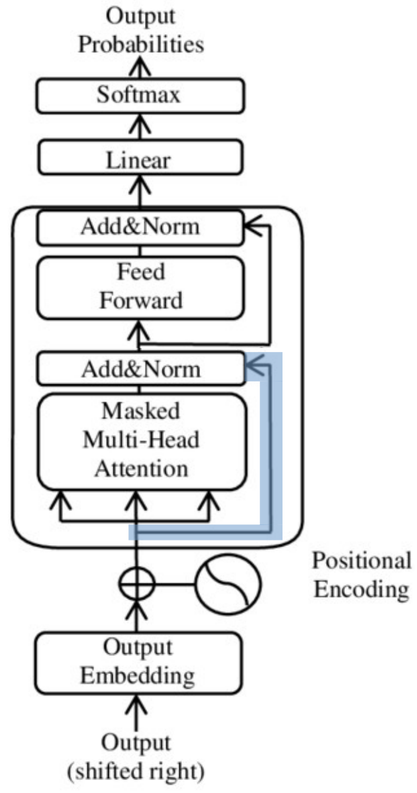
The transformer - Decoder

- The decoder is composed of these components:
  - **Token embedding** (Represent a token with a vector).
  - **Positional encoding** (Preserves the token order).
  - **Self attention** (Keeps track of the relationship between tokens).



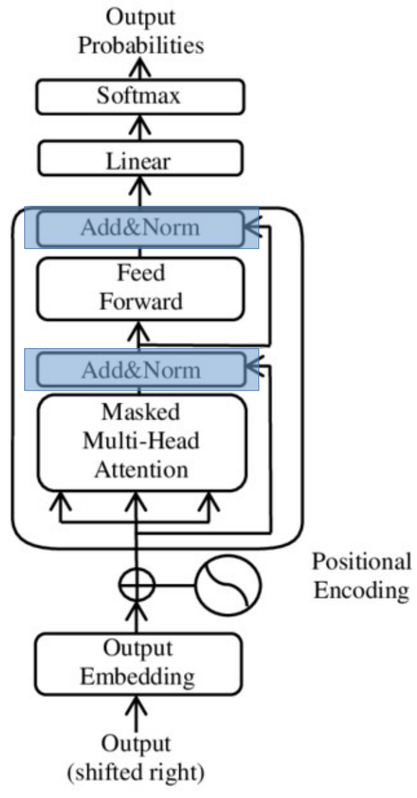
The transformer - Decoder

- The decoder is composed of these components:
  - **Token embedding** (Represent a token with a vector).
  - **Positional encoding** (Preserves the token order).
  - **Self attention** (Keeps track of the relationship between tokens).
  - **Residual connections** (Helps the gradient flow easily through the network).



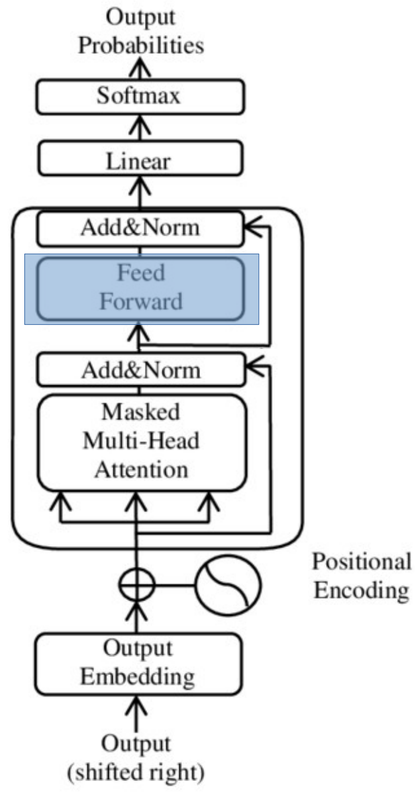
The transformer - Decoder

- The decoder is composed of these components:
  - **Token embedding** (Represent a token with a vector).
  - **Positional encoding** (Preserves the token order).
  - **Self attention** (Keeps track of the relationship between tokens).
  - **Residual connections** (Helps the gradient flow easily through the network).
  - **Layer normalization** (Normalizes the input).



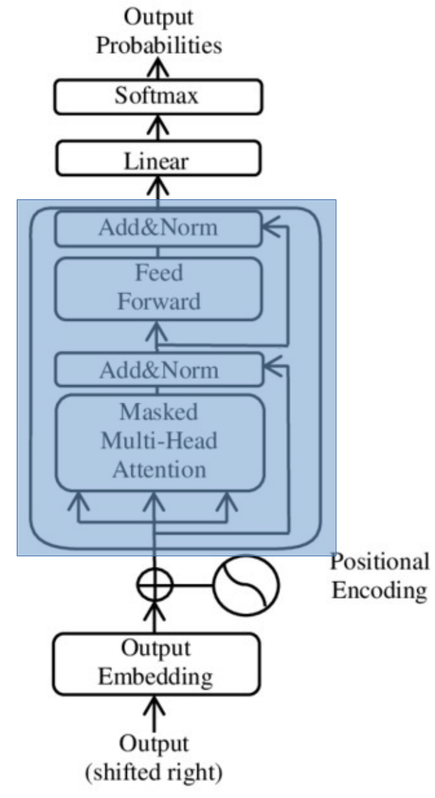
The transformer - Decoder

- The decoder is composed of these components:
  - **Token embedding** (Represent a token with a vector).
  - **Positional encoding** (Preserves the token order).
  - **Self attention** (Keeps track of the relationship between tokens).
  - **Residual connections** (Helps the gradient flow easily through the network).
  - **Layer normalization** (Normalizes the input).
  - **MLP layers** (Expands the model's capacity to learn).



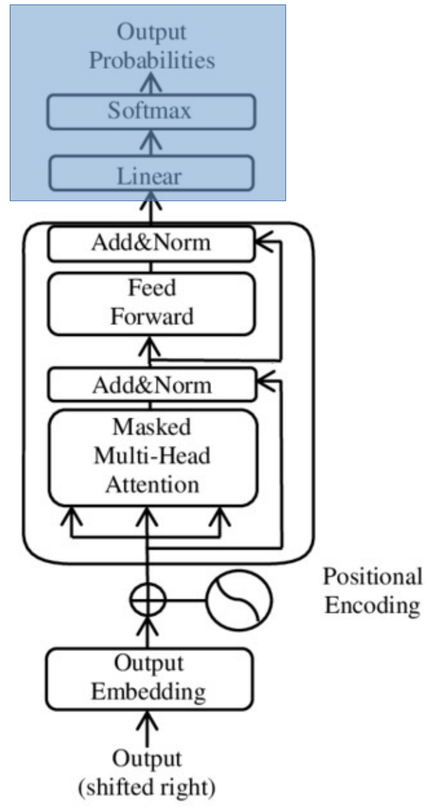
The transformer - Decoder

- The decoder is composed of these components:
  - **Token embedding** (Represent a token with a vector).
  - **Positional encoding** (Preserves the token order).
  - **Self attention** (Keeps track of the relationship between tokens).
  - **Residual connections** (Helps the gradient flow easily through the network).
  - **Layer normalization** (Normalizes the input).
  - **MLP layers** (Expands the model's capacity to learn).
  - **Block**



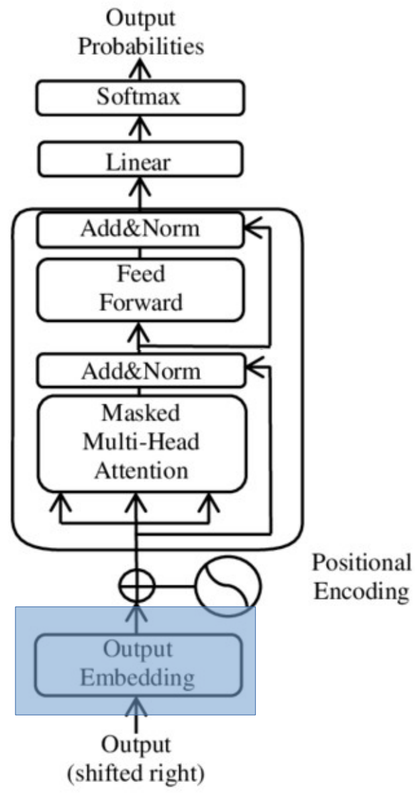
The transformer - Decoder

- The decoder is composed of these components:
  - **Token embedding** (Represent a token with a vector).
  - **Positional encoding** (Preserves the token order).
  - **Self attention** (Keeps track of the relationship between tokens).
  - **Residual connections** (Helps the gradient flow easily through the network).
  - **Layer normalization** (Normalizes the input).
  - **MLP layers** (Expands the model's capacity to learn).
  - **Block**
  - **Final layers** (To get the predictions).



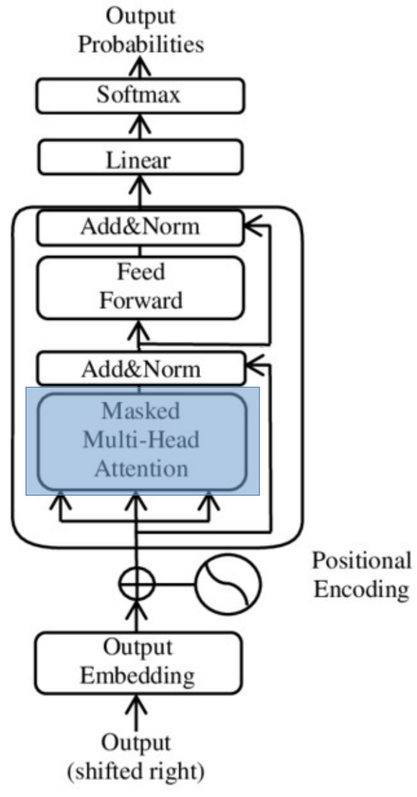
The transformer - Parameters

- **Block size** : Maximum sequence length.
- **Embedding size**



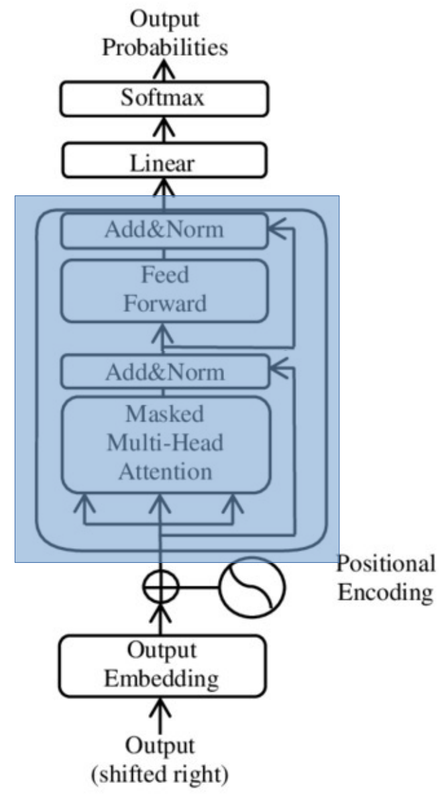
The transformer - Parameters

- **Block size** : Maximum sequence length.
- **Embedding size**
- **Number of heads & head size**

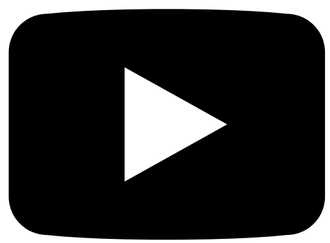


The transformer - Parameters

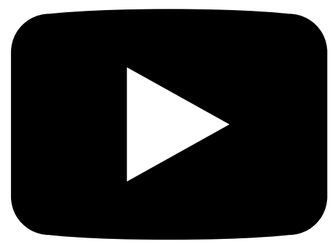
- **Block size** : Maximum sequence length.
- **Embedding size**
- **Number of heads & head size**
- **Number of blocks (layers)**



The transformer – Useful links



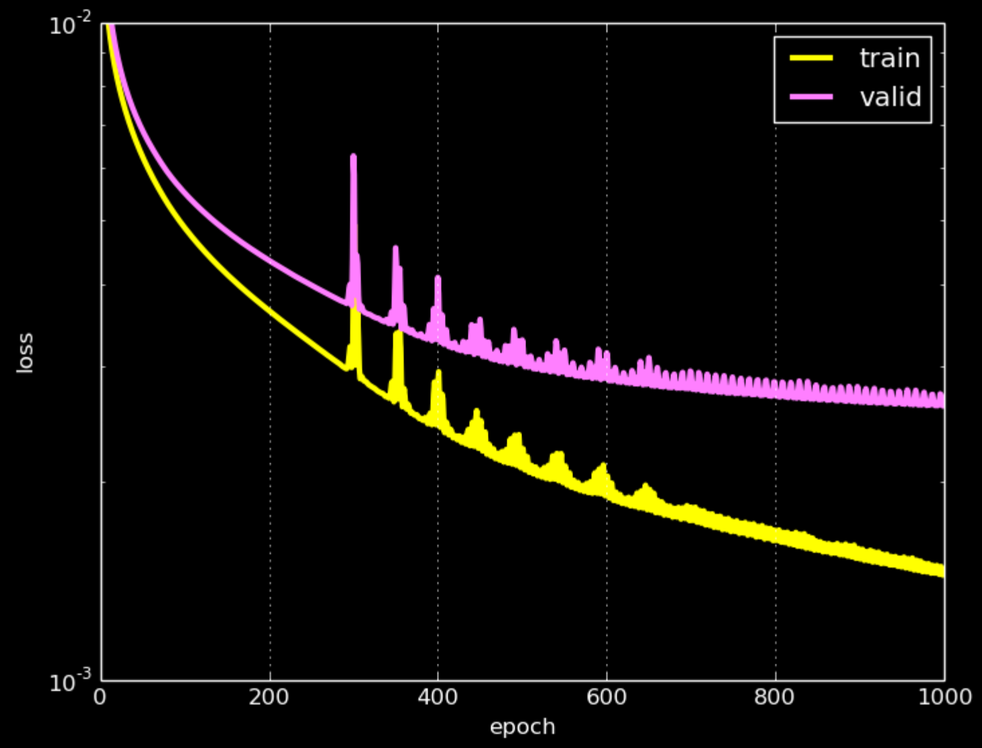
StatQuest playlist



Andrej Karpathy

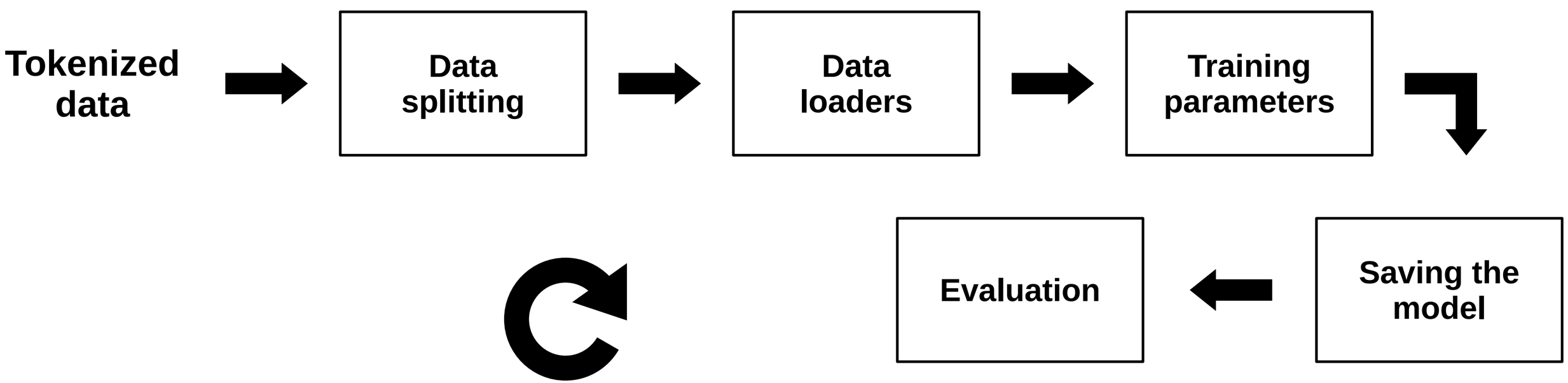
Let’s code

MODEL TRAINING



Model training

Training follows this process:



Model training – Data loaders

- Data loaders are used during training and evaluation.
- During training you can design your data loader in 2 ways:
  - **Random batches retrieval** 
  - **Use all the batches** 

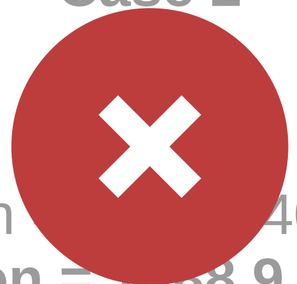
Model training – Data loaders

- Training set contains **4000 batches** of size 64.
- Number of training iterations is **1000**.
- Processing one batch takes **30 seconds**.
- We need to find a balance.

Case 1

duration = 1000 * 1 * 30
duration = 8.33 hours

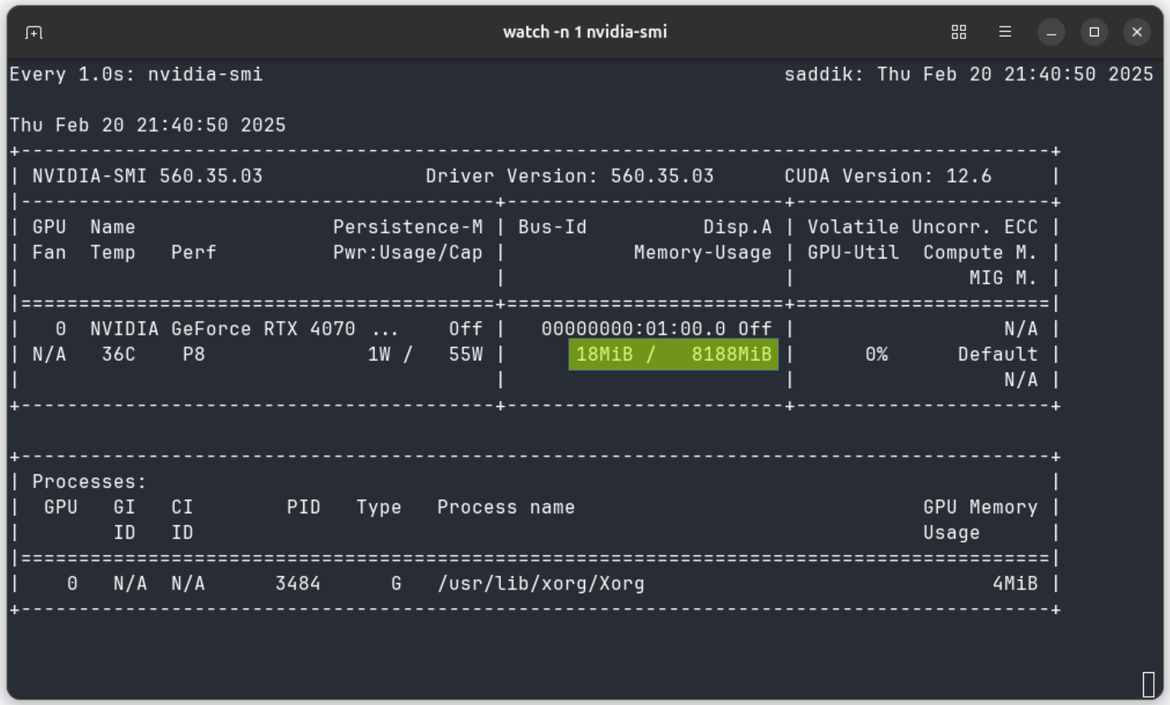
Case 2



duration = 1000 * 4000 * 30
Duration = 1388.9 days

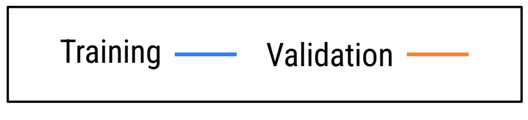
Model training – Training loop

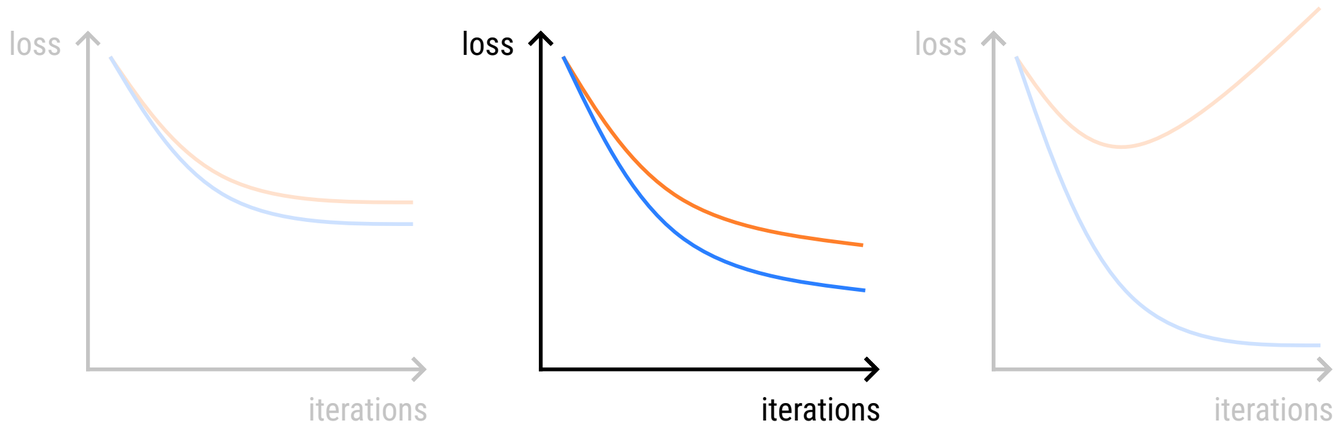
- Make sure that the memory is fully utilized **> 90%**



Model training – Training loop

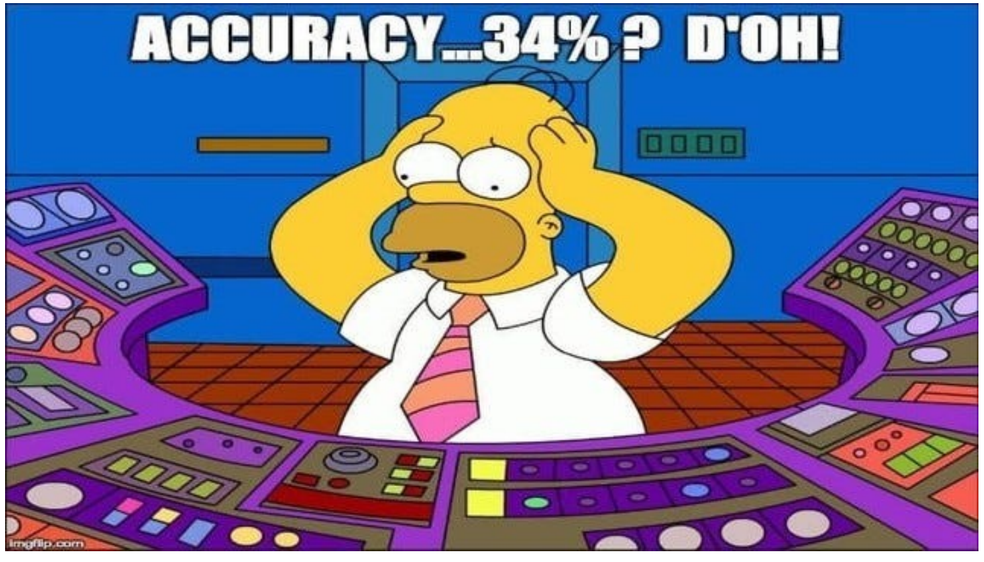
- Make sure that the **memory** is fully utilized **> 90%.**
- Start with a small **batch size** and keep increasing until **OOM (Out Of Memory).**
- **Save** the model regularly, every **x iteration**.
- **Monitor the loss**.





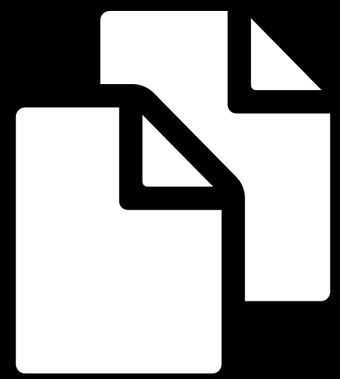
Model training - Evaluation

- **Generate from the model** and evaluate it yourself.
- Not satisfied? Repeat the process, this is how deep learning works.
- See if the model is able to **form coherent text** or not.
- The model does not need to be very very good.
- Because it will be **fine-tuned** later.



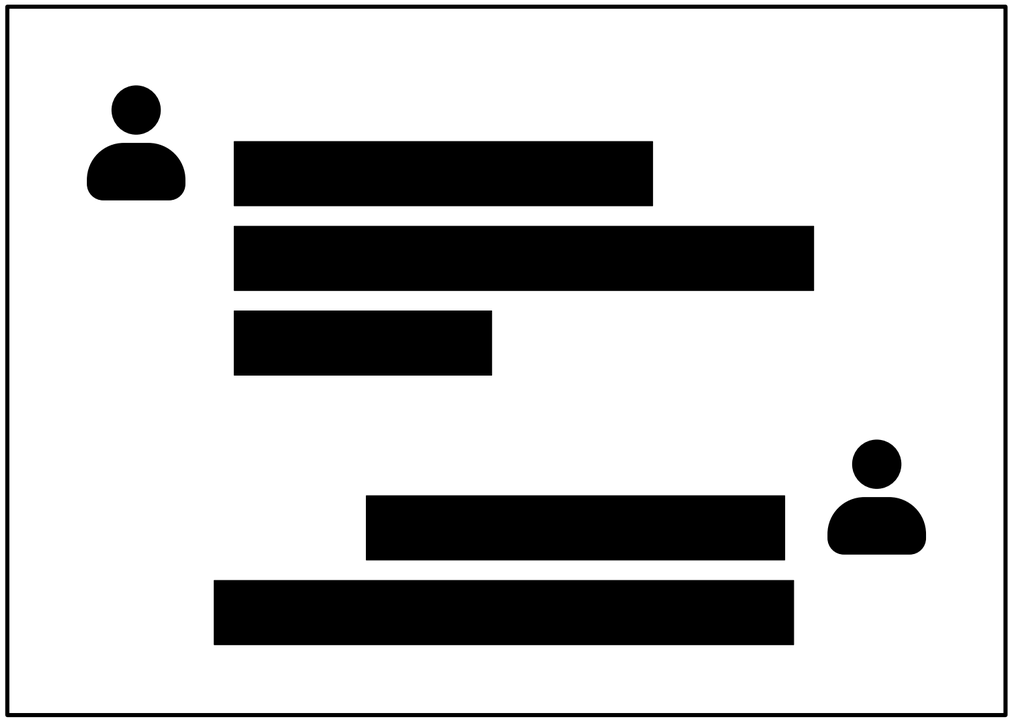
Let’s code

FINE-TUNING DATASET



Fine-tuning dataset

- This dataset is designed to **teach the base model** to be good at a **specific task**.
- In our case, the dataset should **mimic the way a person talks**.
- The **special tokens** are used here.



Raw data

<|startoftext|>User<|separator|>Hi, how are you?<|endoftext|>

<|startoftext|>Assistant<|separator|>Good, is there anything I can help with?<|endoftext|>

...

Fine-tuned data

Fine-tuning dataset

- This dataset is designed to **teach the base model** to be good at a **specific task**.
- In our case, the dataset should **mimic the way a person talks**.
- The **special tokens** are used here.
- The fine-tuning dataset **does not need to be big.**
- Create a high quality dataset if you can.

Fine-tuning dataset

- The input should not exceed the **block size**.
- Think of showing the model **multi-turn** examples.

<|startoftext|>User**<|separator|>**Hi, how are you?**<|endoftext|>**

<|startoftext|>Assistant**<|separator|>**Good, is there anything I can help with?**<|endoftext|>**

No context

<|startoftext|>User**<|separator|>**Hi, how are you?**<|endoftext|>**

<|startoftext|>Assistant**<|separator|>**Good, is there anything I can help with?**<|endoftext|>**

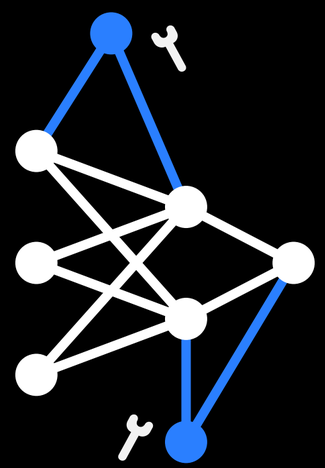
<|startoftext|>User**<|separator|>**I am preparing a new course.**<|endoftext|>**

<|startoftext|>Assistant**<|separator|>**Interesting! Can you tell me what does the course touch?**<|endoftext|>**

Clear context

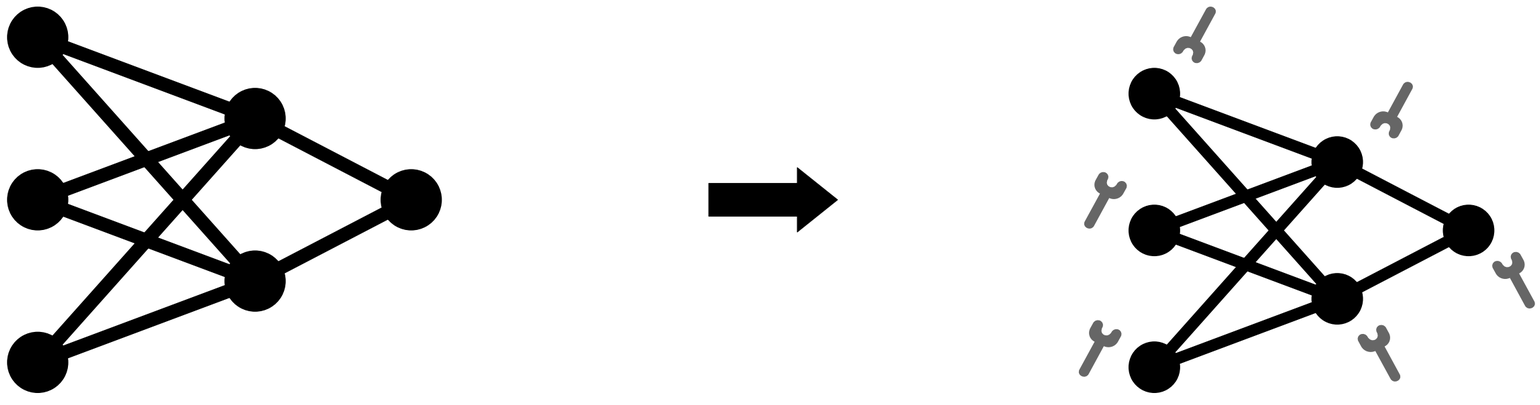
Let’s code

FINE-TUNING



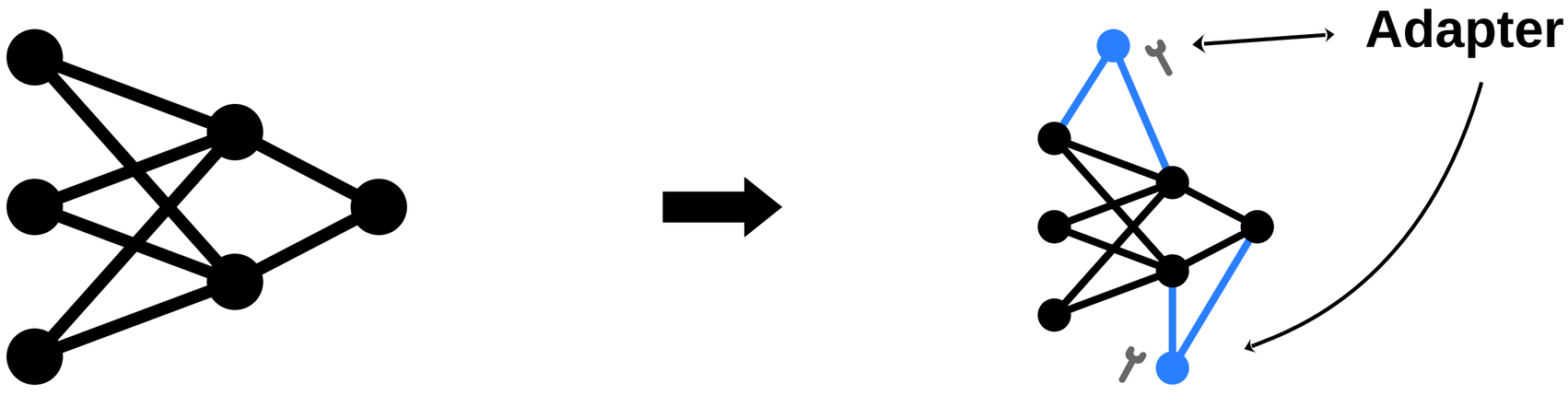
Fine-tuning

- Fine-tuning is the **additional training** performed on the base model.
- Assistant that **excels at a specific task**.
- Methods of fine-tuning:
  - **Full fine-tuning** (Instruction fine-tuning)



Fine-tuning

- Fine-tuning is the **additional training** performed on the base model.
- Assistant that **excels at a specific task**.
- Methods of fine-tuning:
  - **Full fine-tuning** (Instruction fine-tuning)
  - **PEFT** (Parameter Efficient Fine-Tuning)
  - **LORA or QLORA** (Low Rank Adaptation)

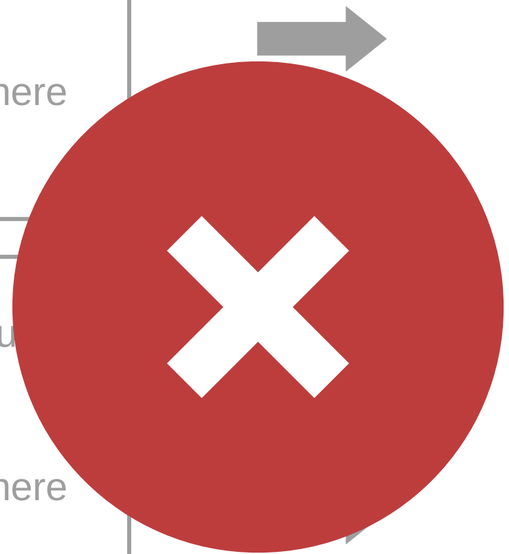


Fine-tuning – Efficient training

- Training on one example at a time is **inefficient**.

<|startoftext|>User<|separator|>Hi, how are you?
<|endoftext|>

<|startoftext|>Assistant<|separator|>Good, is there anything I can help with?<|endoftext|>



Input sequence = 43

<|startoftext|>User<|separator|>Hi, how are you
<|endoftext|>

<|startoftext|>Assistant<|separator|>Good, is there anything I can help with?<|endoftext|>

<|startoftext|>User<|separator|>What does 42 mean?<|endoftext|>

Input sequence = 86

Fine-tuning – Efficient training

Block size = 256

- Training on one example at a time is **inefficient**.
- We want to train on **batches**.
- **Solution**: Add **padding** tokens to make the sequences have the same size.

<|startoftext|>User**<|separator|>**Hi, how are you?**<|endoftext|>**

<|startoftext|>Assistant**<|separator|>**Good, is there anything I can help with?**<|endoftext|>**

[12, 543, 689, 430, 69, 44, 189]

<|startoftext|>User**<|separator|>**Hi, how are you?**<|endoftext|>**

<|startoftext|>Assistant**<|separator|>**Good, is there anything I can help with?**<|endoftext|>**

<|startoftext|>User**<|separator|>**What does 42 mean?**<|endoftext|>**

[12, 543, 689, 430, 69, 44, 189, 890, 1000, 321, 33, 543, 765, 89]

Fine-tuning – Efficient training

Block size = 256

- Training on one example at a time is **inefficient**.
- We want to train on **batches**.
- **Solution**: Add **padding** tokens to make the sequences have the same size.

<|startoftext|>User**<|separator|>**Hi, how are you?**<|endof**

<|startoftext|>Assistant**<|separator|>**Good, is there anyt
can help with?**<|endoftext|>**

[12, 543, 689, 430, 69, 44, 189, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]

[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 12, 543, 689, 430, 69, 44, 189]

<|startoftext|>User**<|separator|>**Hi, how are you?**<|endoftext|>**

<|startoftext|>Assistant**<|separator|>**Good, is there anything I can help with?**<|endoftext|>**

<|startoftext|>User**<|separator|>**What does 42 mean?**<|endoftext|>**

→

[12, 543, 689, 430, 69, 44, 189, 890, 1000, 321, 33, 543, 765, 89, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]

[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 12, 543, 689, 430, 69, 44, 189, 890, 1000, 321, 33, 543, 765, 89]

Fine-tuning – Efficient training

- Training on one example at a time is **inefficient**.
- We want to train on **batches**.
- **Solution**: Add **padding** tokens to make the sequences have the same size.
- The **padding** tokens are **not considered when calculating the loss**.

Fine-tuning – X & Y

- Each pair (x, y) is formed like the first stage.

<|startoftext|>User**<|separator|>**Hi, how are you?**<|endoftext|>**

<|startoftext|>Assistant**<|separator|>**Good, is there anything I can help with?**<|endoftext|>**

[12, 543, 689, 430, 69, 44, 189, -1, -1, -1, -1, -1]

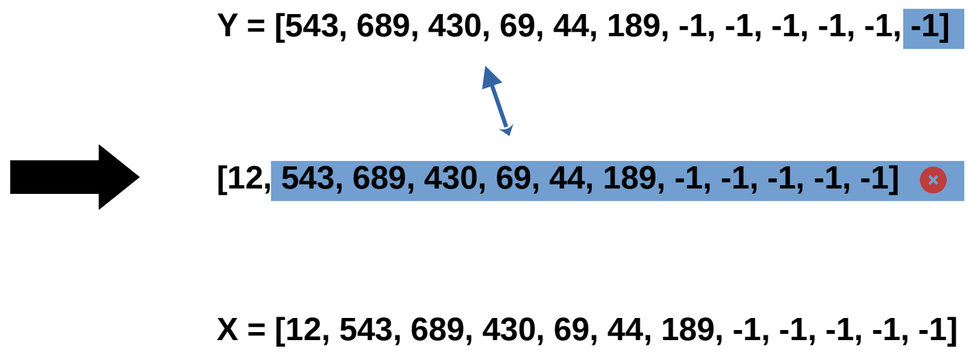
X = [12, 543, 689, 430, 69, 44, 189, -1, -1, -1, -1, -1]

Fine-tuning – X & Y

- Each pair (x, y) is formed like the first stage.
- Repeat the process & create the data loaders.

<|startoftext|>User**<|separator|>**Hi, how are you?**<|endoftext|>**

<|startoftext|>Assistant**<|separator|>**Good, is there anything I can help with?**<|endoftext|>**

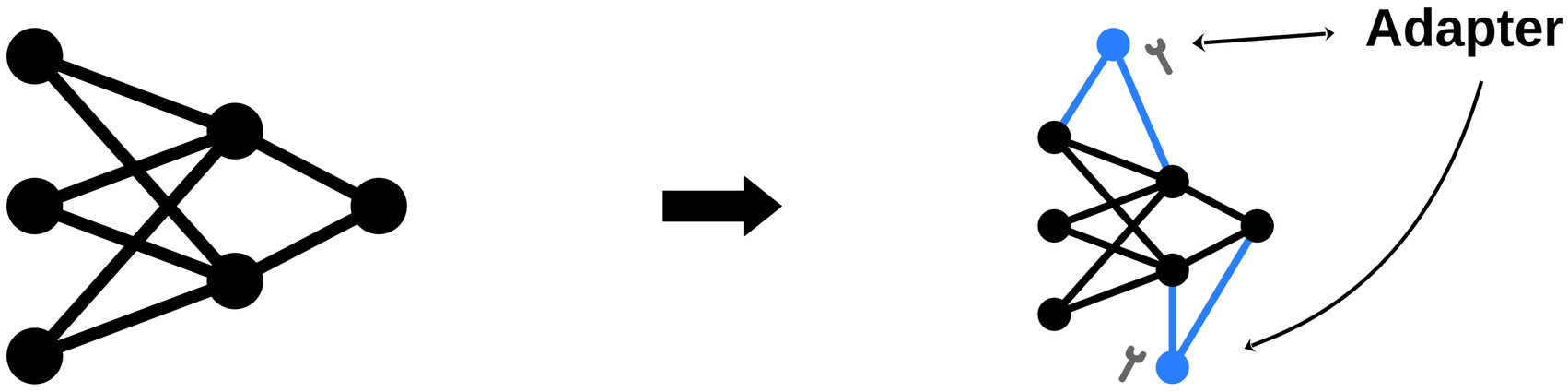


Let’s code

Low Rank Adaptation (LoRA, QLoRA)

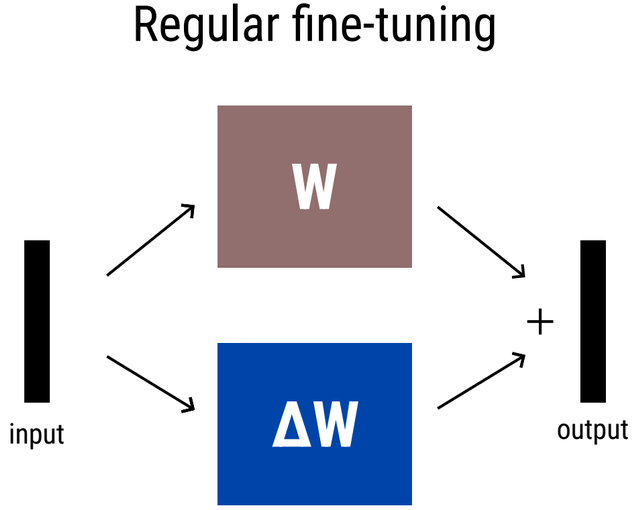
Low Rank Adaptation

- This is a **form of fine-tuning**.
- It is **efficient** and **trains few parameters**.
- It generates **adapters** that can be **attached to the base model**.



Low Rank Adaptation

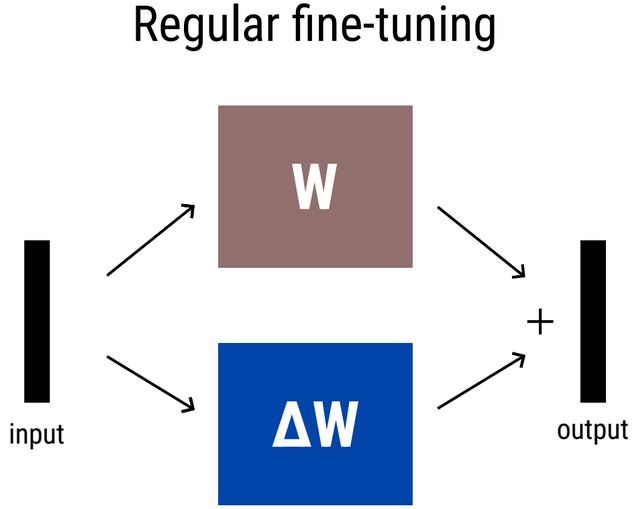
- In **instruction fine-tuning** we update **all model weights**.
- During backpropagation, **we learn a ΔW matrix**.
- If W is a 2048x2048 matrix, then **ΔW has 4,194,304 parameters**.

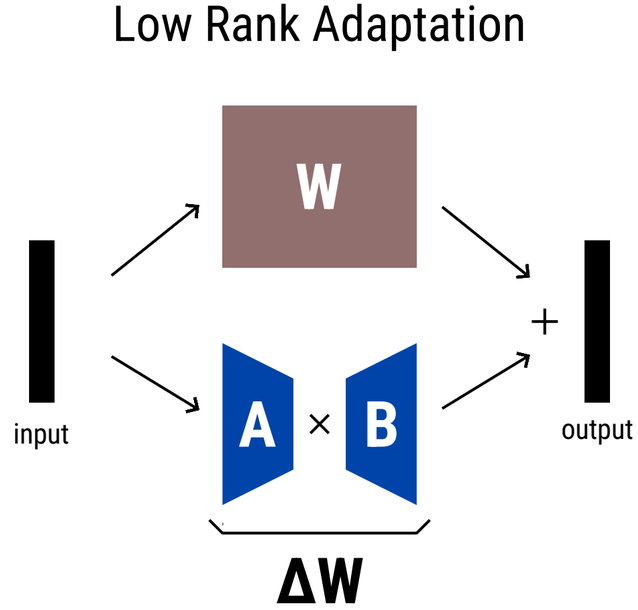


$$W_{new} = W + \Delta W$$

Low Rank Adaptation

- The **LoRA** method aims to **approximate ΔW as AB**.
- Matrix A has dimensions n × r, while Matrix B has dimensions r × m.
- **r represents the rank** and determines the number of **additional parameters**.
- If r = 2 and W is a 2048 × 2048 matrix, the number of parameters to train would be 2 × 2 × 2048 = **8,192**.





Let’s code

Let’s scale

Let’s scale - data

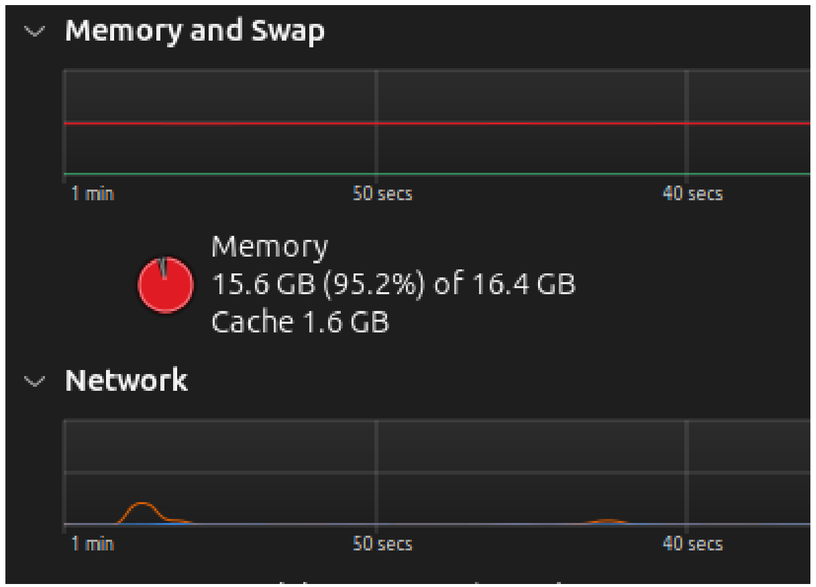
- We started with a small example.
- I will use the AtlaSet corpus.
- This is **600 times** bigger.
- It contains **1.17M rows** in the train set and **2.68K rows** in the test set.

1.5M characters

~900M characters

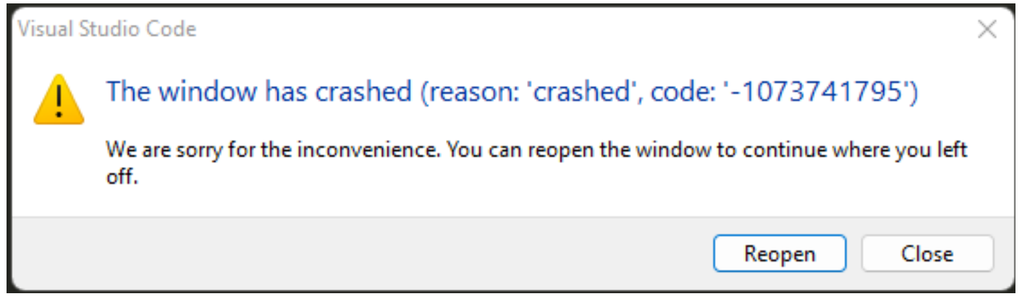
Let’s scale - challenges

- We started with a small example, **we didn’t need a lot of resources**.
- But, **trying to load** the big dataset will **require a lot of RAM**.



Let’s scale - challenges

- We started with a small example, **we didn’t need a lot of resources**.
- But, **trying to load** the big dataset will **require a lot of RAM**.
- **Training the tokenizer** on 900M characters **will crash your program**.



Let’s scale - challenges

- We started with a small example, **we didn’t need a lot of resources**.
- But, **trying to load** the big dataset will **require a lot of RAM**.
- **Training the tokenizer** on 900M characters **will crash your program**.
  - Train on a **sample data (1-5%)** to **estimate** the data distribution.
  - **Training might take hours**, in my case it took **6h.**

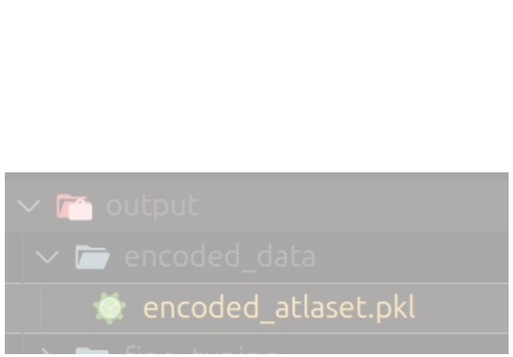
Let’s scale – file formats

- After **encoding the text data**, you need to **save it to disk** so that you don’t repeat this operation.
- What **file format** do we need to use?
- Maybe **.pkl** ?

NO

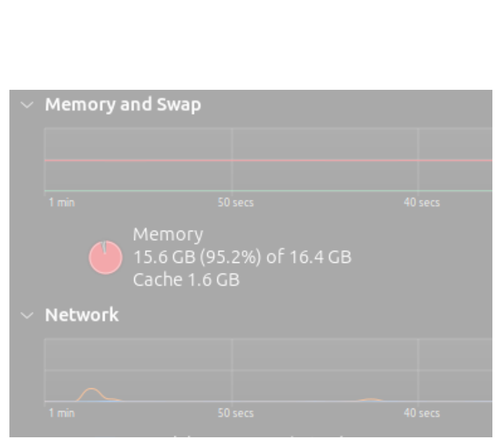
Let’s scale – file formats

- After **encoding the text data**, you need to **save it to disk** so that you don’t repeat this operation.
- What **file format** do we need to use?
- Maybe **.pkl** ?



Stored data

Load data



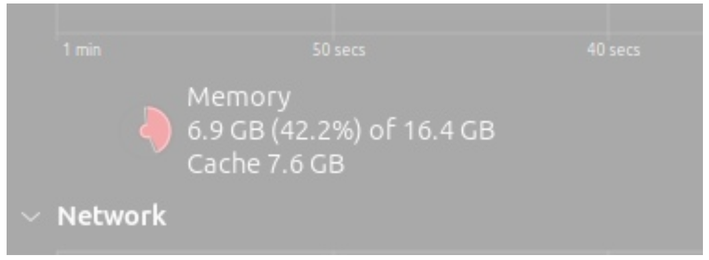
OOM

Let’s scale – file formats

- After **encoding the text data**, you need to **save it to disk** so that you don’t repeat this operation.
- What **file format** do we need to use?
- Maybe **.pkl** ? **NO**.
- We use **Memory-mapped file objects.**

Stored data

Load portions
of the data



RAM is happy :)

Let’s scale - training

- Training will take **forever** to finish


```
                optimizer.ze
16.6s

Processing:   0%|                                  | 6984:13:51,  4.06s/it]
```

Let’s scale - training

- Training will take **forever** to finish.
- Rent a GPU and **do the training on the cloud**.

Lambda

kaggle

Let’s code

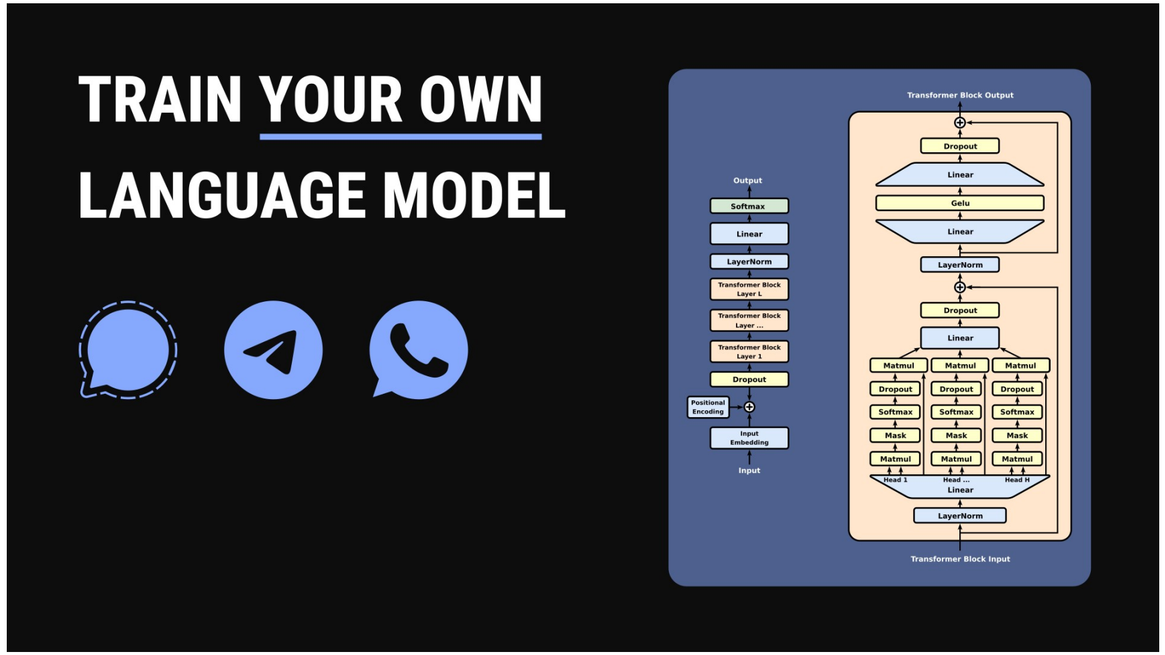
END OF COURSE 1

The Transformer Journey
2017 to 2025

Introduction

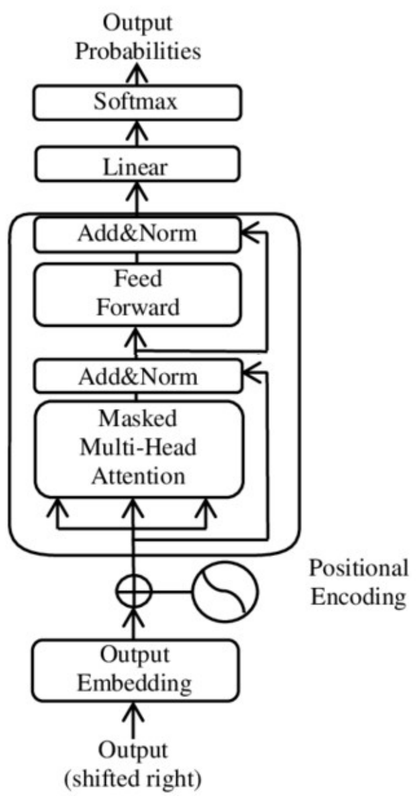
Introduction

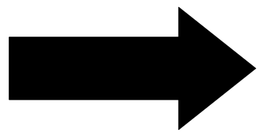
- In the previous course, I used the techniques from the 2017 paper **Attention is All You Need**.

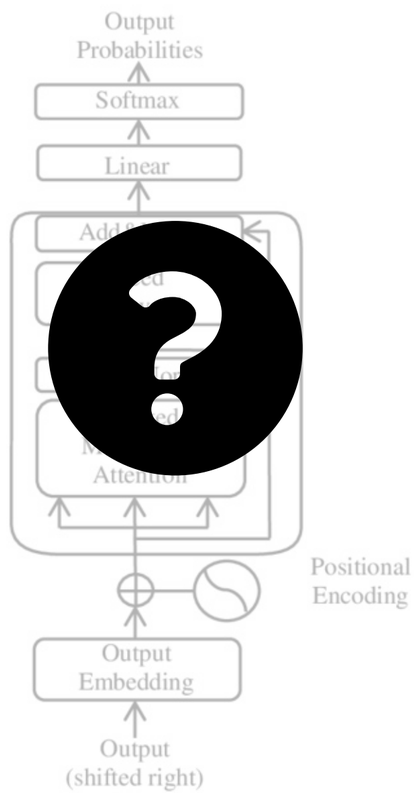


Introduction

- In the previous course, I used the techniques from the 2017 paper **Attention is All You Need**.
- Over the past 7 years, researchers have improved original transformer architecture a lot.
- In this course, I will explain to you those improvements.



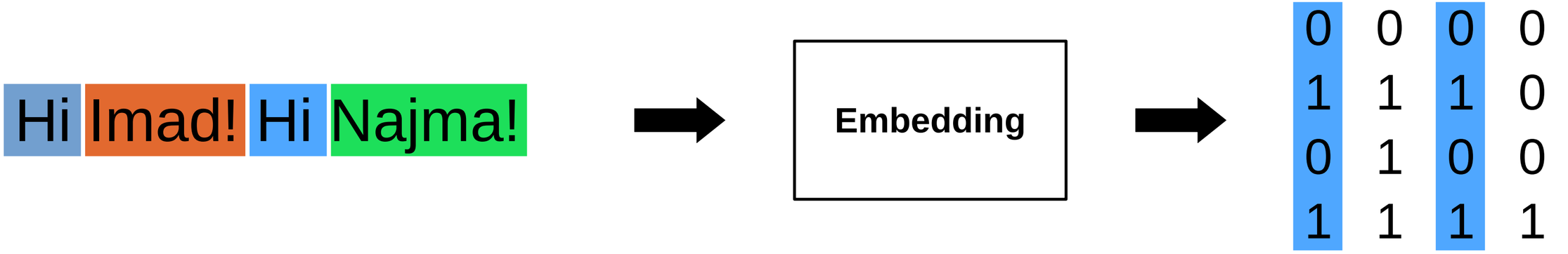




Positional encoding

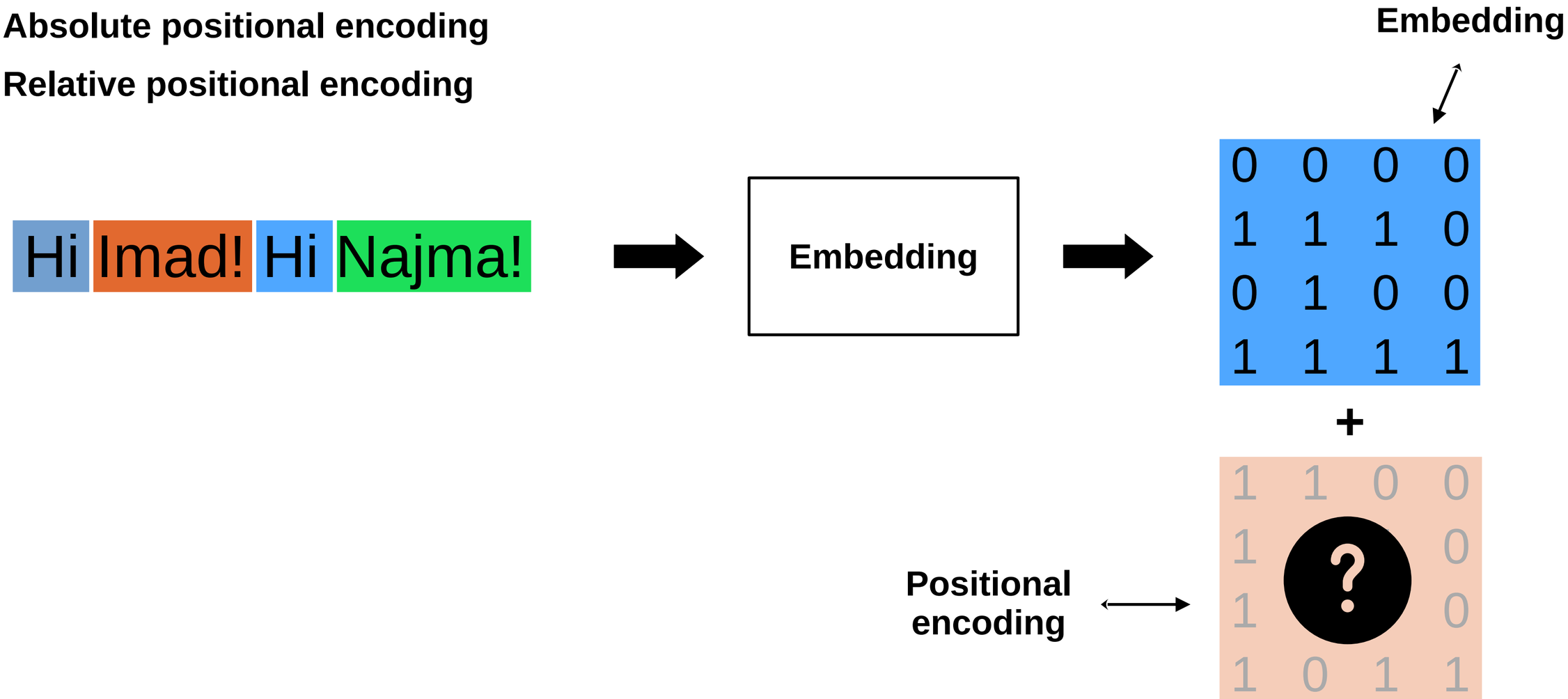
Positional encoding

- Why do we need positional encoding in the first place?
- Without positional encoding the **output is identical for the same token in different positions.**
- The **self attention** layer needs to determine **relationships between words encoded by their positions.**
- How do we achieve that?



Positional encoding

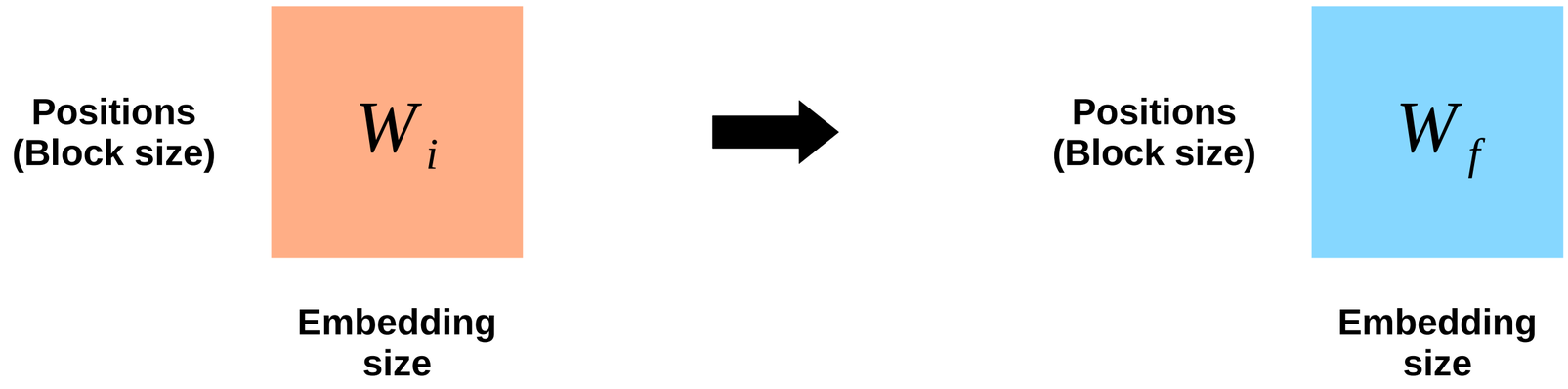
- We have many techniques to apply positional encoding:
  - **Absolute positional encoding**
  - **Relative positional encoding**



Positional encoding

Absolute positional encoding:

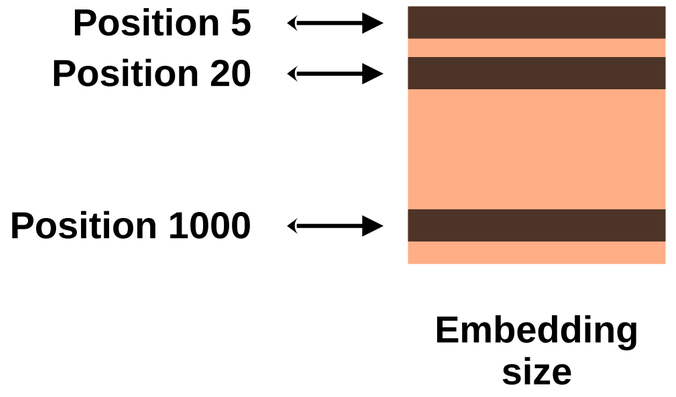
- **Learning from data:**
  - Consider position encodings as **trainable parameters**.
  - The model **learns the best position vector** for each position **during training**.
  - It is **limited** to representing sequences that are not longer than the block size.



Positional encoding

Absolute positional encoding:

- **Learning from data:**
  - Consider position encodings as **trainable parameters**.
  - The model **learns the best position vector** for each position **during training**.
  - It is **limited** to representing sequences that are not longer than the block size.
  - The embedding vectors are **independent of each other**.



Positional encoding

Absolute positional encoding:

- **Sinusoidal:**
  - Uses **fixed sine and cosine waves** of **different frequencies** to create unique position vectors.
  - Used in the original Transformer paper.
  - **It's not learned**, can handle longer sequences than seen in training.

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

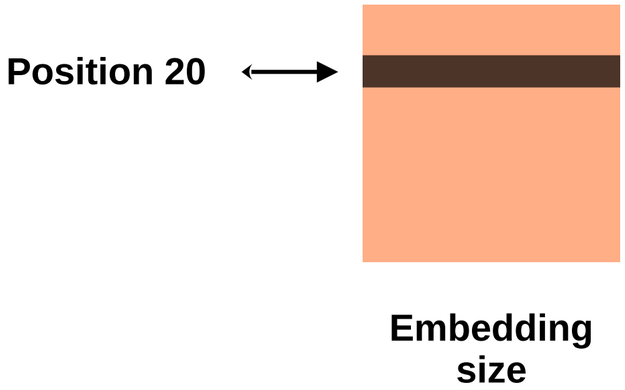
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

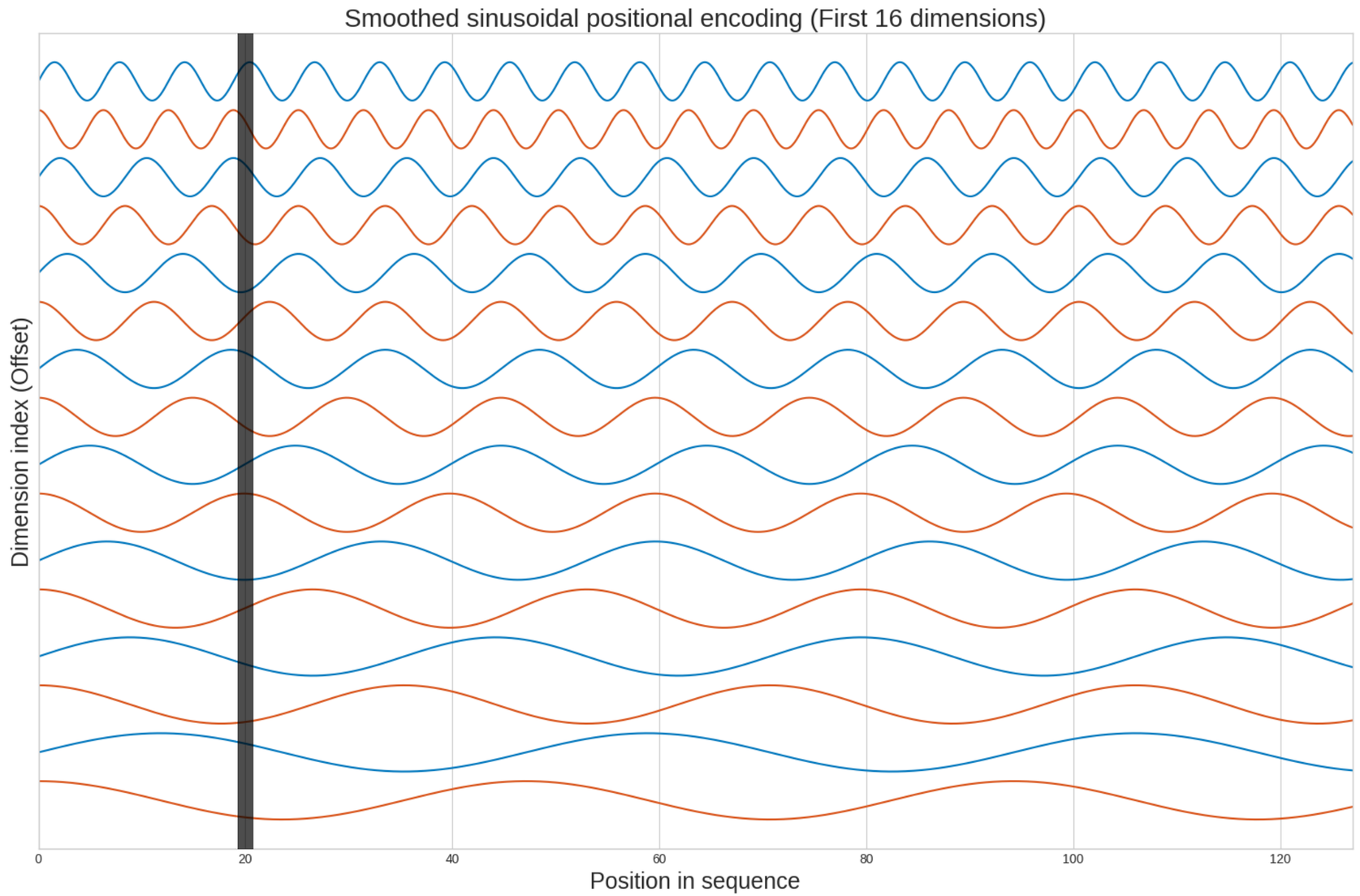
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Positional encoding



$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

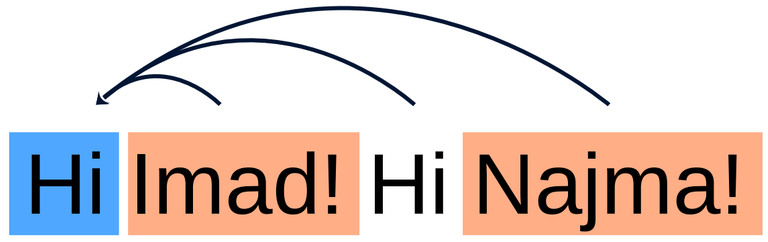
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

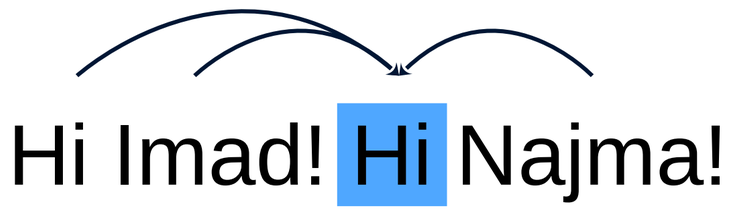


Positional encoding

Relative positional encoding:

- Focuses on the **relationship between words**, rather than just their absolute positions.
- This method **does not add a position vector** to the word embedding vector directly.
- It **alters the attention mechanism** to incorporate relative positional information.

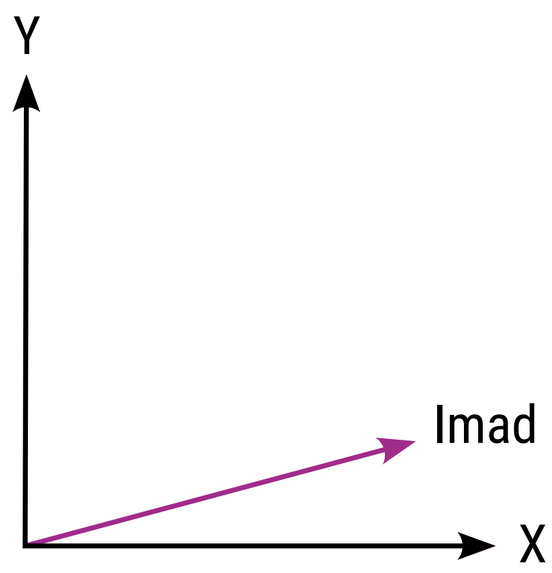


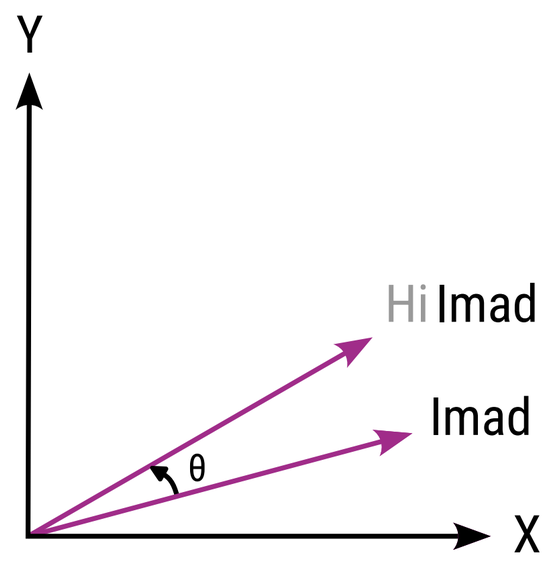


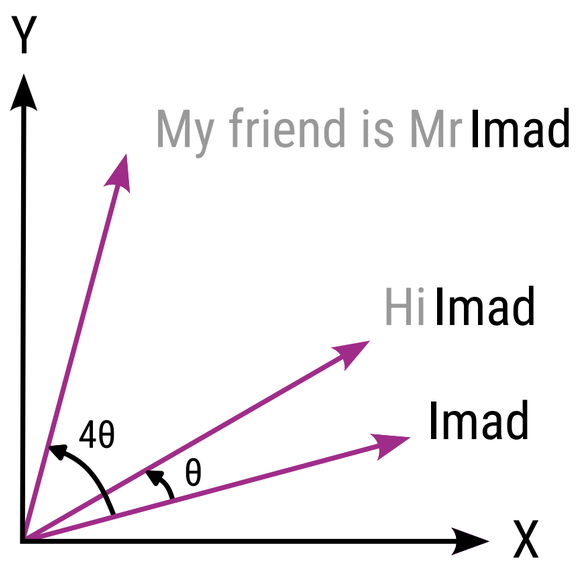
Positional encoding

Rotary Position Embedding (RoPE):

- Combines both absolute position of tokens and their relative distances.
- Each position in the sequence is represented by a **rotation in the embedding space**.
- θ is the rotation angle.



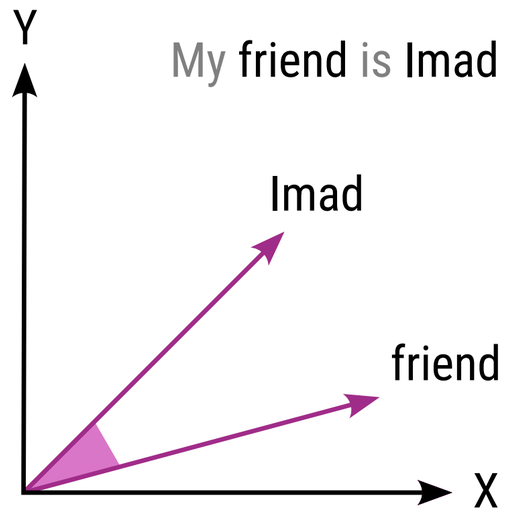


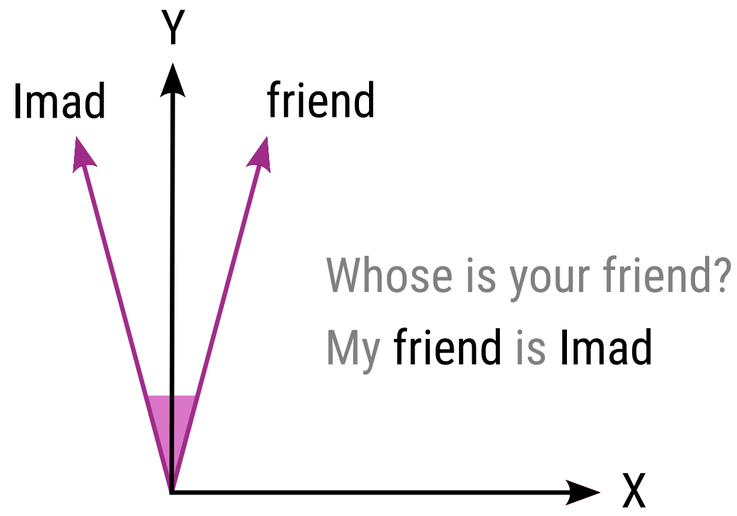


Positional encoding

Rotary Position Embedding (RoPE):

- Combines both absolute position of tokens and their relative distances.
- Each position in the sequence is represented by a **rotation in the embedding space**.
- θ is the rotation angle.
- Relative position of words is preserved.

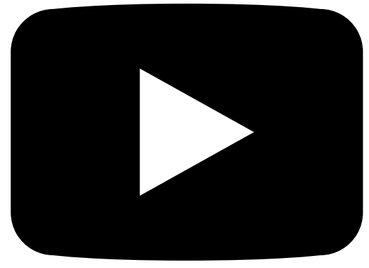




Positional encoding

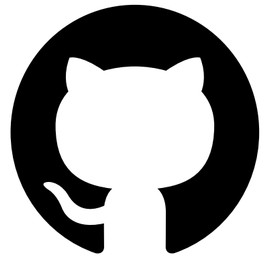
Resources

Jake Tae (Relative encoding)



Efficient NLP (RoPE)

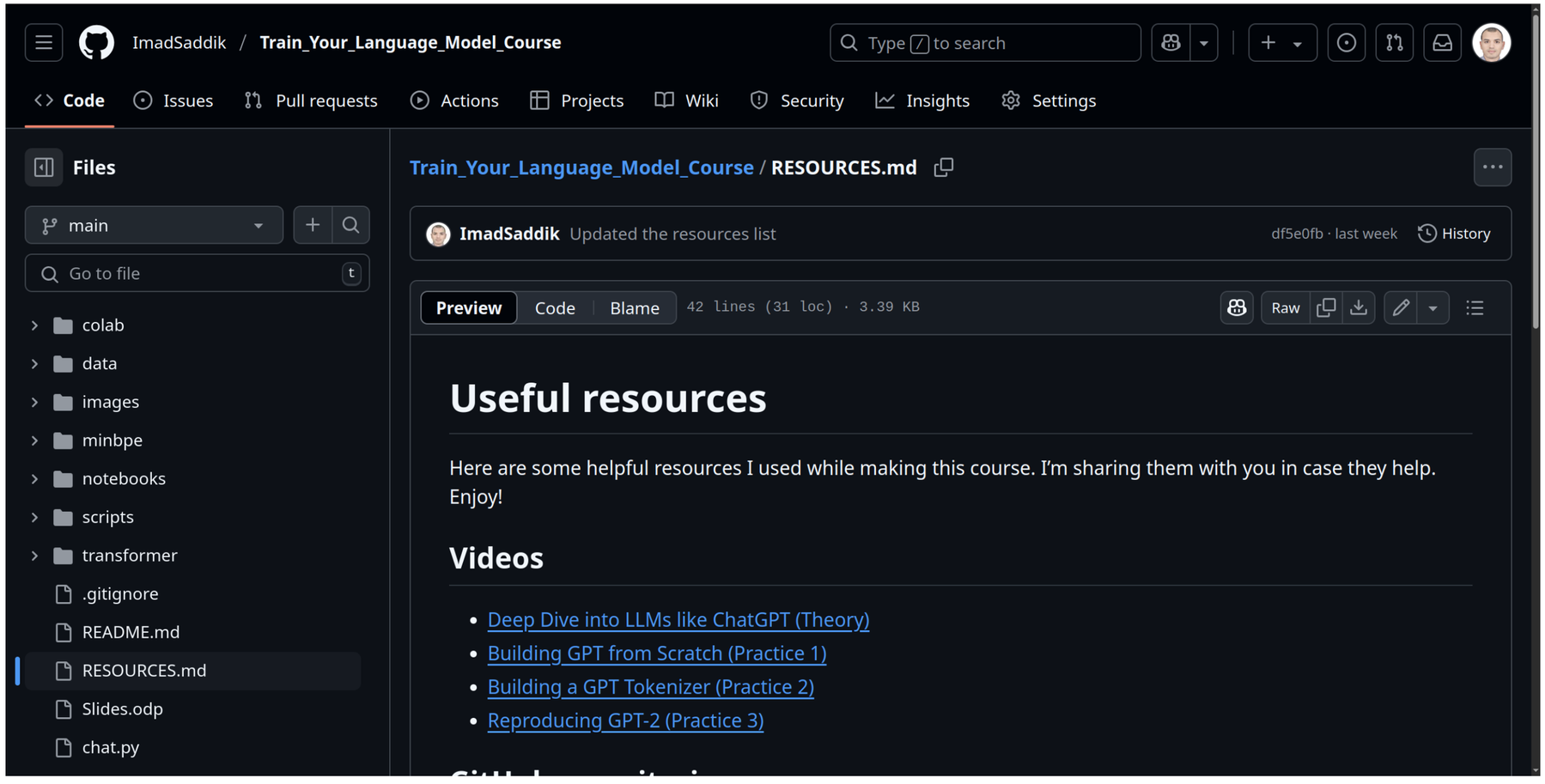
Christopher Fleetwood



My GitHub repository

Positional encoding

Resources



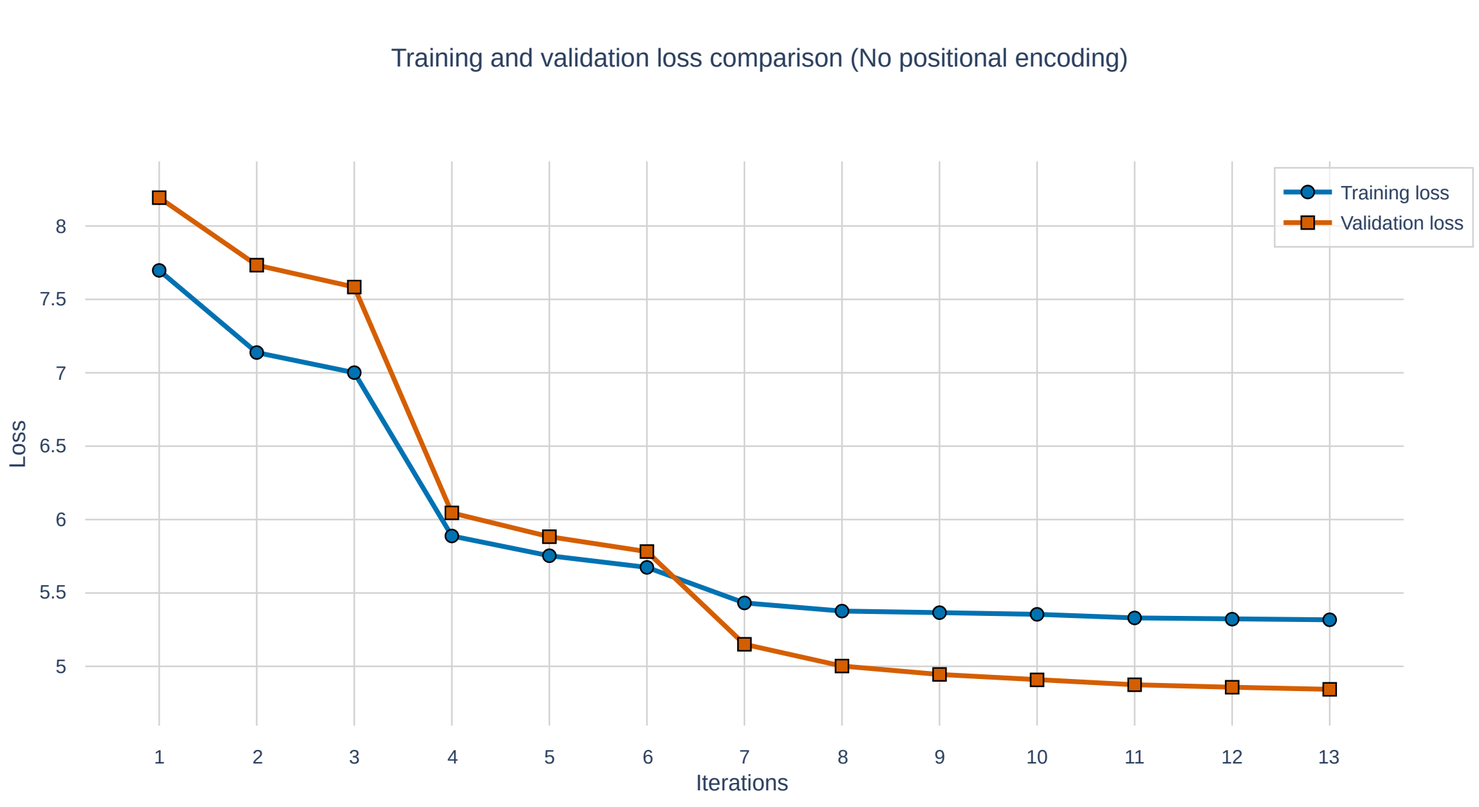
Positional encoding – The experiment

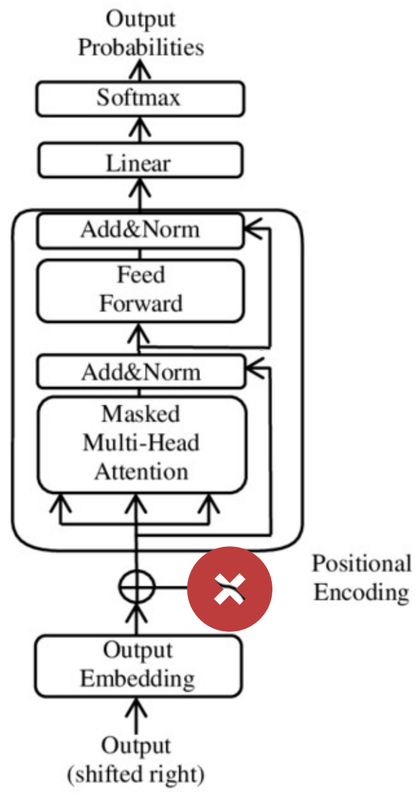
- Create a **small model** and train it for **1 epoch** on **AtlaSet**.
- Do that for every method:
  - **No positional encoding.**
  - **Absolute positional encoding.**
  - **Relative positional encoding.**
  - **Sinusoidal positional encoding.**
  - **Rotary positional encoding.**
- Store the training and validation losses at the end of training.

Positional encoding – The experiment

No positional encoding

- **Don’t add** positional encoding to the embedding tensor.
- **Training time ~ 2h:10m**.

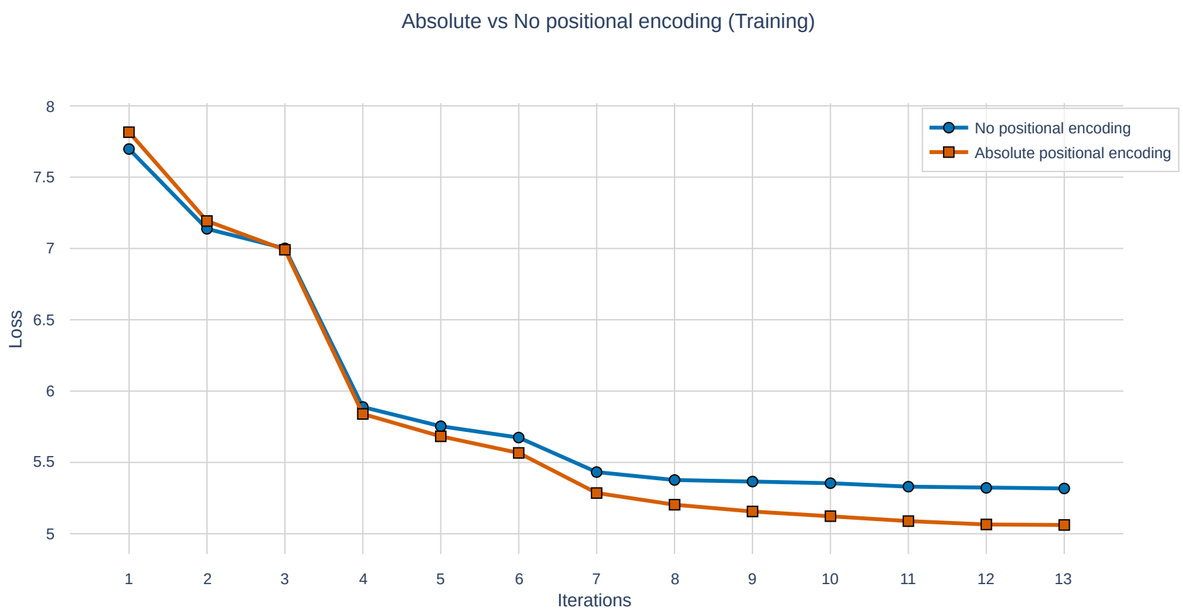


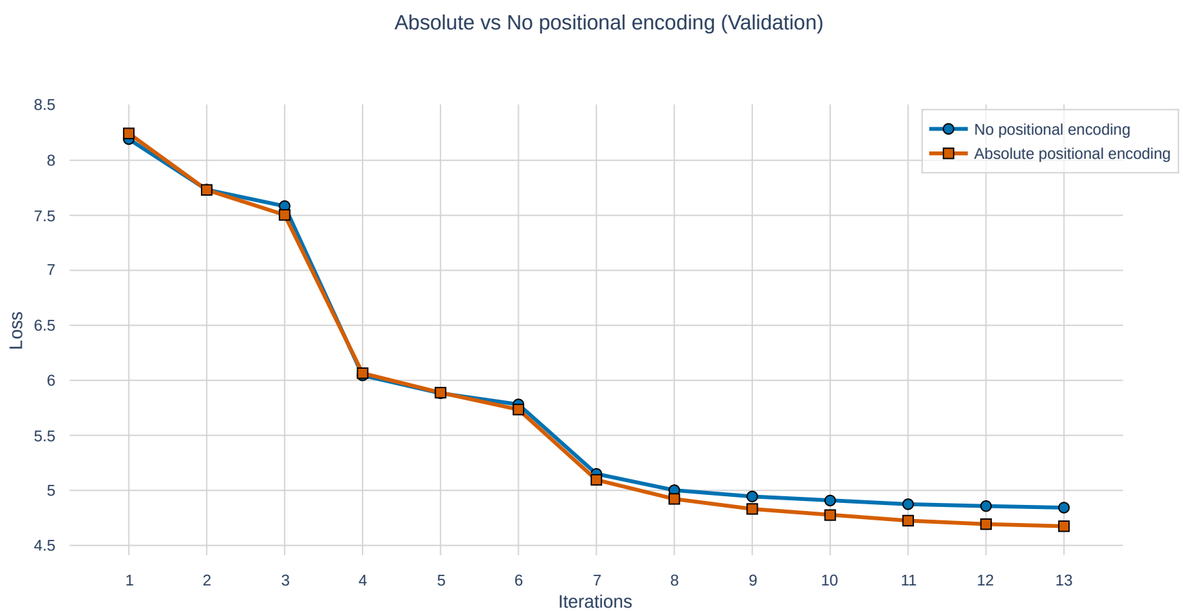


Positional encoding – The experiment

Absolute positional encoding (learnable)

- No change is needed.
- **Training time ~ 2h:10m**.

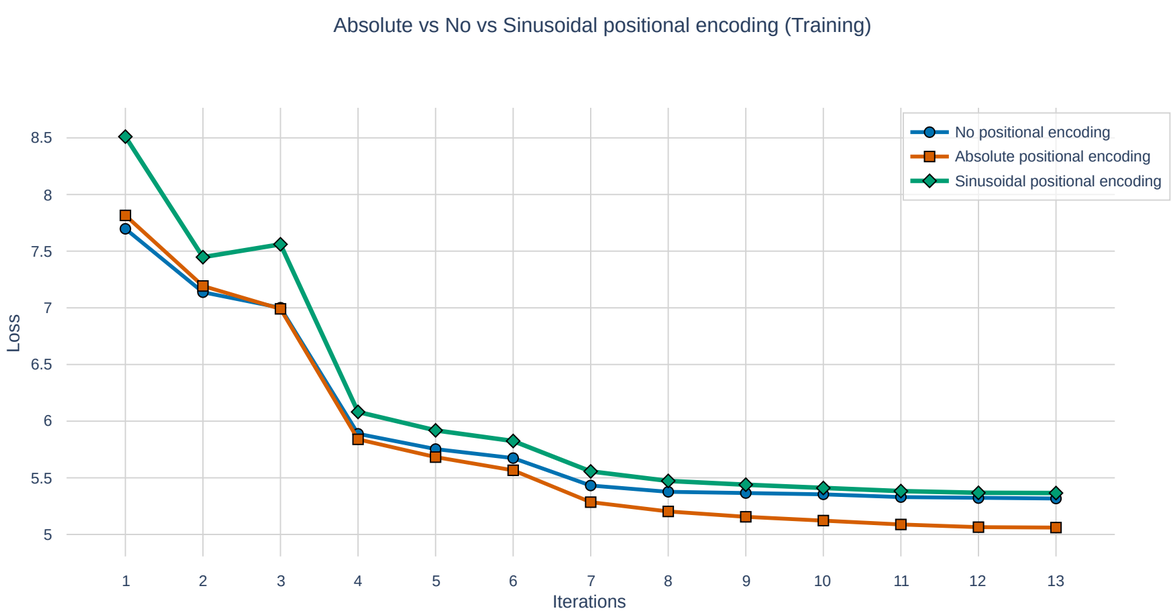


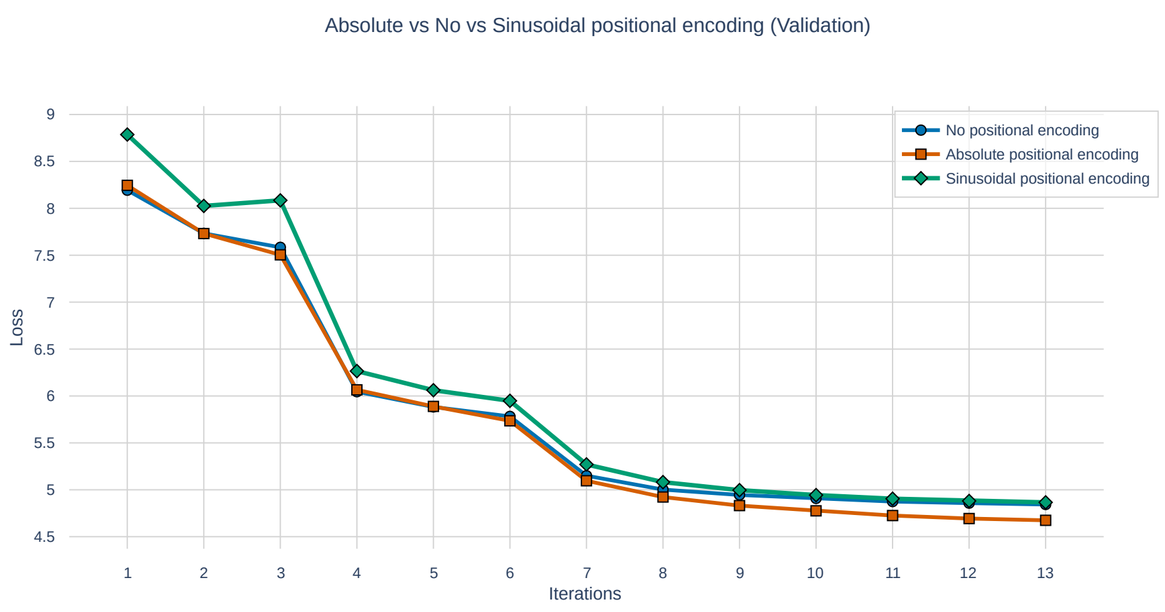


Positional encoding – The experiment

Absolute positional encoding (sinusoidal)

- No learnable parameters are needed.
- **Training time ~ 2h:10m**.

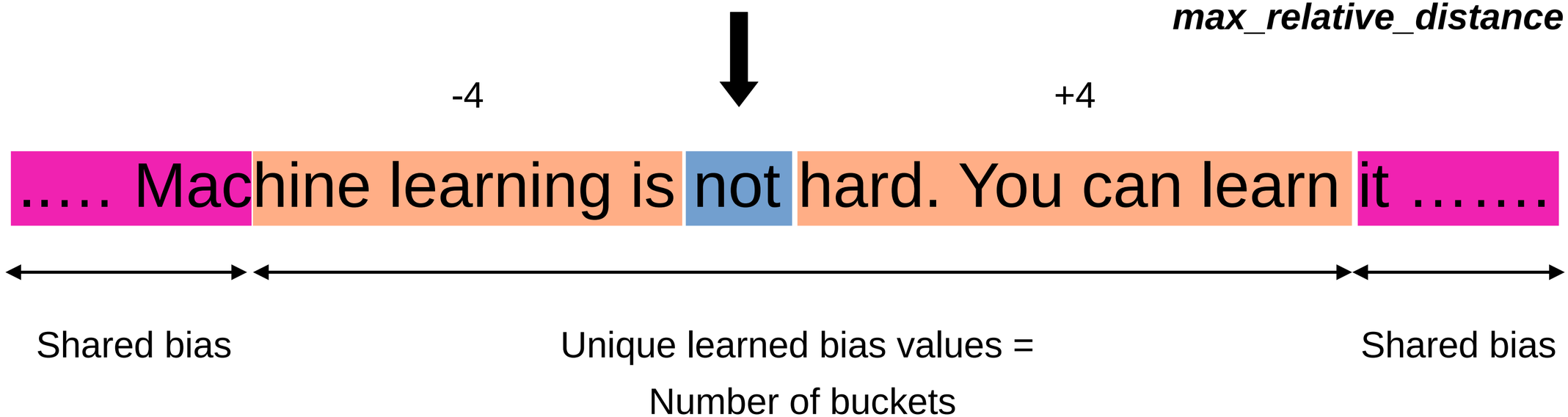




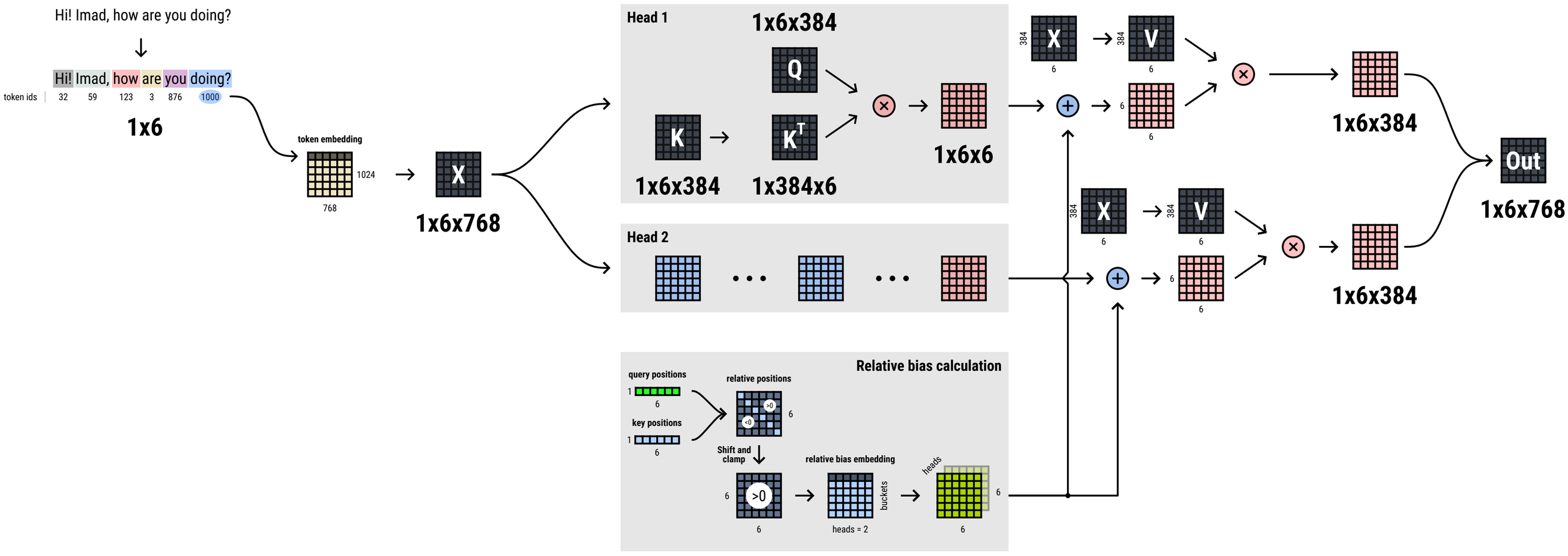
Positional encoding – The experiment

Relative positional encoding (learnable)

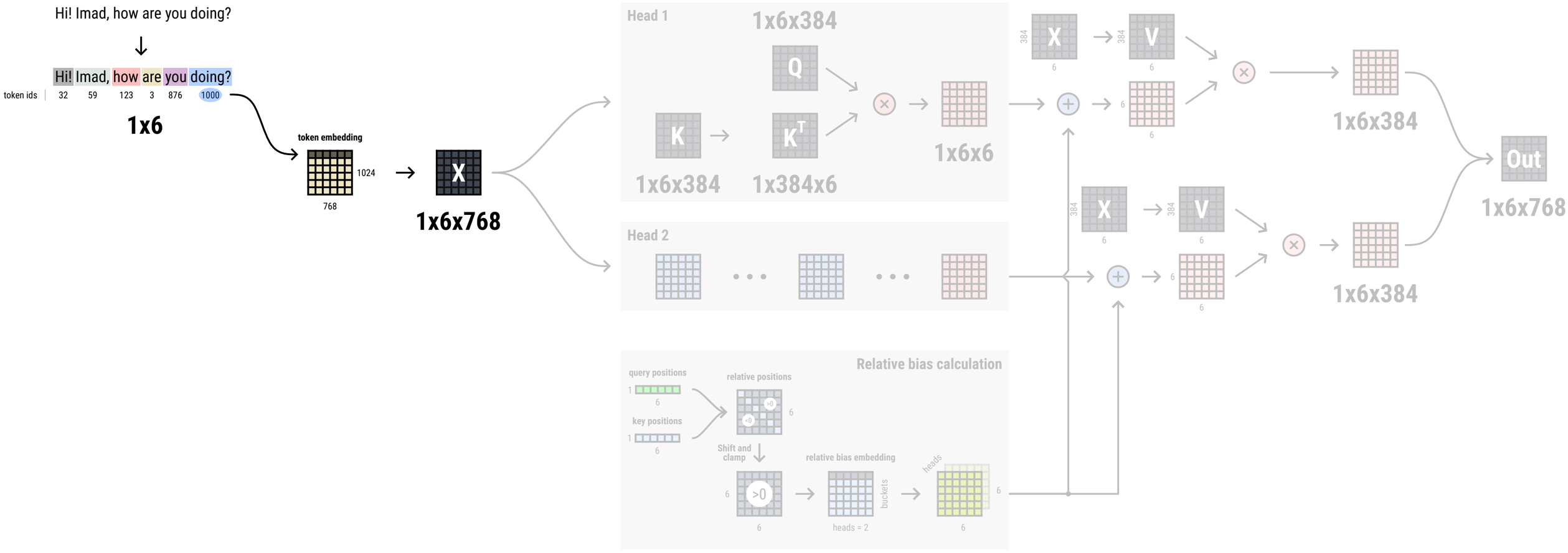
- Learnable parameters are present in this method.
- Works by defining the **number of buckets** for relative distances (***max_relative_distance***).



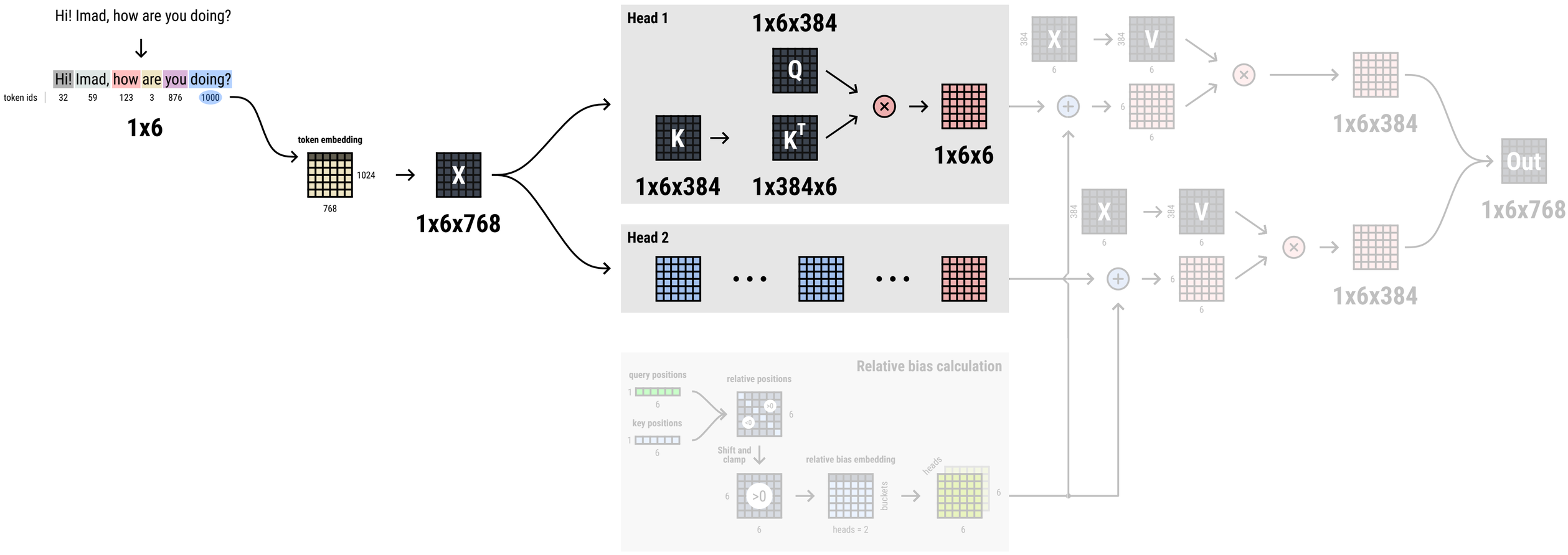
Positional encoding – The experiment



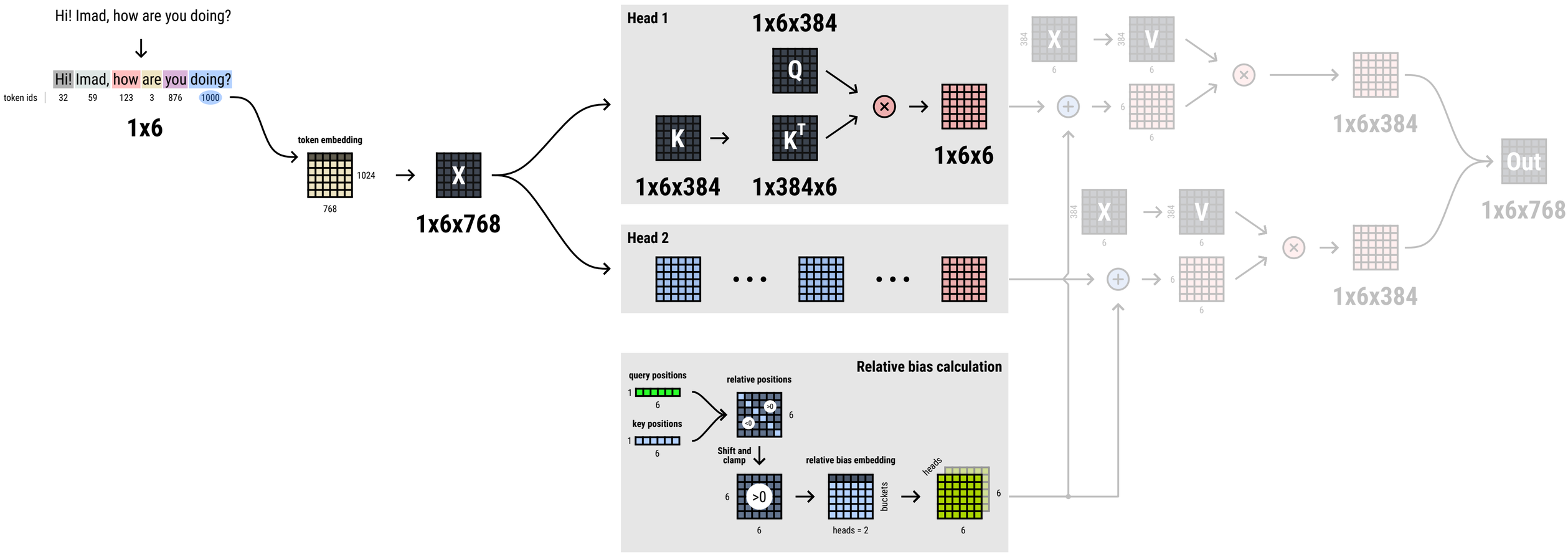
Positional encoding – The experiment



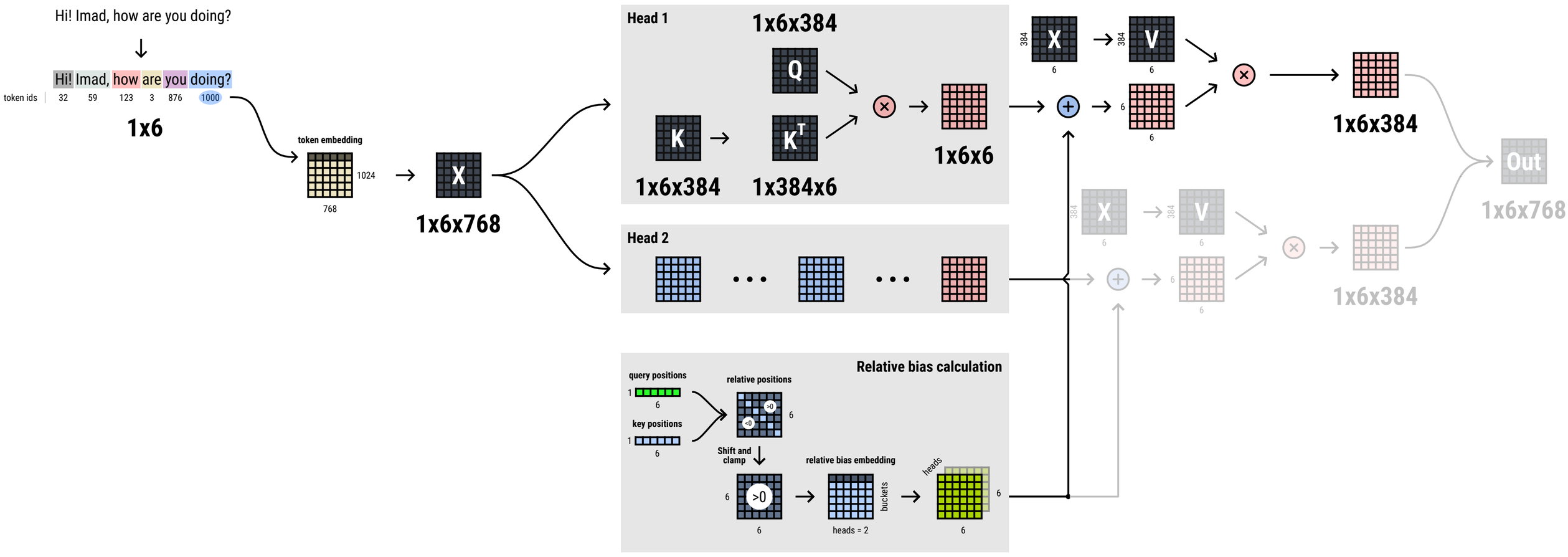
Positional encoding – The experiment



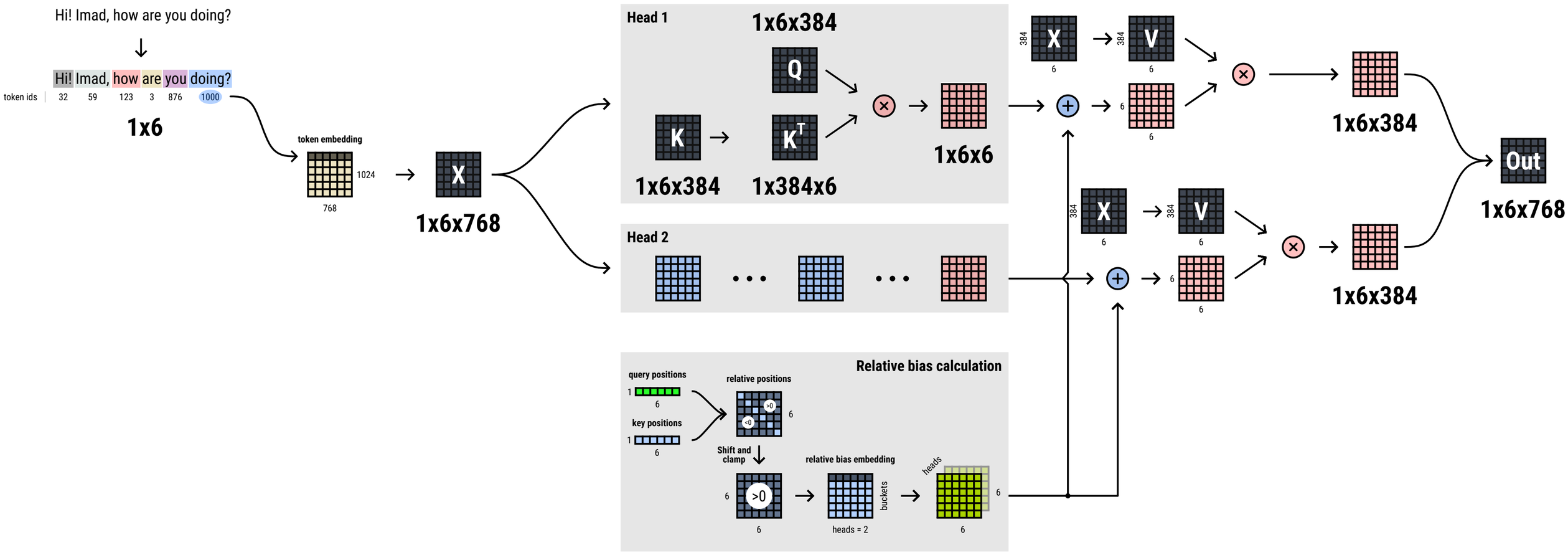
Positional encoding – The experiment



Positional encoding – The experiment



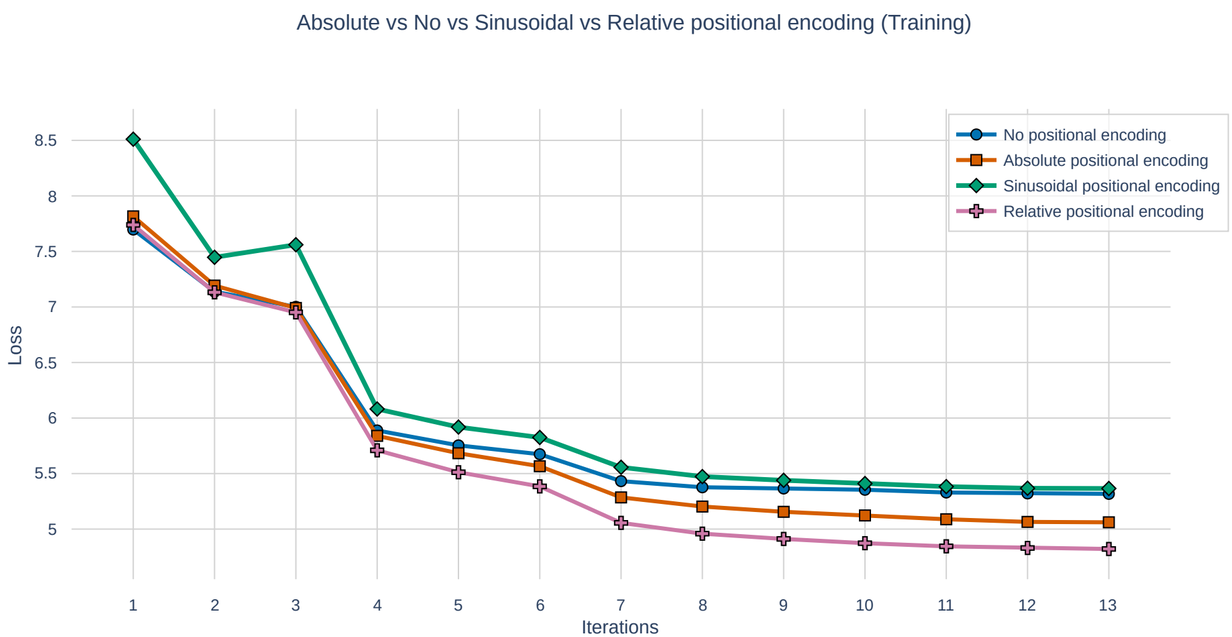
Positional encoding – The experiment

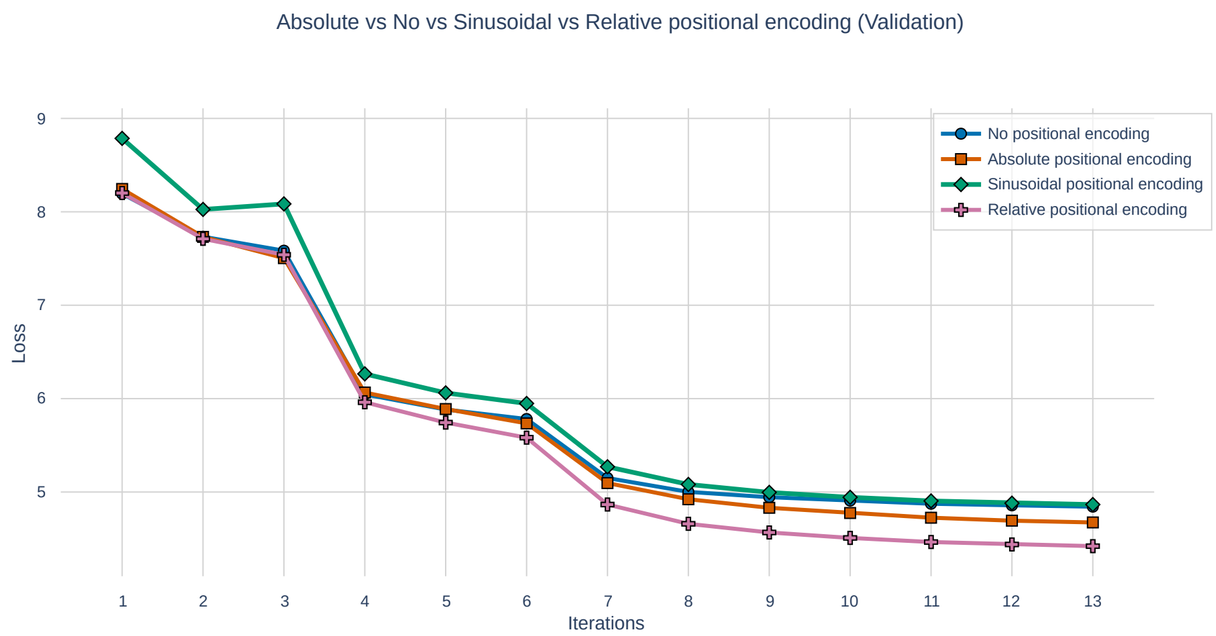


Positional encoding – The experiment

Relative positional encoding (learnable)

- **Training time ~ 5h**.

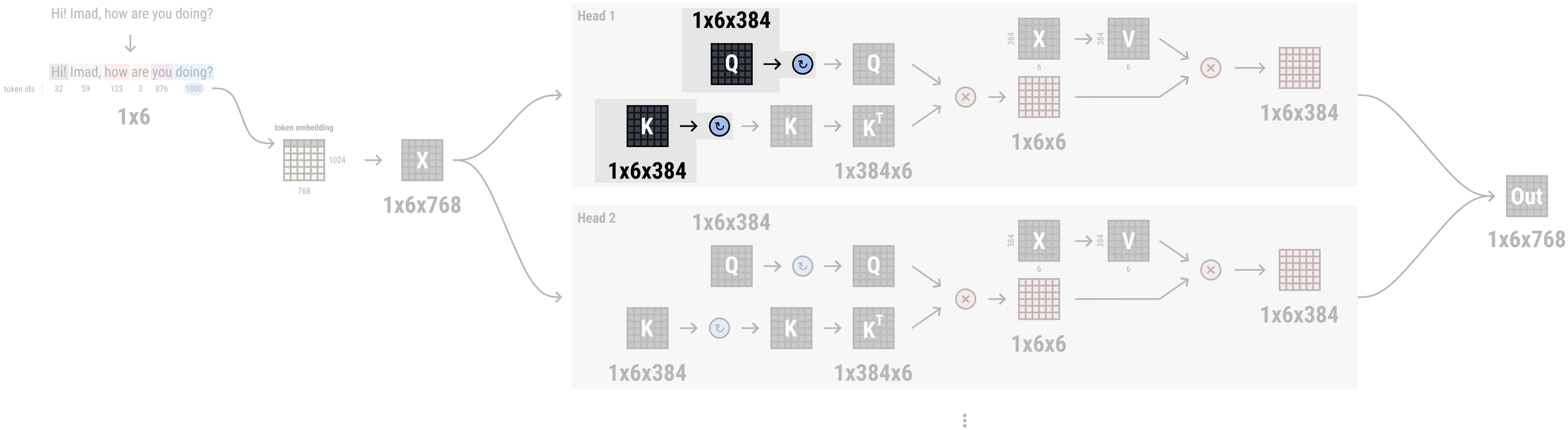




Positional encoding – The experiment

Rotary positional encoding (RoPE)

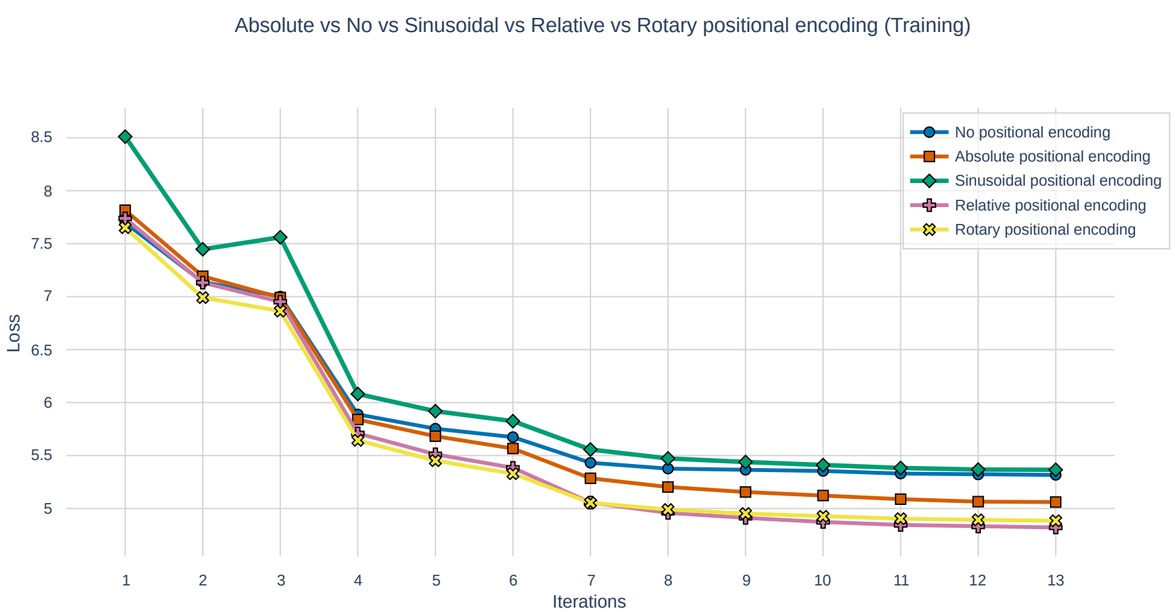
- No learnable parameters are needed.

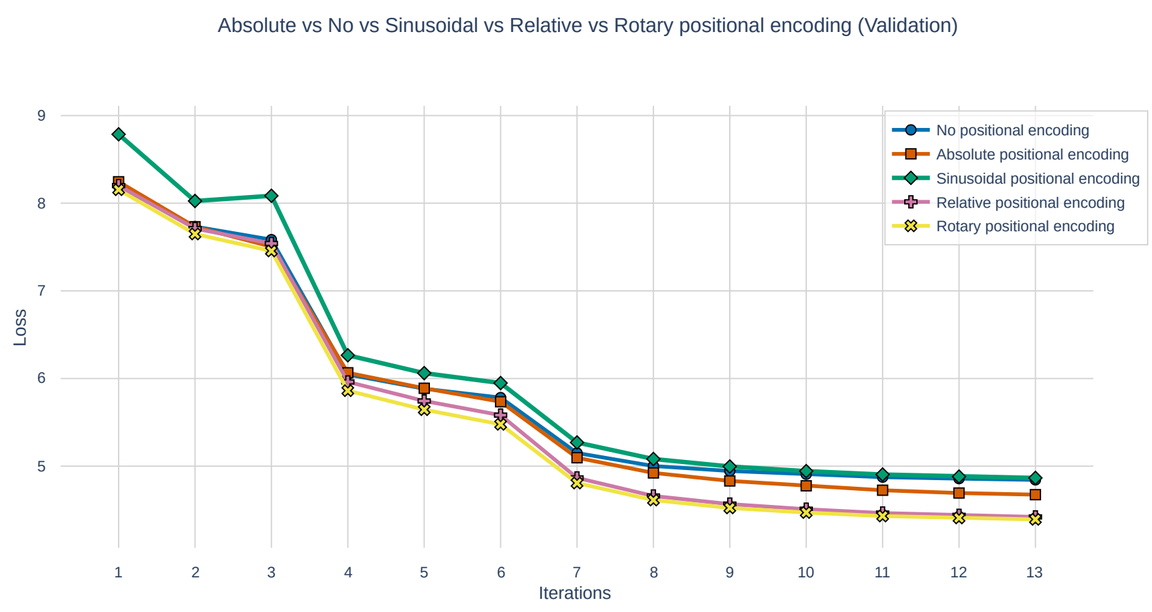


Positional encoding – The experiment

Rotary positional encoding (RoPE)

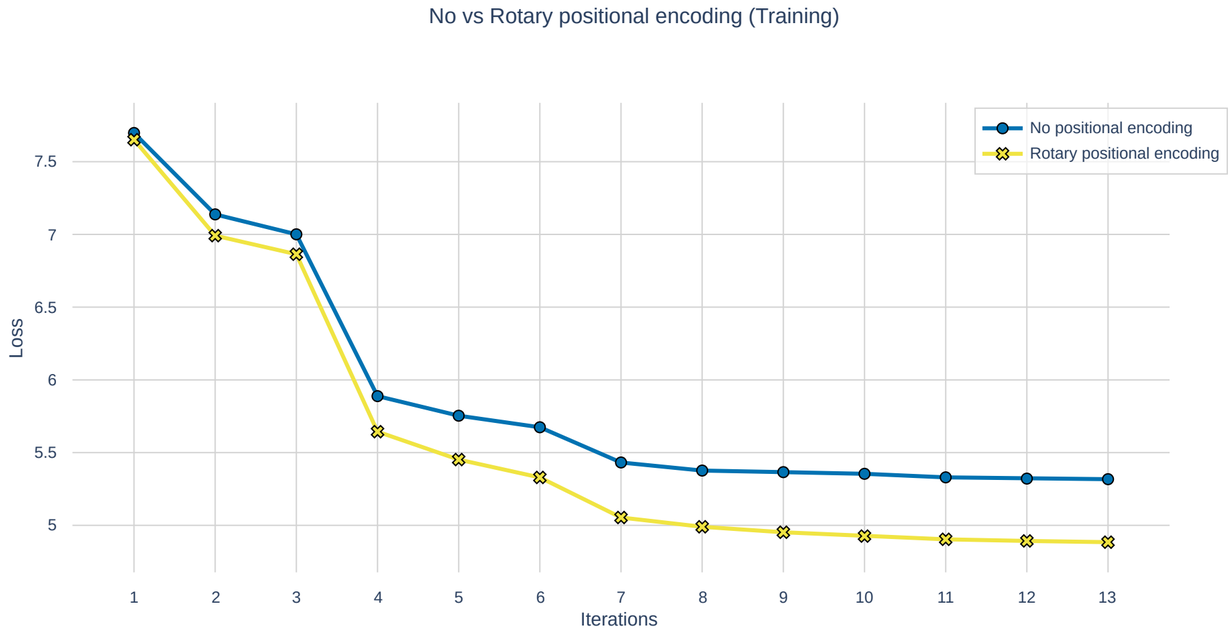
- No learnable parameters are needed.
- **Training time ~ 2h**.

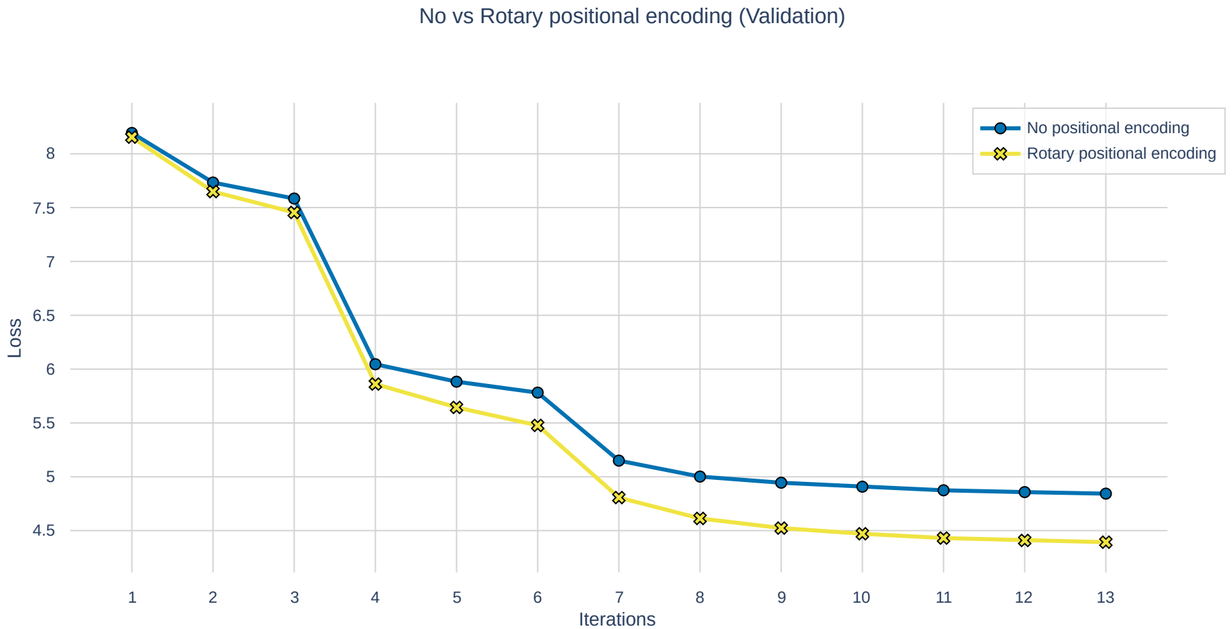




Positional encoding – Conclusion

Positional encoding definitely helps the model learn better.





Attention layer

Attention layer

- **Attention** helps the model **focus on the most relevant** pieces of information in the input.
- We will compare the following methods:

Sparse attention (Big Bird)

Grouped Query Attention (GQA)

Local Attention

Multi Head Attention (MHA)

Linear Attention

Latent Attention

Attention layer – Multi Head Attention

- **MHA** was introduced in the **Attention Is All You Need** paper.

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

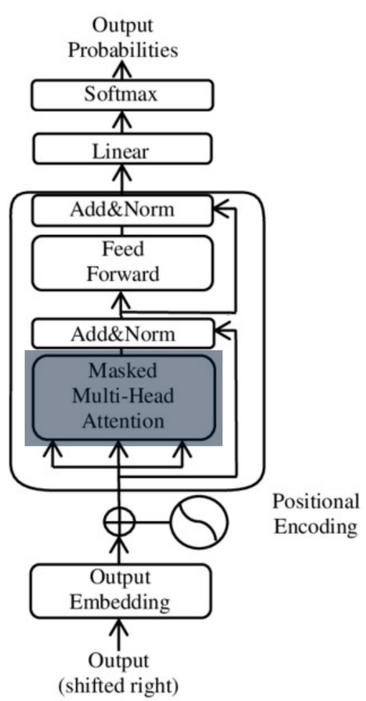
Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

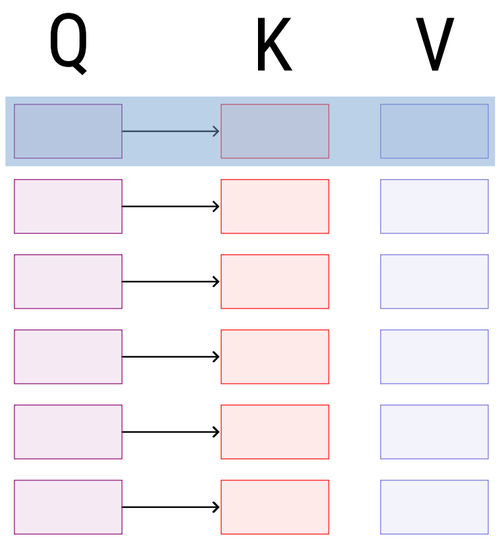
Attention layer – Multi Head Attention

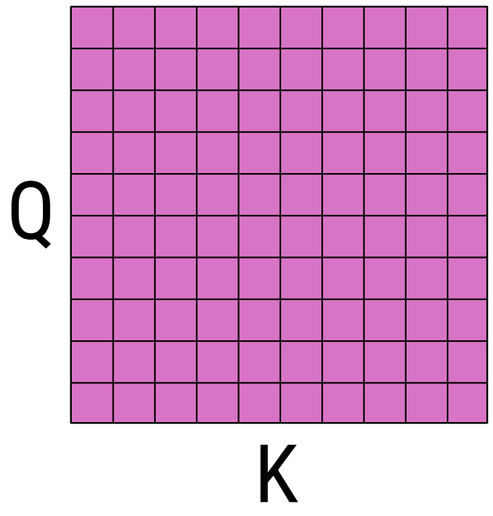
- **MHA** is the **foundational attention** mechanism from the **Attention Is All You Need** paper.
- MHA computes **attention scores** independently across multiple heads in **parallel**.
- It learns **diverse representations** by allowing different heads to focus on various input aspects.

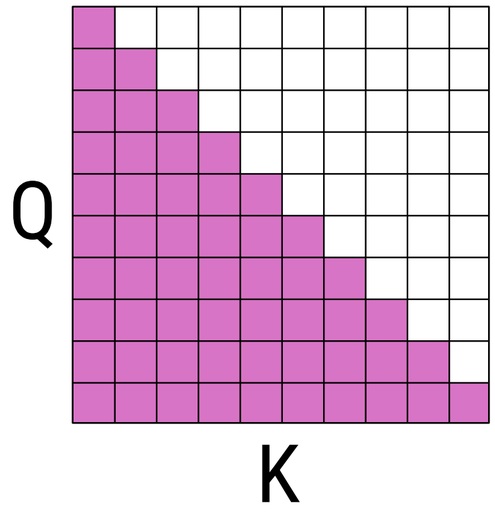


Attention layer – Multi Head Attention

- Within each attention head, **input is projected into query (Q), key (K), and value (V)** matrices.
- Each query vector is then compared with all key vectors in that head to measure similarity.
- This **comparison yields an attention score matrix**.
- In the decoder, we use a mask to prevent queries from attending to future keys. (**Masked attention**)

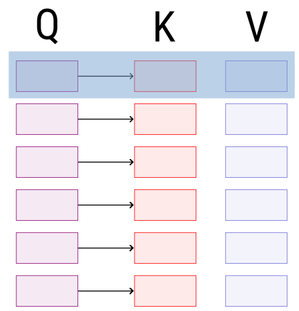


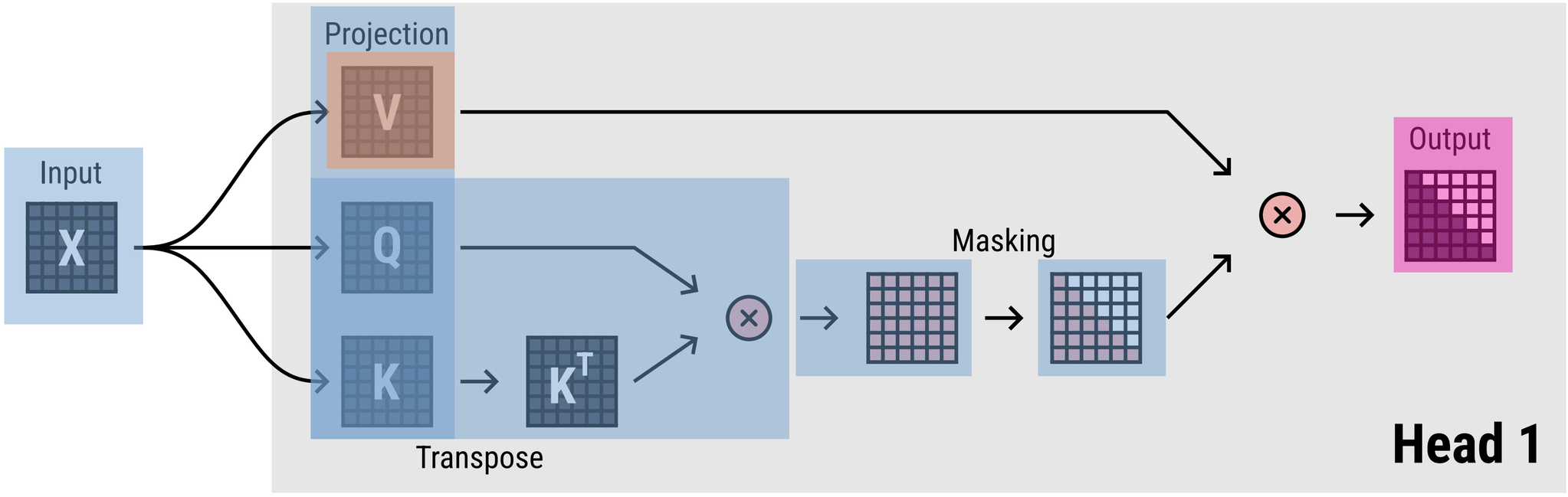




Attention layer – Multi Head Attention

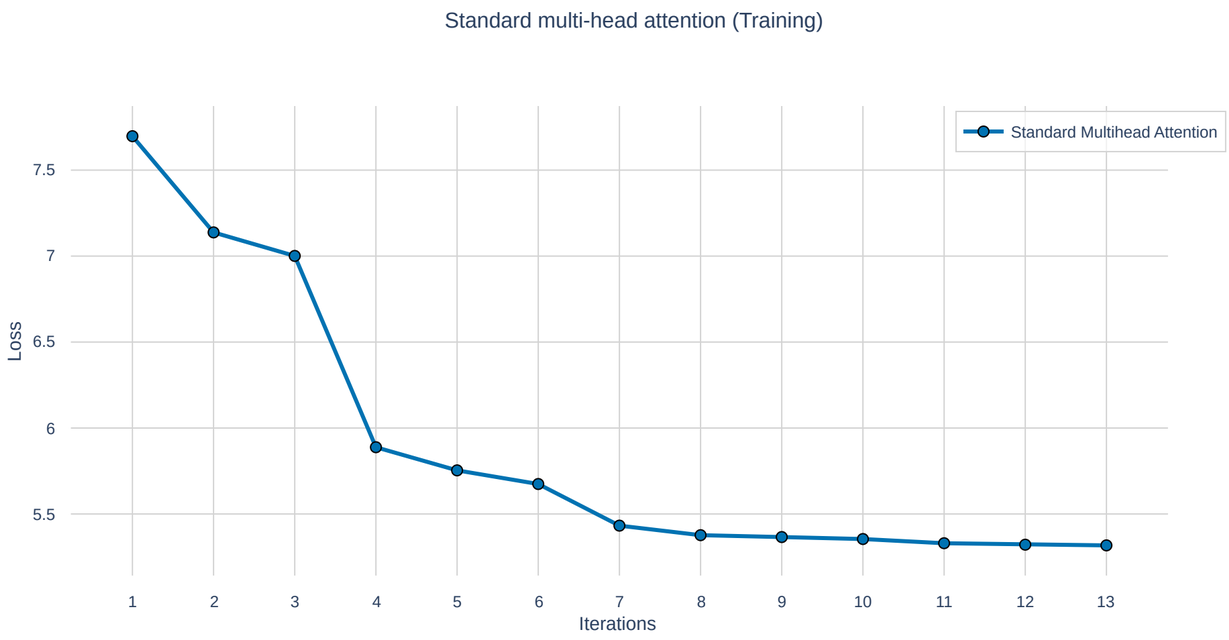
$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

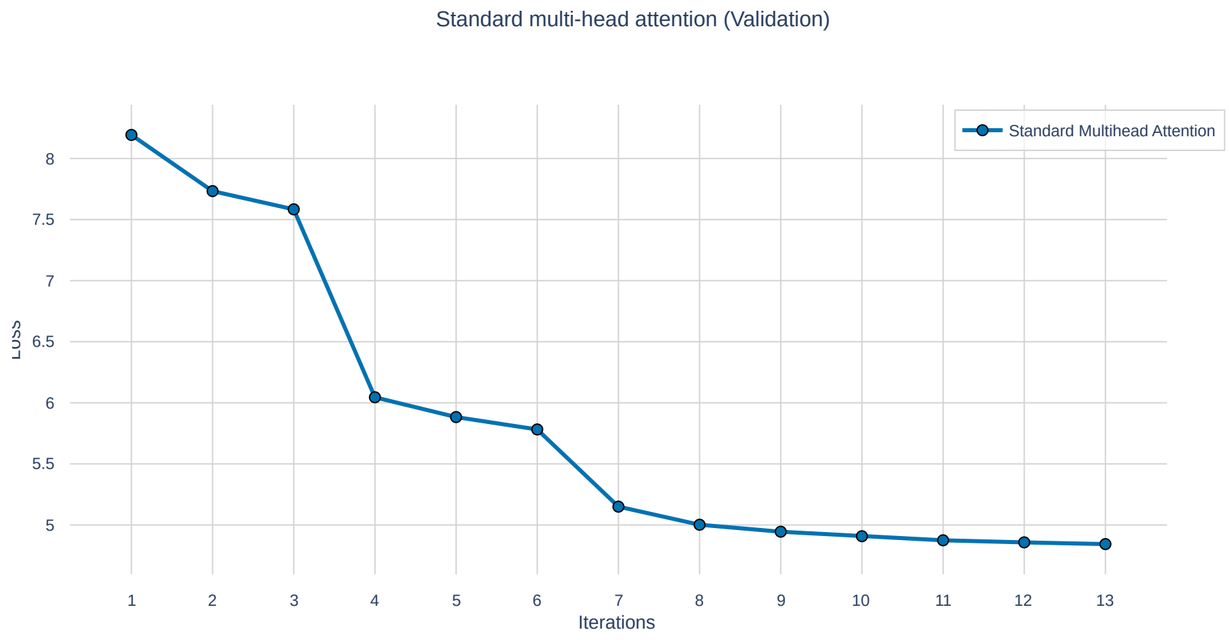




Attention layer – Multi Head Attention

Baseline = Performance of the model with **MHA**





Attention layer – Multi Query Attention

• **MQA** was introduced in the **Fast Transformer Decoding** paper.

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

Noam Shazeer
Google
noam@google.com

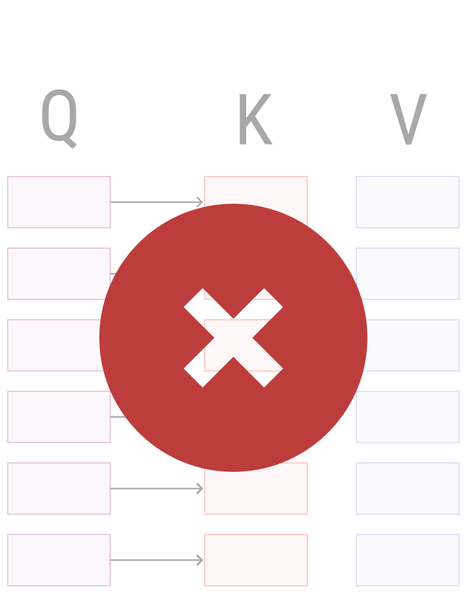
November 7, 2019

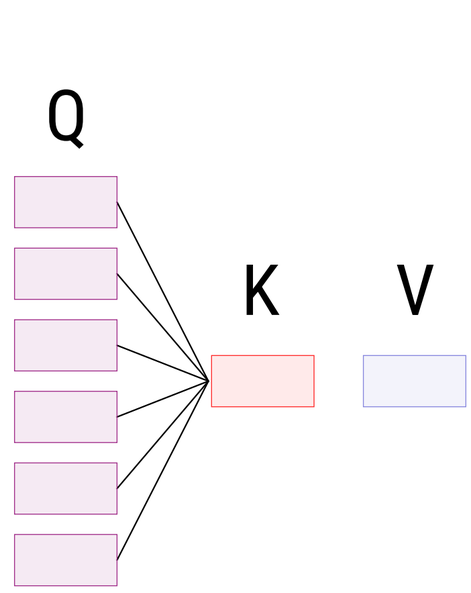
Abstract

Multi-head attention layers, as used in the Transformer neural sequence model, are a powerful alternative to RNNs for moving information across and between sequences. While training these layers is generally fast and simple, due to parallelizability across the length of the sequence, incremental inference (where such paralleization is impossible) is often slow, due to the memory-bandwidth cost of repeatedly loading the large "keys" and "values" tensors. We propose a variant called multi-query attention, where the keys and values are shared across all of the different attention "heads", greatly reducing the size of these tensors and hence the memory bandwidth requirements of incremental decoding. We verify experimentally that the resulting models can indeed be much faster to decode, and incur only minor quality degradation from the baseline.

Attention layer – Multi Query Attention

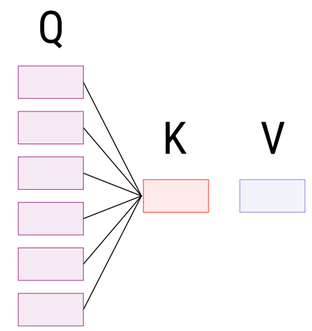
- **MQA** simplifies MHA to reduce memory usage (shrinks the **KV cache**).
- Uses **one key (K) and value (V)** for all query heads.
- K and V are calculated only once.
- **MQA**’s performance degrades slightly compared to **MHA**.
- **MQA** is faster than **MHA** in inference.

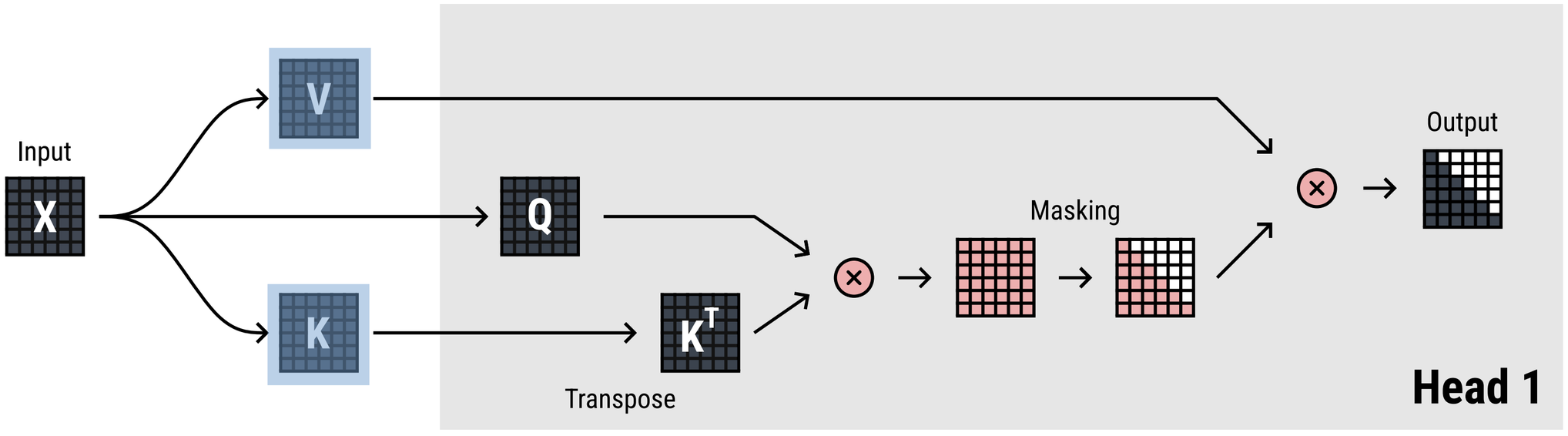




Attention layer – Multi Query Attention

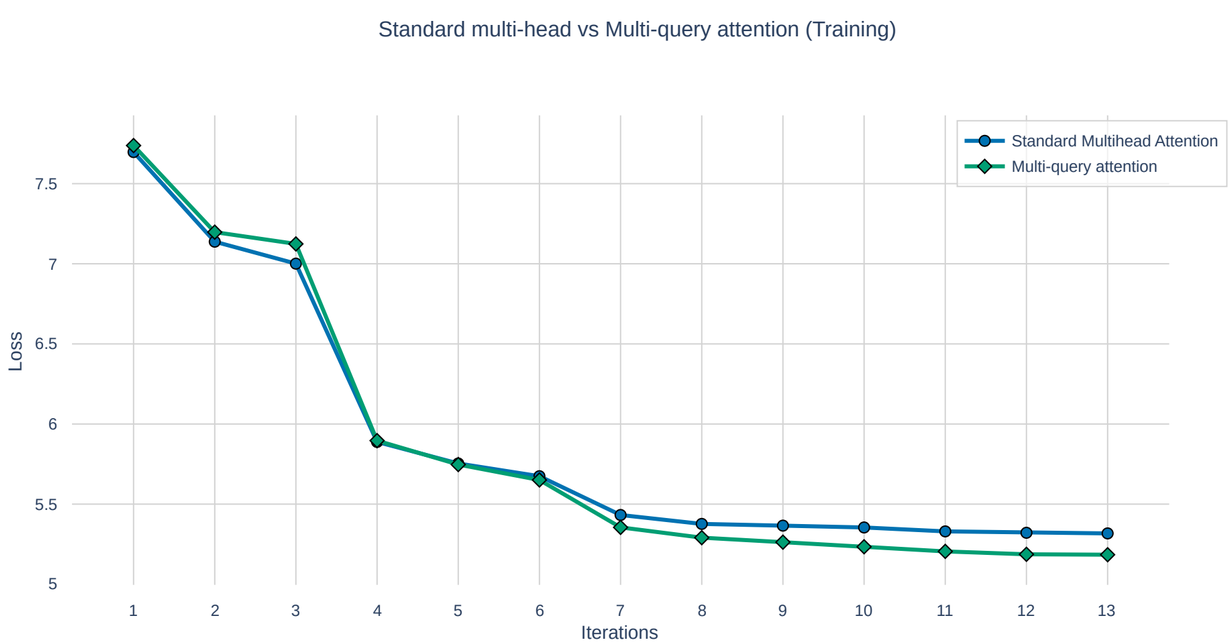
$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

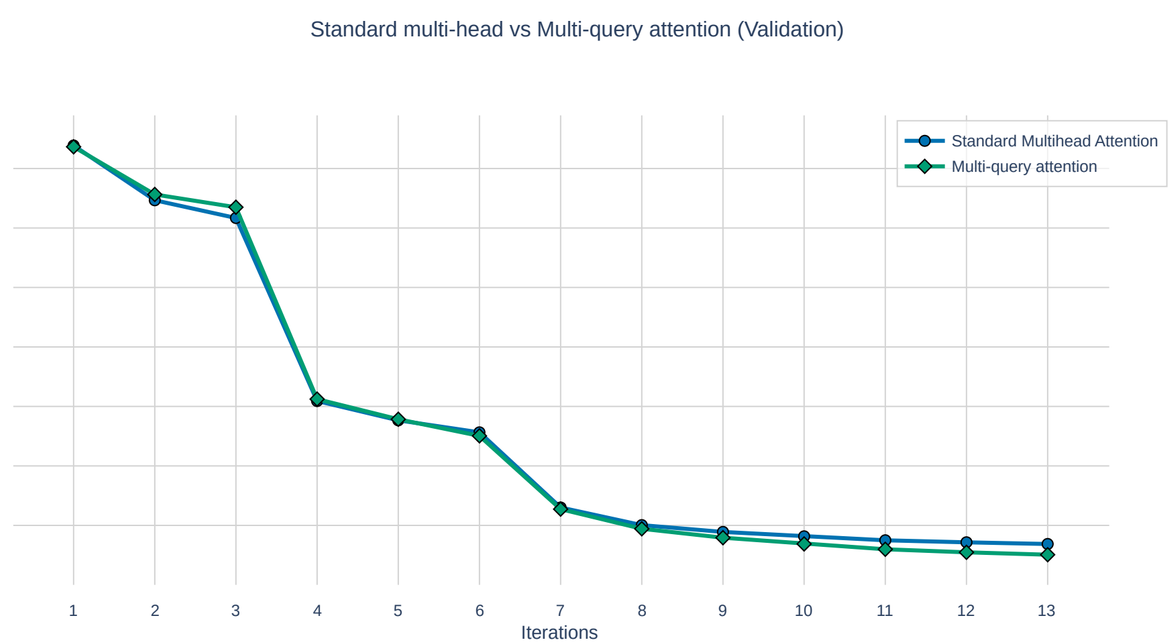




Attention layer – Multi Query Attention

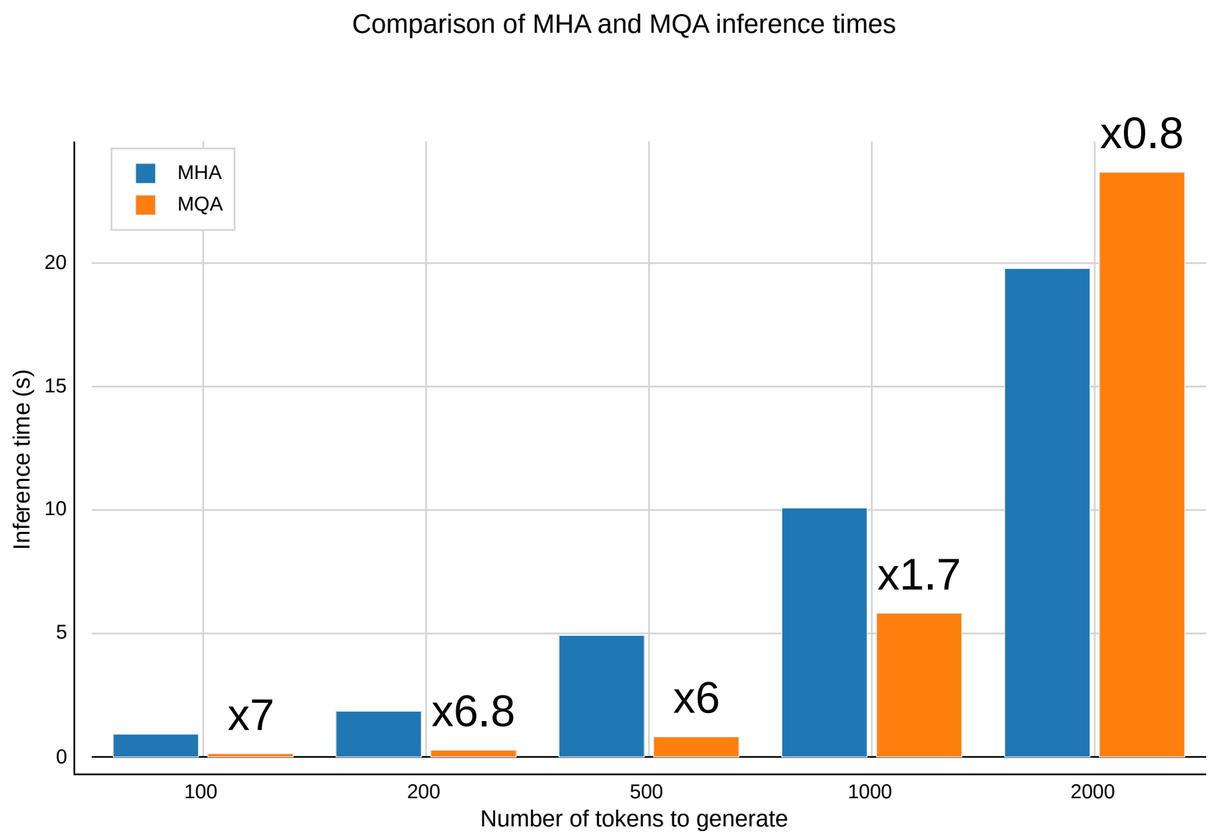
Comparison





Attention layer – Multi Query Attention

Inference speed



Attention layer – Local Attention

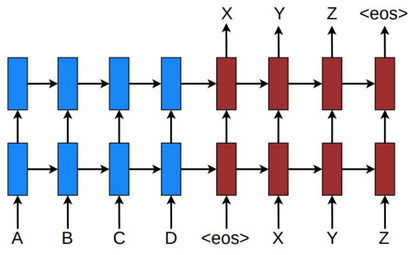
- **Local Attention** was mentioned in these 2 papers.

Effective Approaches to Attention-based Neural Machine Translation

Minh-Thang Luong **Hieu Pham** **Christopher D. Manning**
Computer Science Department, Stanford University, Stanford, CA 94305
{lmthang,hyhieu,manning}@stanford.edu

Abstract

An attentional mechanism has lately been used to improve neural machine translation (NMT) by selectively focusing on parts of the source sentence during translation. However, there has been little work exploring useful architectures for attention-based NMT. This paper examines two simple and effective classes of at-

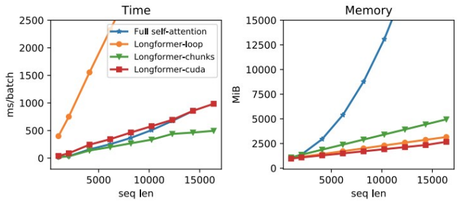


Longformer: The Long-Document Transformer

Iz Beltagy* **Matthew E. Peters*** **Arman Cohan***
Allen Institute for Artificial Intelligence, Seattle, WA, USA
{beltagy,matthewp,armanc}@allenai.org

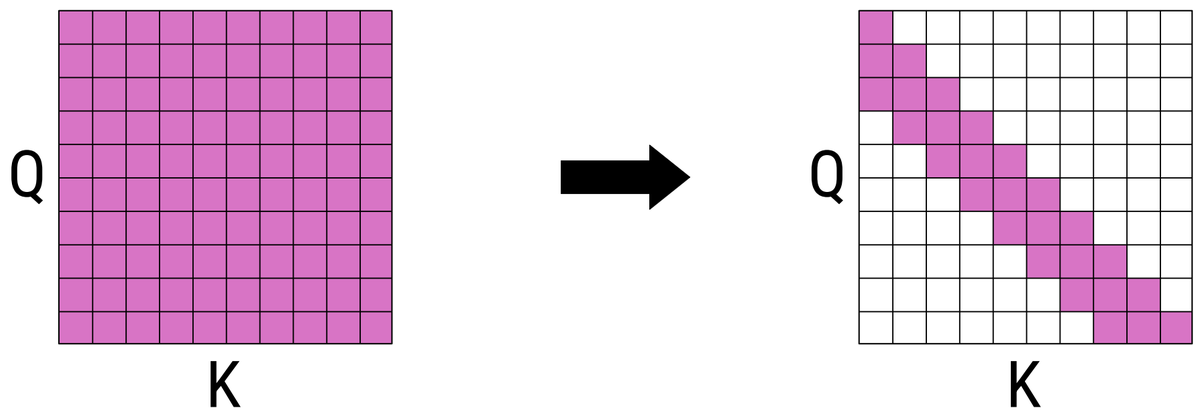
Abstract

Transformer-based models are unable to process long sequences due to their self-attention operation, which scales quadratically with the sequence length. To address this limitation, we introduce the Longformer with an attention mechanism that scales linearly with sequence



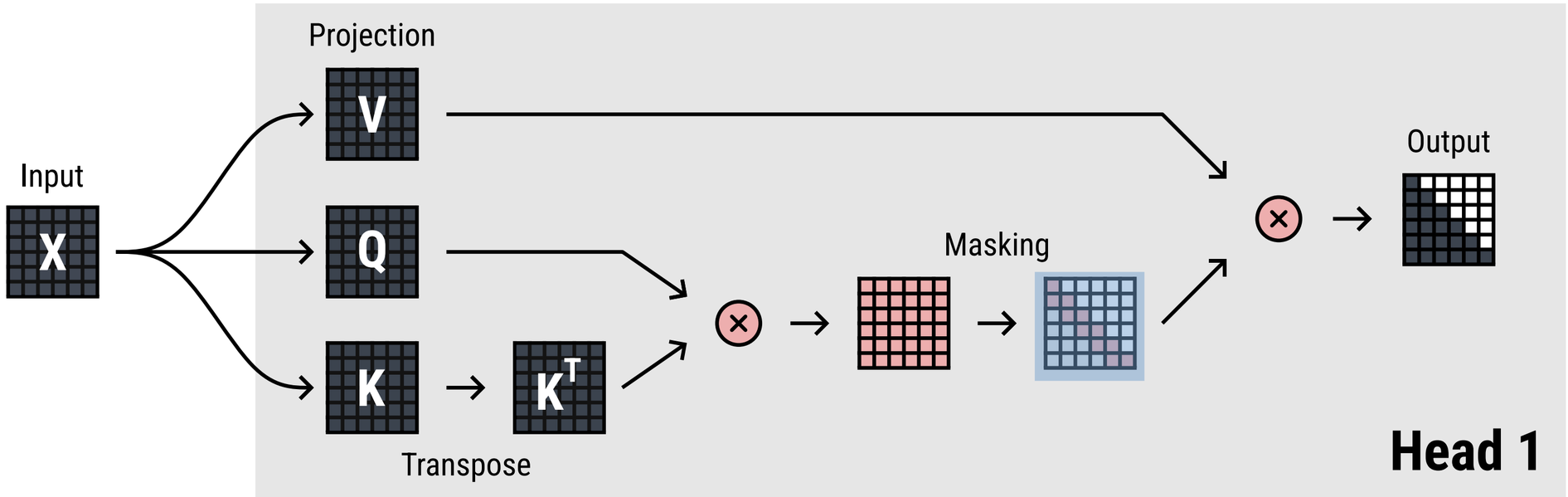
Attention layer – Local Attention

- **Local Attention** limits the attention span of each token to a **fixed-size window**.
- Local Attention is **computationally efficient**.
- **Long range dependencies are not captured** because the model focuses on the local context.
- Local Attention may lead to **degradation in performance**.



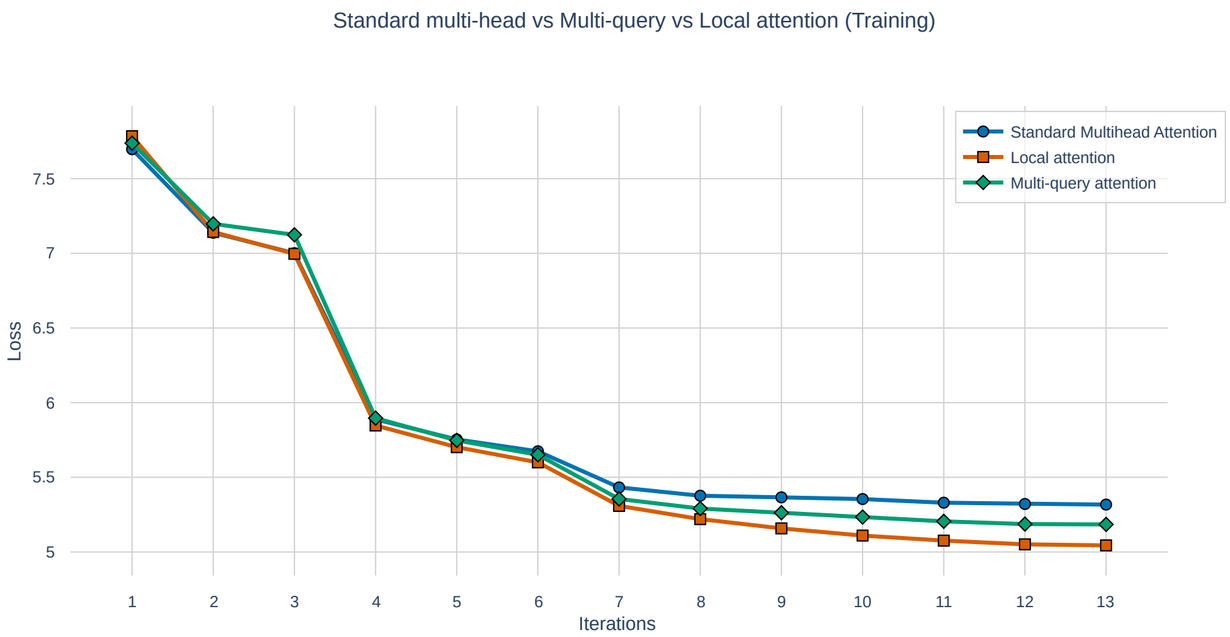
Attention layer – Local Attention

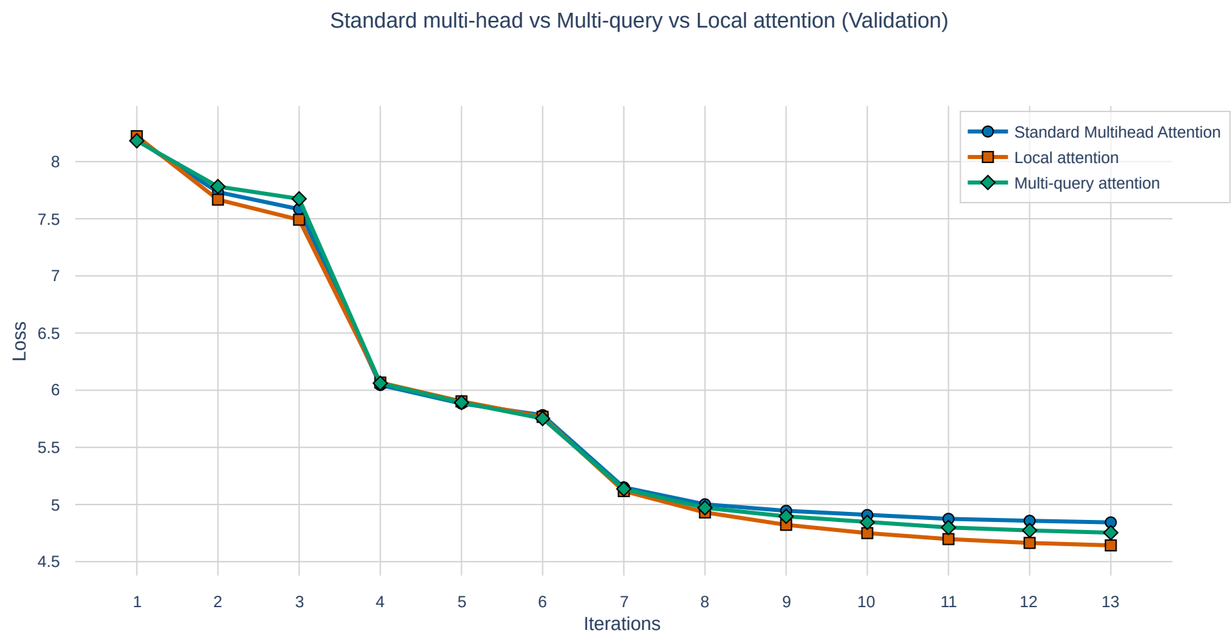
$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



Attention layer – Local Attention

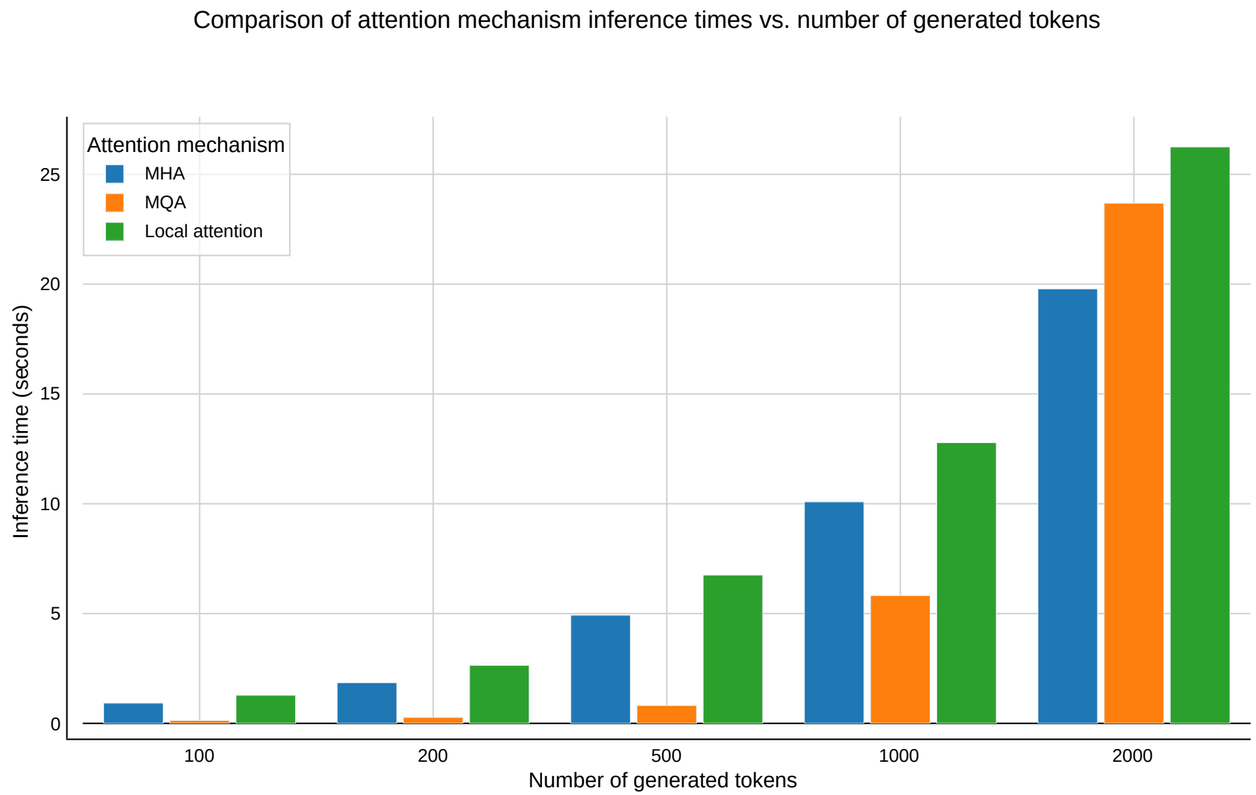
Comparison





Attention layer – Local Attention

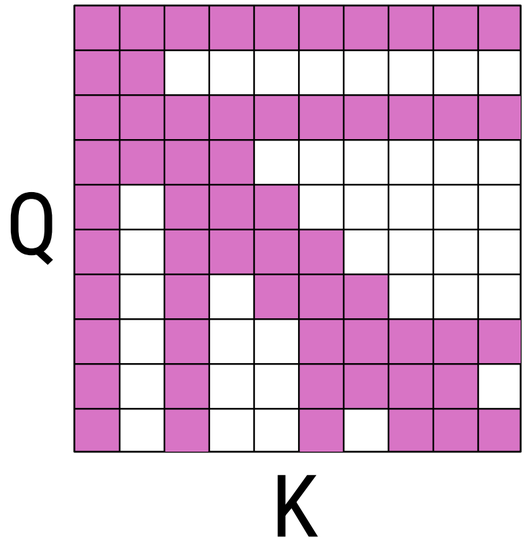
Inference speed

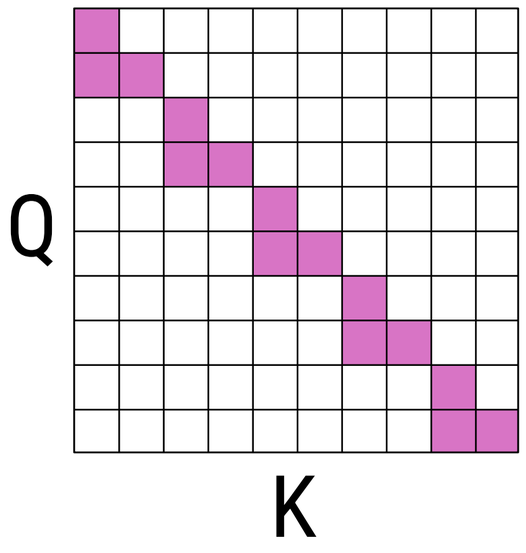


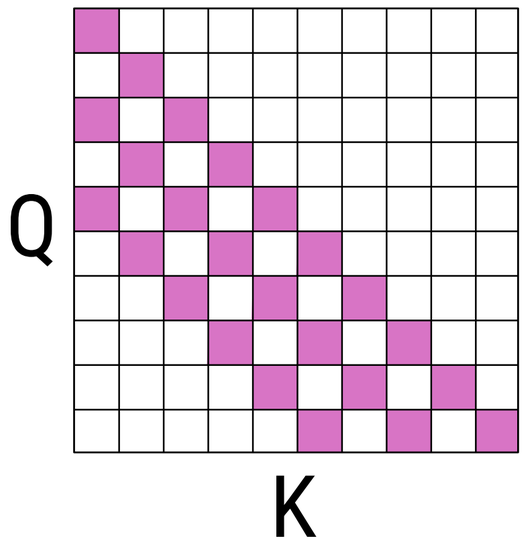
Attention layer – Local Attention

Other variants:

- Dilated sliding window.
- Chunked sliding window.
- Global + sliding window.







Attention layer – Grouped Query Attention

- **GQA** was mentioned in this paper.

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

Abstract

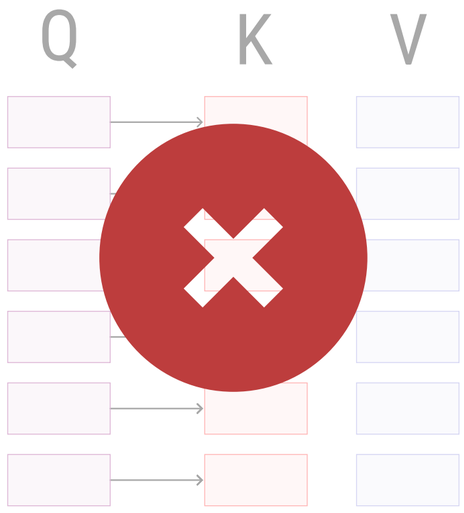
Multi-query attention (MQA), which only uses a single key-value head, drastically speeds up decoder inference. However, MQA can lead to quality degradation, and moreover it may not be desirable to train a separate model just for faster inference. We (1) propose a recipe for uptraining existing multi-head language model checkpoints into models with MQA using 5% of original pre-training compute, and (2) intro-

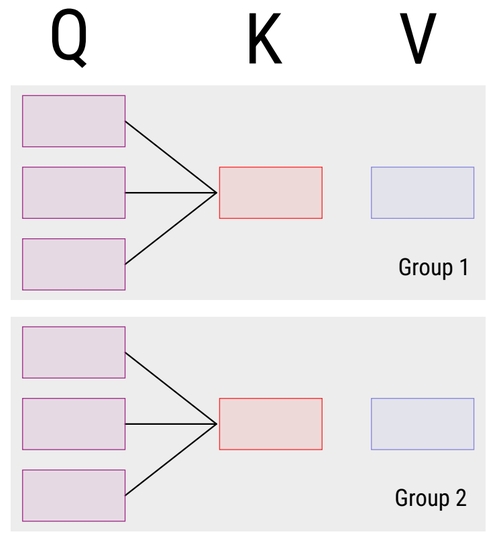
show that language model checkpoints with multi-head attention (MHA) can be *uptrained* (Komatsuzaki et al., 2022) to use MQA with a small fraction of original training compute. This presents a cost-effective method to obtain fast multi-query as well as high-quality MHA checkpoints.

Second, we propose grouped-query attention (GQA), an interpolation between multi-head and multi-query attention with single key and value heads *per subgroup of query heads*. We show that

Attention layer – Grouped Query Attention

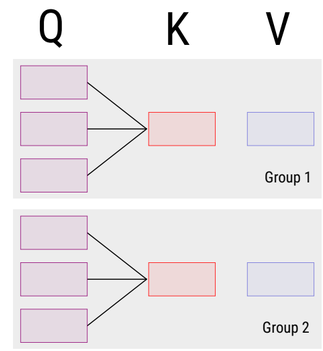
- **GQA** is the generalization of **MHA** and **MQA**.
- **GQA** groups queries together.
- **GQA** offers an ideal trade-off between performance and speed.
- **GQA** reduces the number of calculations.

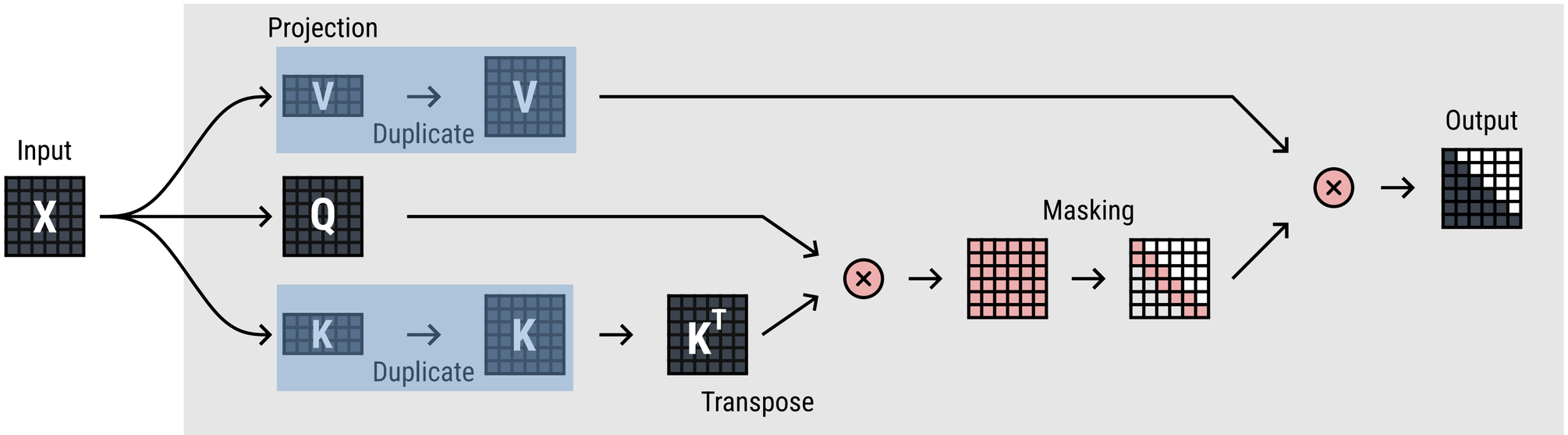




Attention layer – Grouped Query Attention

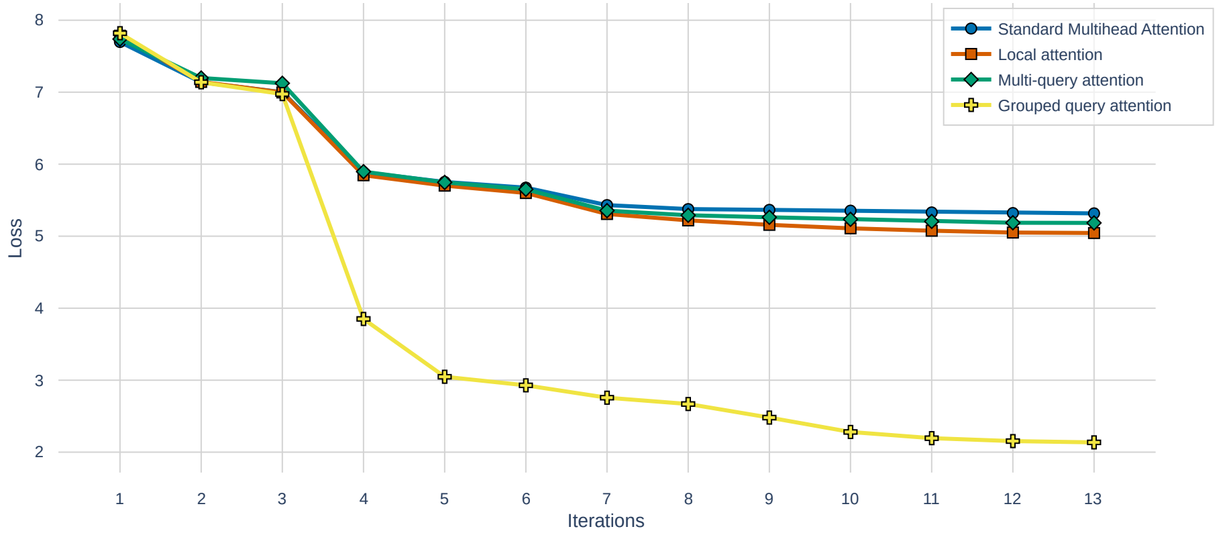
$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

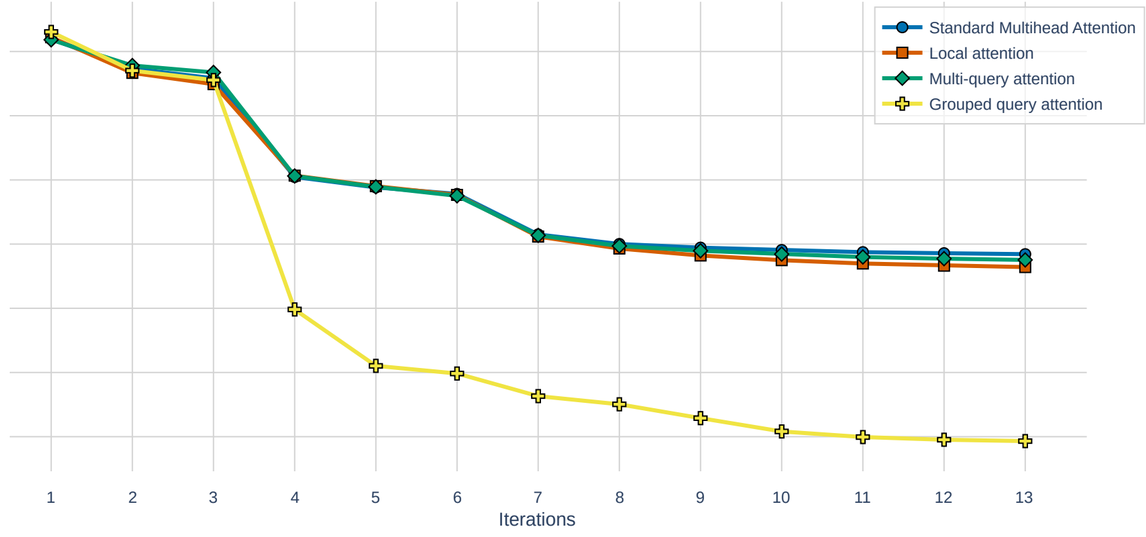




Attention layer – Grouped Query Attention

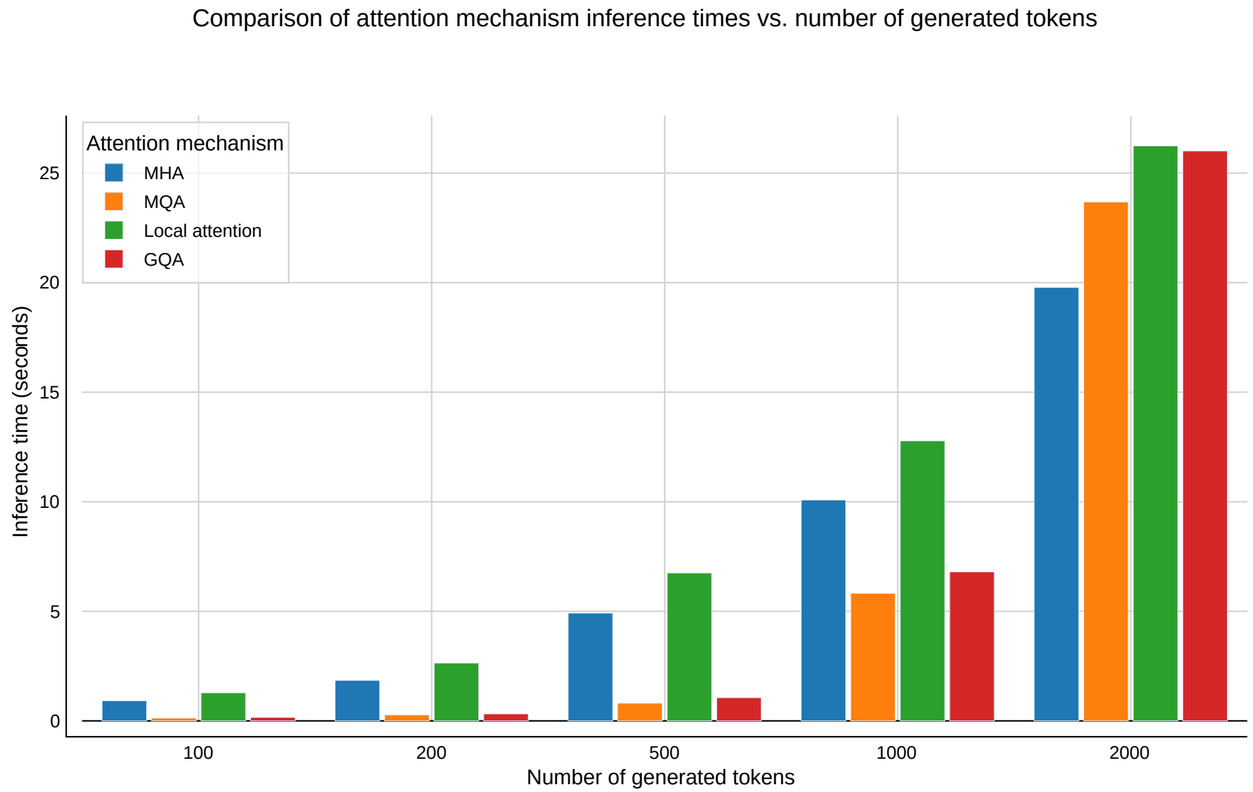
Comparison





Attention layer – Grouped Query Attention

Inference speed



Attention layer – Linear Attention

• Read the following papers to understand **Linear Attention**.

Transformers are RNNs:
Fast Autoregressive Transformers with Linear Attention

Angelos Katharopoulos [1 2] **Apoorv Vyas** [1 2] **Nikolaos Pappas** [3] **François Fleuret** [2 4 *]

Abstract

Transformers achieve remarkable performance in several tasks but due to their quadratic complexity, with respect to the input's length, they are prohibitively slow for very long sequences. To address this limitation, we express the self-attention as a linear dot-product of kernel feature maps and make use of the associativity property of matrix products to reduce the complexity from $\mathcal{O}\left(N^2\right)$ to $\mathcal{O}\left(N\right)$, where N is the sequence length. We show that this formulation permits an iterative implementation that dramatically accelerates autoregressive transformers and reveals their relationship to recurrent neural networks. Our *lin*

by the global receptive field of self-attention, whic
cesses contexts of N inputs with a quadratic memor
time complexity $\mathcal{O}\left(N^2\right)$. As a result, in practice
formers are slow to train and their context is *limited*
disrupts temporal coherence and hinders the capturi
long-term dependencies. Dai et al. (2019) addressed t
ter by attending to memories from previous contexts
at the expense of computational efficiency.

Lately, researchers shifted their attention to approache
increase the context length without sacrificing effic
Towards this end, Child et al. (2019) introduced s
factorizations of the attention matrix to reduce the
attention complexity to $\mathcal{O}\left(N\sqrt{N}\right)$. Kitaev et al. (202

LINEAR ATTENTION IS (MAYBE) ALL YOU NEED
(TO UNDERSTAND TRANSFORMER OPTIMIZATION)

Kwangjun Ahn*
MIT EECS/LIDS
kjahn@mit.edu

Xiang Cheng*
MIT LIDS
chengx@mit.edu

Minhak Song*
KAIST ISysE/Math
minhaksong@kaist.ac.kr

Chulhee Yun
KAIST AI
chulhee.yun@kaist.ac.kr

Ali Jadbabaie
MIT CEE/LIDS
jadbabai@mit.edu

Suvrit Sra
MIT EECS/LIDS
suvrit@mit.edu

ABSTRACT

Transformer training is notoriously difficult, requiring a careful design of optimizers and use of various heuristics. We make progress towards understanding the subtleties of training Transformers by carefully studying a simple yet canonical linearized *shallow* Transformer model. Specifically, we train linear Transformers to solve regression tasks, inspired by J. von Oswald et al. (ICML 2023), and K. Ahn et al. (NeurIPS 2023). Most importantly, we observe that our proposed linearized models can reproduce several prominent aspects of Transformer training dynamics. Consequently, the results obtained in this paper suggest that a simple

Attention layer – Linear Attention

- **MHA scales badly** with long sequence lengths.
- **Linear Attention** solves this issue by **using less memory** and approximating **(1)** with **(2).**
- **Linear Attention** can give you good performance even with that approximation.

$$(1)\quad Attention(Q,K,V)=softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$(2)\quad Linear\ Attention(Q,K,V)=\frac{\phi(Q_i)^T\sum_{j=1}^{N}\phi(K_j)V_j^T}{\phi(Q_i)^T\sum_{j=1}^{N}\phi(K_j)}$$

Transformers are RNNs

3.1. Transformers

Let $x \in \mathbb{R}^{N\times F}$ denote a sequence of N feature vectors of dimensions F. A transformer is a function $T : \mathbb{R}^{N\times F} \rightarrow \mathbb{R}^{N\times F}$ defined by the composition of L transformer layers $T_1(\cdot), \ldots, T_L(\cdot)$ as follows,

$$T_l(x) = f_l(A_l(x) + x). \tag{1}$$

The function $f_l(\cdot)$ transforms each feature independently of the others and is usually implemented with a small two-layer feedforward network. $A_l(\cdot)$ is the self attention function and is the only part of the transformer that acts across sequences.

The self attention function $A_l(\cdot)$ computes, for every position, a weighted average of the feature representations of all other positions with a weight proportional to a similarity score between the representations. Formally, the input sequence x is projected by three matrices $W_Q \in \mathbb{R}^{F\times D}$

Given such a kernel with a feature representation $\phi(x)$ we can rewrite equation 2 as follows,

$$V_i' = \frac{\sum_{j=1}^{N}\phi(Q_i)^T\phi(K_j)V_j}{\sum_{j=1}^{N}\phi(Q_i)^T\phi(K_j)}, \tag{4}$$

and then further simplify it by making use of the associative property of matrix multiplication to

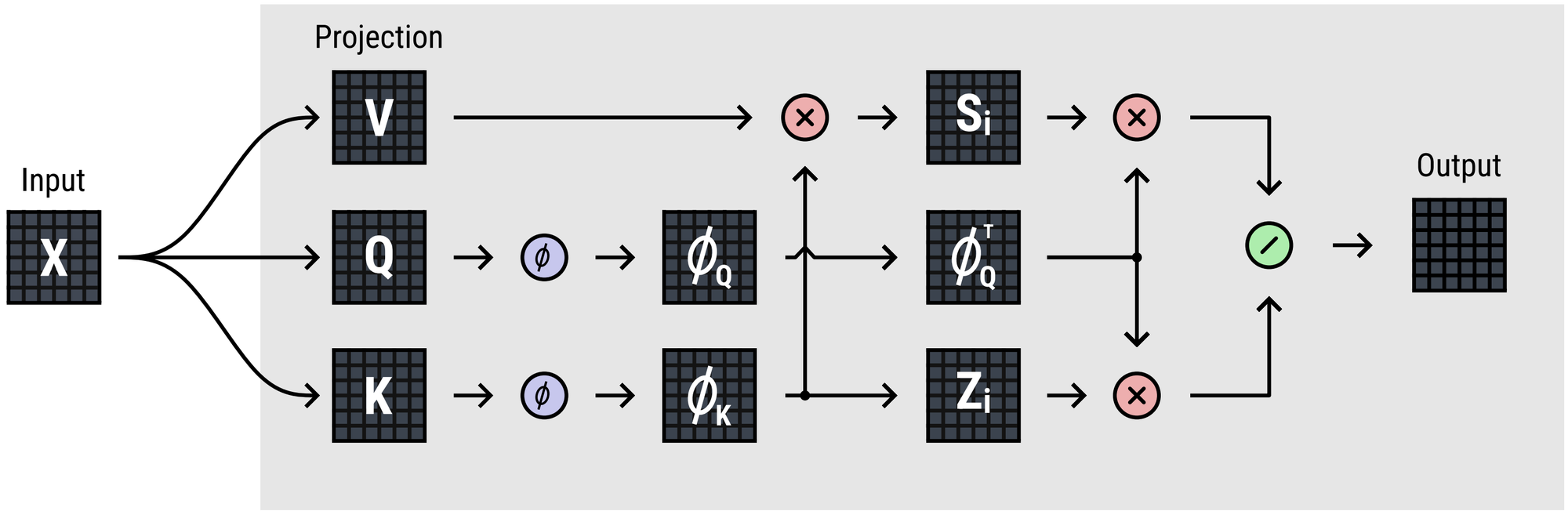
$$V_i' = \frac{\phi(Q_i)^T\sum_{j=1}^{N}\phi(K_j)V_j^T}{\phi(Q_i)^T\sum_{j=1}^{N}\phi(K_j)}. \tag{5}$$

The above equation is simpler to follow when the numerator is written in vectorized form as follows,

$$\left(\phi(Q)\phi(K)^T\right)V = \phi(Q)\left(\phi(K)^T V\right). \tag{6}$$

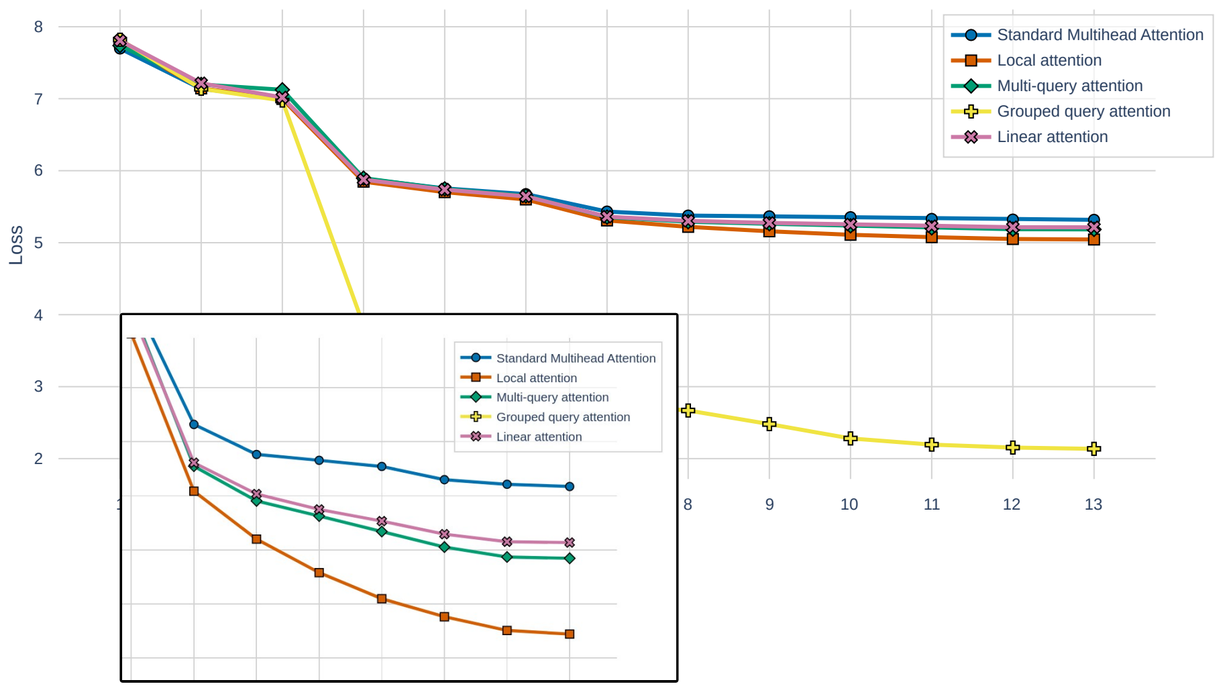
Attention layer – Linear Attention

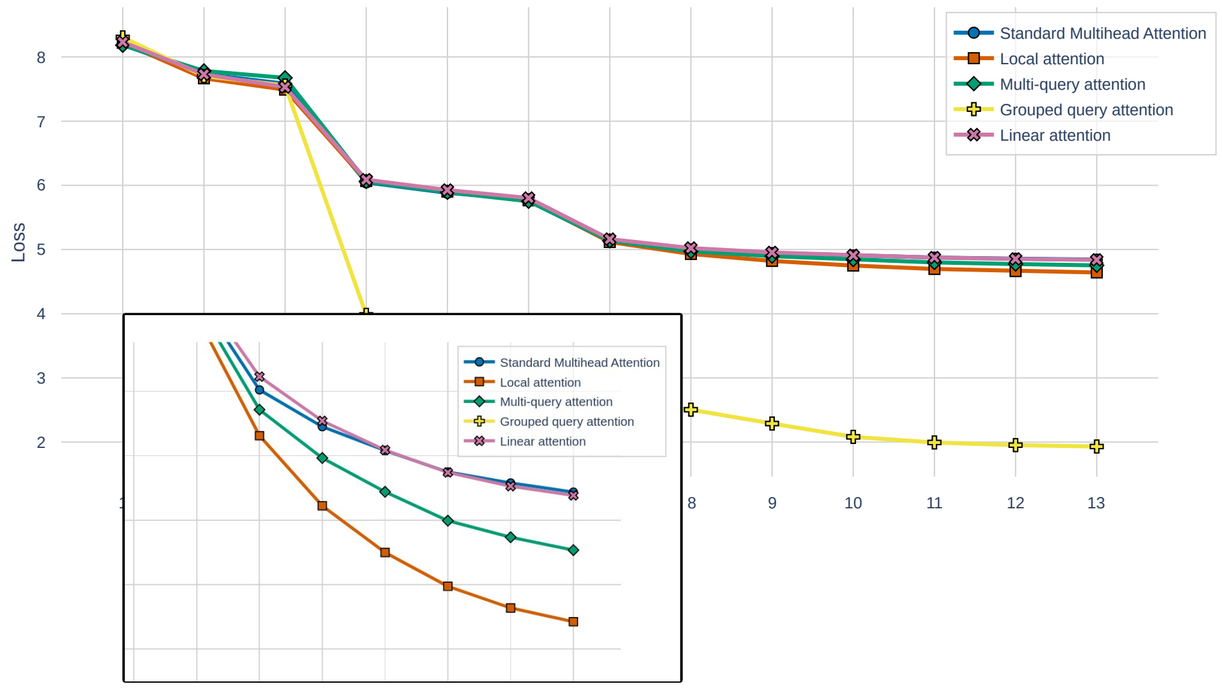
$$Linear\ Attention(Q,K,V)=\frac{\phi(Q_i)^T\sum_{j=1}^{N}\phi(K_j)V_j^T}{\phi(Q_i)^T\sum_{j=1}^{N}\phi(K_j)}=\frac{\phi(Q_i)^T S_i}{\phi(Q_i)^T Z_i}$$



Attention layer – Linear Attention

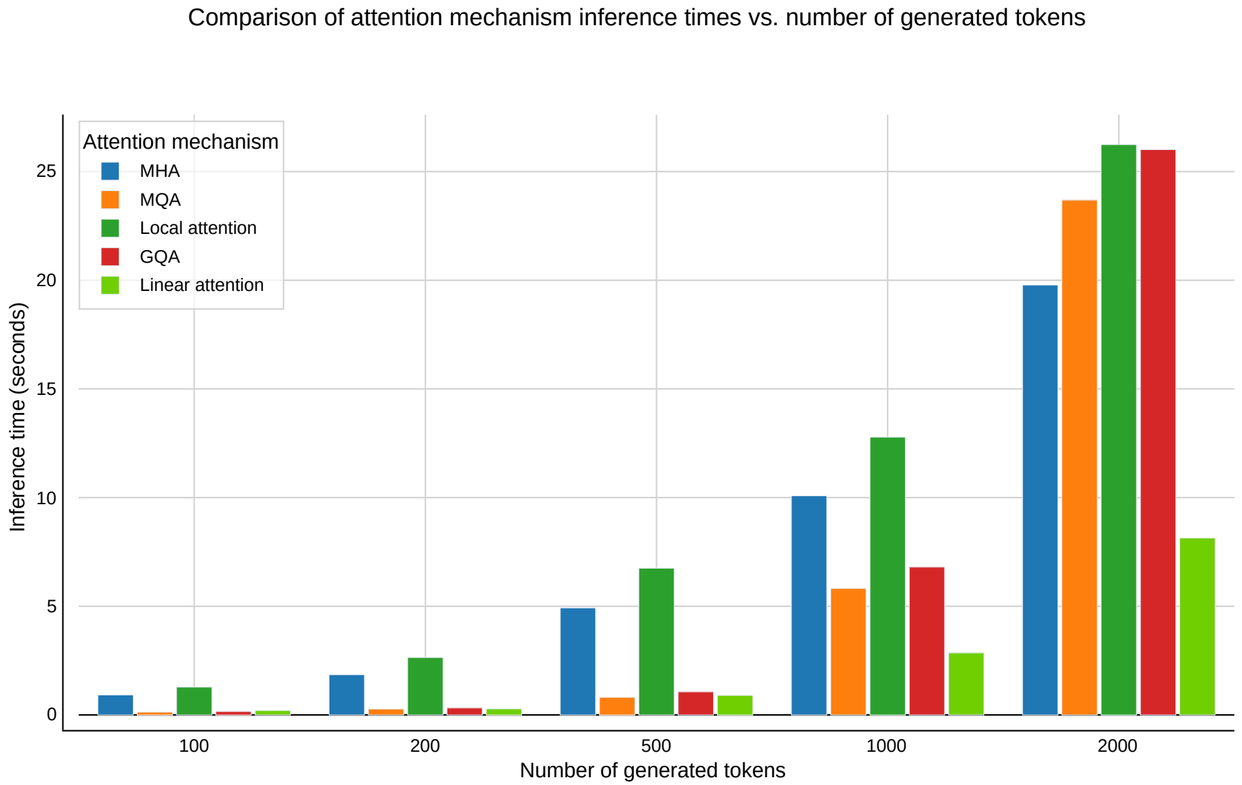
Comparison





Attention layer – Linear Attention

Inference speed



Attention layer – BigBird (Sparse Attention)

- Read about **BigBird** in this paper.

Big Bird: Transformers for Longer Sequences

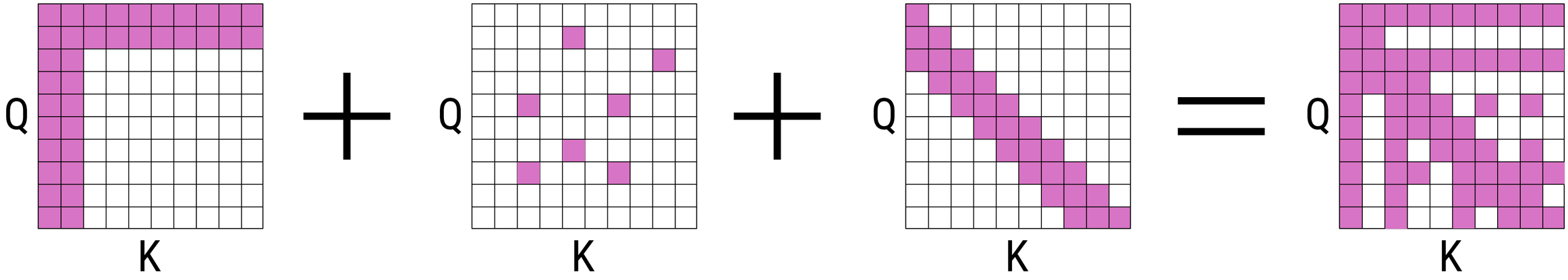
Manzil Zaheer, Guru Guruganesh, Avinava Dubey,
Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham,
Anirudh Ravula, Qifan Wang, Li Yang, Amr Ahmed
Google Research
{manzilz, gurug, avinavadubey}@google.com

Abstract

Transformers-based models, such as BERT, have been one of the most successful deep learning models for NLP. Unfortunately, one of their core limitations is the quadratic dependency (mainly in terms of memory) on the sequence length due to their full attention mechanism. To remedy this, we propose, BIGBIRD, a sparse attention mechanism that reduces this quadratic dependency to linear. We show that BIGBIRD is a universal approximator of sequence functions and is Turing complete, thereby preserving these properties of the quadratic, full attention model. Along the way, our theoretical analysis reveals some of the benefits of having $O(1)$ global tokens (such as CLS), that attend to the entire sequence as part of the sparse attention mechanism. The proposed sparse attention can handle sequences of length up to 8x of what was previously possible using similar hardware. As a consequence of the capability to handle longer context, BIGBIRD drastically improves performance on various NLP tasks such as question answering and summarization. We also propose novel applications to genomics data.

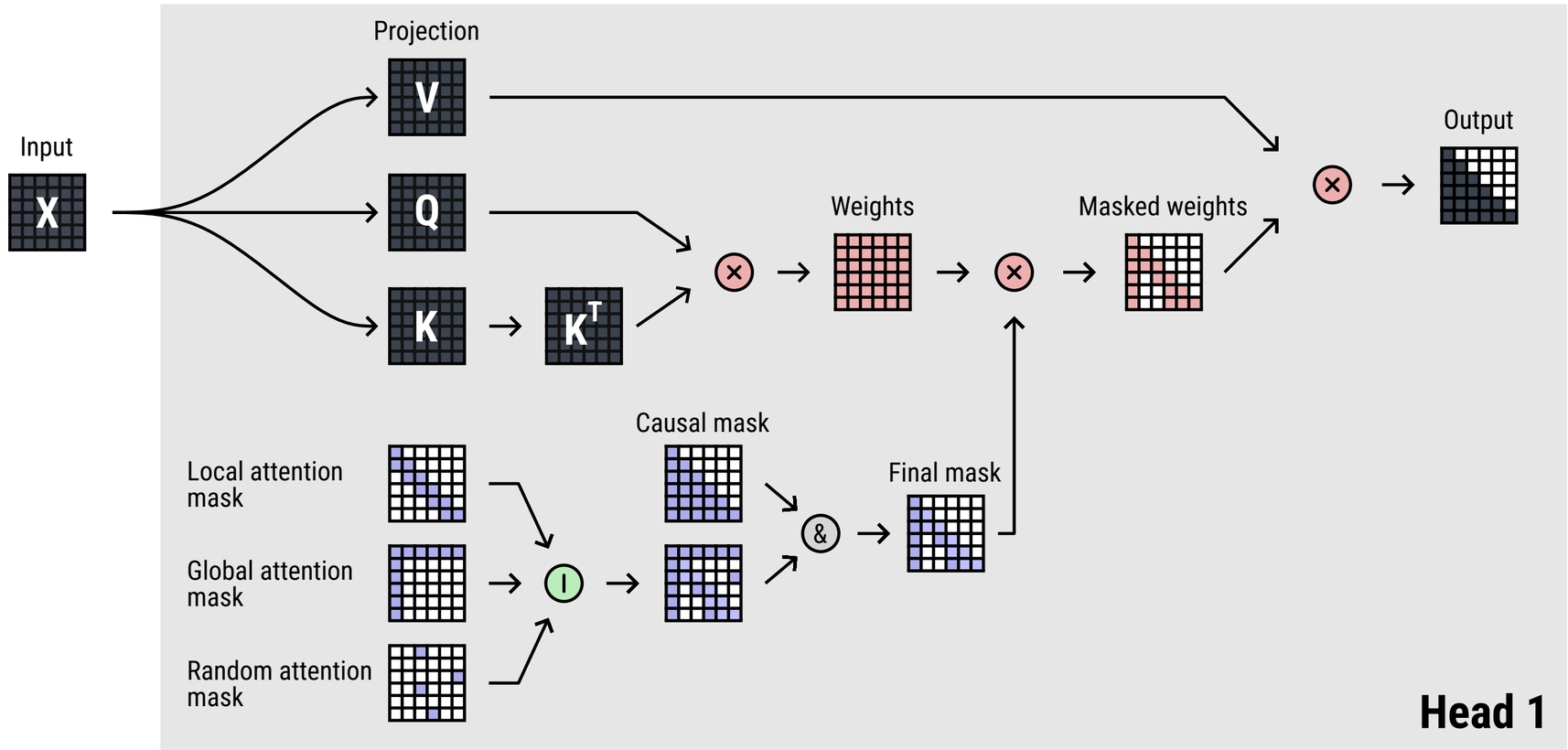
Attention layer – BigBird (Sparse Attention)

- **BigBird** is designed to **process large sequences** of text.
- **Sparse attention** reduces the computational and memory complexity to **O(n)**.
- **BigBird = Global + Random + Window attention.**
- **BigBird captures the global and local dependencies** in the sequence.



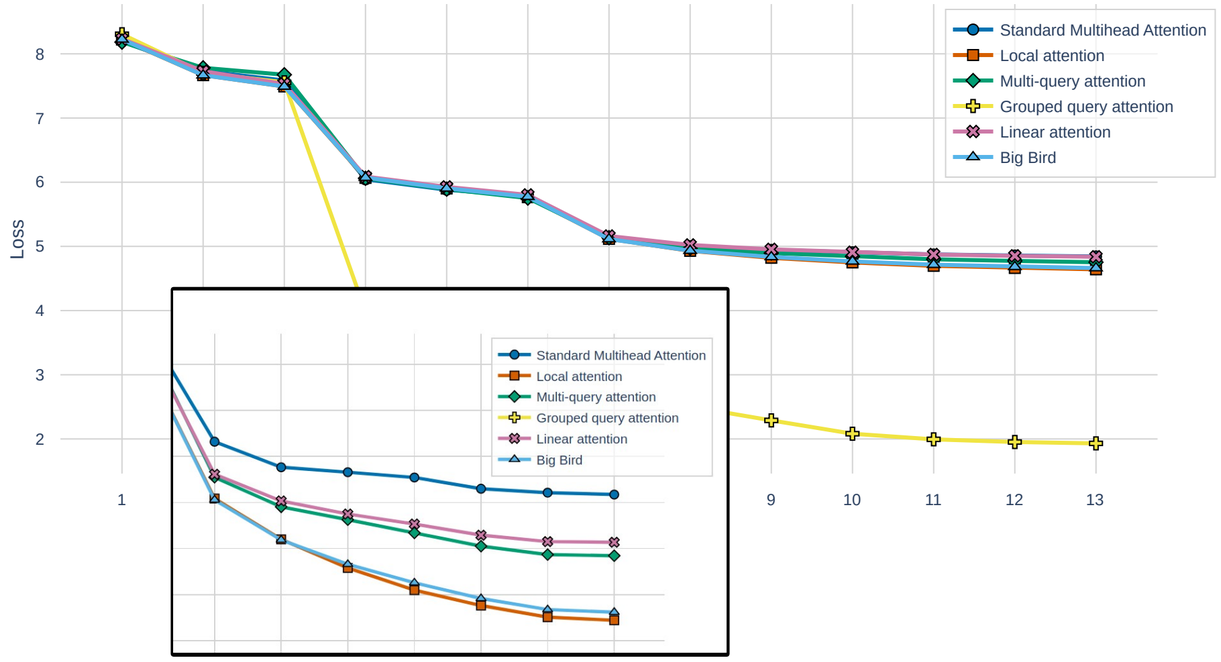
Attention layer – BigBird (Sparse Attention)

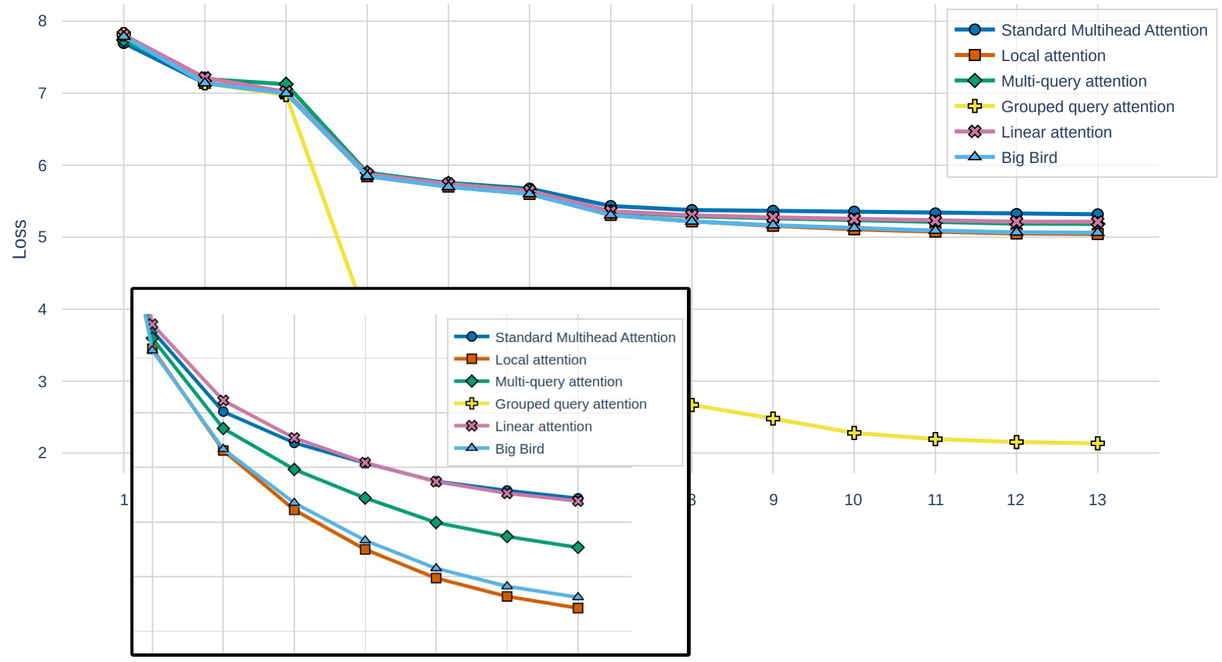
$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



Attention layer – BigBird (Sparse Attention)

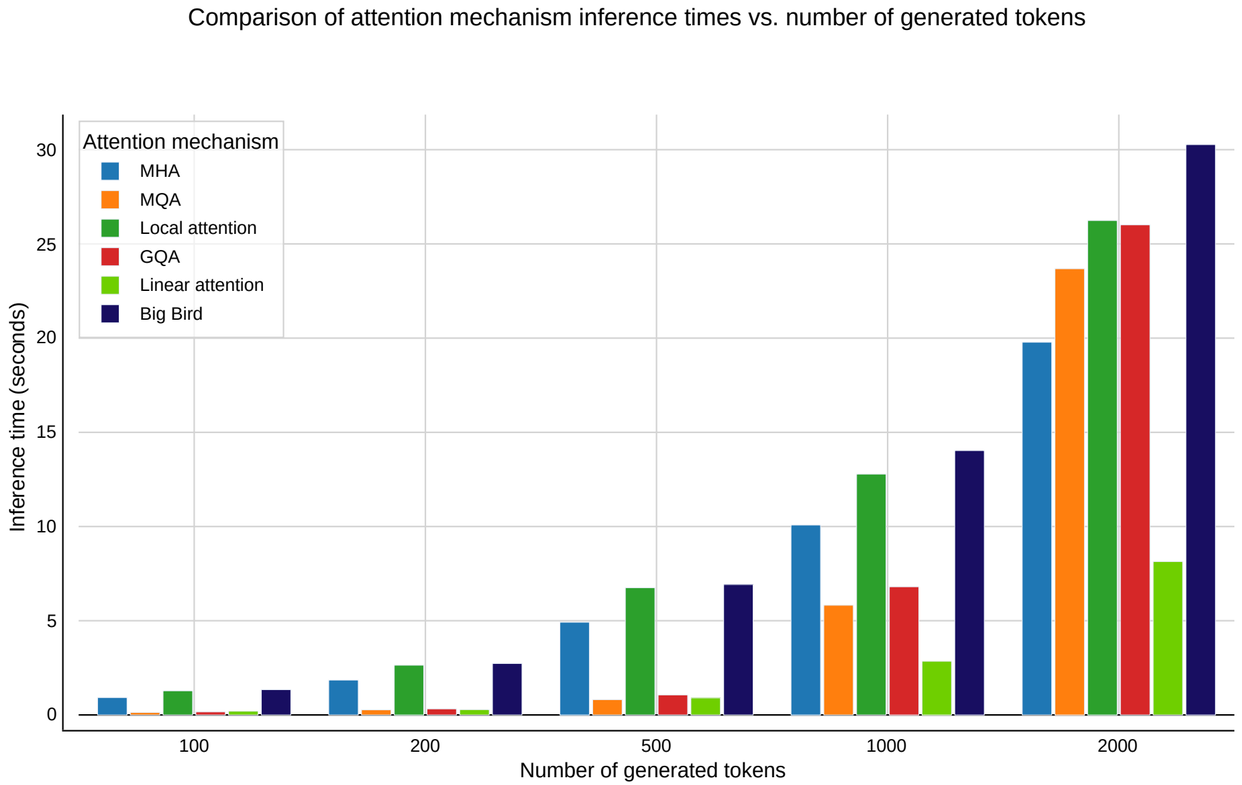
Comparison





Attention layer – BigBird (Sparse Attention)

Inference speed



Attention layer – Multi-head Latent Attention

- **MLA** was introduced in the **DeepSeek V2** paper.

deepseek

DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model

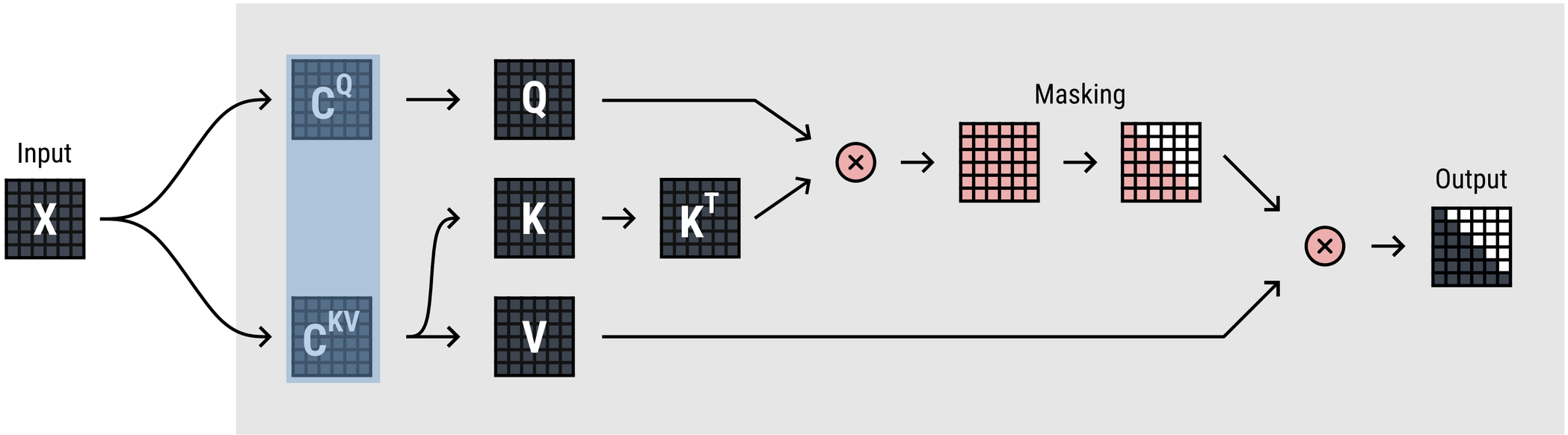
DeepSeek-AI

research@deepseek.com

Attention layer – Multi-head Latent Attention

- **MLA** is an efficient method that **compresses the KV cache**.
- **MLA** does not sacrifice performance for faster inference.
- **MLA** incorporates **latent representations** into the attention mechanism.
- **MLA is very fast** compared to **MHA**.

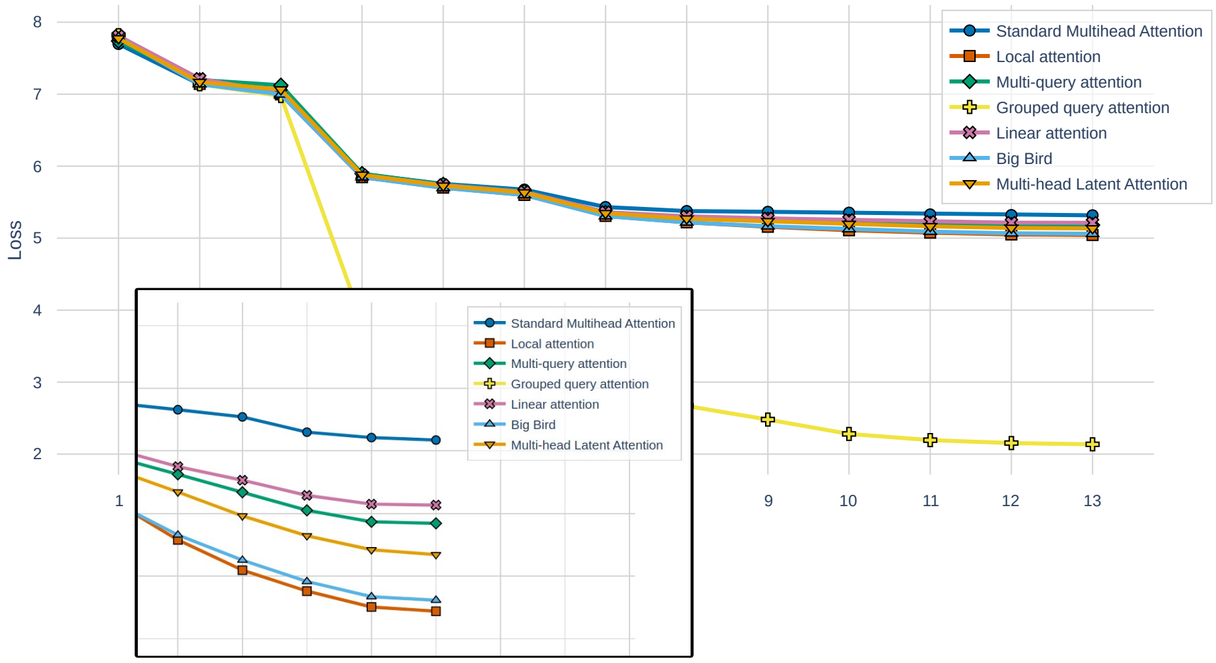
Attention layer – Multi-head Latent Attention



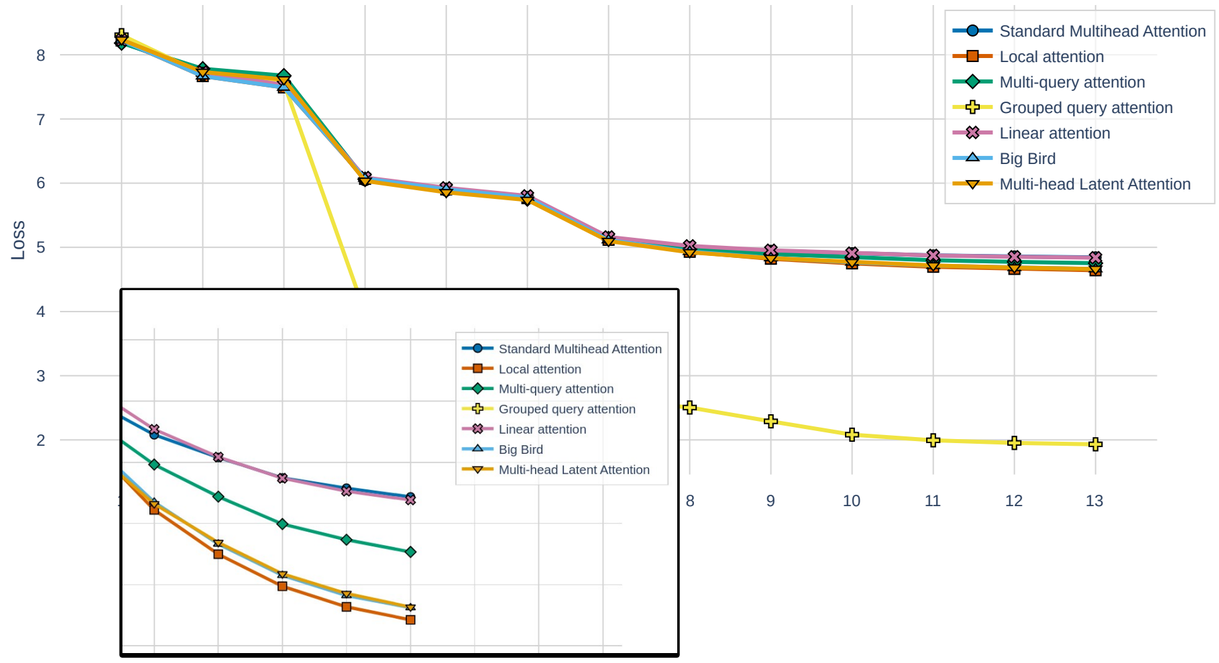
Attention layer – Multi-head Latent Attention

Comparison

Standard multi-head vs Multi-query vs Grouped query vs Local vs Linear vs Big Bird vs Latent attention (Training)

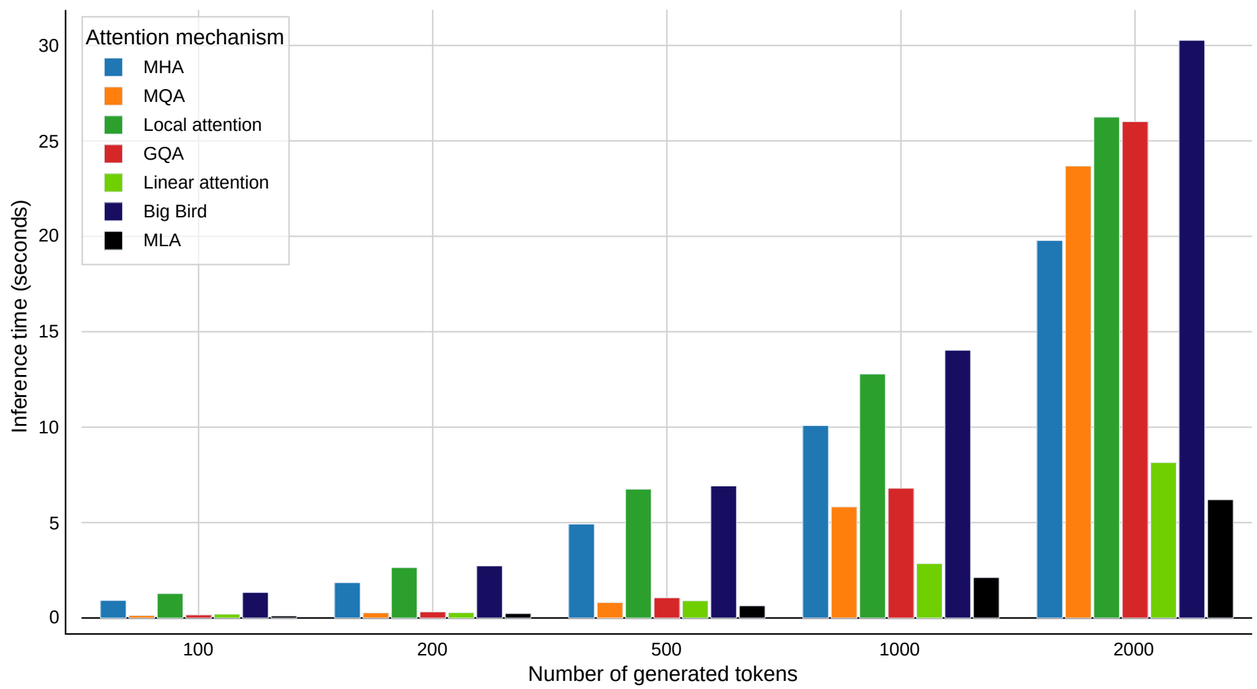


Standard multi-head vs Multi-query vs Grouped query vs Local vs Linear vs Big Bird vs Latent attention (Validation)



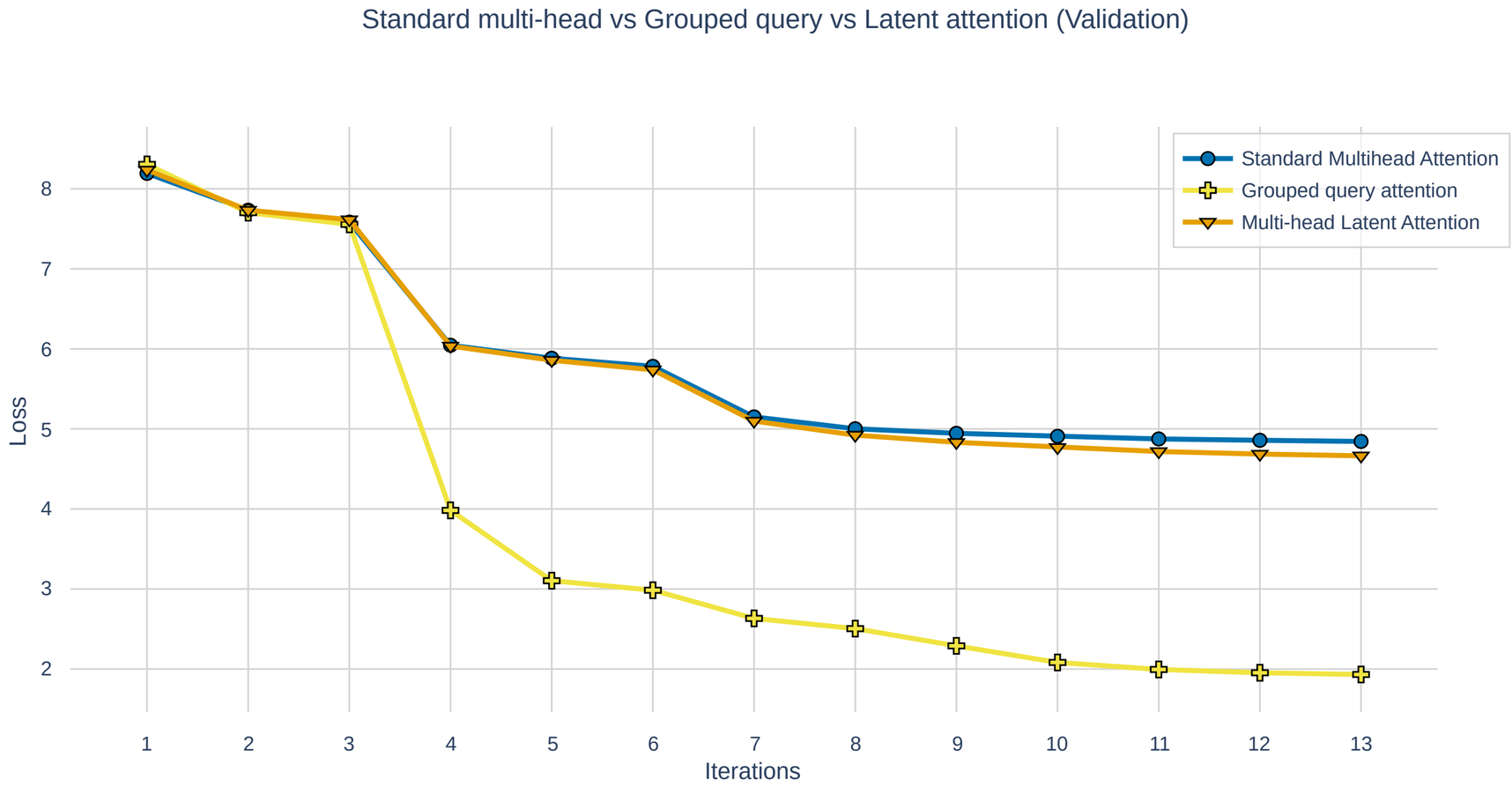
Attention layer – Multi-head Latent Attention

Inference speed



Attention layer – The end

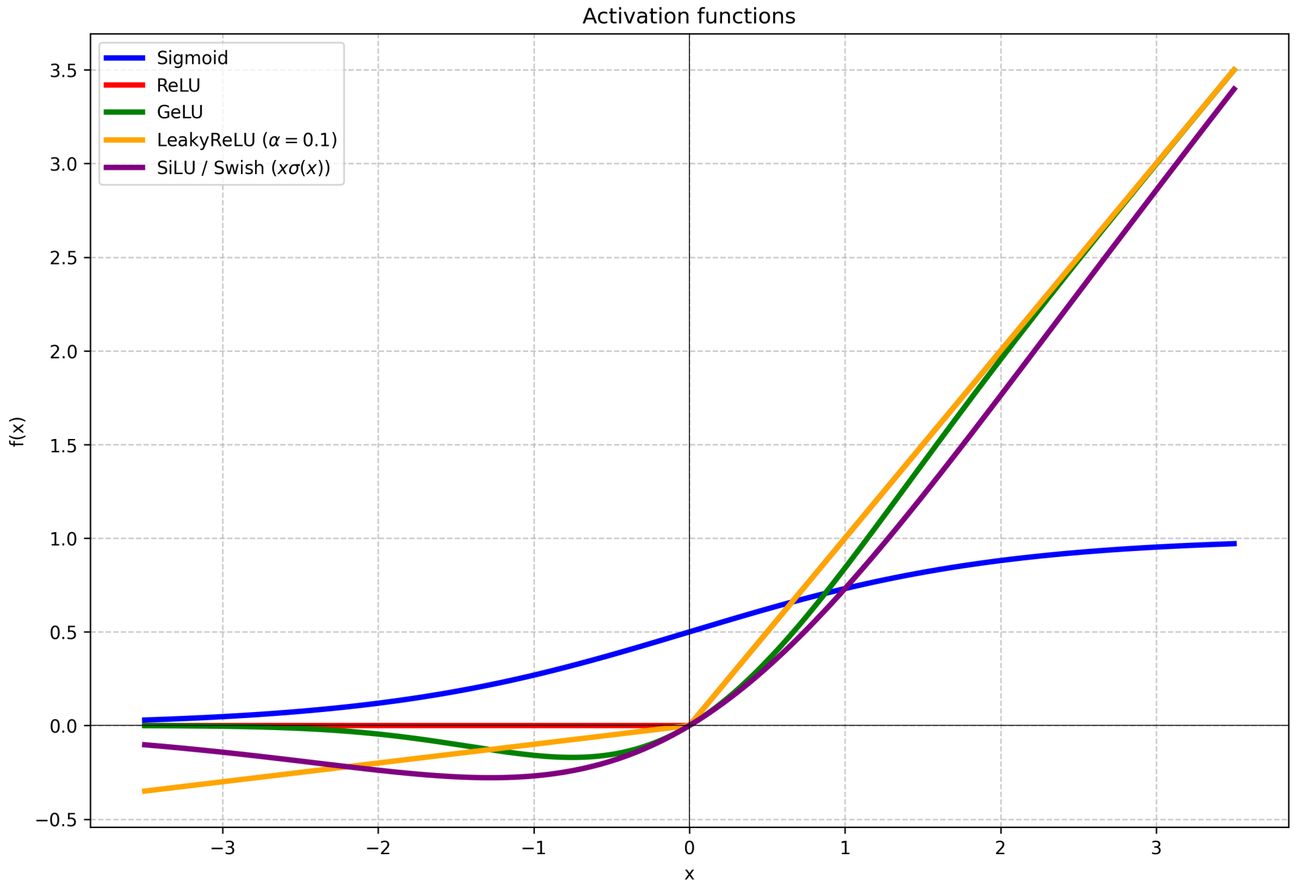
Before vs After



Small refinements

Small refinements

- Experiment with different **activation functions**.



Small refinements

- Experiment with different **activation functions**.
- Play with the **normalization methods**.

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

LayerNorm

$$y_i = \frac{x_i}{\sqrt{\frac{1}{d}\sum_{j=1}^{d} x_j^2 + \epsilon}} g_i$$

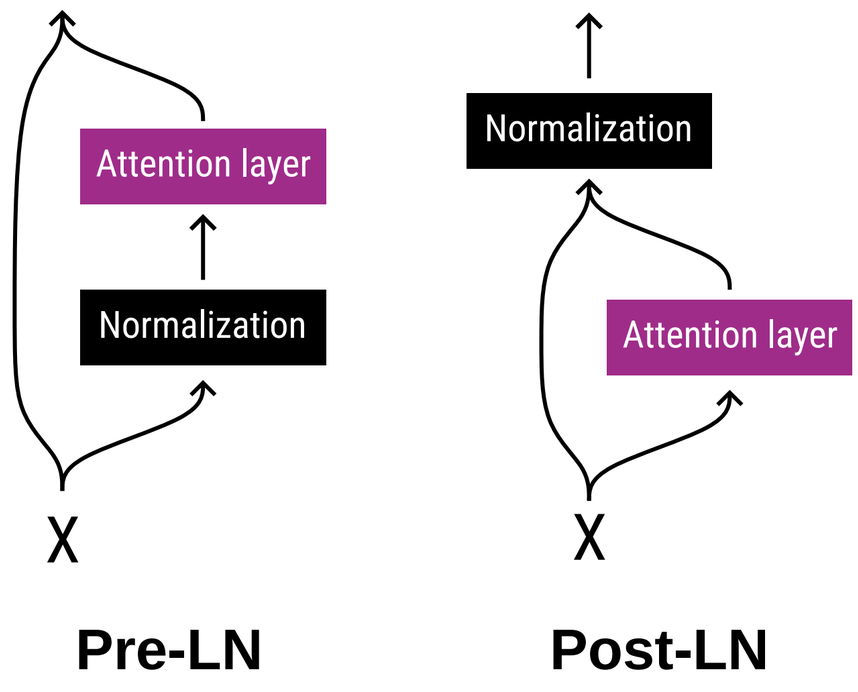
RMSNorm

$$y_i = \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

BatchNorm

Small refinements

- Experiment with different **activation functions**.
- Play with the **normalization methods**.
- **Where to put** the normalization layer?

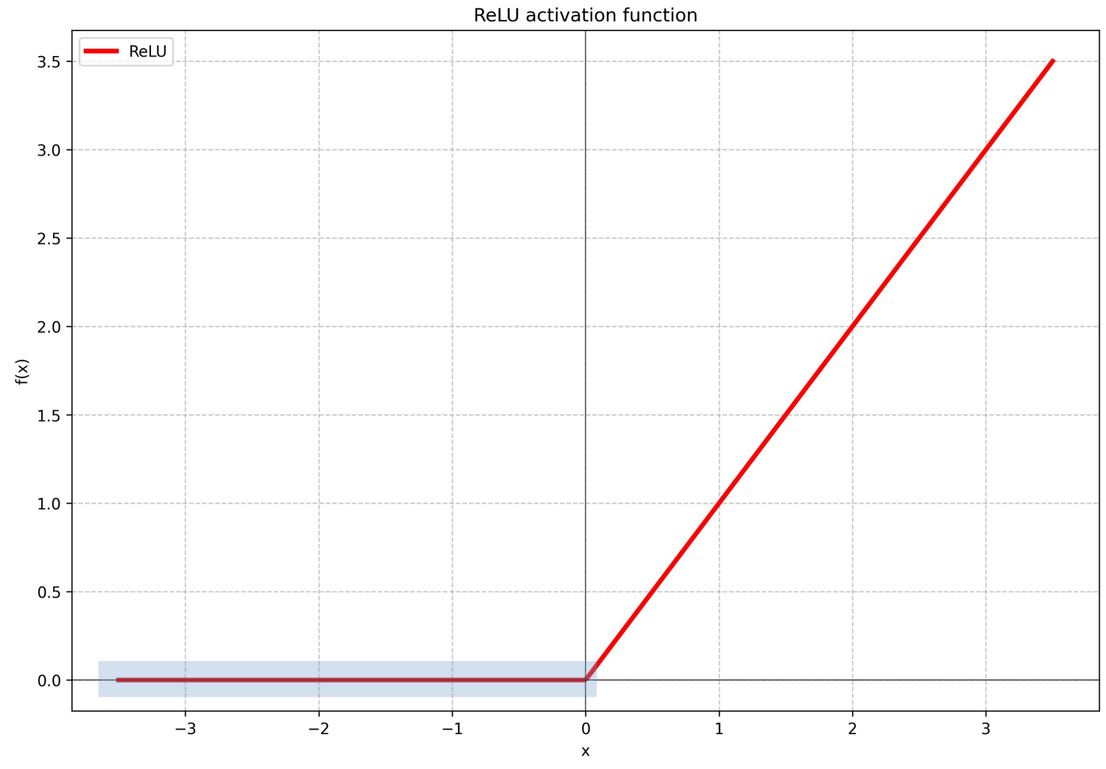


Small refinements

- Experiment with different **activation functions**.
- Play with the **normalization methods**.
- **Where to put** the normalization layer?
- Should we use **dropout**?

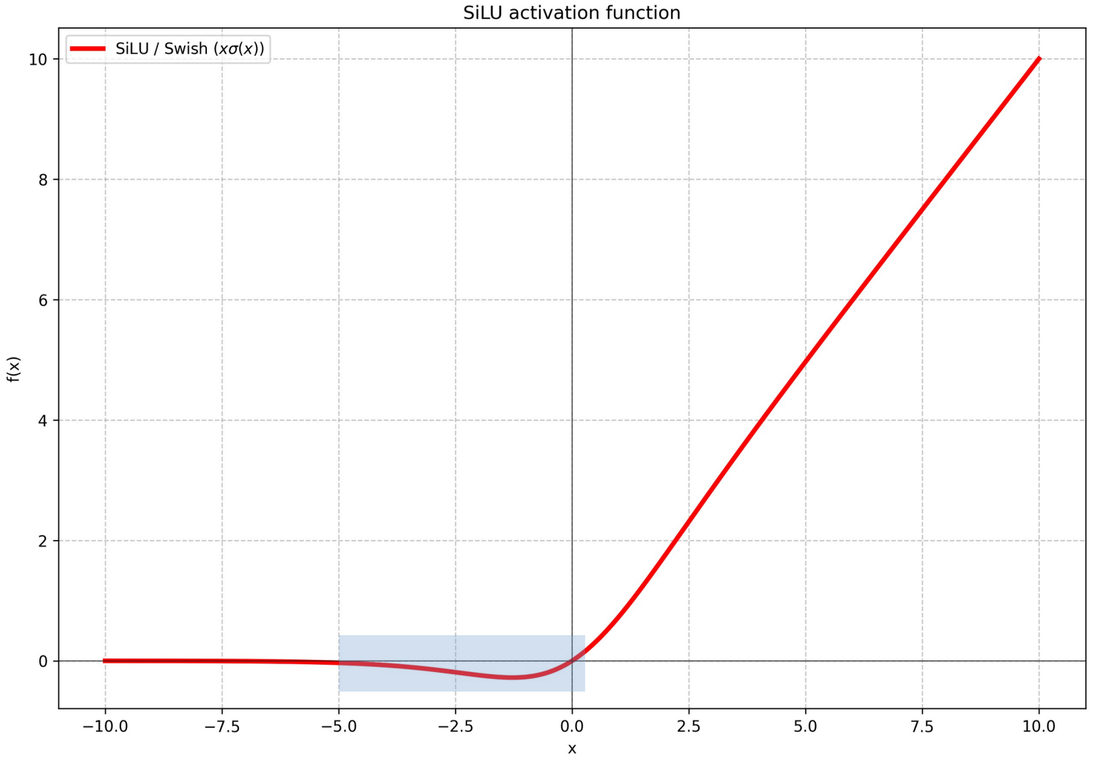
Activation functions

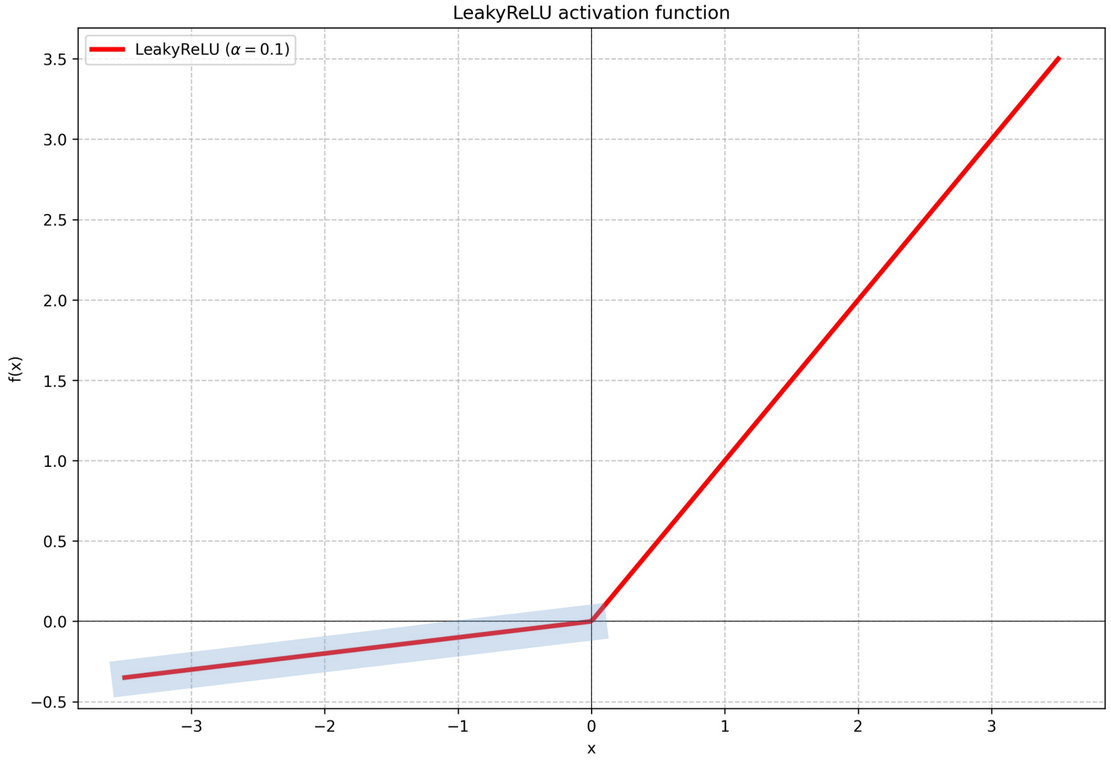
- The most **common practice** was to just use **ReLU** in deep learning.
- Now, every **fancy LLM** uses a different activation function.
- Why **ReLU** is not used much?
- The problem is that **negative values are clamped to 0**.
- This limits the **representational capacity** of ReLU.



Activation functions

- **LeakyReLU** mitigated the issue by allowing some negative values to pass through to the next layer.
- However, with **LeakyReLU**, negative values can continue to **grow uncontrollably**.
- **SiLU** and similar activation functions help by controlling the extent of negative value propagation.





Activation functions

- **SwiGLU** is an activation function that is used by the **Llama** model.
- **SwiGLU** is composed of 2 parts: **Swish** and **GLU**.
- **Swish** has been shown to outperform **ReLU** in many applications.
- **GLU** enables the network to **focus on important features** by either passing or blocking information.

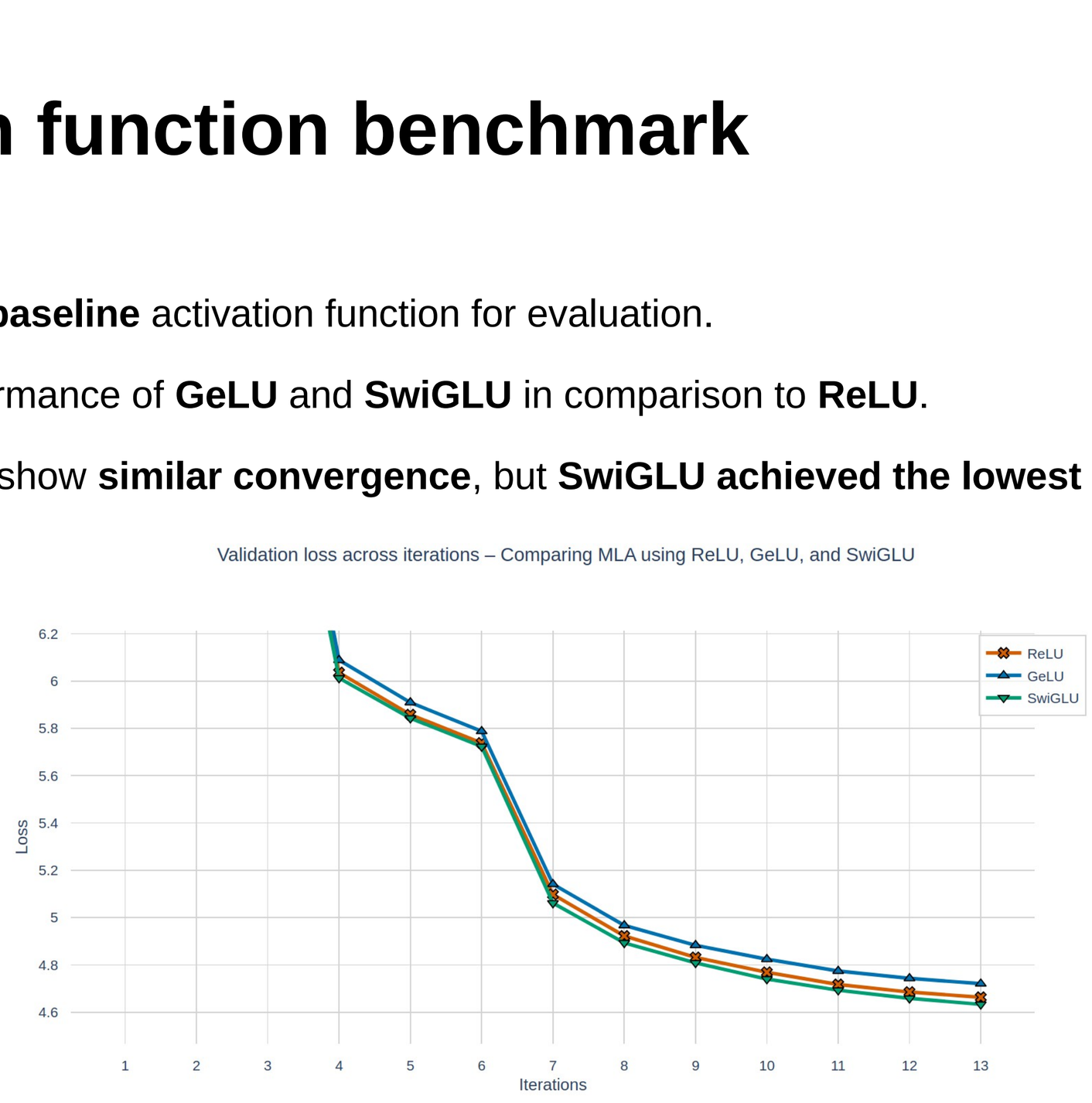
$$\text{GLU}(x) = (xW + b) \otimes \sigma(xV + c)$$

$$\text{Swish}_\beta(x) = x \cdot \sigma(\beta x)$$

$$\text{SwiGLU}(x) = (xW + b) \cdot \sigma(xW + b) \otimes (xV + c)$$

Activation function benchmark

- Use **ReLU** as the **baseline** activation function for evaluation.
- Evaluate the performance of **GeLU** and **SwiGLU** in comparison to **ReLU**.
- All three functions show **similar convergence**, but **SwiGLU achieved the lowest loss value.**

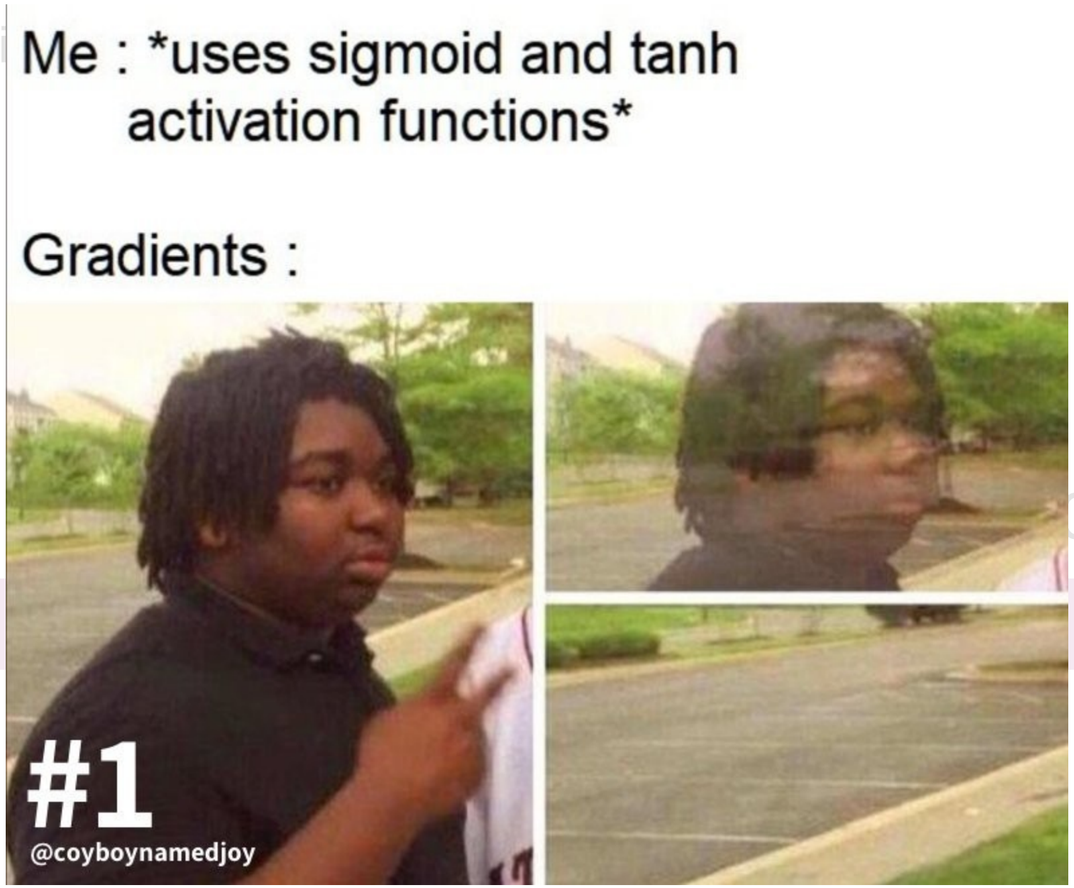


Normalization methods

- **Normalization** play a crucial role in **stabilizing and accelerating** the training process.
- It helps mitigate the issues of **vanishing gradients**.

Normalization methods

- **Normalization** play a cruci ng process.
- It helps mitigate the issues



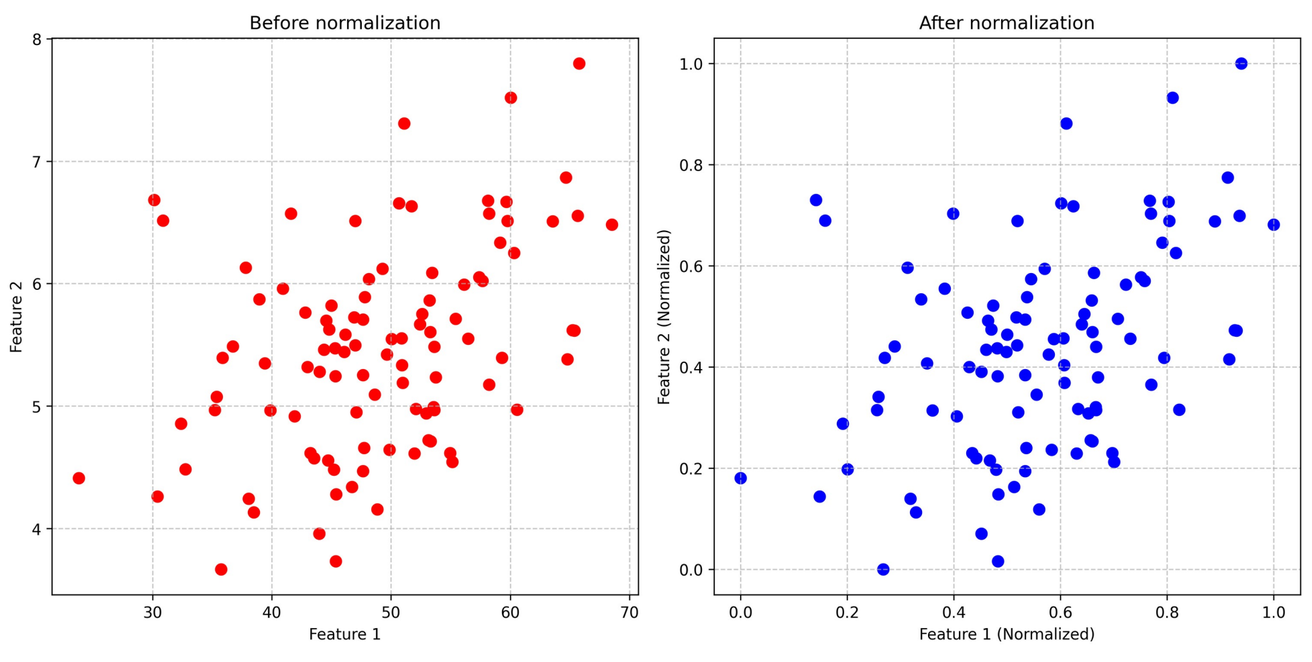
Input Output

Normalization methods

- **Normalization** play a crucial role in **stabilizing and accelerating** the training process.
- It helps mitigate the issues of **vanishing gradients and exploding gradients**.

Normalization methods

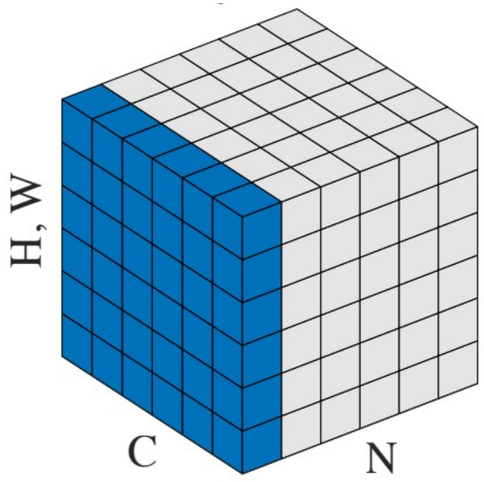
- **Normalization** play a crucial role in **stabilizing and accelerating** the training process.
- It helps mitigate the issues of **vanishing gradients and exploding gradients**.
- **Normalization** adjusts the **scale** of the data **without changing its shape**.



Normalization methods

- There are many normalization methods: **LayerNorm**, **BatchNorm**, **RMSNorm**, etc.
- **LayerNorm (LN)** is the method used in the **original Transformer**.
- **LN** normalizes the activations of each layer **across the feature dimension**.
- **LN** is suitable in scenarios where the **mini-batch size is small**.

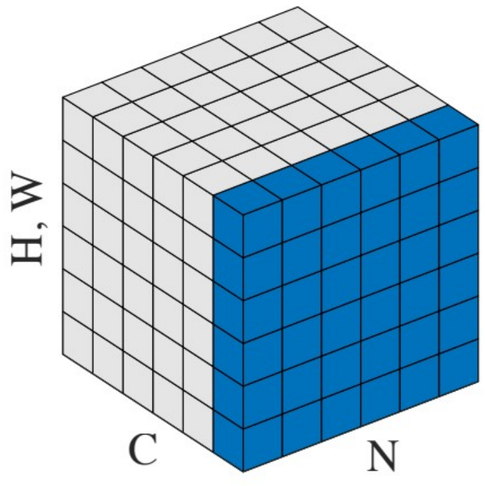
$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$



Normalization methods

- **BatchNorm (BN)** normalizes across the **mini-batch dimension**.
- **BN** uses **learnable parameters**, allowing the model to **adaptively scale and shift** the normalized activations.

$$y_i = \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$



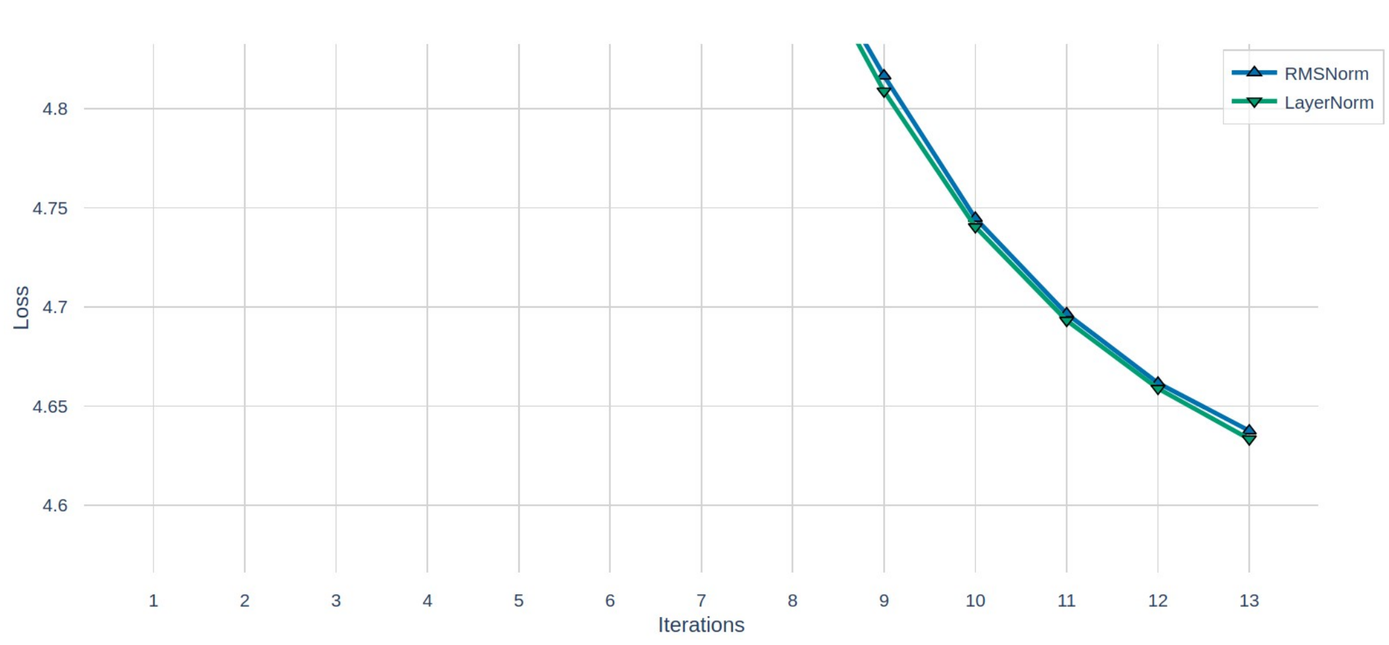
Normalization methods

- **RMSNorm** normalizes activations based on the **Root Mean Square** of the activations themselves.
- Unlike **LN**, **RMSNorm does not center the activations** before normalization.
- **RMSNorm** reduces computational complexity without sacrificing performance.

$$y_i = \frac{x_i}{\sqrt{\frac{1}{d}\sum_{j=1}^{d} x_j^2 + \epsilon}} g_i$$

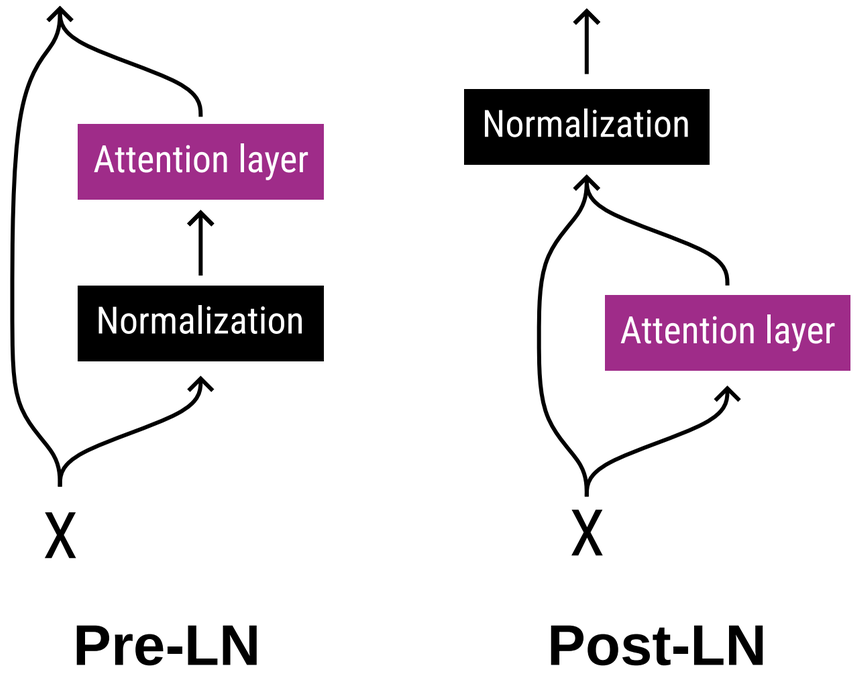
Normalization methods benchmark

- Use **LayerNorm** as the **baseline** normalization method for evaluation.
- Evaluate the performance of **RMSNorm** in comparison to **LayerNorm**.
- **LayerNorm** performed better by **0.07%**.



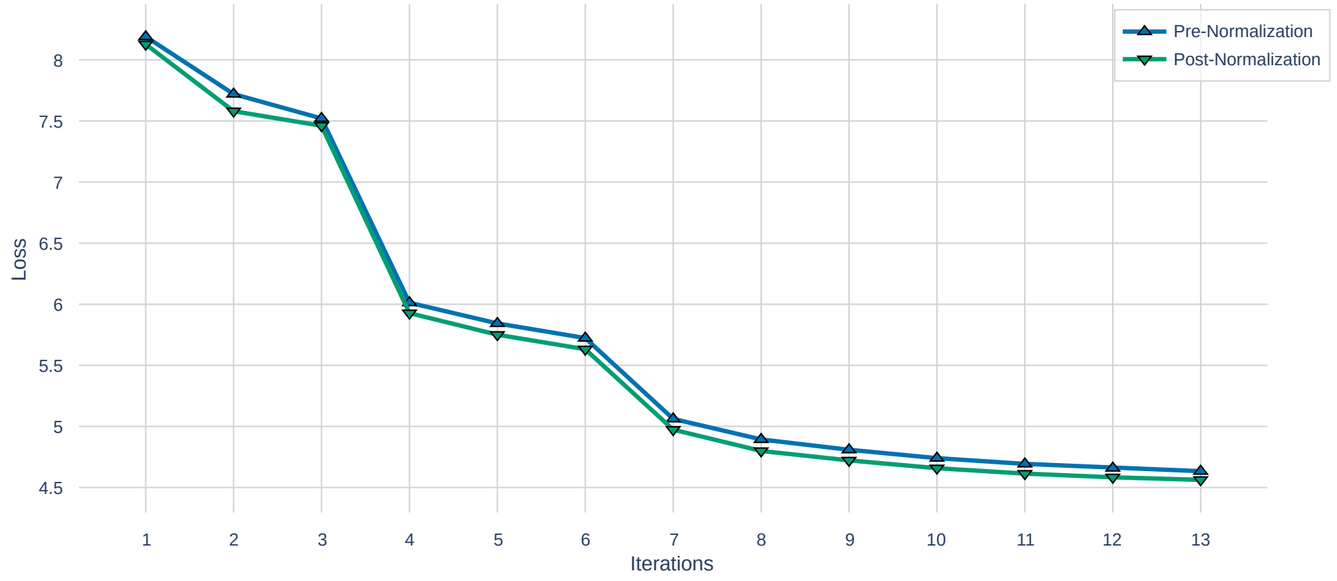
Normalization positioning

- **Normalization** can be placed before or after the **attention** or **feed forward** layers.
- **Post-LN** can encounter **stability issues** as the number of layers **grow**.
- **Post-LN** can achieve **better final performance** but finding the right hyper-parameters is **hard**.
- **Pre-LN** offers **better training stability**.
- **Pre-LN** is **less sensitive** to hyper-parameters choices.
- **Pre-LN** works well where the number of layers is big.



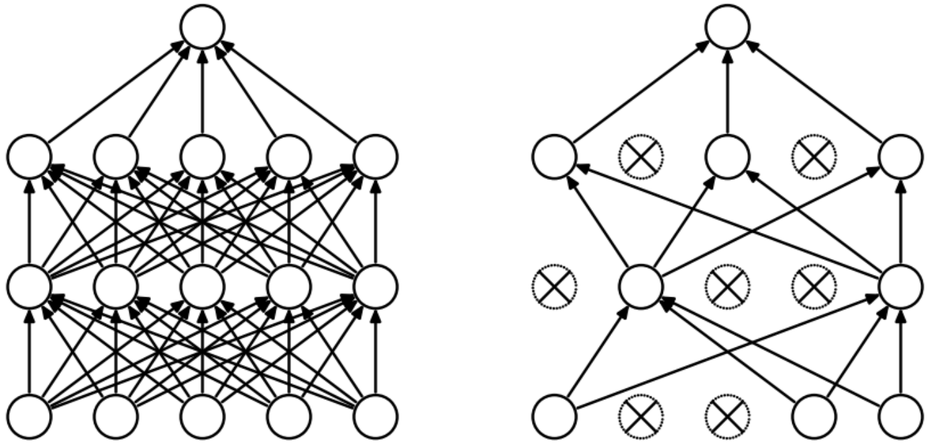
Normalization positioning benchmark

- Use **Pre-Normalization** as the **baseline** for evaluation.
- Evaluate the performance of **Post-Normalization** in comparison to **Pre-Normalization**.
- **Post-Normalization** performed better than **Pre-Normalization**.



Dropout

- **Dropout** is useful when you want to **avoid overfitting**.
- **Dropout** should be used if **training on a small dataset for few epochs**.
- **LLM**s train for **one epoch** because the **dataset size is enormous**. In this case, dropout is not needed.



(a) Standard Neural Net (b) After applying dropout.

Read the paper

Dropout benchmark

- Use **Dropout** as the **baseline** for evaluation.
- Evaluate the performance of **No Dropout** in comparison to **Dropout**.
- **No Dropout** performed better than **Dropout**.