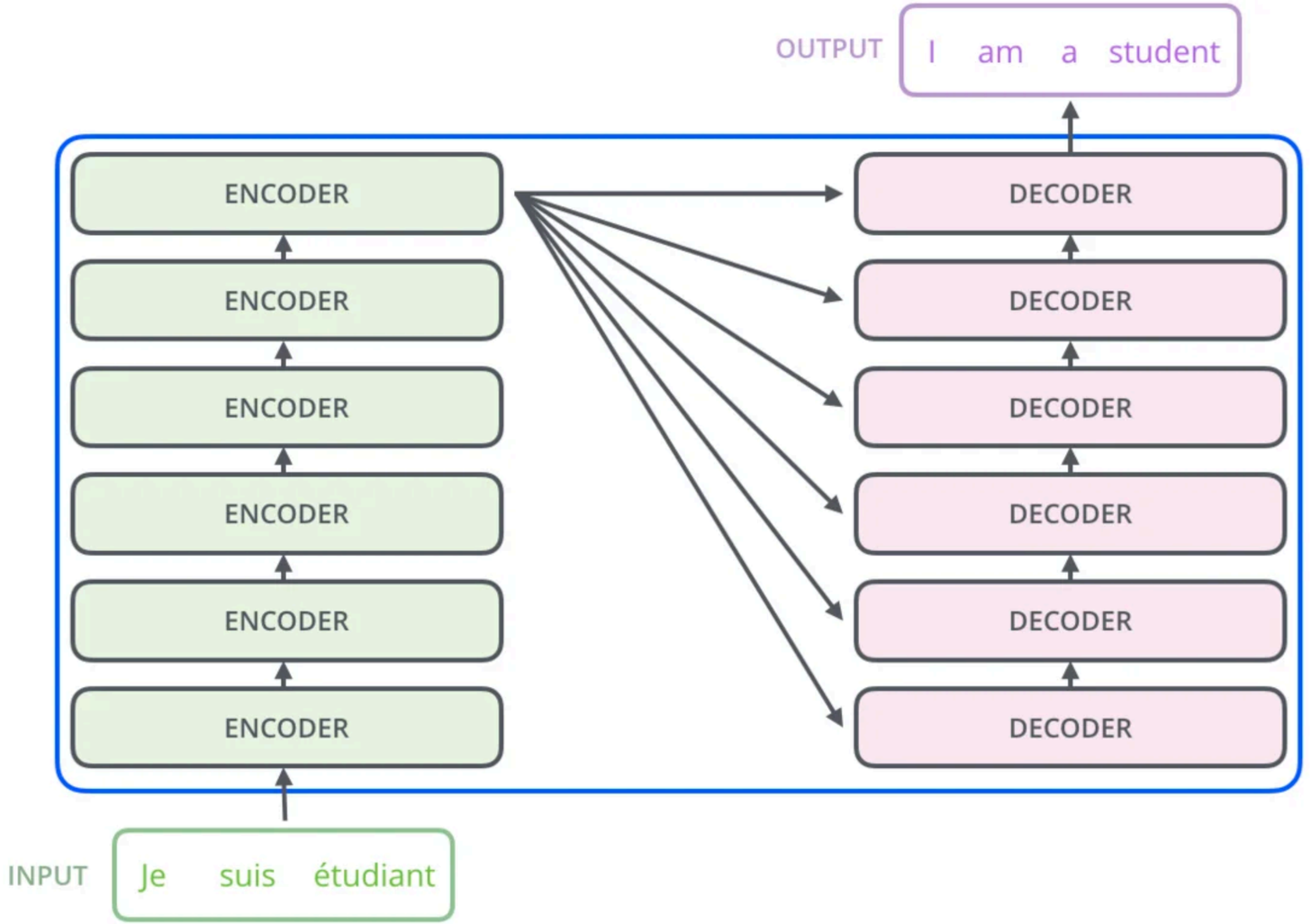
Reformer

The Efficient Transformer

Pavel Fakanov

Transformer

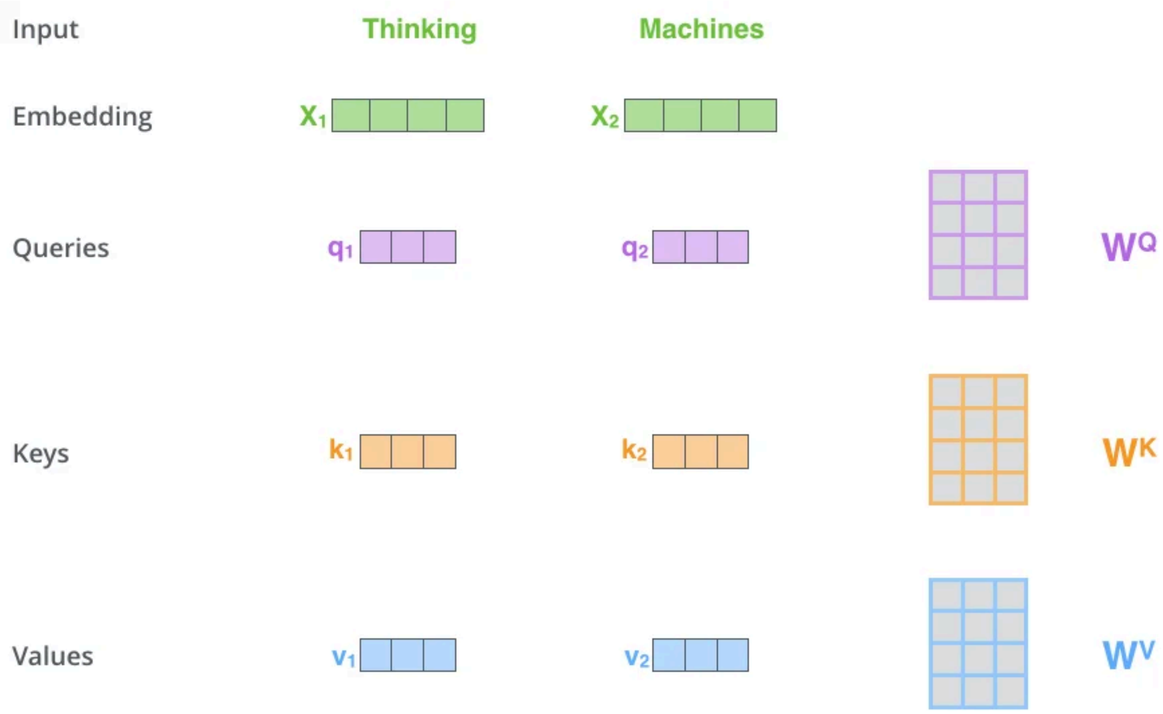
A High-Level Look



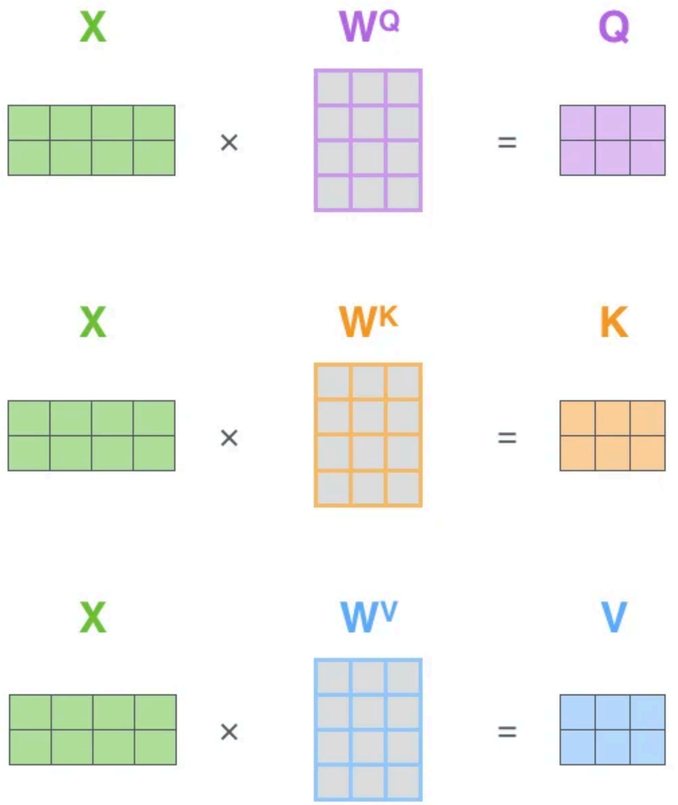
Self-Attention

Keys, Queries, Values

Single tensors

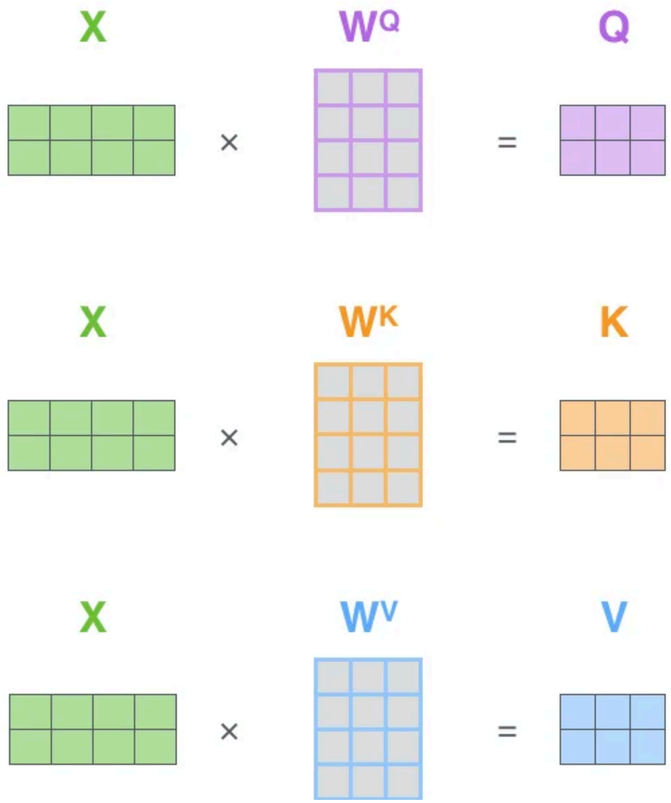


Matrix level



Self-Attention

Query, Key, Value matrixes



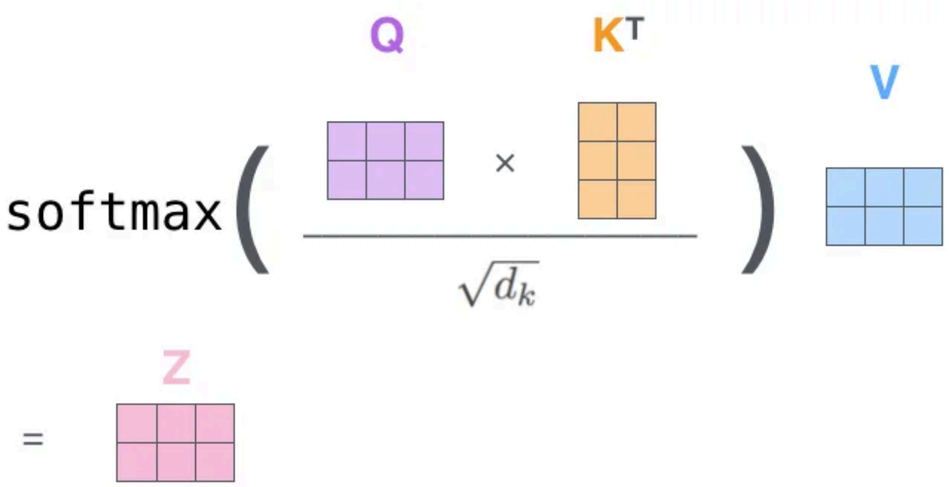
Self-Attention matrix

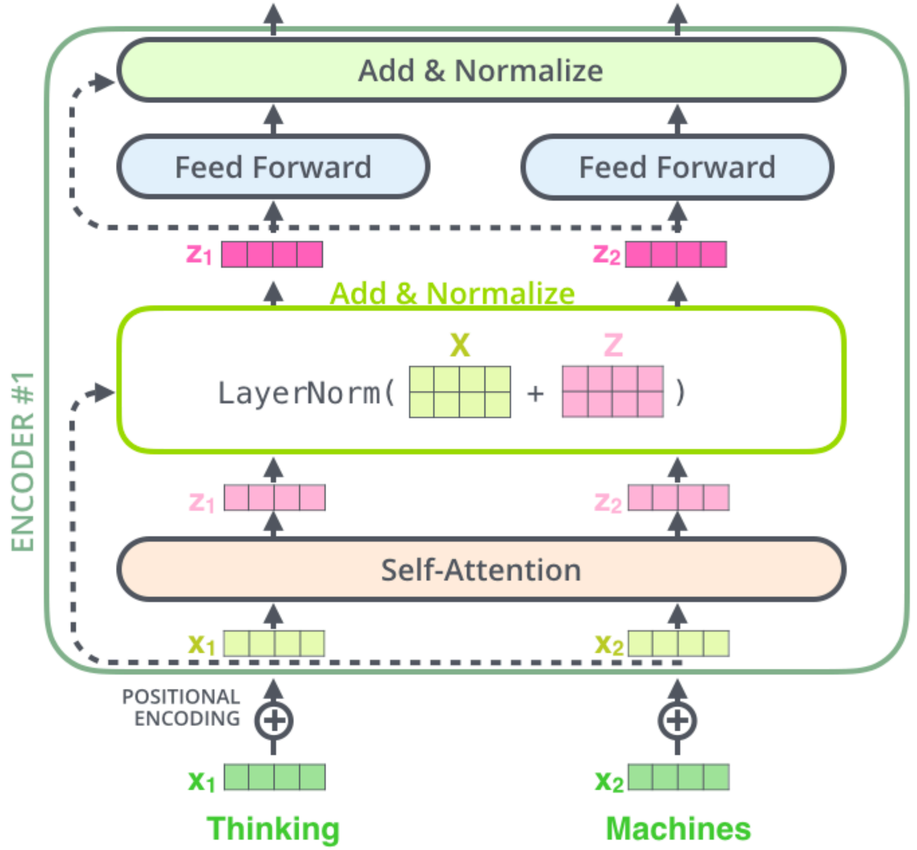
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

What is the complexity of self-attention operation?

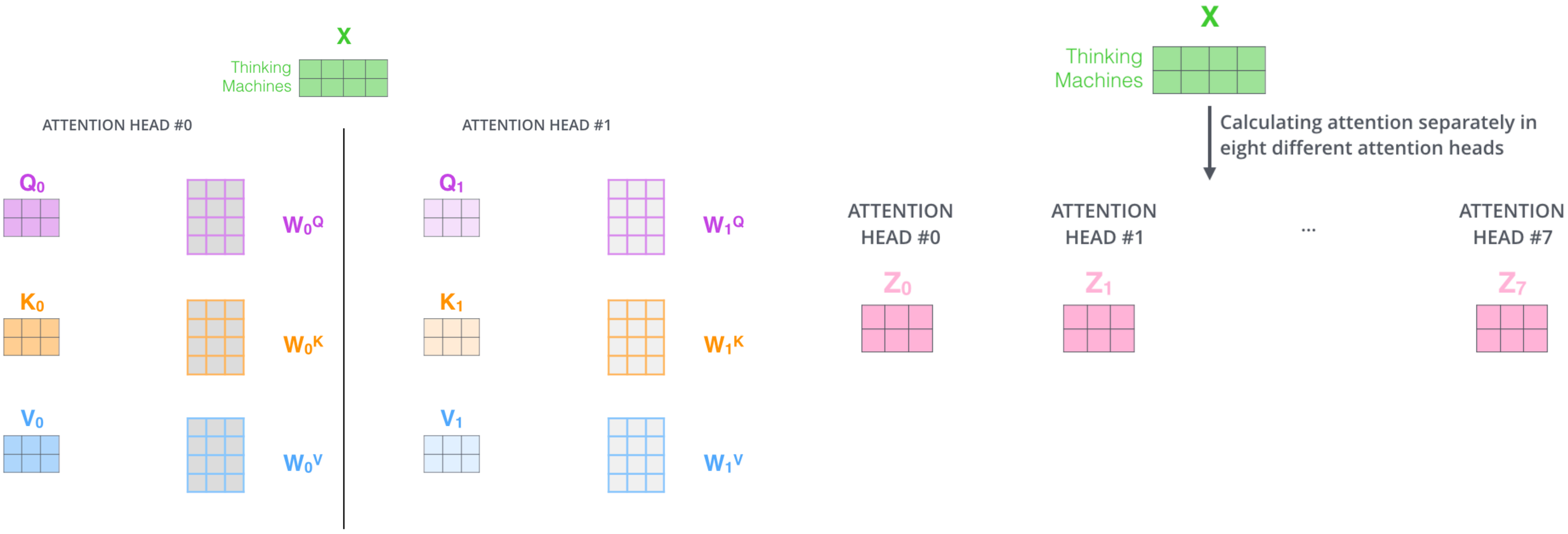
Encoder

Architecture





Multi-head attention



Multi-head attention

1) This is our input sentence*

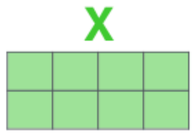
2) We embed each word*

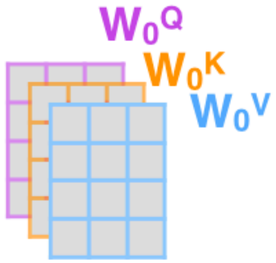
3) Split into 8 heads. We multiply X or R with weight matrices

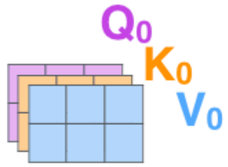
4) Calculate attention using the resulting Q/K/V matrices

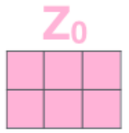
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

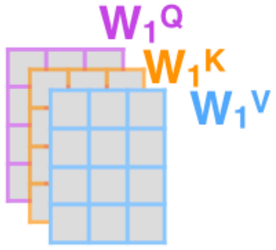
Thinking Machines

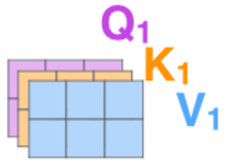


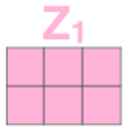


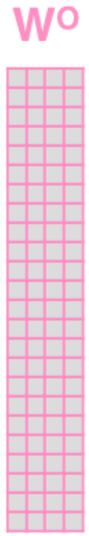


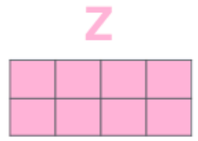






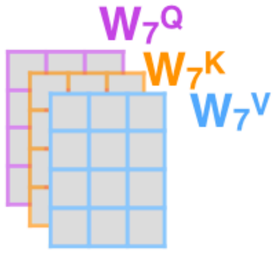




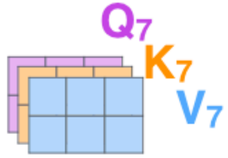


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

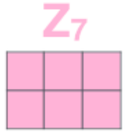
...



...



...



Transformer

Number of parameters

- 5 billion parameters
- 2 GB of memory

Mesh-TensorFlow: Deep Learning for Supercomputers

Noam Shazeer, Youlong Cheng, Niki Parmar,
Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee
Mingsheng Hong, Cliff Young, Ryan Sepassi, Blake Hechtman
Google Brain
{noam, ylc, nikip, trandustin, avaswani, penporn, phawkins,
hyouklee, hongm, cliffy, rsepassi, blakehechtman}@google.com

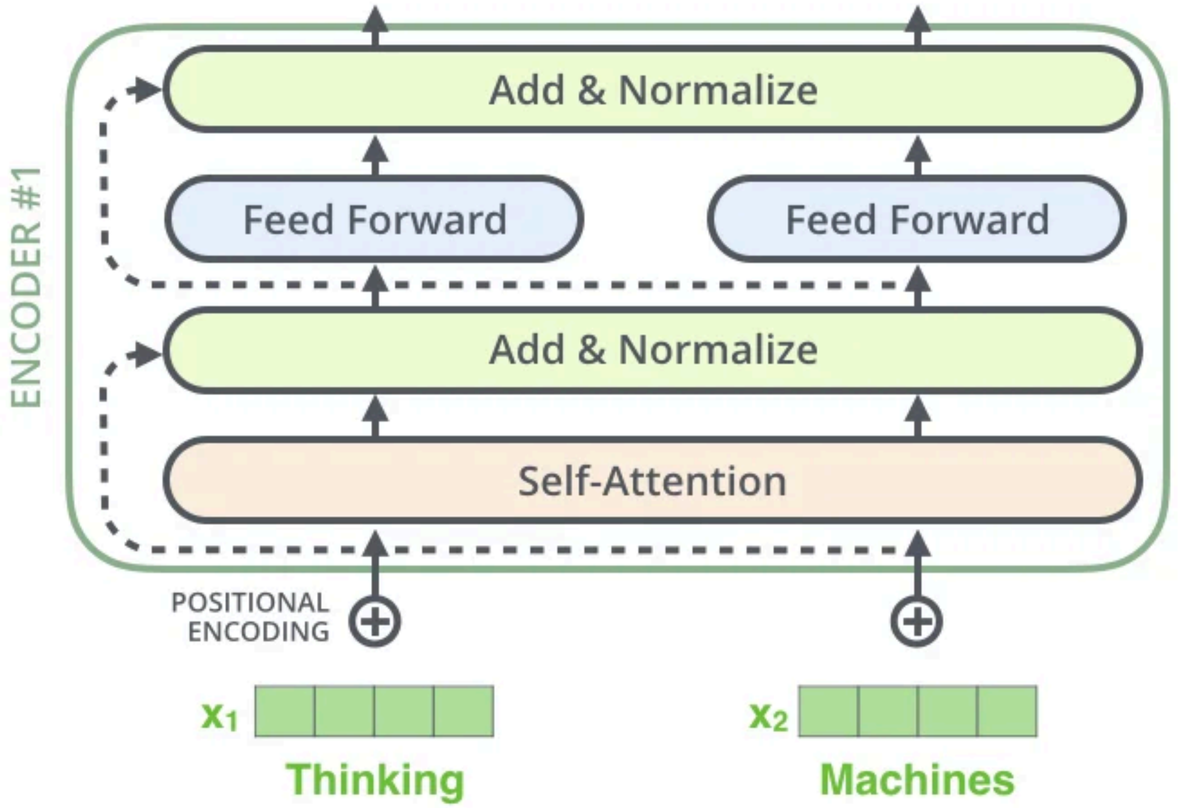
Abstract

Batch-splitting (data-parallelism) is the dominant distributed Deep Neural Network (DNN) training strategy, due to its universal applicability and its amenability to Single-Program-Multiple-Data (SPMD) programming. However, batch-splitting suffers from problems including the inability to train very large models (due to memory constraints), high latency, and inefficiency at small batch sizes. All of these can be solved by more general distribution strategies (model-parallelism). Unfortunately, efficient model-parallel algorithms tend to be complicated to discover, describe, and to implement, particularly on large clusters. We introduce Mesh-TensorFlow, a language for specifying a general class of distributed tensor computations. Where data-parallelism can be viewed as splitting tensors and operations along the "batch" dimension, in Mesh-TensorFlow, the user can specify any tensor-dimensions to be split across any dimensions of a multi-dimensional mesh of processors. A Mesh-TensorFlow graph compiles into a SPMD program consisting of parallel operations coupled with collective communication primitives such as Allreduce. We use Mesh-TensorFlow to implement an efficient data-parallel, model-parallel version of the Transformer [21] sequence-to-sequence model. Using TPU meshes of up to 512 cores, we train Transformer models with up to 5 billion parameters, surpassing state of the art results on WMT'14 English-to-French translation task and the one-billion-word language modeling benchmark. Mesh-Tensorflow is available at https://github.com/tensorflow/mesh .

Encoder

Memory resources for one layer

- Batch size: 8
- Sequence length: 64 K
- Embedding size: 1024



Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences

Encoder

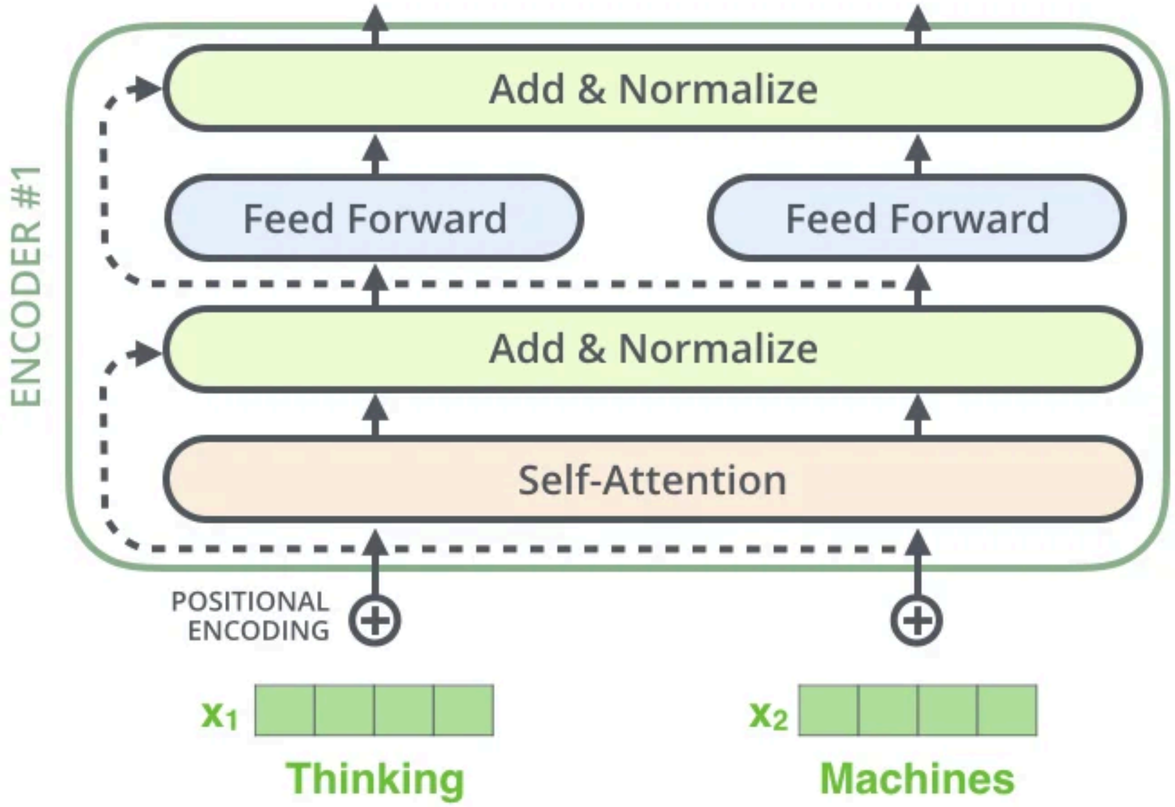
Memory resources for one layer

- Batch size: 8
- Sequence length: 64 K
- Embedding size: 1024

$64k \times 1k \times 8 = 0.5B$ floats

2 Gb of memory

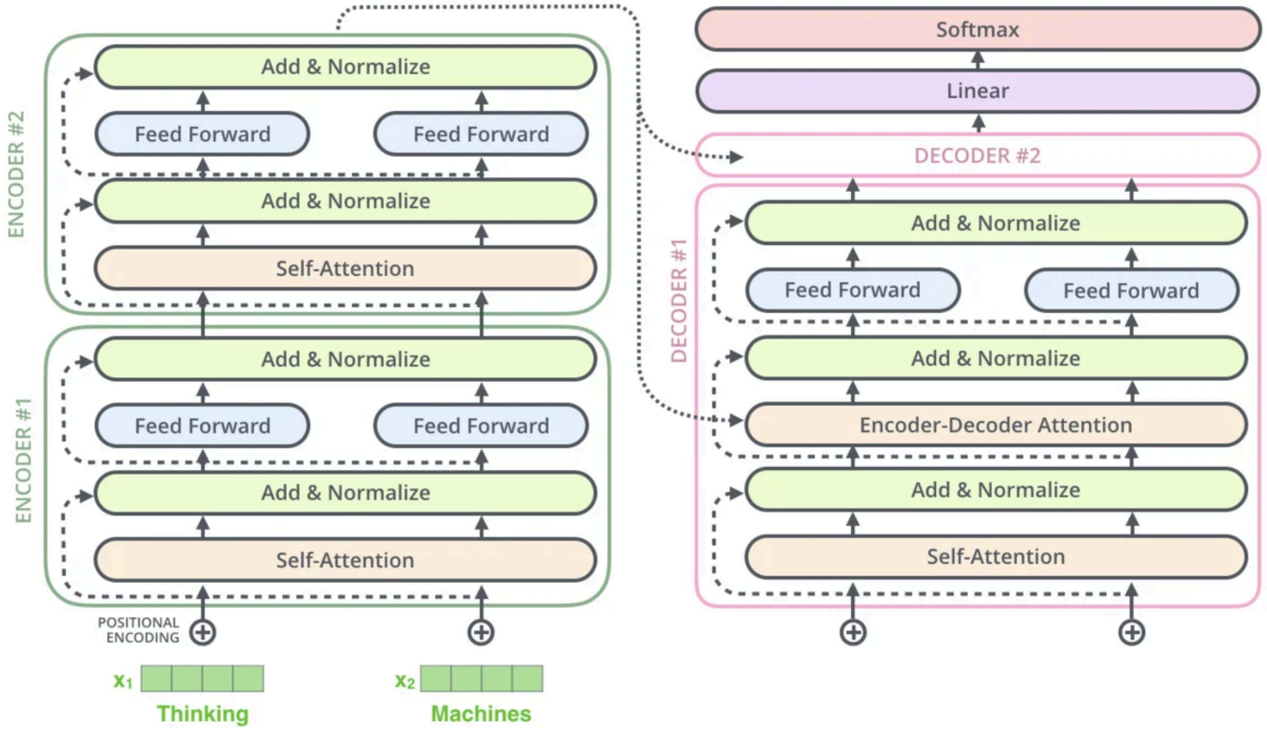
So what is the problem?



Transformer

Fair memory calculation

- Store activations on each of N layers for back-propagation
- Activations for fully-connected layers
- Attention operation - $O(L^2)$ both in computational and memory



Transformer

- Store activations on each of N layers for back-propagation
- Activations for fully-connected layers
- Attention operation - $O(L^2)$

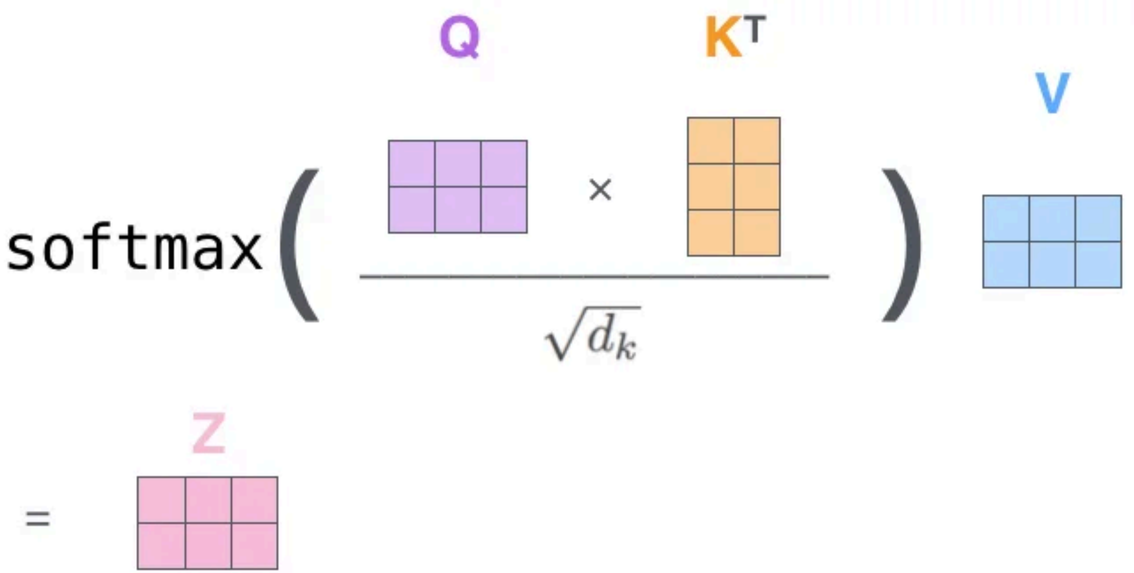
Reformer

- Reversible layers, enable storing only a single copy of activations
- Splitting activations and processing them in chunks
- Attention operation - $O(L \, log \, L)$

Memory-efficient Attention

- $Q: [batch\_size, length, d_{model}]$
- $K: [batch\_size, length, d_{model}]$
- $QK^T: [batch\_size, length, length]$

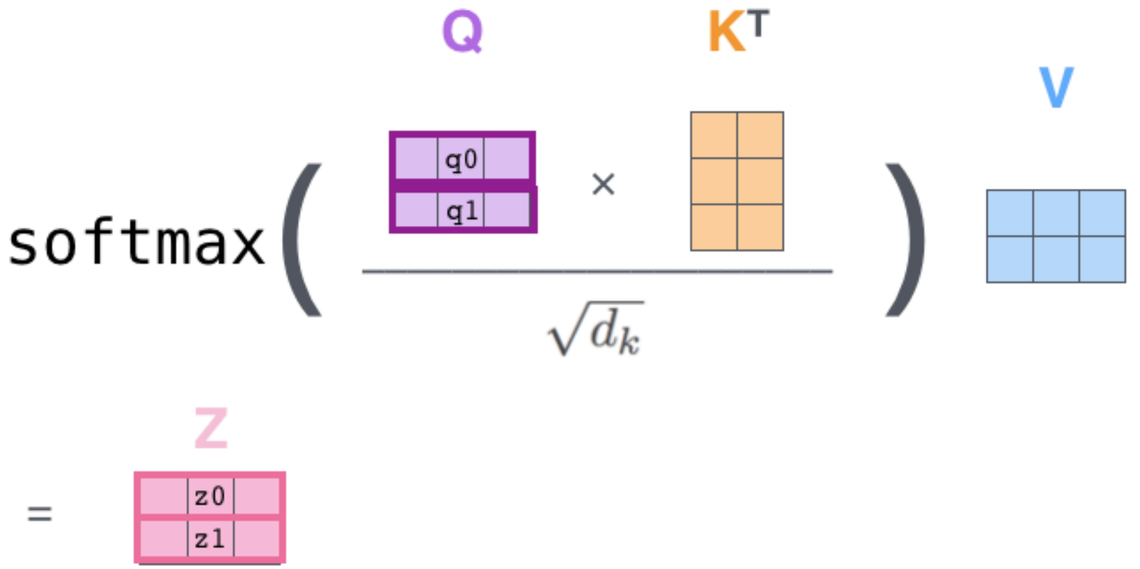
$batch\_size = 1 : 64\text{K} \times 64\text{K}$ matrix
16Gb of memory



Memory-efficient Attention

- $Q: [batch\_size, length, d_{model}]$
- $K: [batch\_size, length, d_{model}]$
- $QK^T: [batch\_size, length, length]$

$batch\_size = 1 : 64\text{K} \times 64\text{K}$ matrix
16Gb of memory

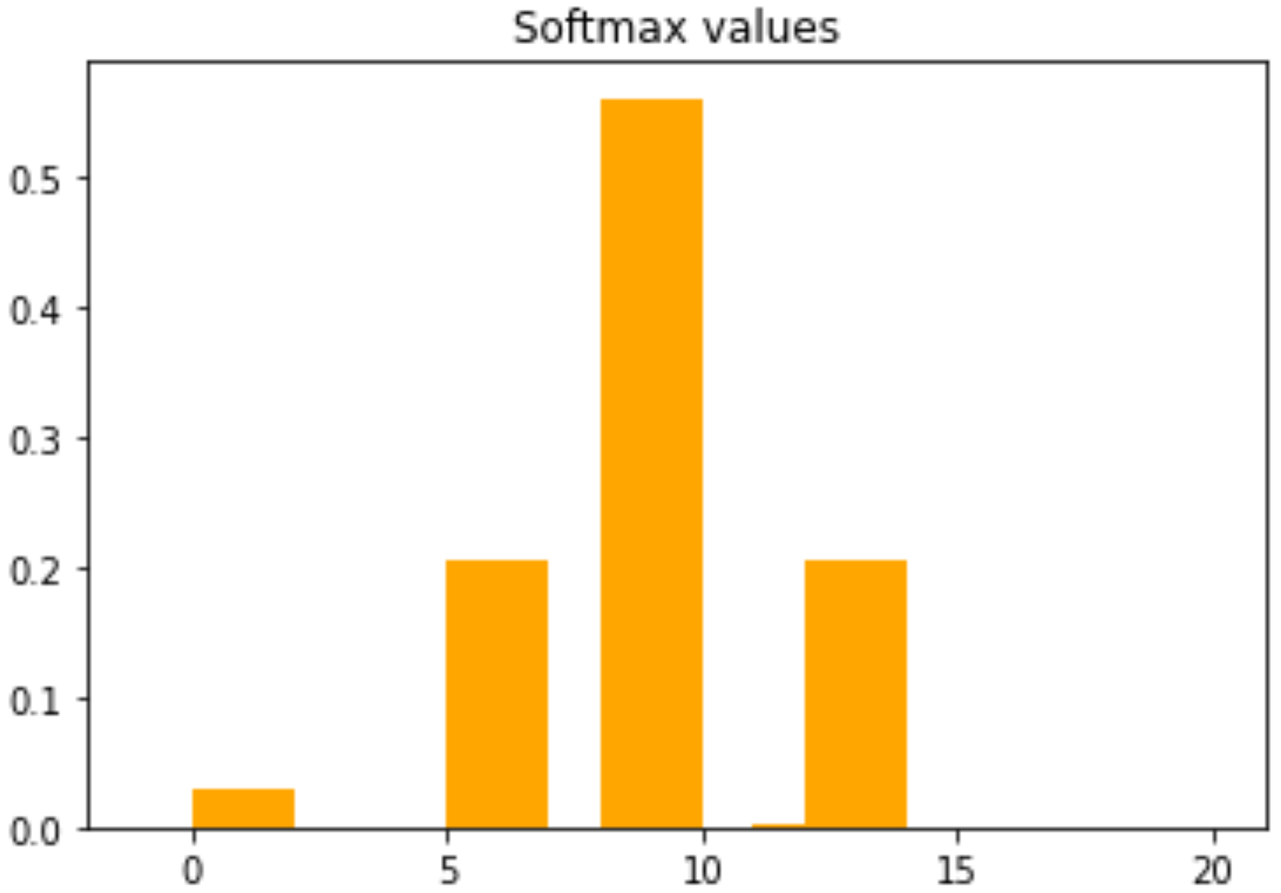


Hashing attention

Idea

- Consider only a small subset of closest keys

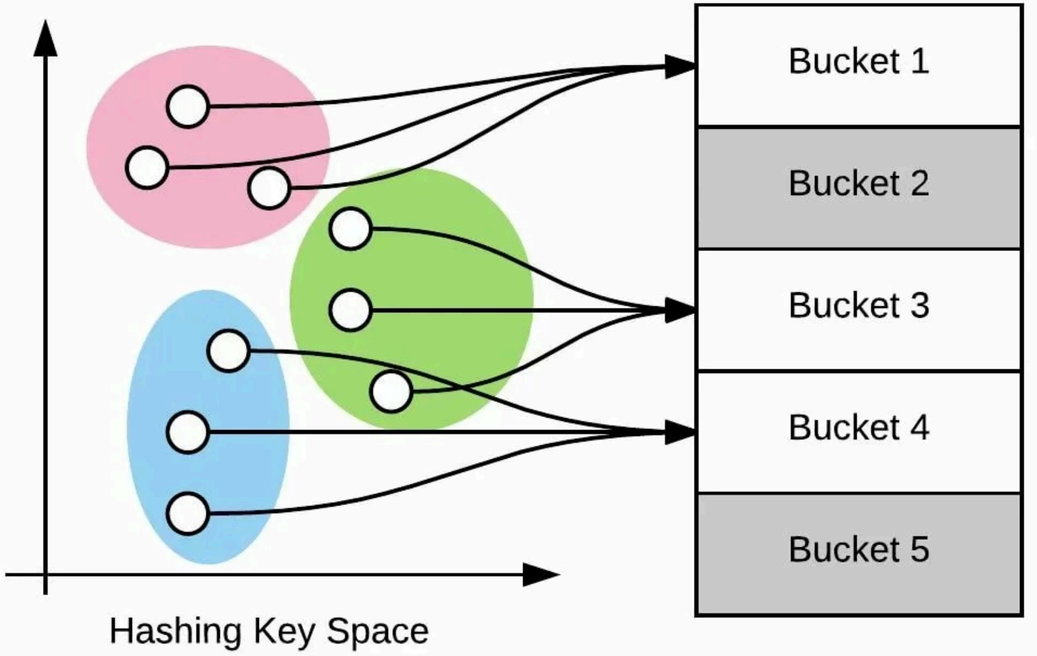
- How to find the closest neighbors for each q_i ?



Locality sensitive hashing

A family $\mathbb{H}$ *is called* $(S_0, cS_0, p_1, p_2) - sensitive$ *if for any two points* $x, y \in R^d$ *and* h *choosen uniformly from* $\mathbb{H}$ *satisfies the following :*

- *if* $Sim(x, y) \geq S_0$ *then* $Pr(h(x) = h(y)) \geq p_1$
- *if* $Sim(x, y) \leq cS_0$ *then* $Pr(h(x) = h(y)) \leq p_2$



Locality sensitive hashing

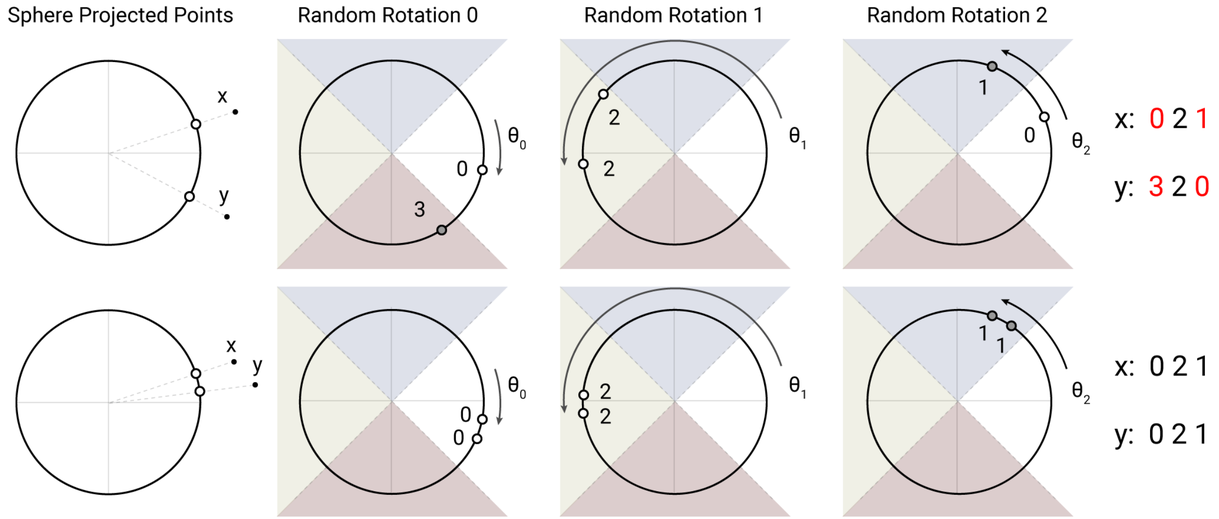
Details

$b - number\ of\ buckets$

$R\ -\ random\ matrix\ of\ size\ [d_k, b/2]$

$h(x)\ =\ argmax([-xR; xR])$

$[u; v]-$ concatenation of two vectors



Locality Sensitive hashing

Formalization

- Normal attention equation:

$$o_i = \sum_{j \in \mathscr{P}_i} \exp\left(q_i \cdot k_j - z\left(i, \mathscr{P}_i\right)\right) v_j \qquad \text{where } \mathscr{P}_i = \{j : i \geq j\}$$

Locality Sensitive hashing

Formalization

- Normal attention equation:

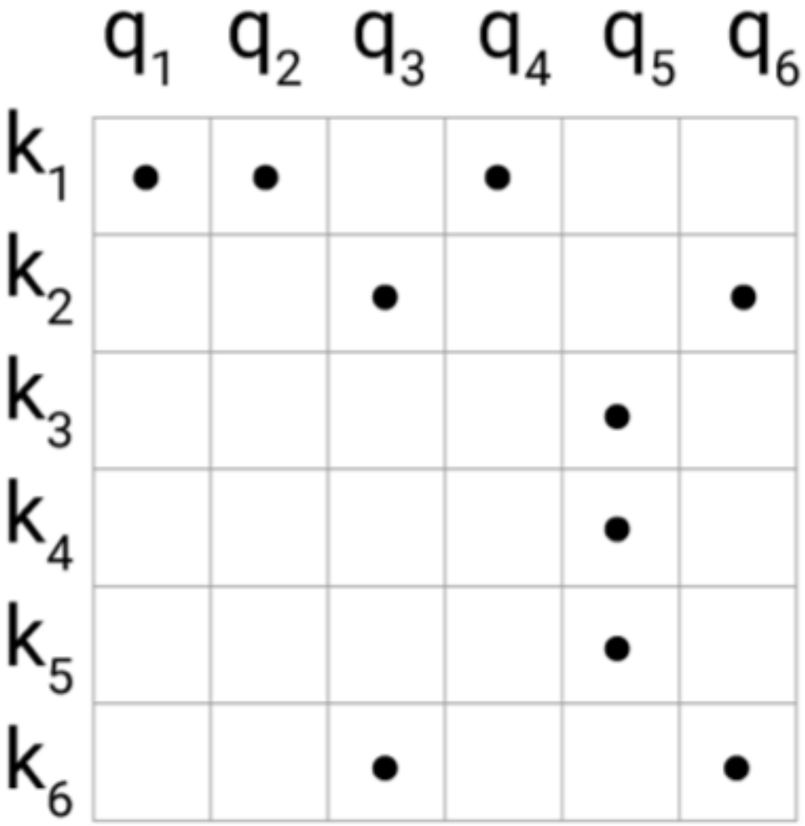
$$o_i = \sum_{j \in \mathscr{P}_i} \exp\left(q_i \cdot k_j - z\left(i, \mathscr{P}_i\right)\right) v_j \qquad \text{where } \mathscr{P}_i = \{j : i \geq j\}$$

- LSH attention:

$$o_i = \sum_{j \in \mathscr{P}_i} \exp\left(q_i \cdot k_j - z\left(i, \mathscr{P}_i\right)\right) v_j \qquad \text{where } \mathscr{P}_i = \{j : h(q_i) = h(k_j)\}$$

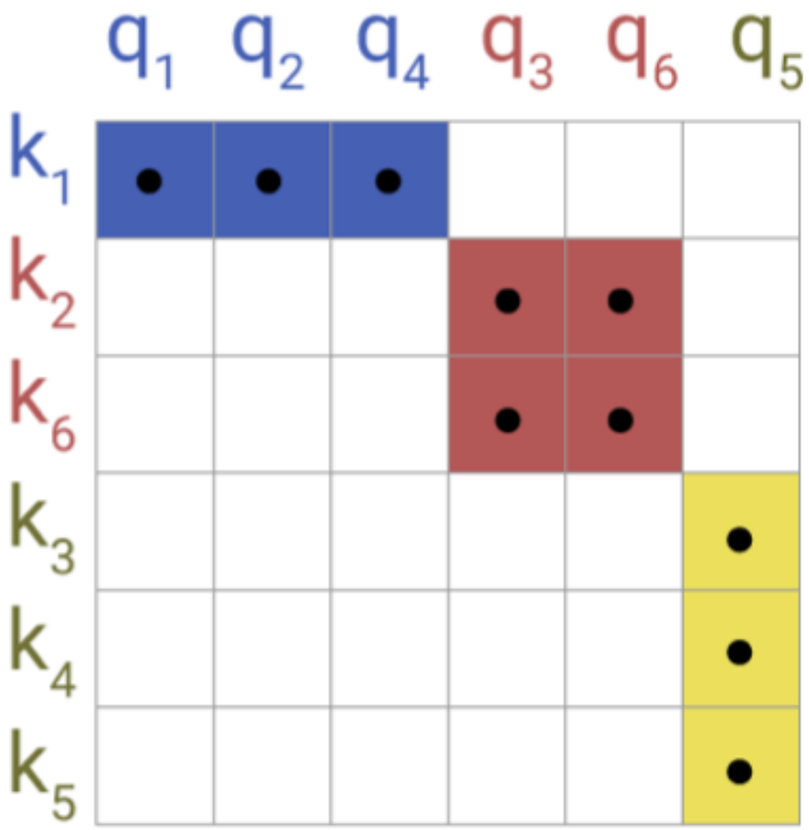
LSH-Attention

Full-attention matrix



Full-attention is typically sparse, but the computation does not take advantage of this sparsity

LSH-attention matrix



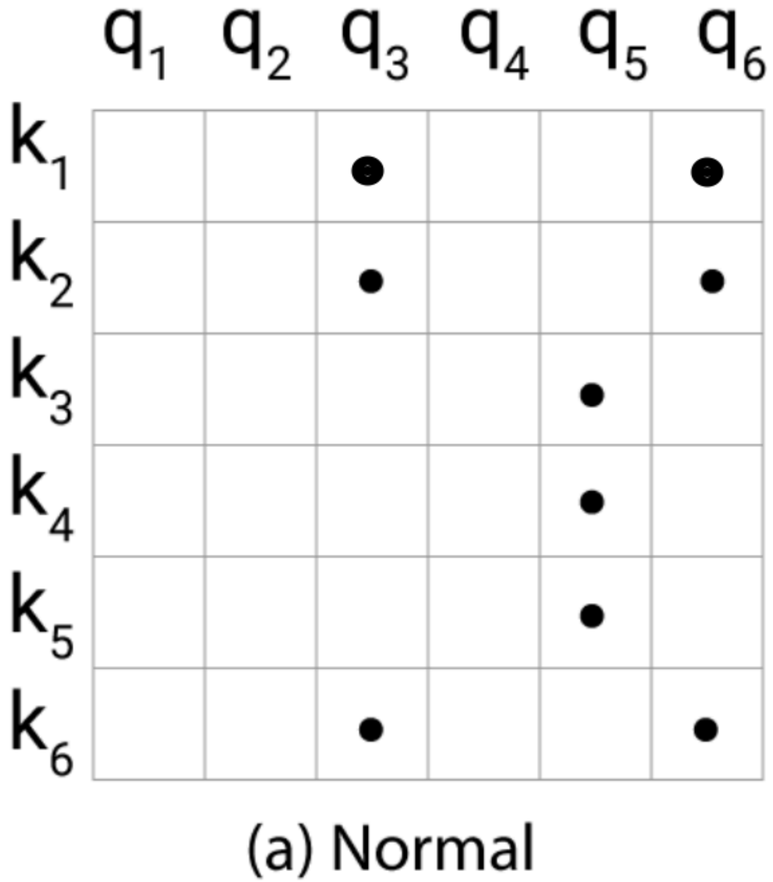
Queries and keys have been sorted according to their hash bucket

LSH-Attention

Difficulties

- There is no guarantee that the number of keys and the number of queries will be equal within a bucket

- Moreover, it is possible a bucket to contain many queries but no keys



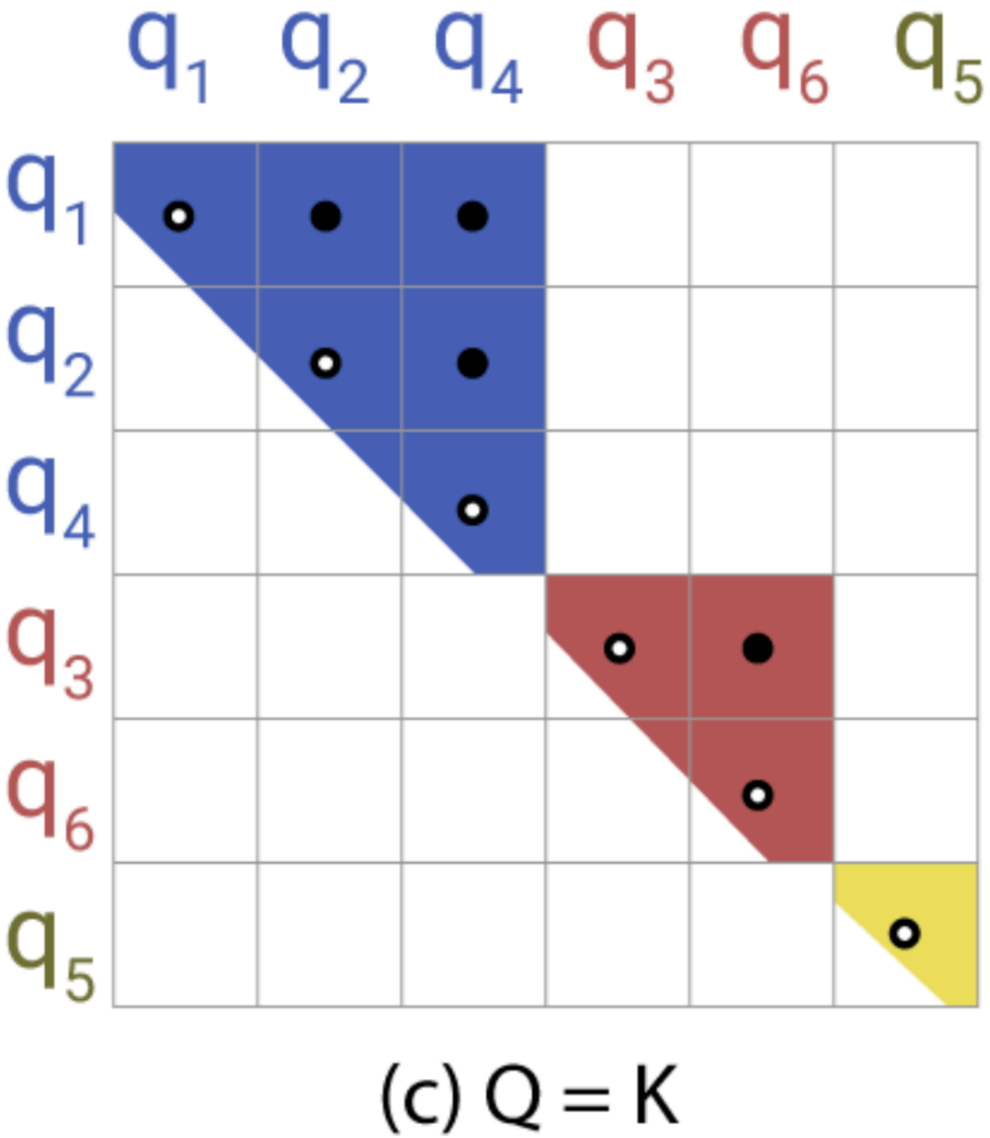
(a) Normal

LSH-Attention

Shared-QK

$$Ensure\ h(k_j) = h(q_j),\ by\ setting\ k_j = \frac{q_j}{||q_j||}$$

Sort the queries by bucket number and, within each bucket, by sequence position



Therefore, we want queries and keys (Q and K) to be identical

a model that behaves like this a *shared-QK Transformer*

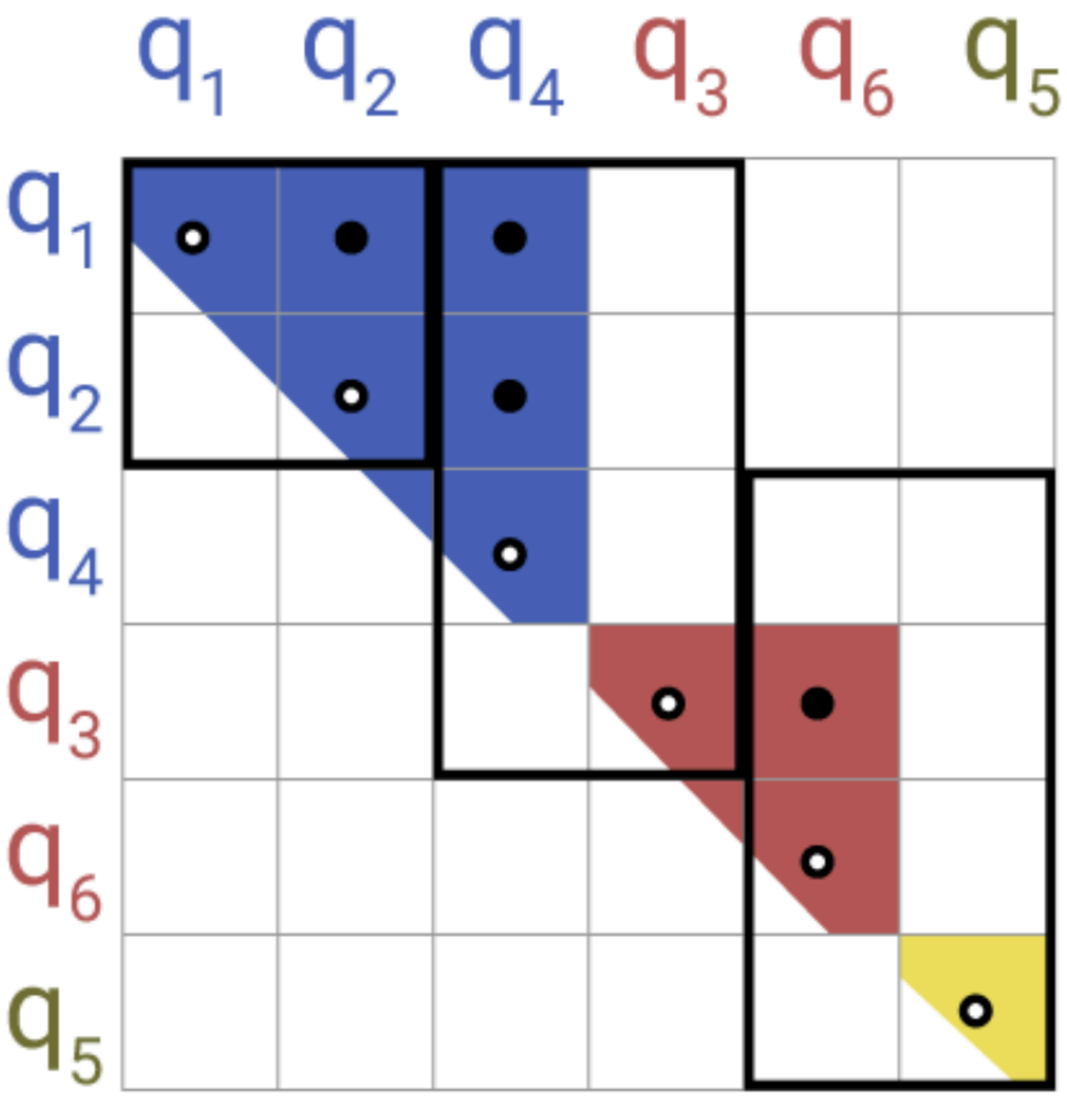
LSH-Attention

Chunks

We can follow a batching approach where chunks of m consecutive queries (after sorting) attend to each other, and one chunk back

In practice we set $m = \frac{2seq\_len}{n_{buckets}}$

Average bucket size is $\frac{seq\_len}{n_{buckets}}$



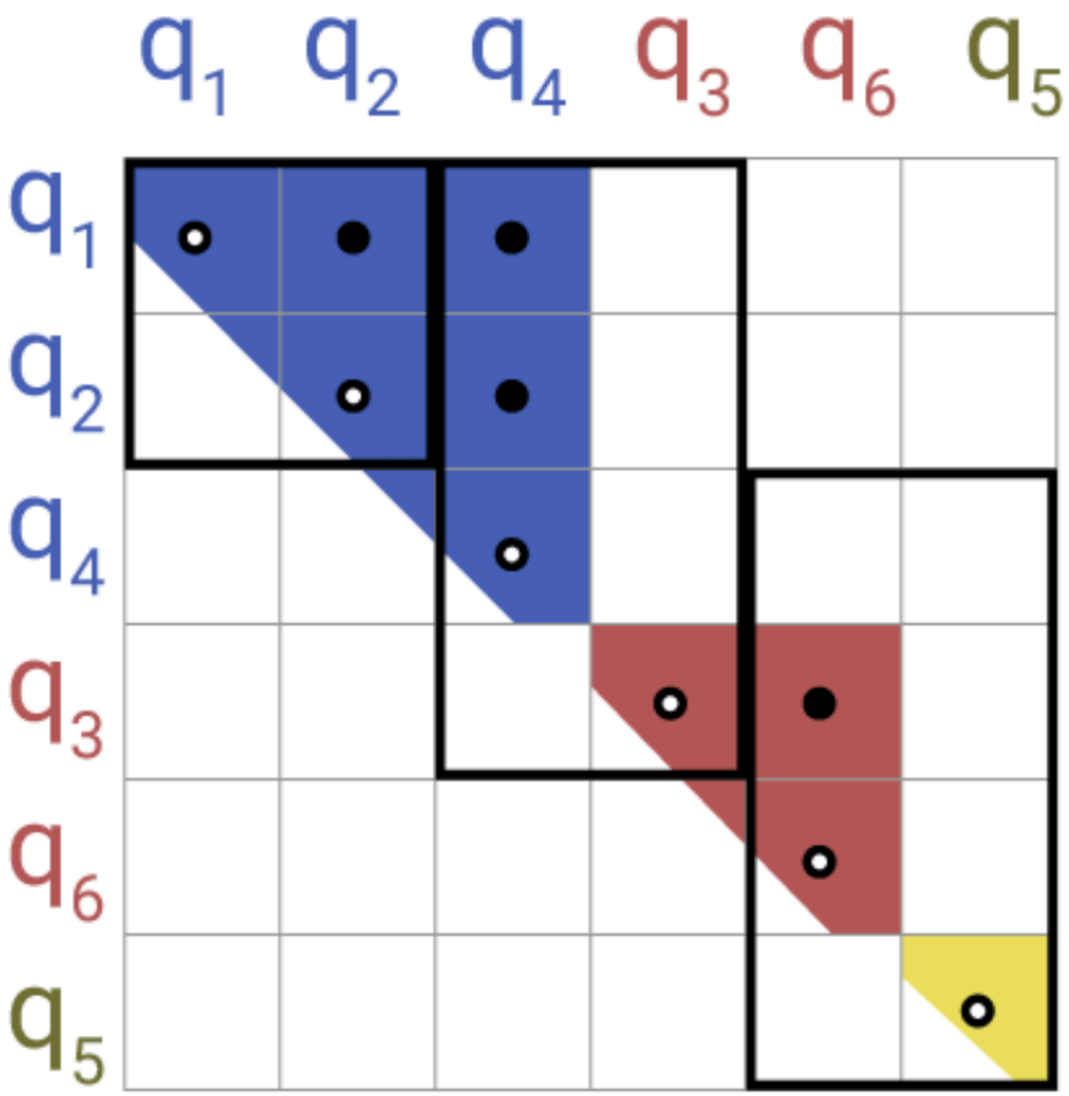
This allows us to suppose that the whole bucket will fit into one chunk

LSH-Attention

Chunks. Formalization

Consider a permutation given by sorting:
$i \mapsto s_i$

i	1	2	3	4	5	6
s_i	1	2	4	3	6	5



LSH-Attention

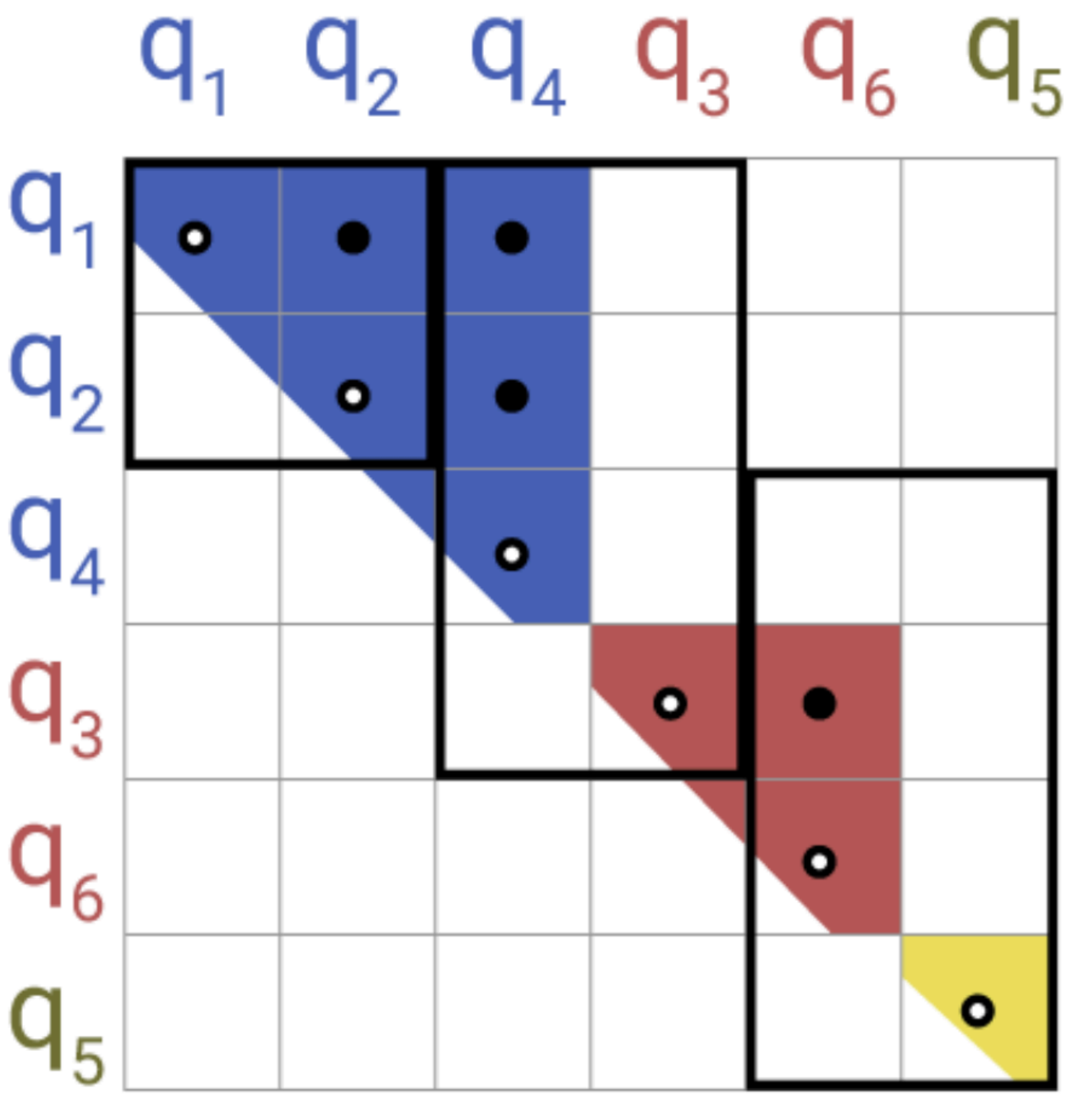
Chunks. Formalization

Consider a permutation given by sorting:
$i \mapsto s_i$

i	1	2	3	4	5	6
s_i	1	2	4	3	6	5

$$\mathcal{P}_i = \{j : h(q_i) = h(k_j)\}$$

$$\widetilde{\mathcal{P}}_i = \left\{ j : \left\lfloor \frac{s_i}{m} \right\rfloor - 1 \leq \left\lfloor \frac{s_j}{m} \right\rfloor \leq \left\lfloor \frac{s_i}{m} \right\rfloor \right\}$$



LSH-Attention

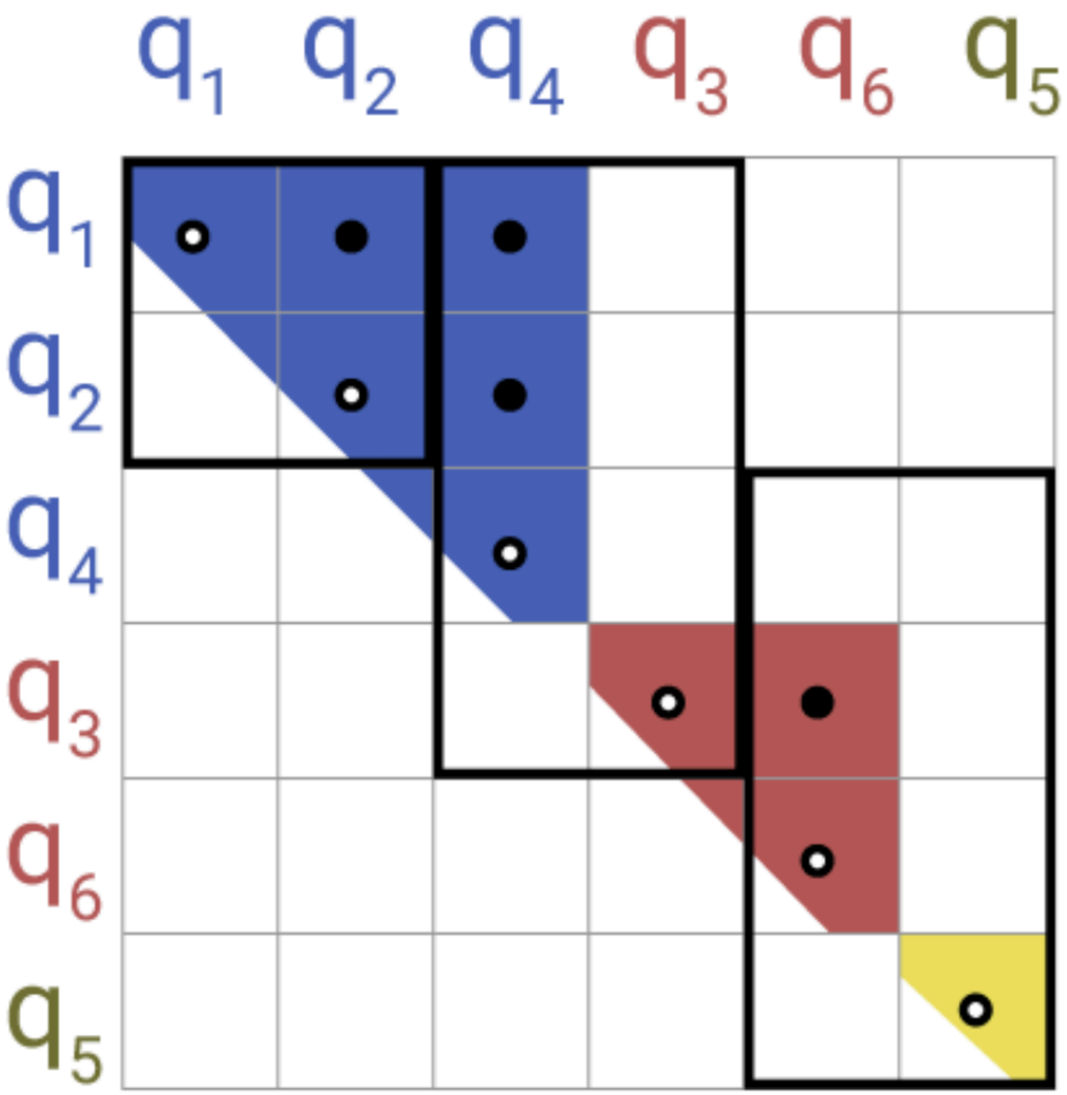
Chunks. Formalization

Consider a permutation given by sorting:
$i \mapsto s_i$

i	1	2	3	4	5	6
s_i	1	2	4	3	6	5

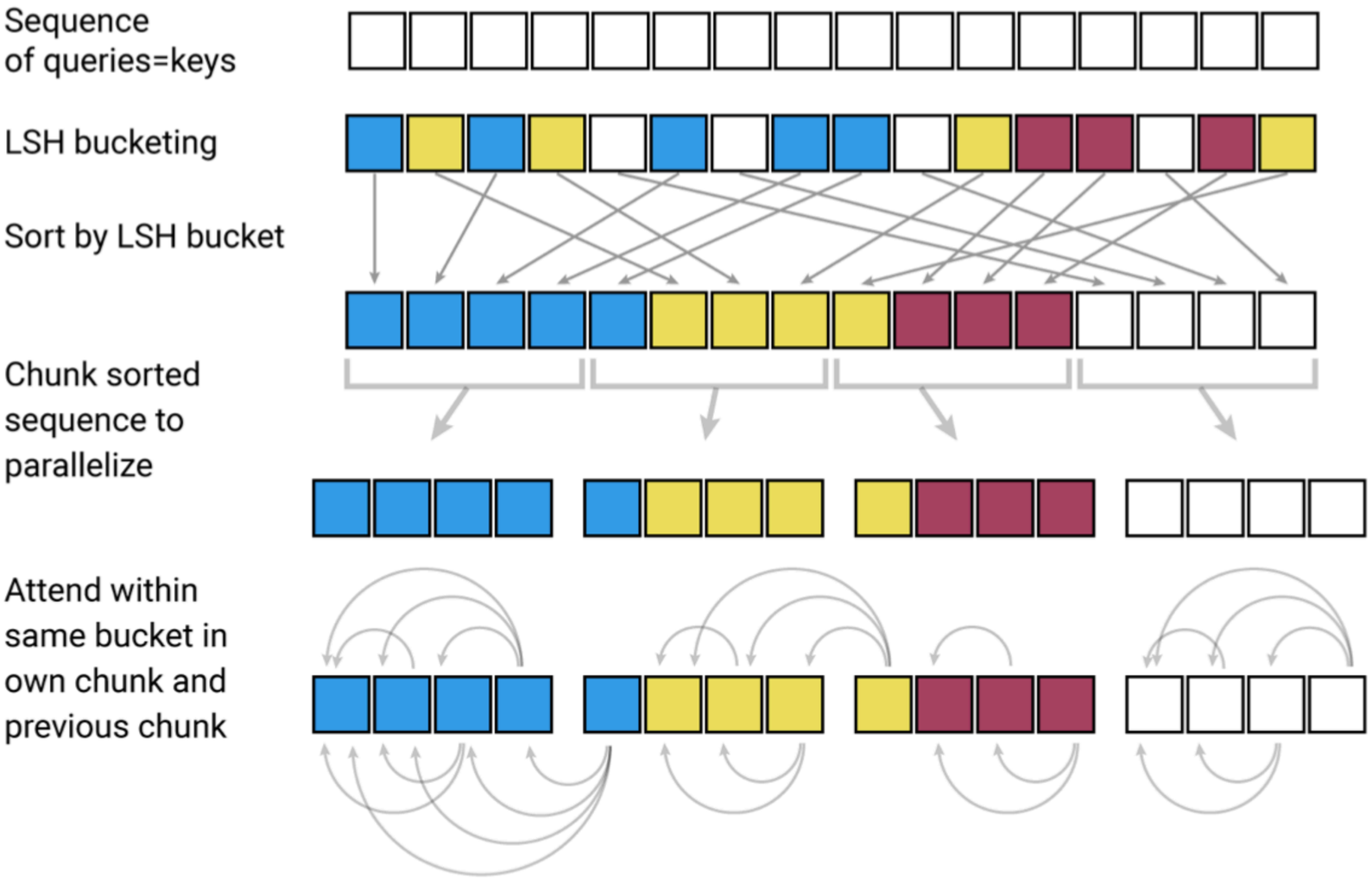
$$\mathcal{P}_i = \{j : h(q_i) = h(k_j)\}$$

$$\widetilde{\mathcal{P}}_i = \left\{ j : \left\lfloor \frac{s_i}{m} \right\rfloor - 1 \leq \left\lfloor \frac{s_j}{m} \right\rfloor \leq \left\lfloor \frac{s_i}{m} \right\rfloor \right\}$$



If $\max_i \left|\mathcal{P}_i\right| < m$, then $\mathcal{P}_i \subseteq \widetilde{\mathcal{P}}_i$

LSH-Attention

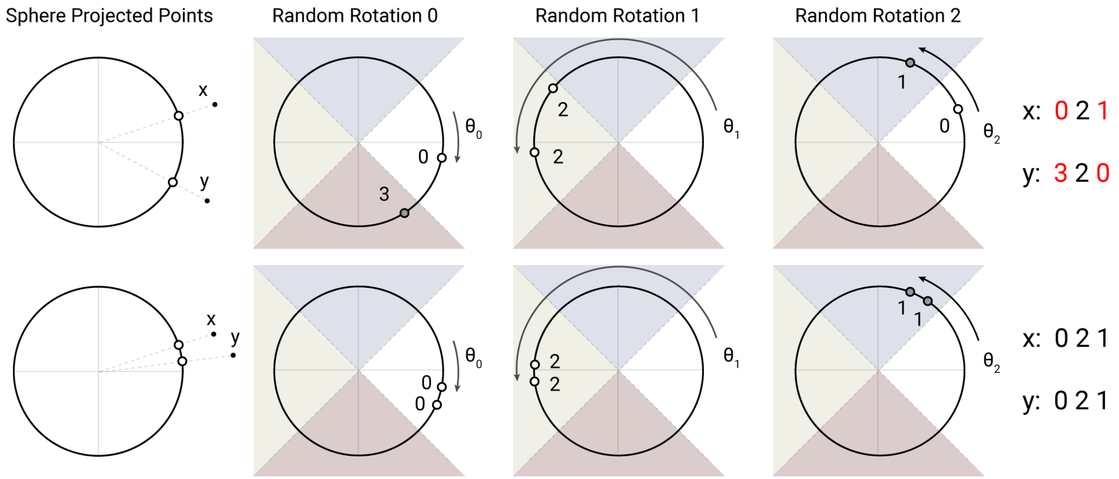


LSH-Attention

Multi-round

- Idea: Reduce probability of similar items falling into different buckets by doing multiple rounds of hashing

$$\{h^{(1)}, h^{(2)}, \ldots, h^{(n\_rounds)}\}$$



$$n\_rounds = 3$$

LSH-Attention

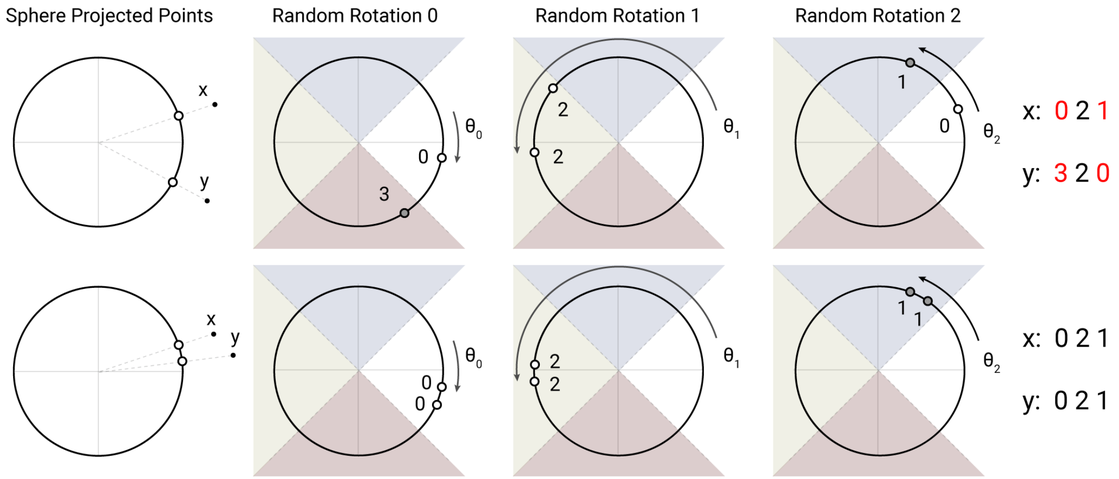
Multi-round

- Idea: Reduce probability of similar items falling into different buckets by doing multiple rounds of hashing

$$\{h^{(1)}, h^{(2)}, \ldots, h^{(n\_rounds)}\}$$

$$\mathcal{P}_i = \bigcup_{r=1}^{n_{rounds}} \mathcal{P}_i^{(r)}$$

where $\mathcal{P}_i^{(r)} = \left\{ j : h^{(r)}\left(q_i\right) = h^{(r)}\left(q_j\right) \right\}$



$n\_rounds = 3$

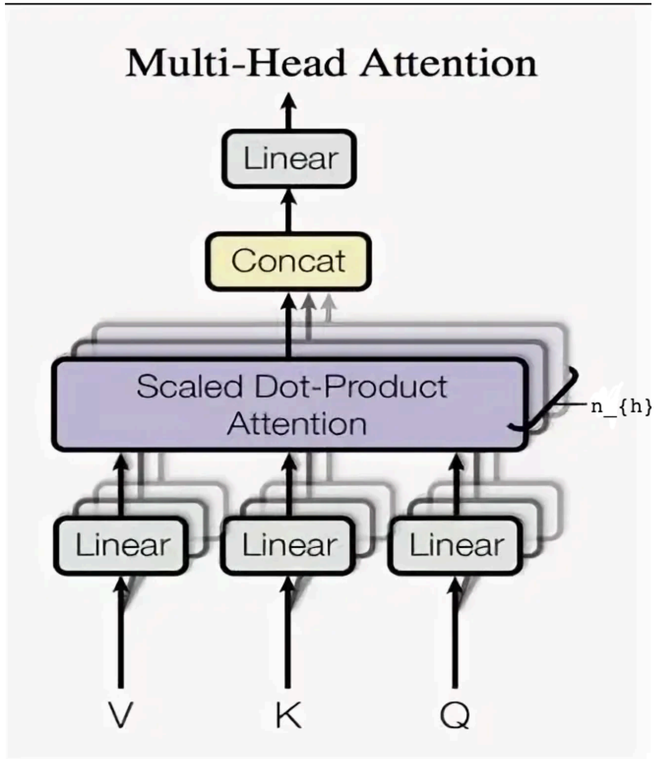
LSH-Attention

Complexity

- Batch size - b
- Sequence length - l
- Number of buckets - n_c
- Chunk length - l_c
- Number of heads - n_h
- Embedding dimension - d_k

Hash computation:

N - random vectors, hash is N bits, $n_c = 2^N$



LSH-Attention

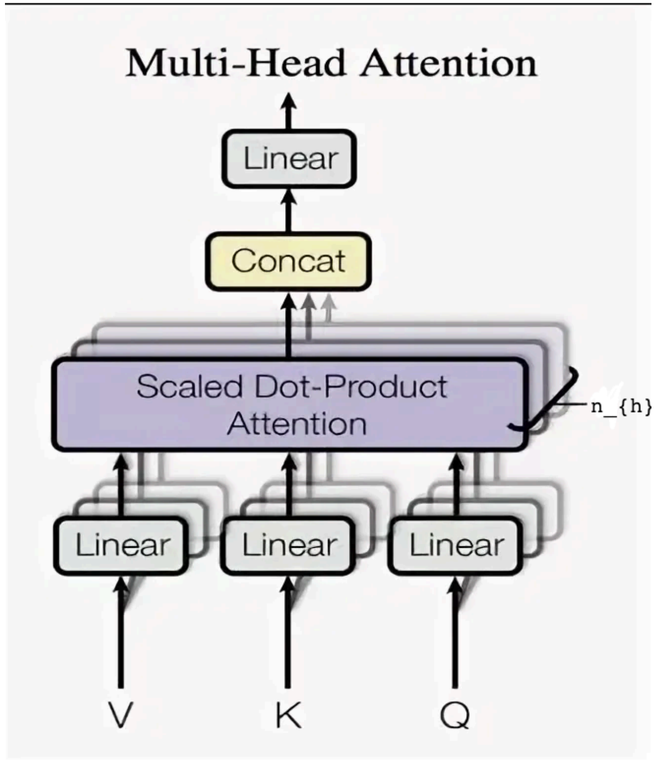
Complexity

- Batch size - b
- Sequence length - l
- Number of buckets - n_c
- Chunk length - l_c
- Number of heads - n_h
- Embedding dimension - d_k

Hash computation:

N - random vectors, hash is N bits, $n_c = 2^N$

$$\mathbb{O}(l \cdot N) = \mathbb{O}(l \cdot log(n_c))$$



LSH-Attention

Complexity

- Batch size - b
- Sequence length - l
- Number of buckets - n_c
- Chunk length - l_c
- Number of heads - n_h
- Embedding dimension - d_k

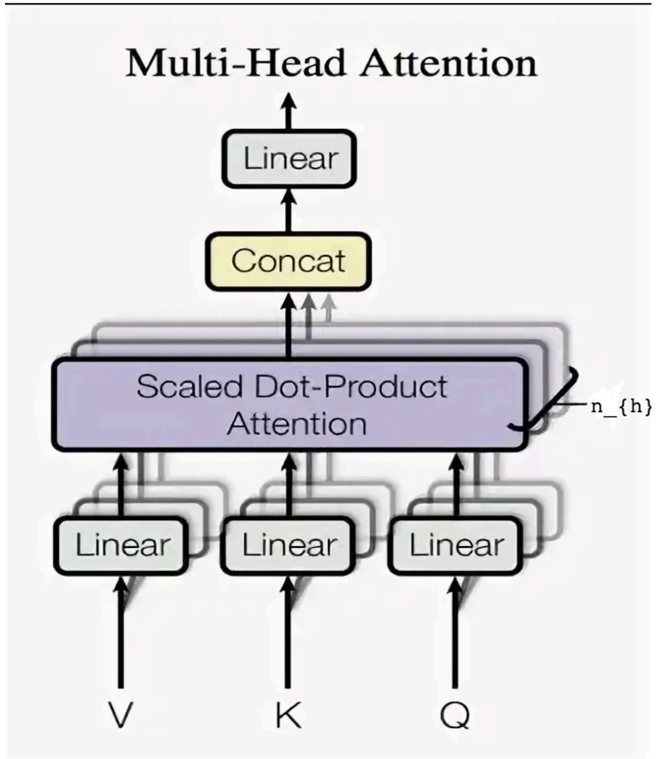
Hash computation:

N - random vectors, hash is N bits, $n_c = 2^N$

$$\mathcal{O}(l \cdot N) = \mathcal{O}(l \cdot log(n_c))$$

LSH-Attention:

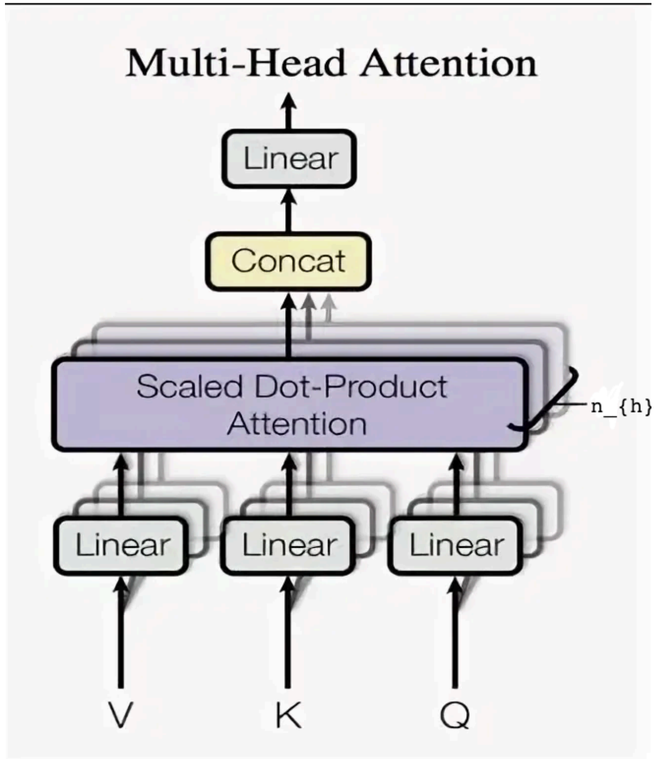
$$\mathcal{O}(n_c \cdot l_c \cdot 2l_c)$$



LSH-Attention

Complexity

- Batch size - b
- Sequence length - l
- Number of buckets - n_c
- Chunk length - l_c
- Number of heads - n_h
- Embedding dimension - d_k



Hash computation:

N - random vectors, hash is N bits, $n_c = 2^N$

$$\mathcal{O}(l \cdot N) = \mathcal{O}(l \cdot log(n_c))$$

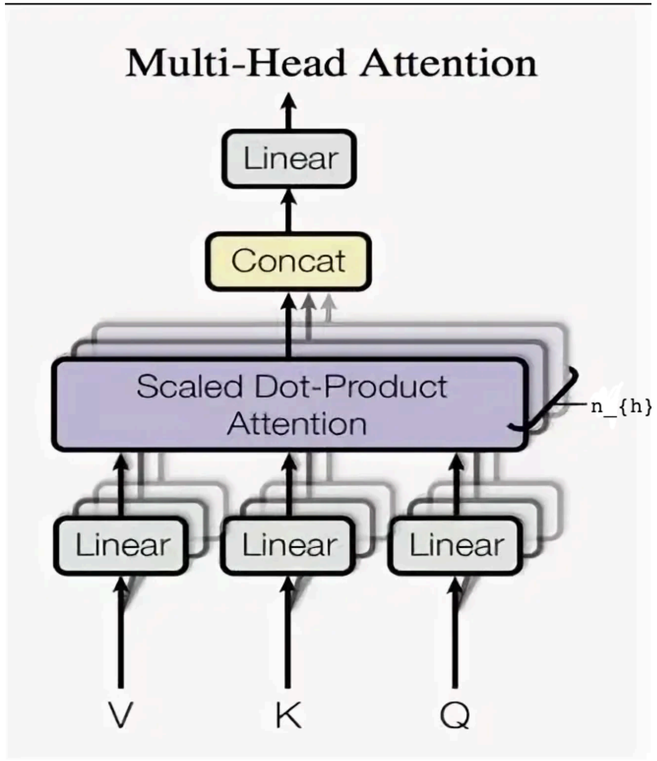
LSH-Attention:

$$\mathcal{O}(n_c \cdot l_c \cdot 2l_c) = \mathcal{O}(n_c \cdot \frac{2l^2}{n_c^2})$$

LSH-Attention

Complexity

- Batch size - b
- Sequence length - l
- Number of buckets - n_c
- Chunk length - l_c
- Number of heads - n_h
- Embedding dimension - d_k



Hash computation:

N - random vectors, hash is N bits, $n_c = 2^N$

$$\mathcal{O}(l \cdot N) = \mathcal{O}(l \cdot log(n_c))$$

LSH-Attention:

$$\mathcal{O}(n_c \cdot l_c \cdot 2l_c) = \mathcal{O}(n_c \cdot \frac{2l^2}{n_c^2}) = \mathcal{O}(\frac{l^2}{n_c})$$

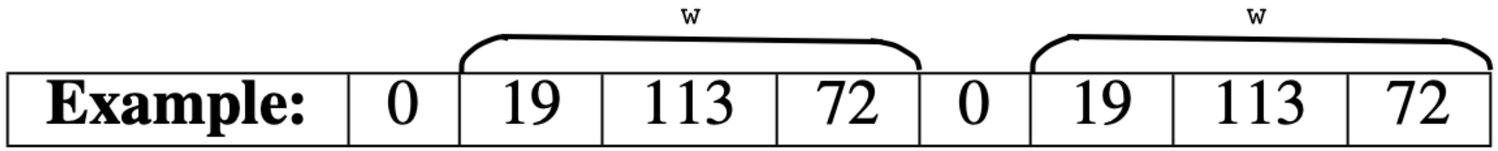
Synthetic task

Data

- Language modeling
- In this task, each training and testing example has the form

  $0w0w$ where $w \in \{1,...,127\}^*$

  is a sequence ranging from 1 to N

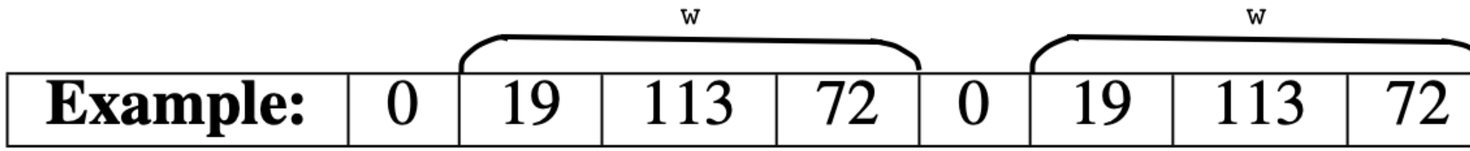


- Each w is of length 511, so the whole input is of length 1024

Synthetic task

Task

- This task can be perfectly solved by 1-layer Transformer model



- But it can’t be solved by any sparse attention models, relying on a limited span

Synthetic task

Model

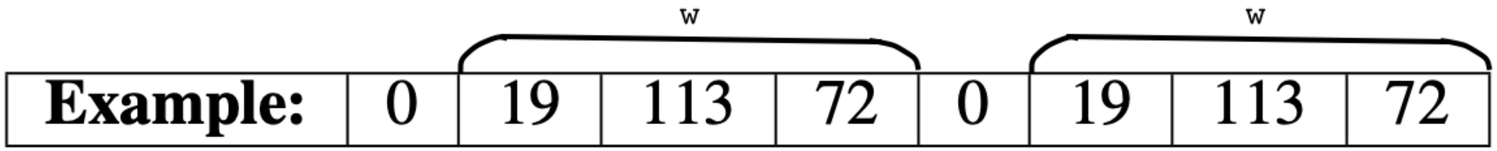
1−layer Transformer

$d_{model} = d_{ff} = 256$

4 heads

Attention options:

- Full-attention
- LSH attention with $n\_rounds = 1$
- LSH attention with $n\_rounds = 2$
- LSH attention with $n\_rounds = 4$



Synthetic task

Results

- We can achieve ideal accuracy using only 2 hashes while training and 8 for evaluation

Train \ Eval	Full Attention	LSH-8	LSH-4	LSH-2	LSH-1
Full Attention	100%	94.8%	92.5%	76.9%	52.5%
LSH-4	0.8%	100%	99.9%	99.4%	91.9%
LSH-2	0.8%	100%	99.9%	98.1%	86.8%
LSH-1	0.8%	99.9%	99.6%	94.8%	77.9%

Reversible Transformer

Motivation

For a feed-forward activations with n_l layers it would be:

$b \cdot l \cdot d_{ff} \cdot n_l$

Memory Complexity
$\max(bn_hld_k, bn_hl^2)$
$\max(bn_hld_k, bn_hl^2)$
$\max(bn_hld_k, bn_hln_r(4l/n_c)^2)$

Reversible Transformer

Motivation

For a feed-forward activations with n_l layers it would be:

$b \cdot l \cdot d_{ff} \cdot n_l$

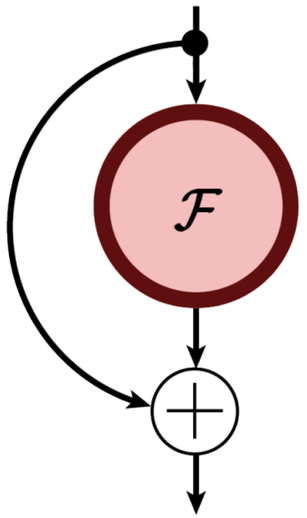
- $b = 8$
- $l = 64K$
- $d_{ff} = 4K$ $16GB$
- $n_l = 16$

Memory Complexity
$\max(bn_h ld_k, bn_h l^2)$
$\max(bn_h ld_k, bn_h l^2)$
$\max(bn_h ld_k, bn_h ln_r(4l/n_c)^2)$

RevNets

- Residual layer performs a following function:

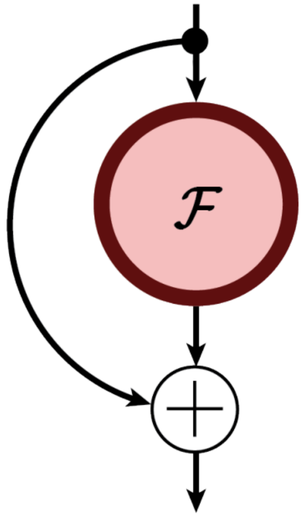
$x \mapsto y \qquad\qquad y = x + F(x)$



RevNets

- Residual layer performs a following function:

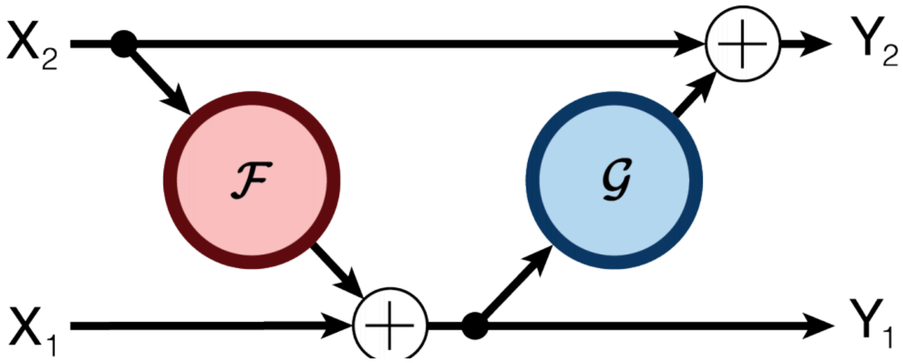
$x \mapsto y \qquad y = x + F(x)$



- Reversible layer works on pairs of inputs/outputs:

$(x_1, x_2) \mapsto (y_1, y_2)$

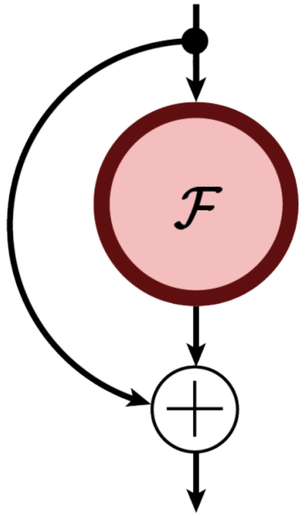
$y_1 = x_1 + F(x_2) \qquad y_2 = x_2 + G(y_1)$



RevNets

- Residual layer performs a following function:

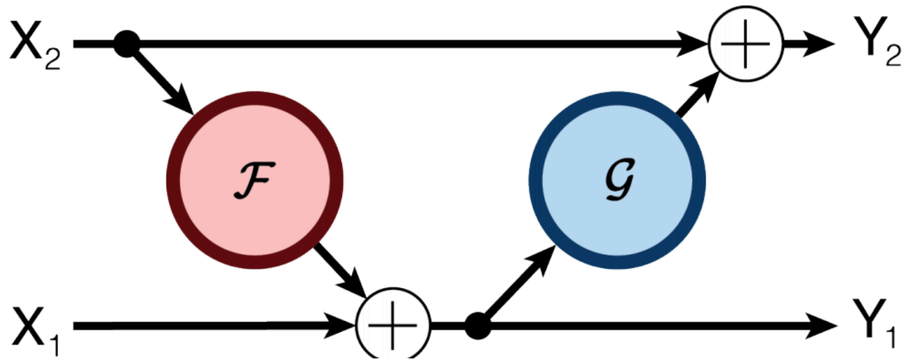
$x \mapsto y \qquad y = x + F(x)$



- Reversible layer works on pairs of inputs/outputs:

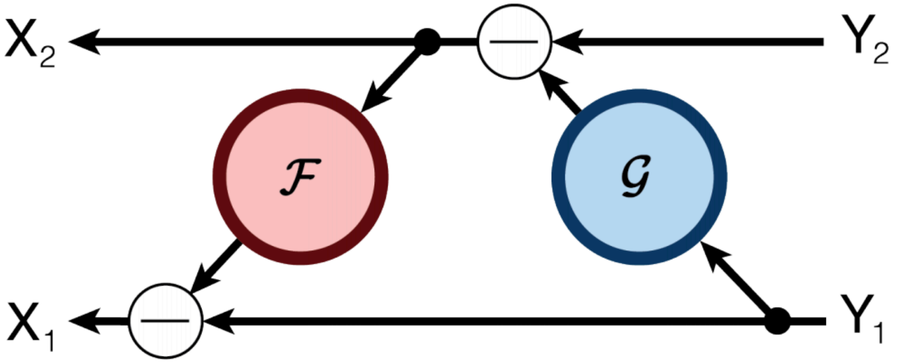
$(x_1, x_2) \mapsto (y_1, y_2)$

$y_1 = x_1 + F(x_2) \qquad y_2 = x_2 + G(y_1)$



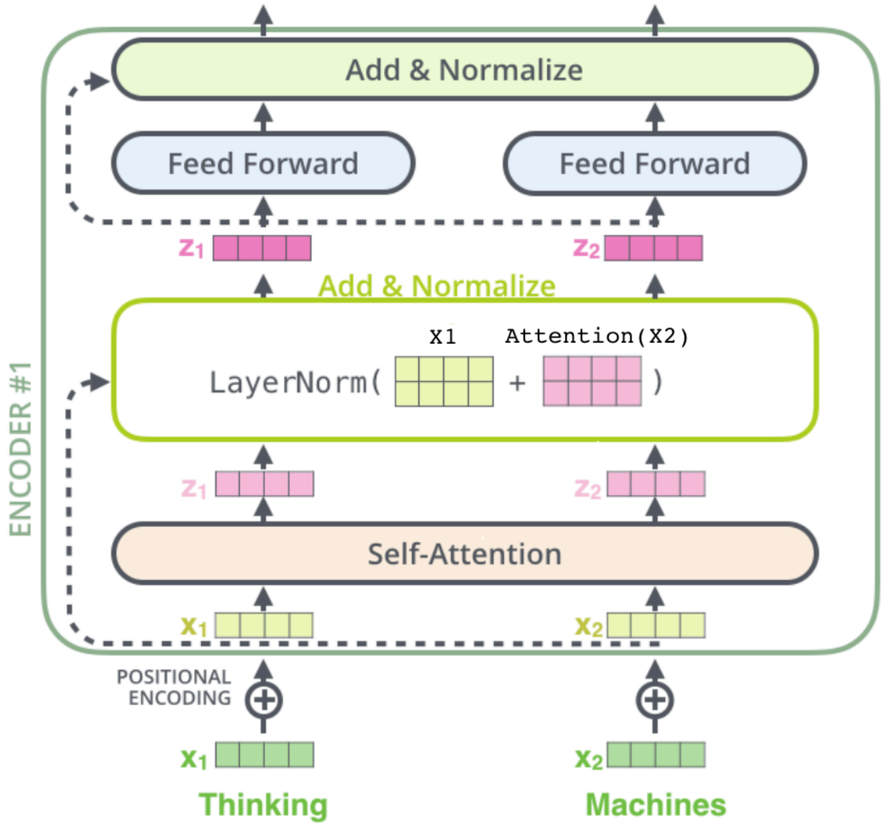
- A layer can be reversed the following way:

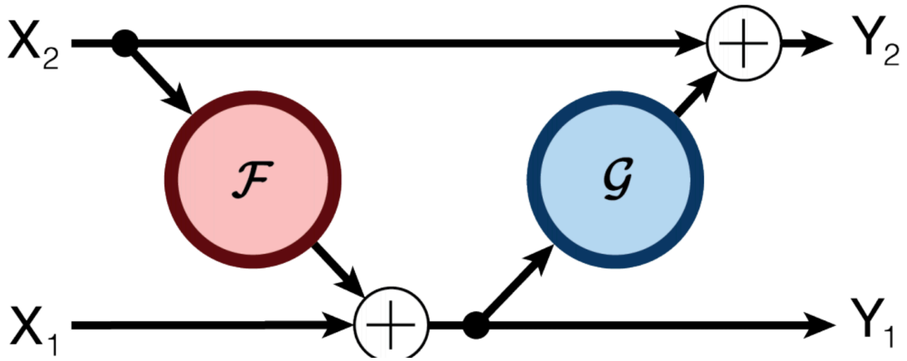
$x_2 = y_2 - G(y_1) \qquad x_1 = y_1 - F(x_2)$



Reversible Transformer

$$Y_1 = X_1 + \text{Attention}\left(X_2\right) \qquad Y_2 = X_2 + \text{FeedForward}\left(Y_1\right)$$



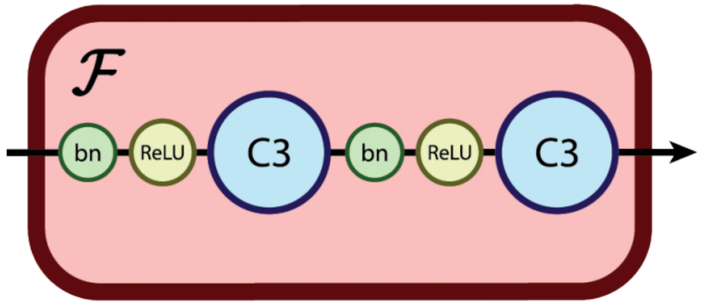


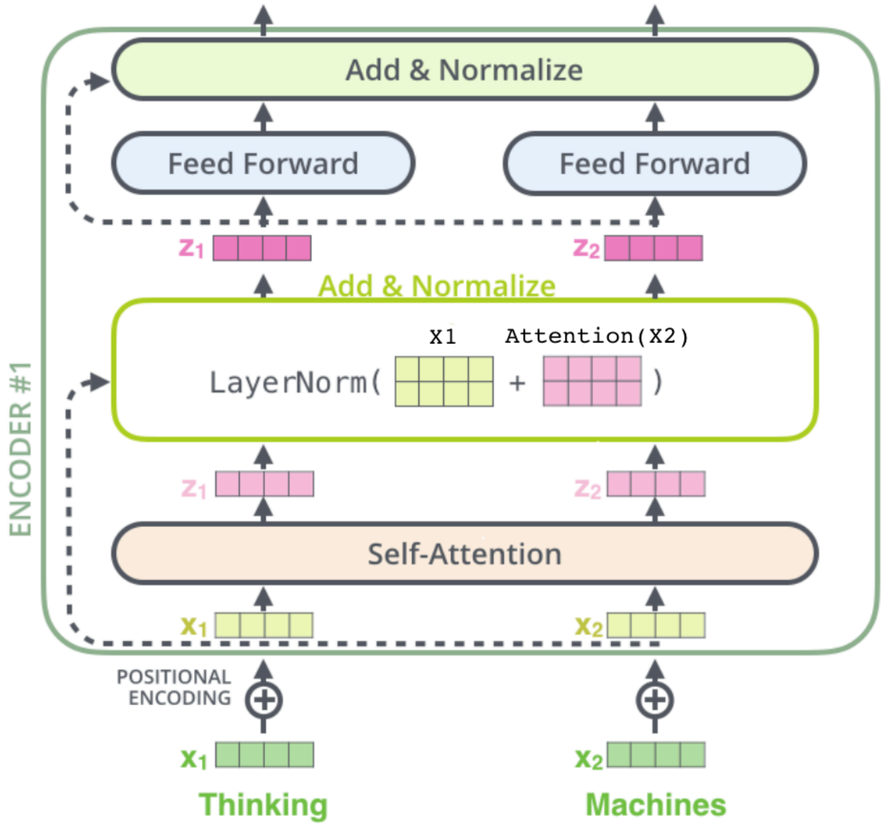
Reversible Transformer

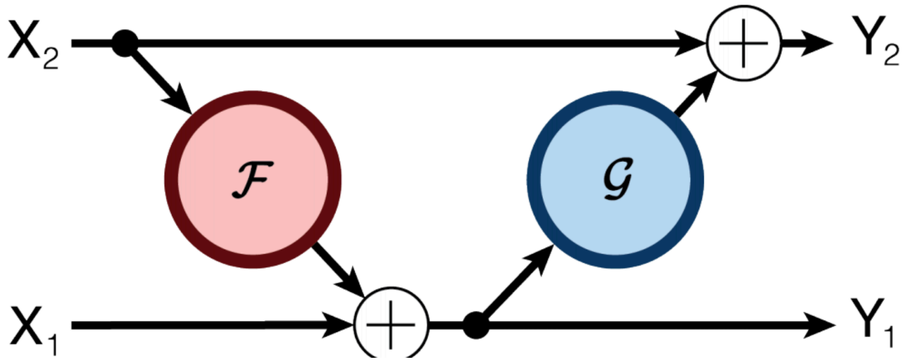
$$Y_1 = X_1 + \text{Attention}\ (X_2) \qquad Y_2 = X_2 + \text{FeedForward}\ (Y_1)$$

Layer normalization is moved

inside residual block







Reversible Transformer

Results

Before

For a feed-forward activations with n_l layers it would be:

$b \cdot l \cdot d_{ff} \cdot n_l$

Now

For a feed-forward activations with n_l layers it would be:

$b \cdot l \cdot d_{ff}$

It performs the same as the normal Transformer

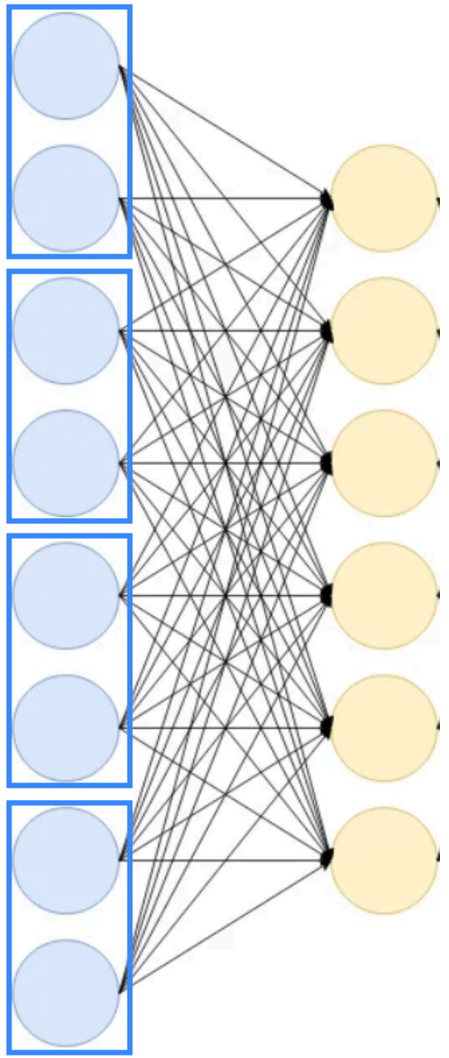
Moreover, we have the same number of parameters

Reversible Transformer

Chunking

- We still may have significant memory usage due to feed-forward layers ($d_{ff} = 4K$)

- Idea - computations if feed-forward layers are completely independent across positions in a sequence

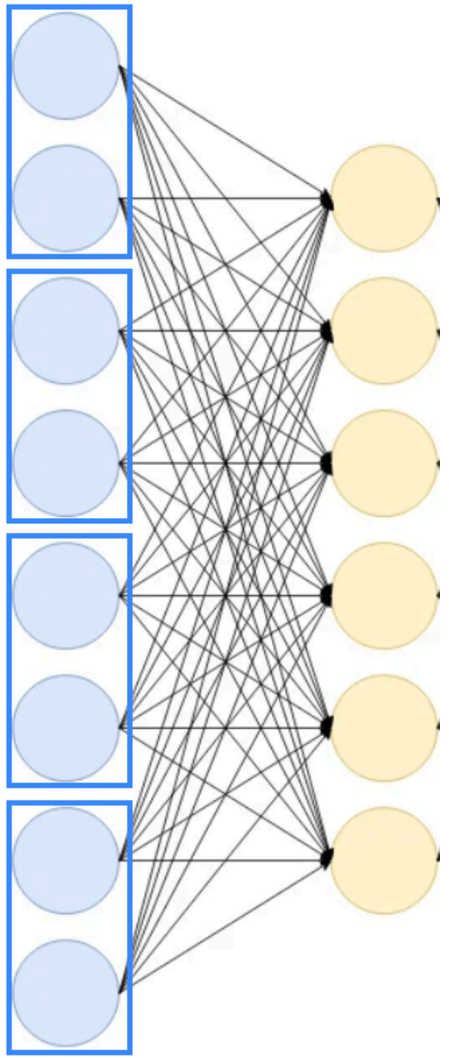


Reversible Transformer

Chunking

- We still may have significant memory usage due to feed-forward layers ($d_{ff} = 4K$)

- Idea - computations if feed-forward layers are completely independent across positions in a sequence



$$Y_2 = \left[Y_2^{(1)}; \ldots; Y_2^{(c)}\right] = \left[X_2^{(1)} + \text{FeedForward}\left(Y_1^{(1)}\right); \ldots; X_2^{(c)} + \text{FeedForward}\left(Y_1^{(c)}\right)\right]$$

The reverse computation and backward pass are also chunked

Reversible Transformer

Is optimization over?

- Chunking and reversible layers - activations are independent on number of layers

- The same is not true for parameters

- Let’s move them to CPU and extract layer parameters only when computing this layer

NB: $batch\_size * seq\_len >> n_{params}$

Transformers

Comparison

Model Type	Memory Complexity	Time Complexity
Transformer	$\max(bld_{ff}, bn_h l^2)n_l$	$(bld_{ff} + bn_h l^2)n_l$
Reversible Transformer	$\max(bld_{ff}, bn_h l^2)$	$(bn_h l d_{ff} + bn_h l^2)n_l$
Chunked Reversible Transformer	$\max(bld_{model}, bn_h l^2)$	$(bn_h l d_{ff} + bn_h l^2)n_l$
LSH Transformer	$\max(bld_{ff}, bn_h l n_r c)n_l$	$(bld_{ff} + bn_h n_r l c)n_l$
Reformer	$\max(bld_{model}, bn_h l n_r c)$	$(bld_{ff} + bn_h n_r l c)n_l$

Experiments

Datasets

- Imagenet64
  Image generation

- enwik8-64K
  Language modeling

- WMT 2014 English-to-German
  Machine translation

Experiments

Models

Models:

- Transformer
- Reformer

Parameters:

- $n_{layers} = 3$
- $d_{model} = 1024$
- $d_{ff} = 4096$
- $n_{heads} = 8$
- $batch\_size = 8$

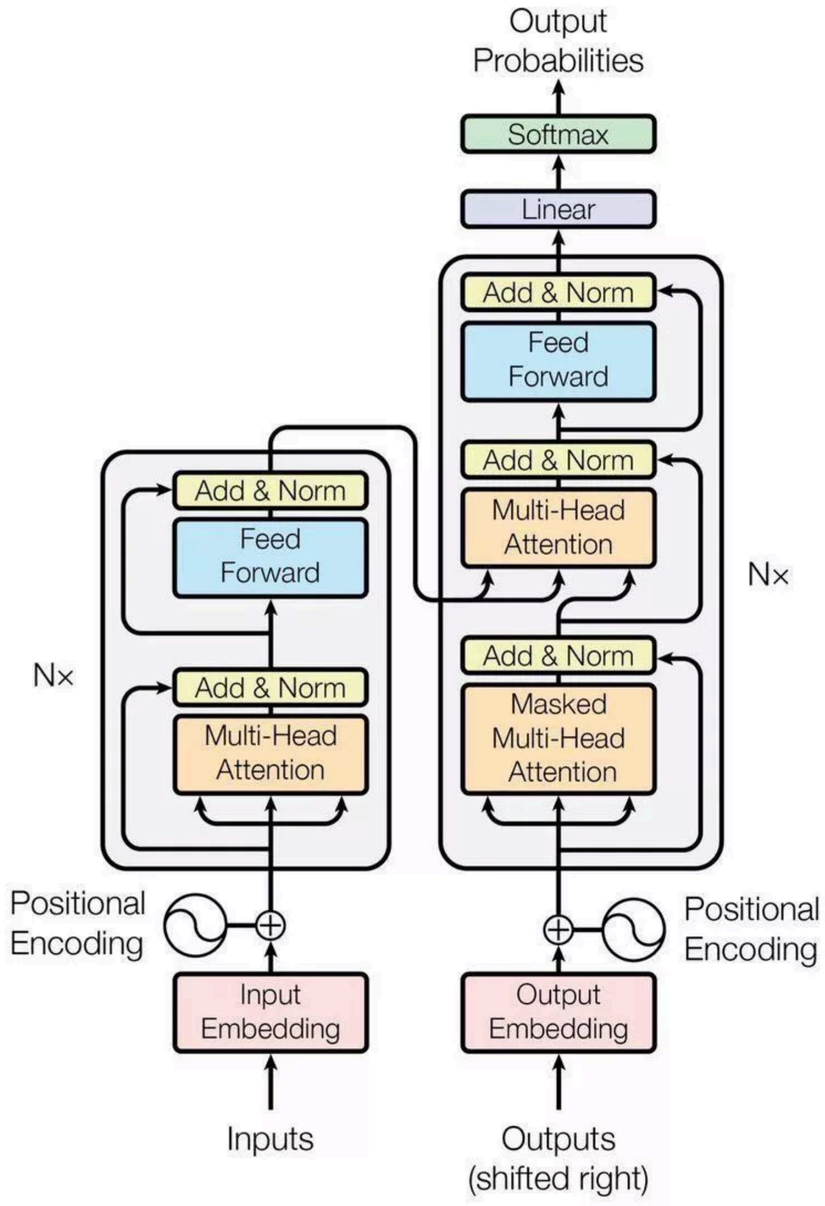
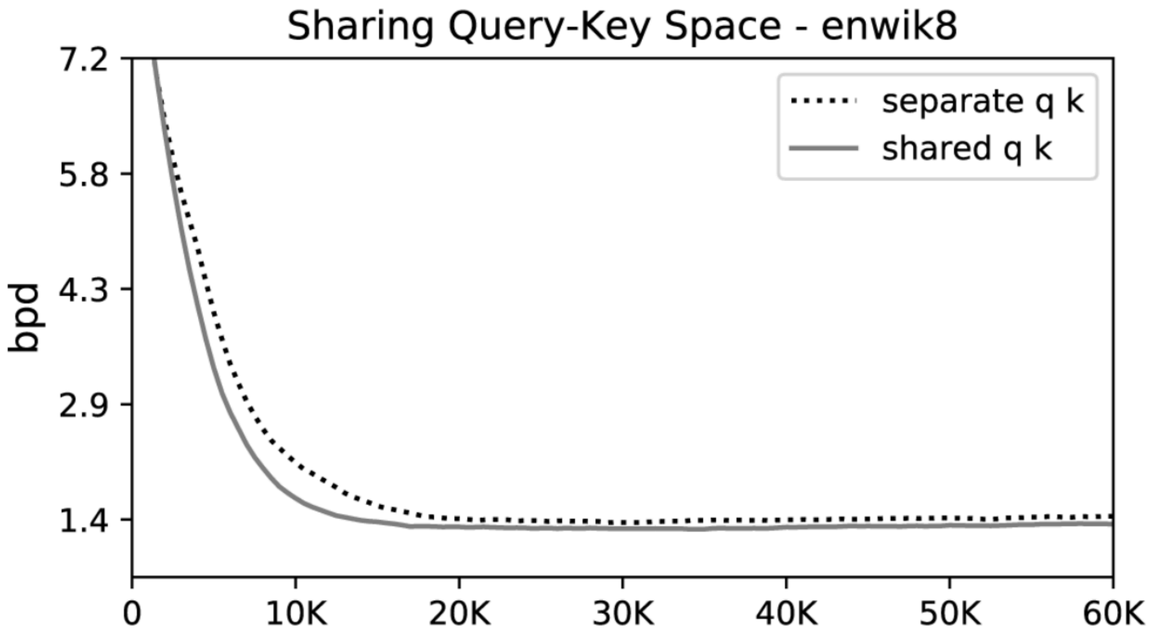
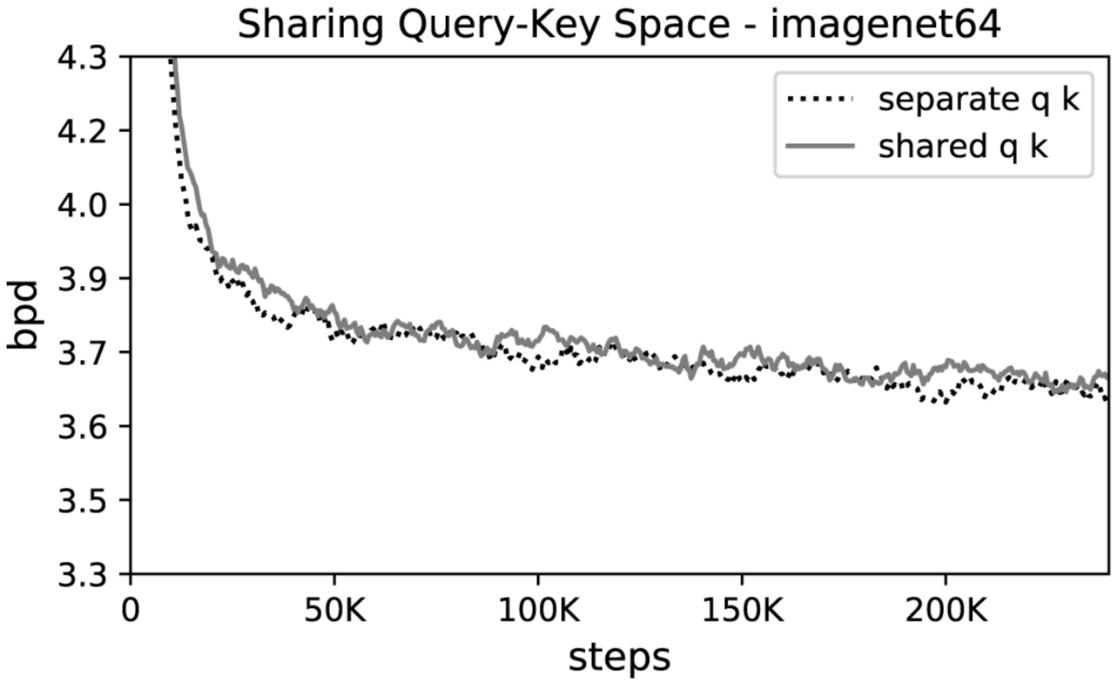


Figure 1: The Transformer - model architecture.

Experiments

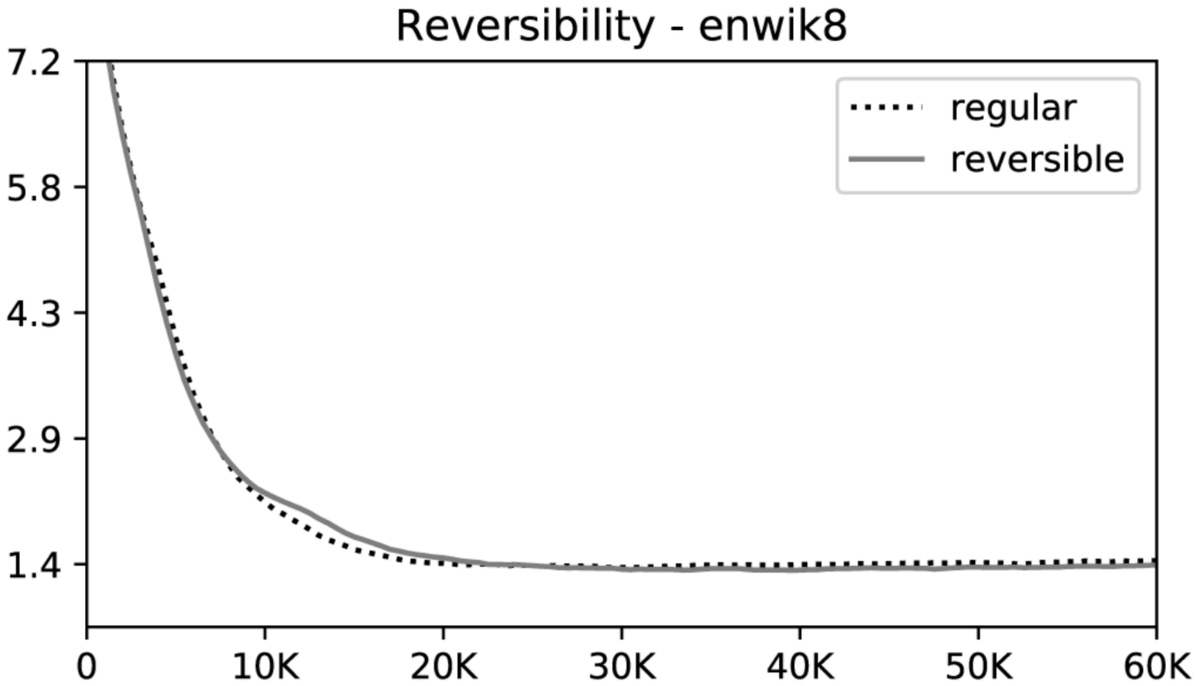
Sharing Query-Key Space

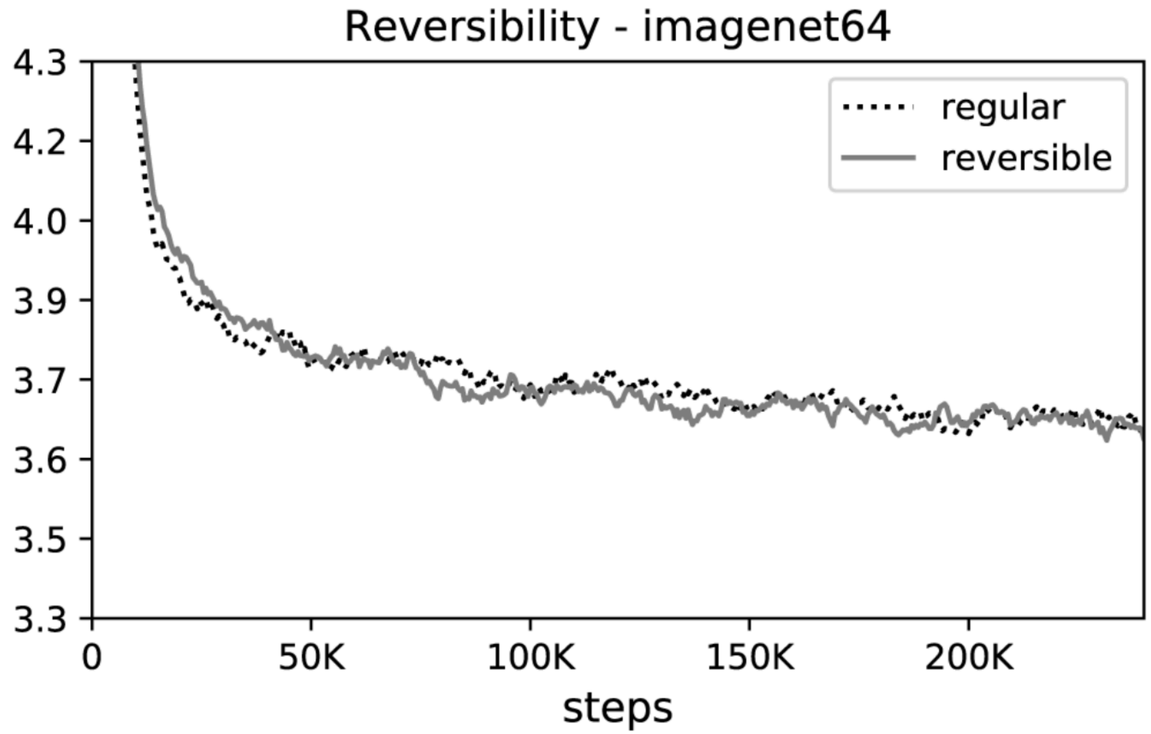




Experiments

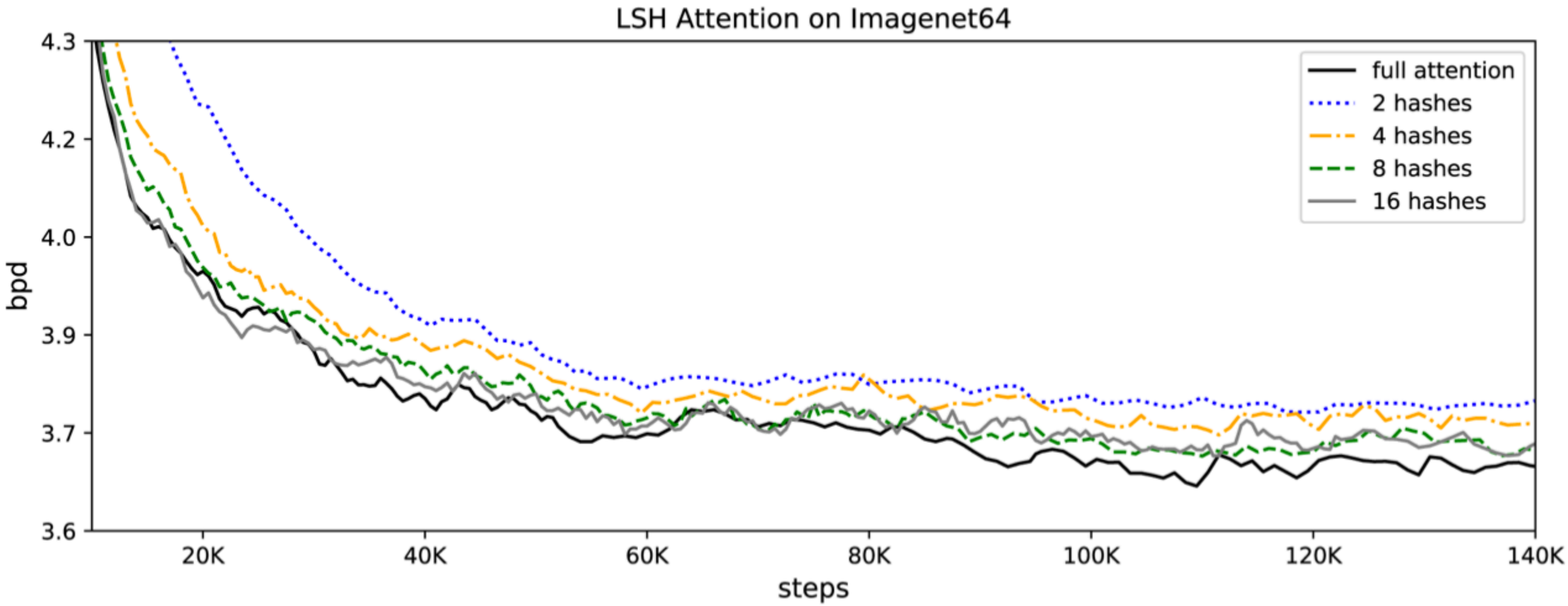
Reversibility





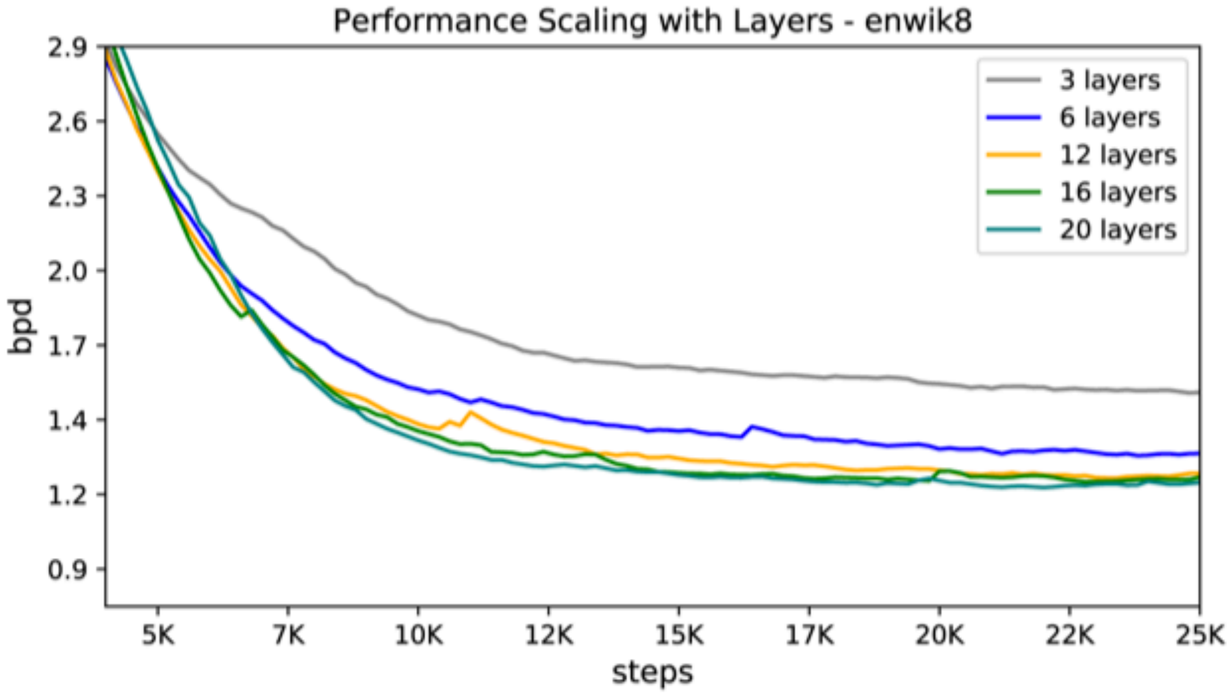
Experiments

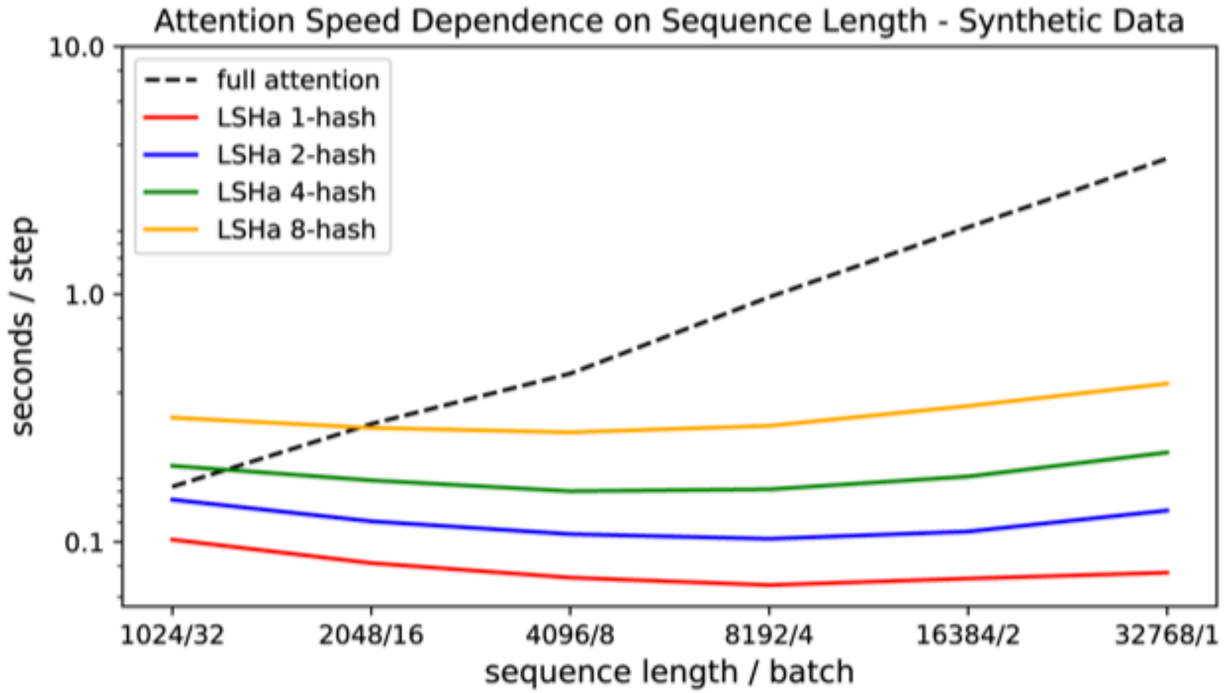
LSH-Attention



Experiments

LSH-Attention





Experiments

Results

Prompt:

Sampled completions:

Questions

1. What are 3 main ideas of Reformer model?

2. Describe in details the procedure of LSH-Attention

3. How RevNets can be applied to Reformer?

References

- Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In ICLR, 2020.

- Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In NIPS, 2015

- Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual net- work: Backpropagation without storing activations. In *Advances in neural information processing systems*, pp. 2214–2224, 2017