Thought for 10 seconds >

Large Language Models on a Tiny Power Budget

Steven Abreu

University of Groningen, Intel Labs

Jason Eshraghian

Assistant Professor, UC Santa Cruz

cool thanks

Search Deep research Create image ...

ChatGPT can make mistakes. Check important info.

ISCAS 2025
IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS
LONDON, UK

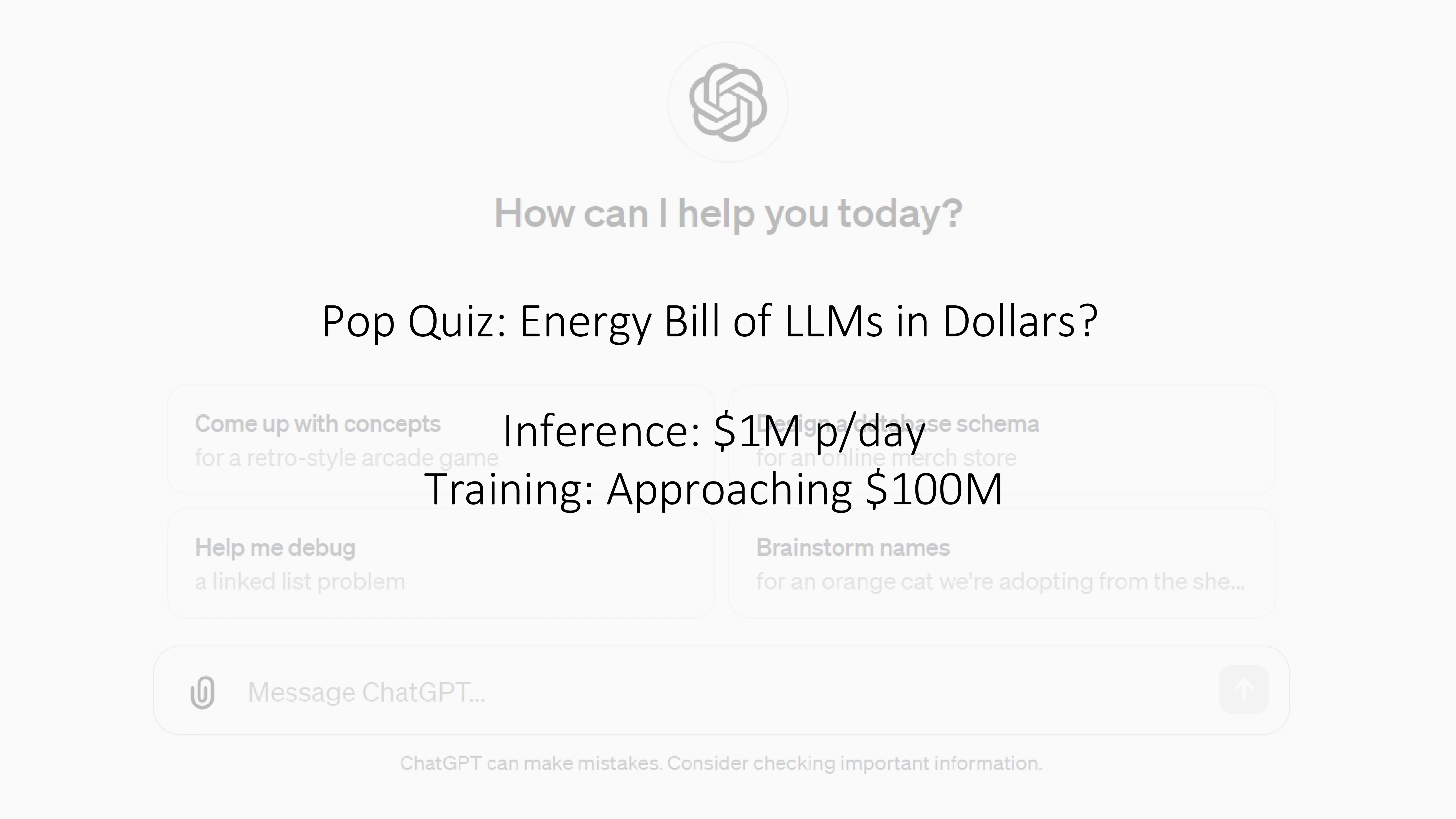
Overview

Part 1: Modern Language Models

- LLMs 101
- Make Transformers Efficient
  - Keeping self-attention
  - Supplementing self-attention
  - Modifying/replacing self-attention

Part 2: Next-Generation Language Models

- State-Space Models
- Neuromorphic Hardware
- MatMul-free LM on Loihi
- What’s next?



Pop Quiz: Energy Bill of LLMs in Dollars?

Inference: $1M p/day

Training: Approaching $100M

Overview

Part 1: Modern Language Models

- LLMs 101
- Make Transformers Efficient
  - Keeping self-attention
  - Supplementing self-attention
  - Modifying/replacing self-attention

Part 2: Next-Generation Language Models

- State-Space Models
- Neuromorphic Hardware
- MatMul-free LM on Loihi
- What’s next?

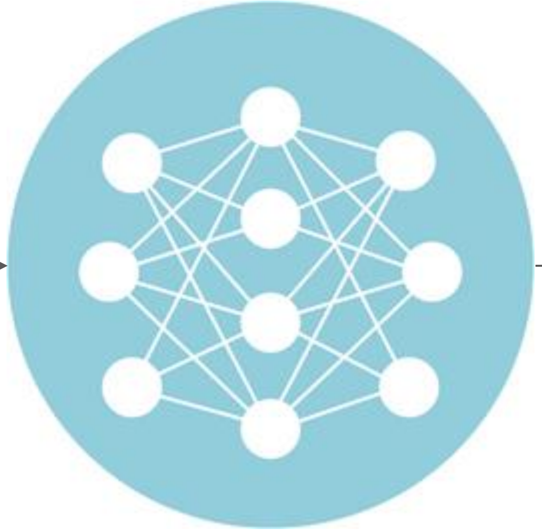
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

“Autoregression”

“The cat in the…”



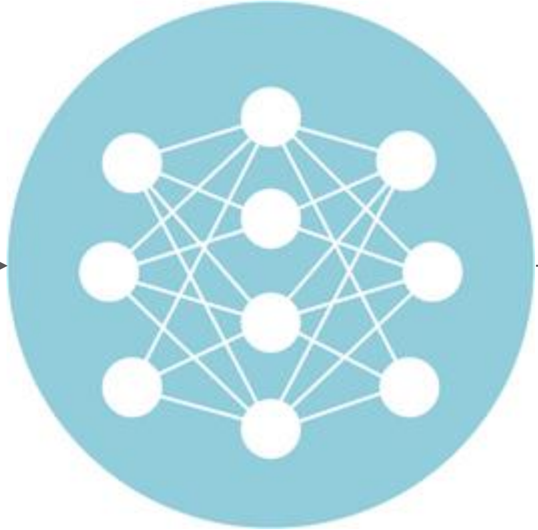
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

"The cat in the…"



hostage-negotiation

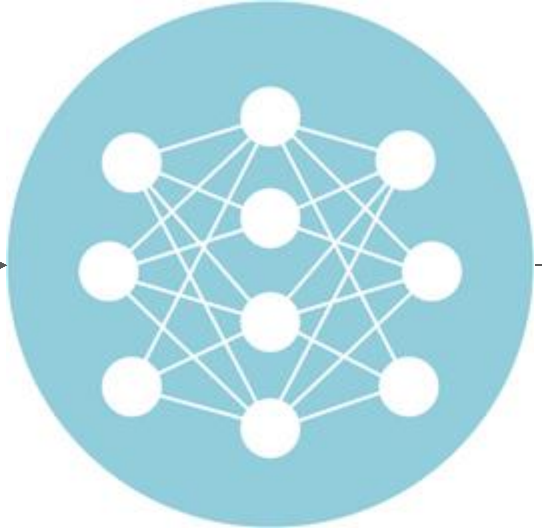
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

"The cat in the… hostage-negotiation"



demanded

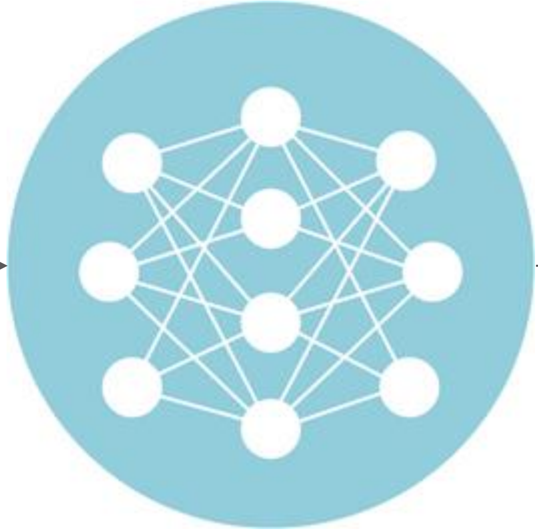
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

"The cat in the… hostage-negotiation demanded"



12

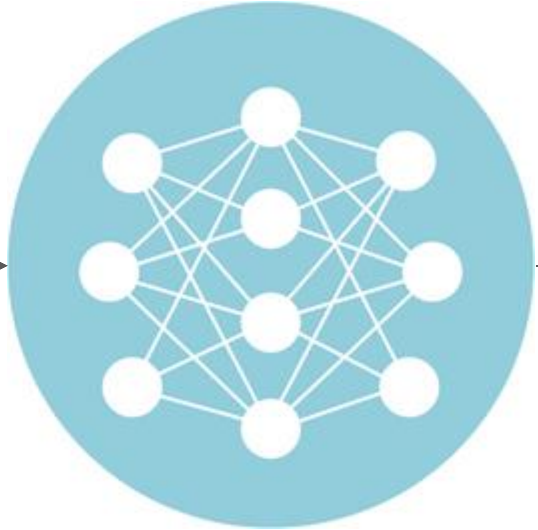
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

"The cat in the… hostage-negotiation demanded 12"



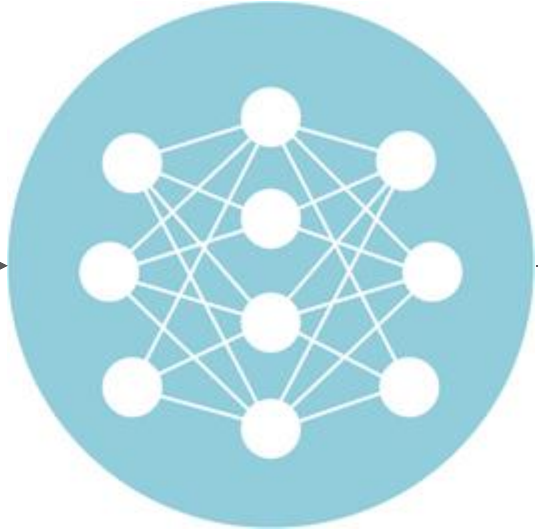
laser-pointers

LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

"The cat in the... hostage-negotiation demanded 12 laser-pointers" →  → <|EndOfText|>

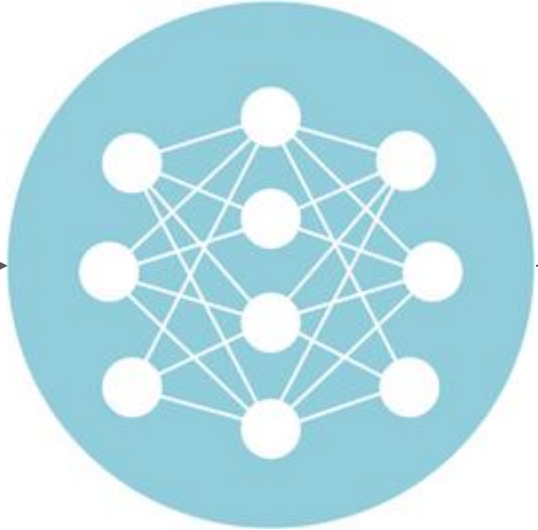
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

"The cat in the…"

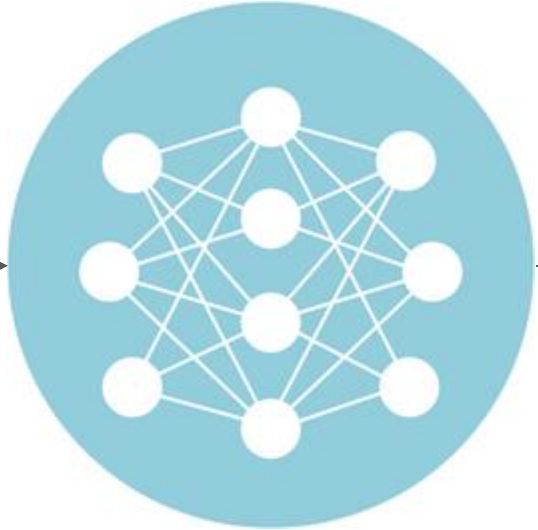


LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

$\Big\}$ *"Autoregression"*

"The cat in the…" →  → hat

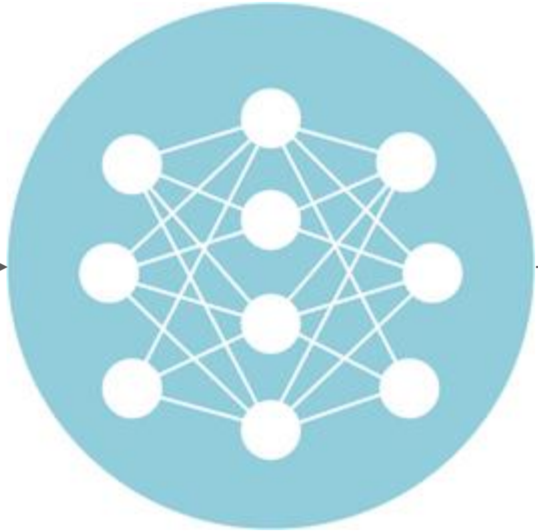
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

“Autoregression”

“The cat in the...
hat



sat

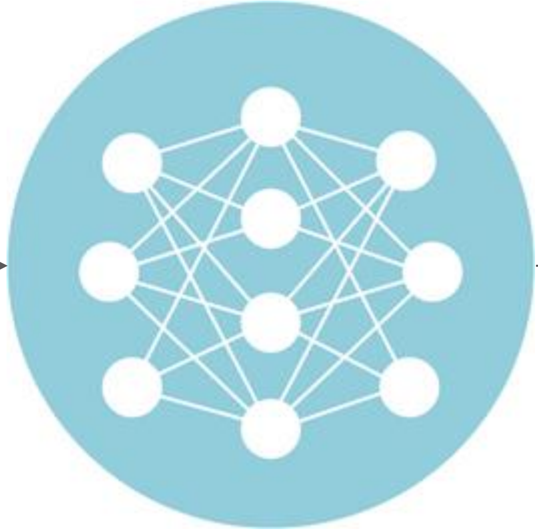
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

“Autoregression”

“The cat in the…
hat sat



on

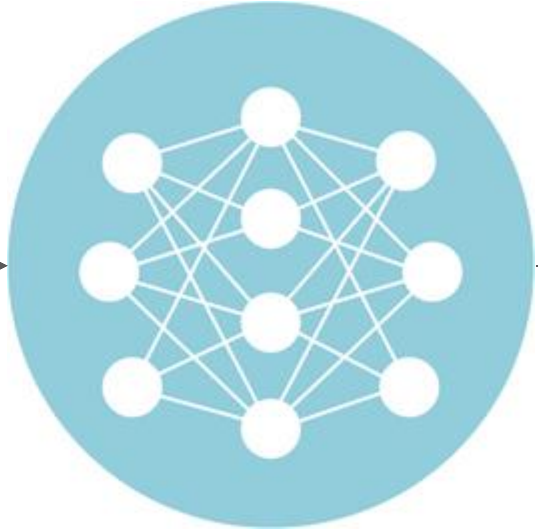
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

"The cat in the…
hat sat on



the

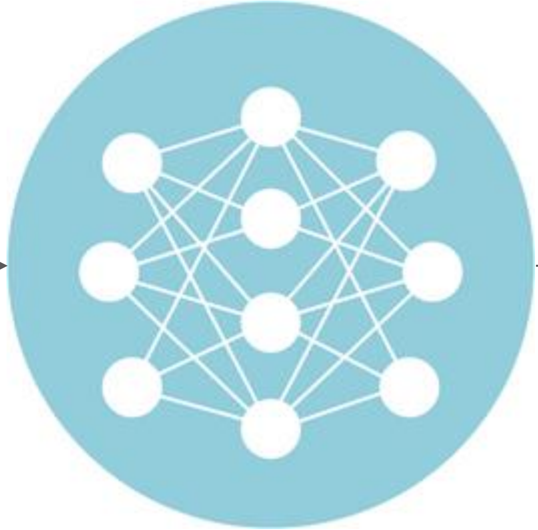
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

"The cat in the… hat sat on the



mat

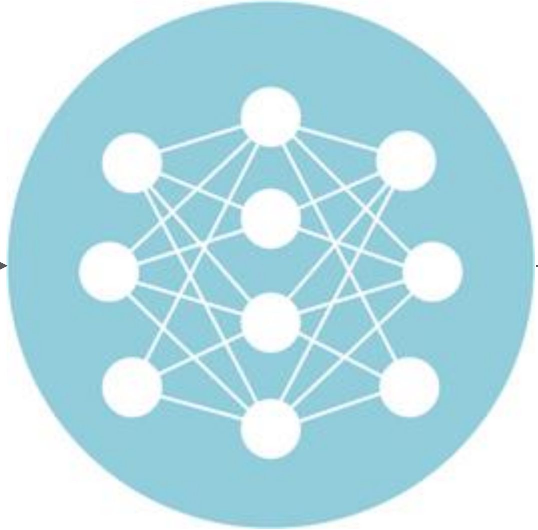
LLMs 101

The objective of a language model is to predict the next token.

Input: A sequence
Output: The next item in that sequence
Repeat.

"Autoregression"

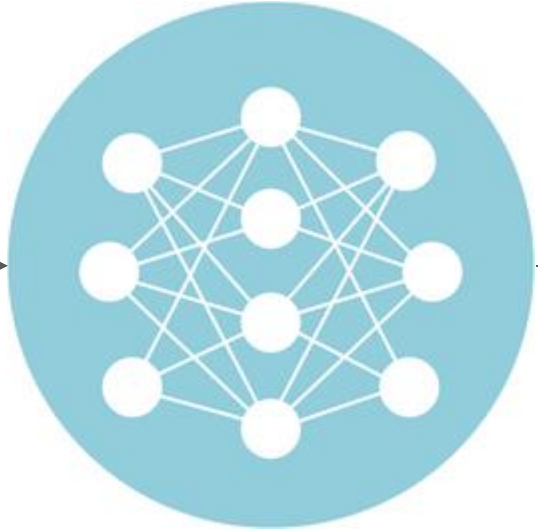
"The cat in the...
hat sat on the mat"



<|EndOfText|>

Teacher Forcing

“The cat in the…



hostage-negotiation

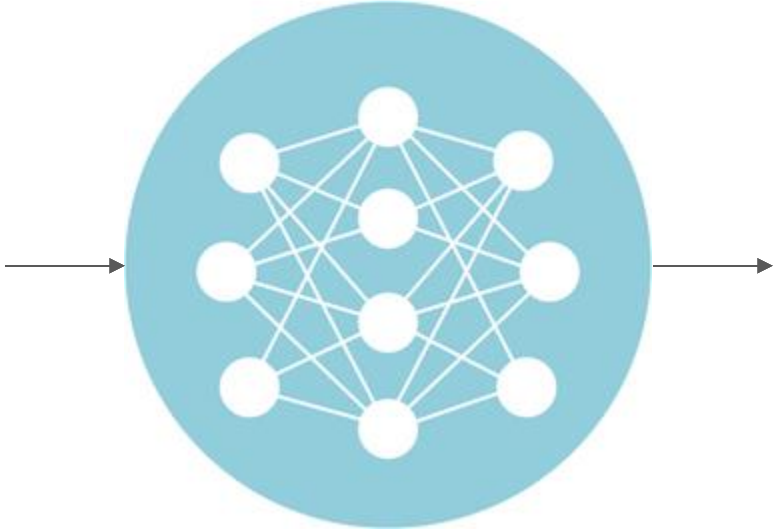
Teacher Forcing

But Neural Networks Need Numbers - Not Words.

Tokenization: *breaking text up into common words, pieces of words, and characters*

The cat in the hostage-negotiation demanded 12 laser pointers

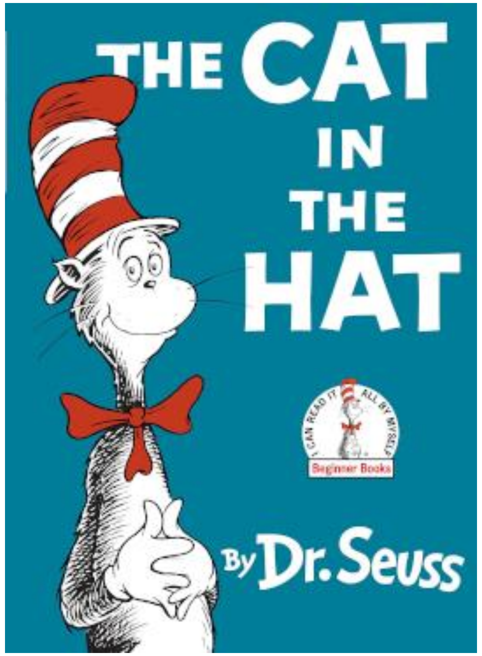
"The cat in the…"



But Neural Networks Need Numbers - Not Words.

Tokenization: *breaking text up into common words, pieces of words, and characters*

The cat in the hostage-negotiation demanded 12 laser pointers



Number of characters: 30,506

Number of tokens (GPT-2): 9,638

But Neural Networks Need Numbers - Not Words.

Tokenization: *breaking text up into common words, pieces of words, and characters*

Embedding: *a vectorized representation of tokens*

The cat in the hostage-negotiation demanded 12 laser pointers

But Neural Networks Need Numbers - Not Words.

Tokenization: *breaking text up into common words, pieces of words, and characters*

Embedding: *a vectorized representation of tokens*

The cat in the hostage-negotiation demanded 12 laser pointers

-0.068
-0.030
0.064
…
-0.01
0.003

But Neural Networks Need Numbers - Not Words.

Tokenization: *breaking text up into common words, pieces of words, and characters*

Embedding: *a vectorized representation of tokens*

The cat in the hostage-negotiation demanded 12 laser pointers

-0.068	-0.016
-0.030	-0.093
0.064	0.242
…	…
-0.01	-0.17
0.003	-0.158

But Neural Networks Need Numbers - Not Words.

Tokenization: *breaking text up into common words, pieces of words, and characters*

Embedding: *a vectorized representation of tokens*

The cat in the hostage-negotiation demanded 12 laser pointers

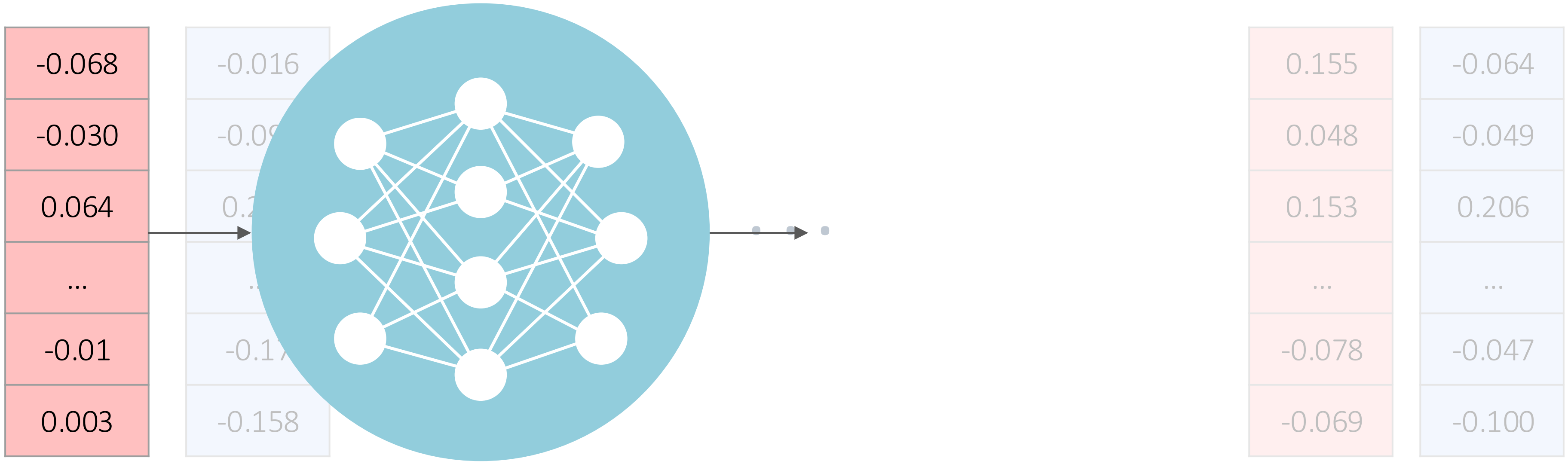
-0.068	-0.016		0.155	-0.064
-0.030	-0.093		0.048	-0.049
0.064	0.242	. . .	0.153	0.206
…	…		…	…
-0.01	-0.17		-0.078	-0.047
0.003	-0.158		-0.069	-0.100

But Neural Networks Need Numbers - Not Words.

Tokenization: *breaking text up into common words, pieces of words, and characters*

Embedding: *a vectorized representation of tokens*

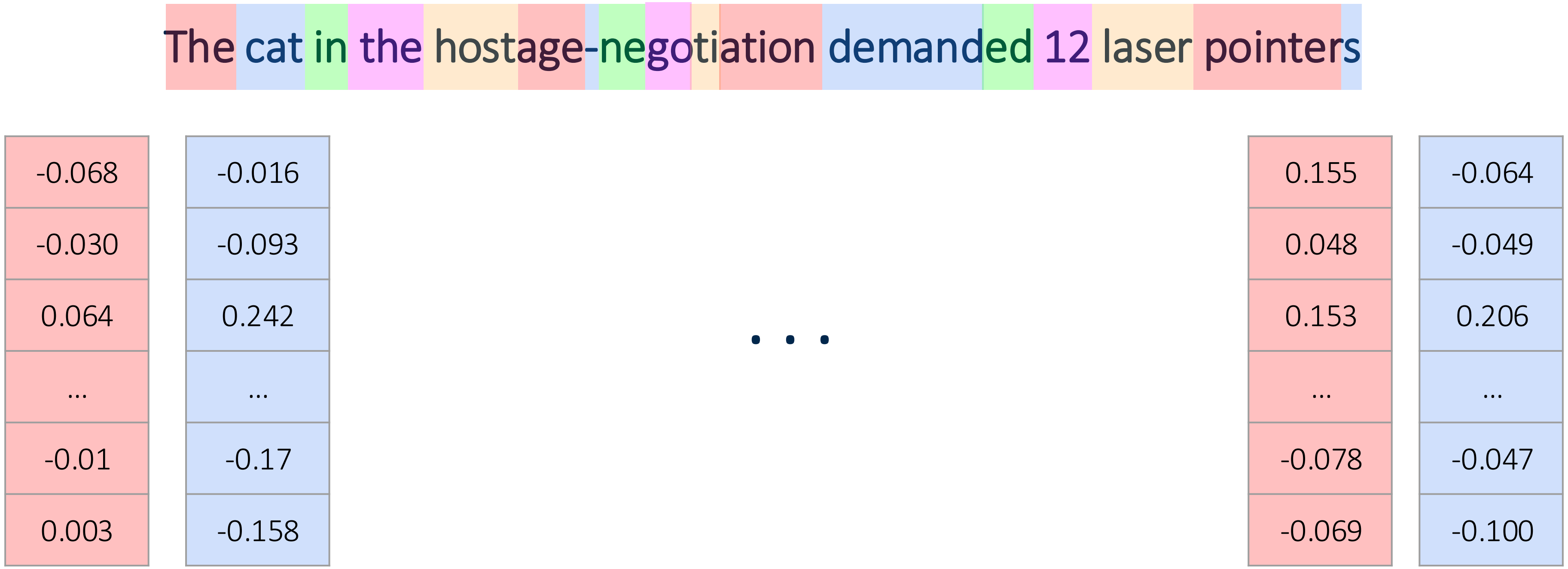
The cat in the hostage-negotiation demanded 12 laser pointers



But Neural Networks Need Numbers - Not Words.

Tokenization: *breaking text up into common words, pieces of words, and characters*

Embedding: *a vectorized representation of tokens*



GPT2’s Tokenizer

Vocabulary Size: 50256

Embedding Size: 768

<table>
<tr><th>!</th><th>“</th><th>#</th><th></th><th>informants</th><th>gazed</th><th><|endoftext|></th></tr>
<tr><td>-0.11</td><td>0.04</td><td>-0.12</td><td>…</td><td>-0.04</td><td>0.19</td><td>-0.05</td></tr>
<tr><td>0.04</td><td>-0.05</td><td>0.05</td><td>…</td><td>-0.05</td><td>0.02</td><td>-0.03</td></tr>
<tr><td>-0.12</td><td>0.05</td><td>0.18</td><td>…</td><td>0.01</td><td>0.05</td><td>0.05</td></tr>
<tr><td>-0.09</td><td>-0.10</td><td>-0.09</td><td>…</td><td>0.14</td><td>-0.01</td><td>-0.04</td></tr>
<tr><td>-0.05</td><td>0.08</td><td>0.08</td><td>…</td><td>0.10</td><td>-0.07</td><td>-0.06</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
</table>

```python
from transformers import GPT2TokenizerFast, GPT2Model

tok    = GPT2TokenizerFast.from_pretrained("gpt2")
```

GPT2’s Tokenizer

Vocabulary Size: 50256

Embedding Size: 768

<table>
<tr><th>!</th><th>“</th><th>#</th><th></th><th>informants</th><th>gazed</th><th><|endoftext|></th></tr>
<tr><td>-0.11</td><td>0.04</td><td>-0.12</td><td>...</td><td>-0.04</td><td>0.19</td><td>-0.05</td></tr>
<tr><td>0.04</td><td>-0.05</td><td>0.05</td><td>...</td><td>-0.05</td><td>0.02</td><td>-0.03</td></tr>
<tr><td>-0.12</td><td>0.05</td><td>0.18</td><td>...</td><td>0.01</td><td>0.05</td><td>0.05</td></tr>
<tr><td>-0.09</td><td>-0.10</td><td>-0.09</td><td>...</td><td>0.14</td><td>-0.01</td><td>-0.04</td></tr>
<tr><td>-0.05</td><td>0.08</td><td>0.08</td><td>...</td><td>0.10</td><td>-0.07</td><td>-0.06</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
</table>

Word-level vocabulary = huge dictionary
Character-level vocabulary = huge context

Modern LLM Tokenizers

- GPT3: 100k x 768
- DeepSeek: 100k x 7k
- Claude: 65k x 1k
- Llama4: 200k x 5k

GPT Token List

0	!
1	"
2	#
3	$
4	%
5	&
6	'
7	(
8	)
9	*
10	+
11	,
12	-
13	.

Word-level vocabulary = huge dictionary
Character-level vocabulary = huge context

<u>Modern LLM Tokenizers</u>

- GPT3: 100k x 768
- DeepSeek: 100k x 7k
- Claude: 65k x 1k
- Llama4: 200k x 5k

GPT Token List

18494	.JSON
18495	441
18496	desk
18497	.substr
18498	//--
18499	lyn
18500	pson
18501	407
18502	disable
18503	Func
18504	9Assert \tAssert
18505	MARK
18506	defeat
18507	blind

Word-level vocabulary = huge dictionary
Character-level vocabulary = huge context

Modern LLM Tokenizers

- GPT3: 100k x 768
- DeepSeek: 100k x 7k
- Claude: 65k x 1k
- Llama4: 200k x 5k

GPT Token List

100247	moden
100248	Icelandic
100249	;d
100250	.allowed
100251	(newUser
100252	merciless
100253	.WaitFor
100254	daycare
100255	Conveyor
100257	<end of text>
100258	<prefix>
100259	<middle>
100260	<suffix>
100276	<end of prompt>

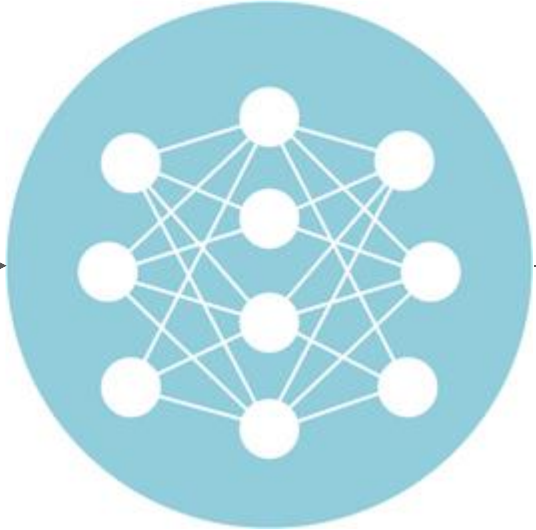
Word-level vocabulary = huge dictionary
Character-level vocabulary = huge context

Modern LLM Tokenizers

- GPT3: 100k x 768
- DeepSeek: 100k x 7k
- Claude: 65k x 1k
- Llama4: 200k x 5k

Unembedding

"The cat in the…" →



→

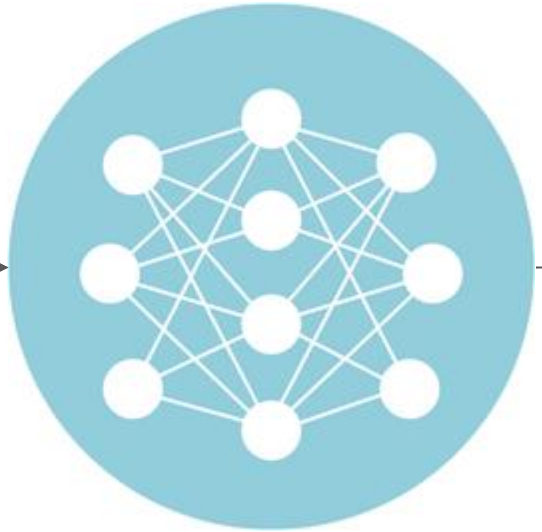
$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{50254} \\ z_{50255} \\ z_{50256} \end{bmatrix}$$

hat

Vocabulary Size: 50256

Unembedding

"The cat in the..." →

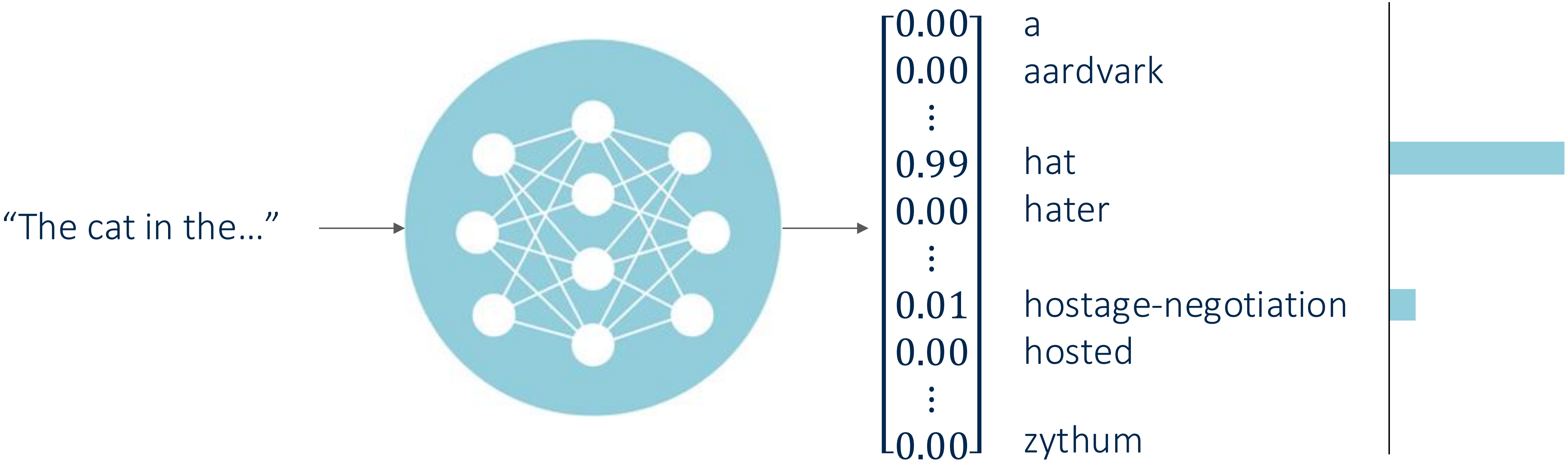


→

$$\begin{array}{cl} \begin{bmatrix} 0.00 \\ 0.00 \\ \vdots \\ 0.99 \\ 0.00 \\ \vdots \\ 0.01 \\ 0.00 \\ \vdots \\ 0.00 \end{bmatrix} & \begin{array}{l} \text{a} \\ \text{aardvark} \\ \\ \text{hat} \\ \text{hater} \\ \\ \text{hostage-negotiation} \\ \text{hosted} \\ \\ \text{zythum} \end{array} \end{array}$$

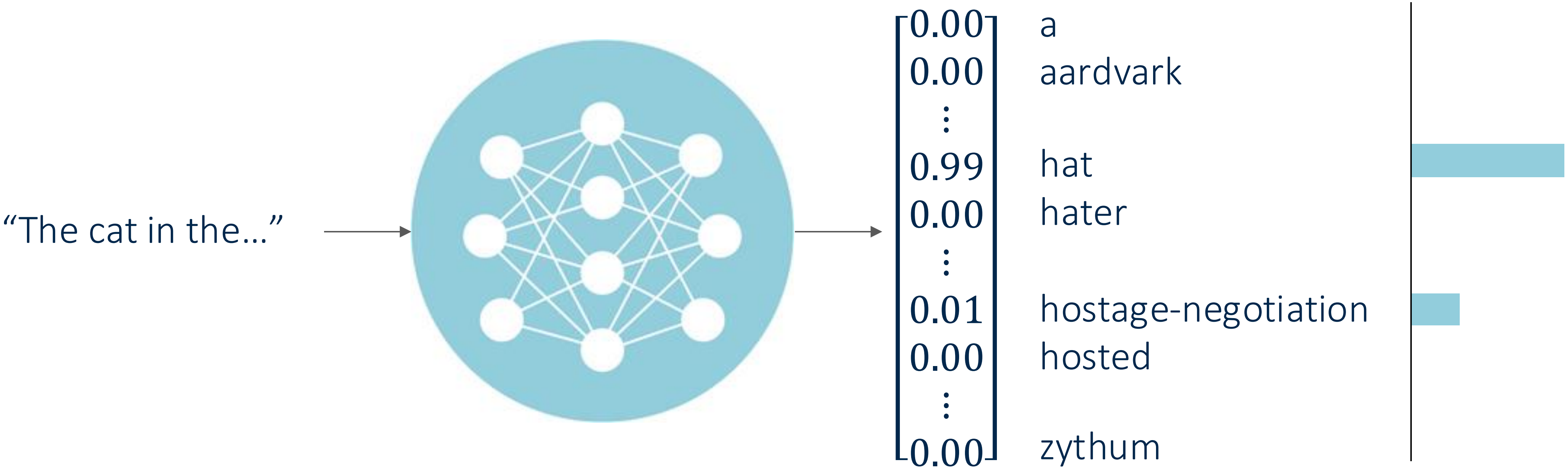
Temperature: T = 0

Unembedding



Temperature: T = 1

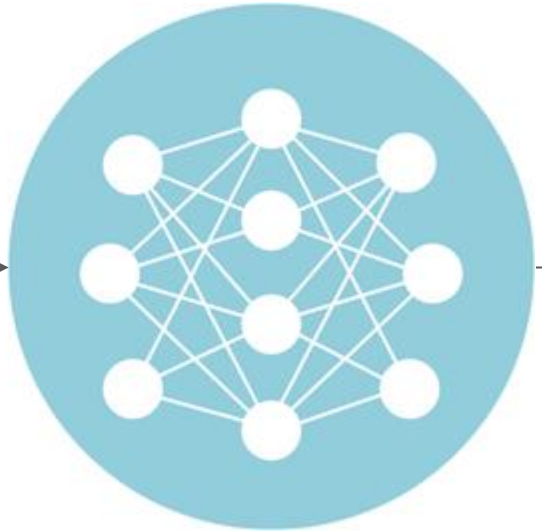
Unembedding



Temperature: T = 2

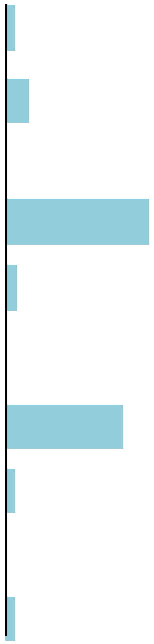
Unembedding

"The cat in the…" →

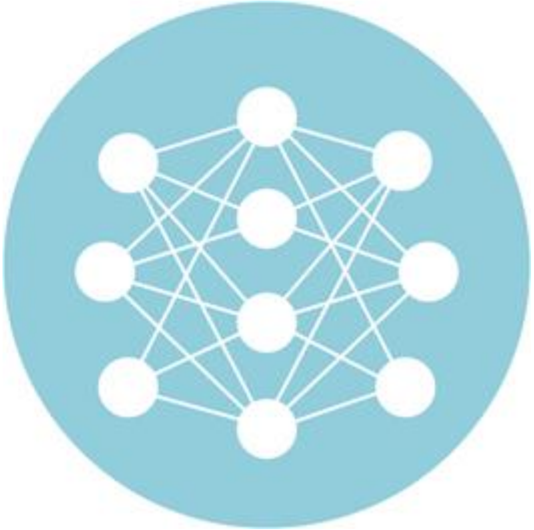


→

$$\begin{array}{cl} \begin{bmatrix} 0.00 \\ 0.00 \\ \vdots \\ 0.99 \\ 0.00 \\ \vdots \\ 0.01 \\ 0.00 \\ \vdots \\ 0.00 \end{bmatrix} & \begin{array}{l} \text{a} \\ \text{aardvark} \\ \\ \text{hat} \\ \text{hater} \\ \\ \text{hostage-negotiation} \\ \text{hosted} \\ \\ \text{zythum} \end{array} \end{array}$$



Temperature: $T = 20$



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

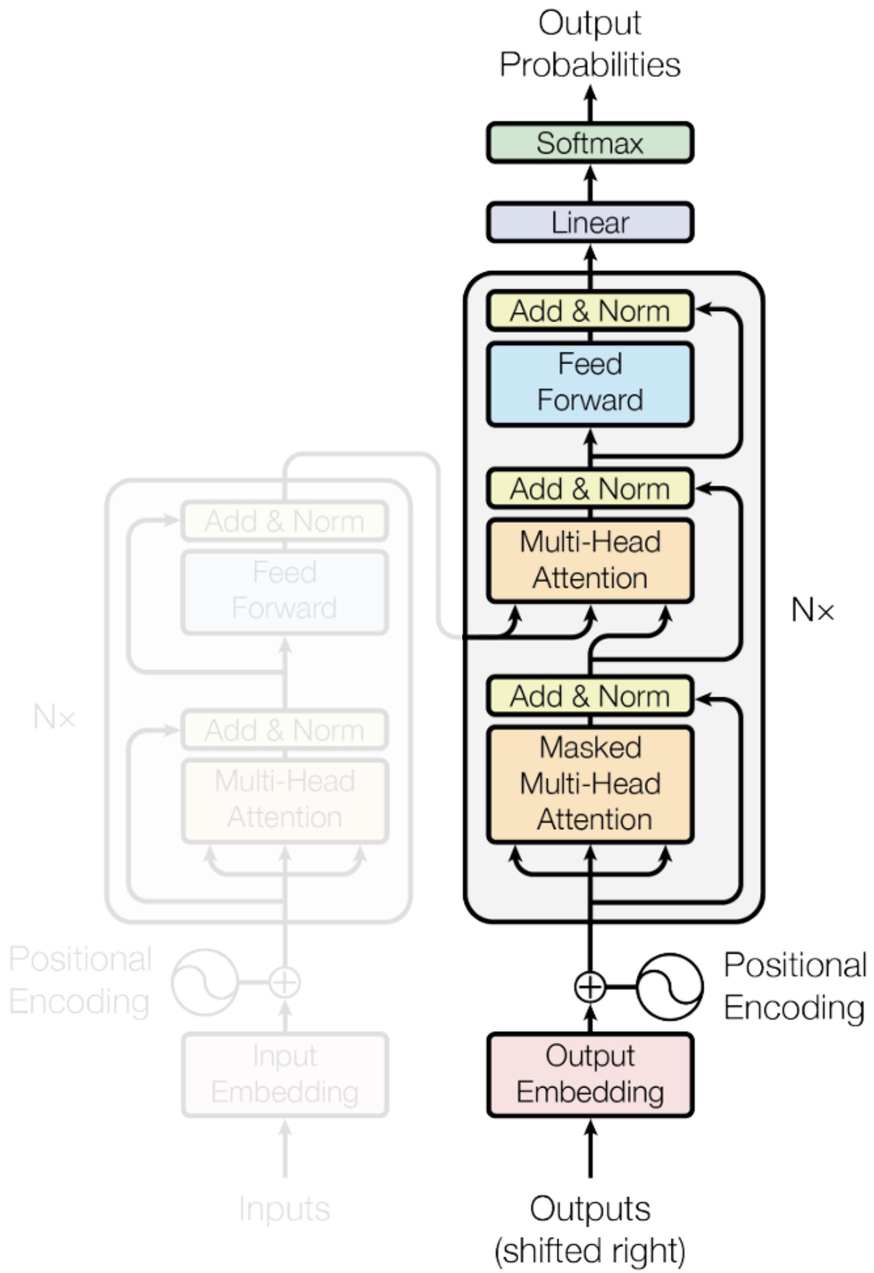
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

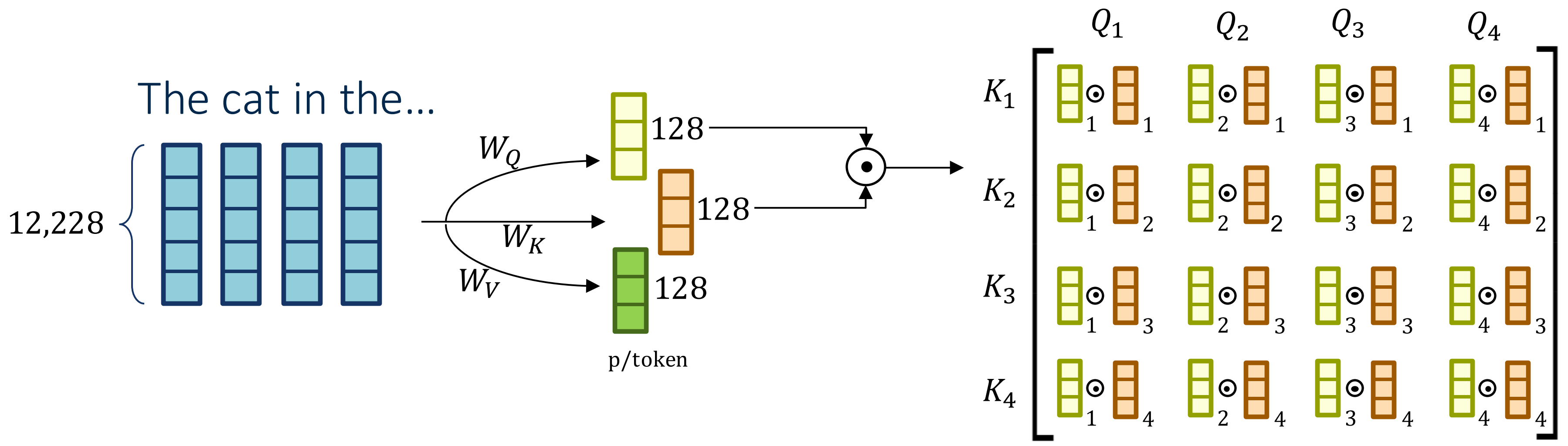
1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [31, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [34, 22, 14].

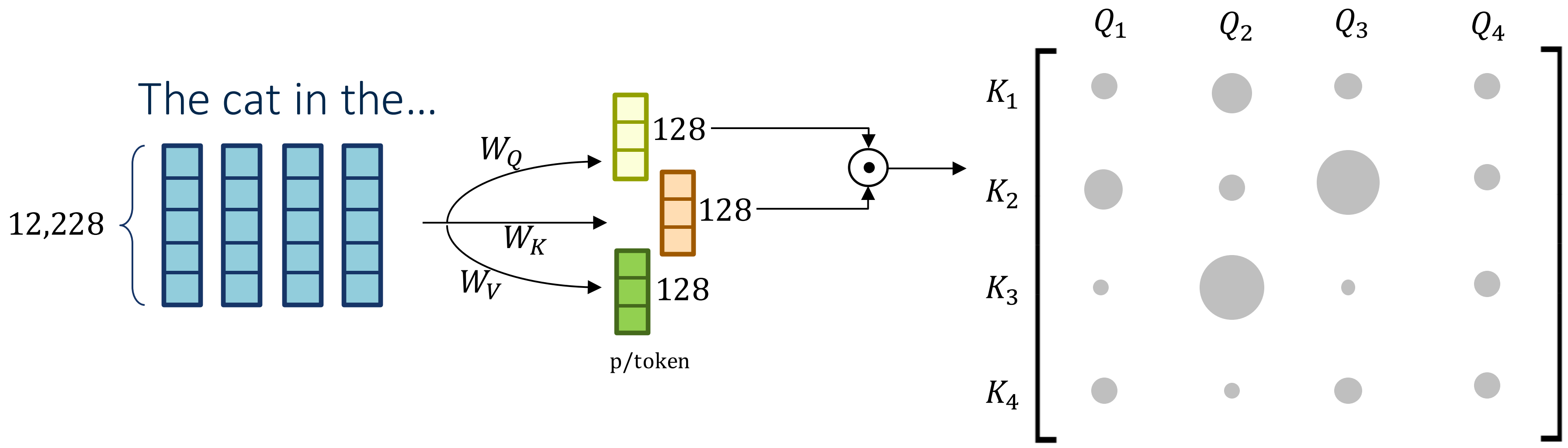
Recurrent models typically factor computation along the symbol positions of the input and output



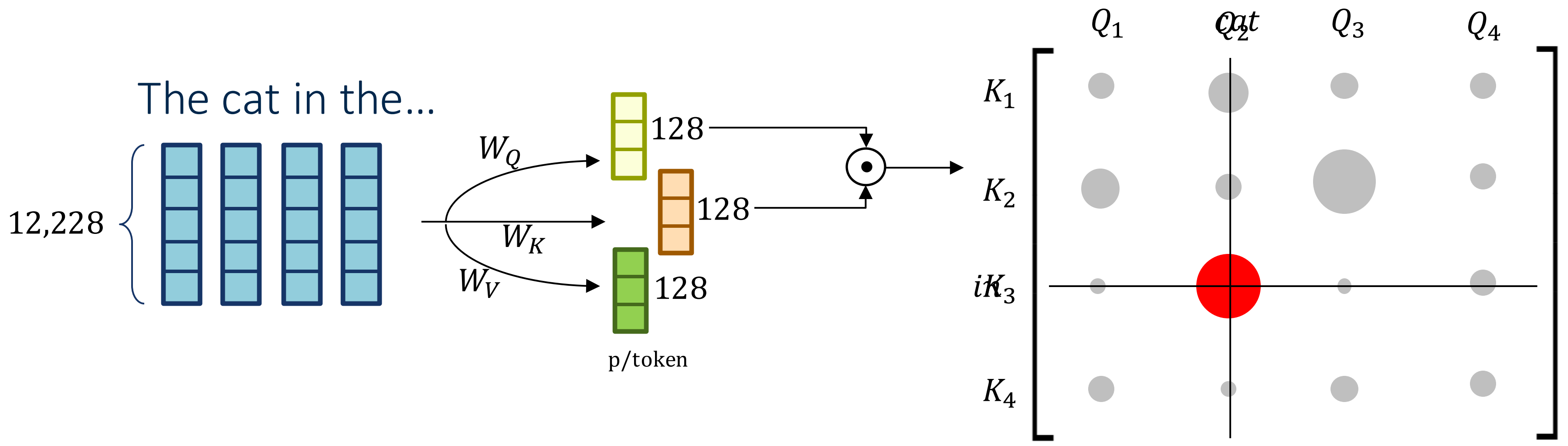
The Anatomy of a Language Model: GPT-3



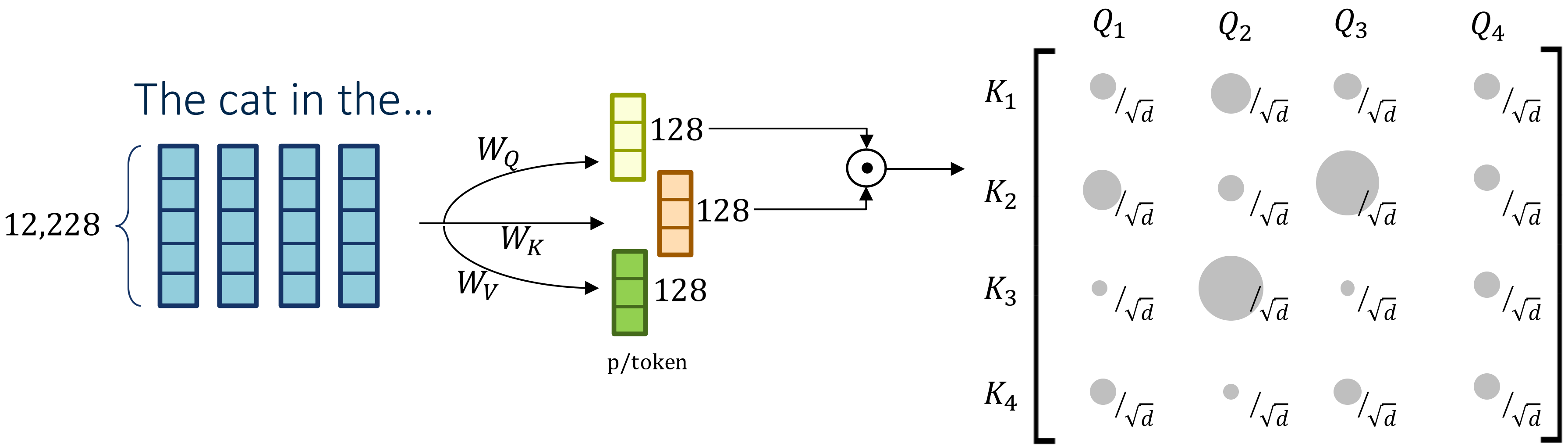
The Anatomy of a Language Model: GPT-3



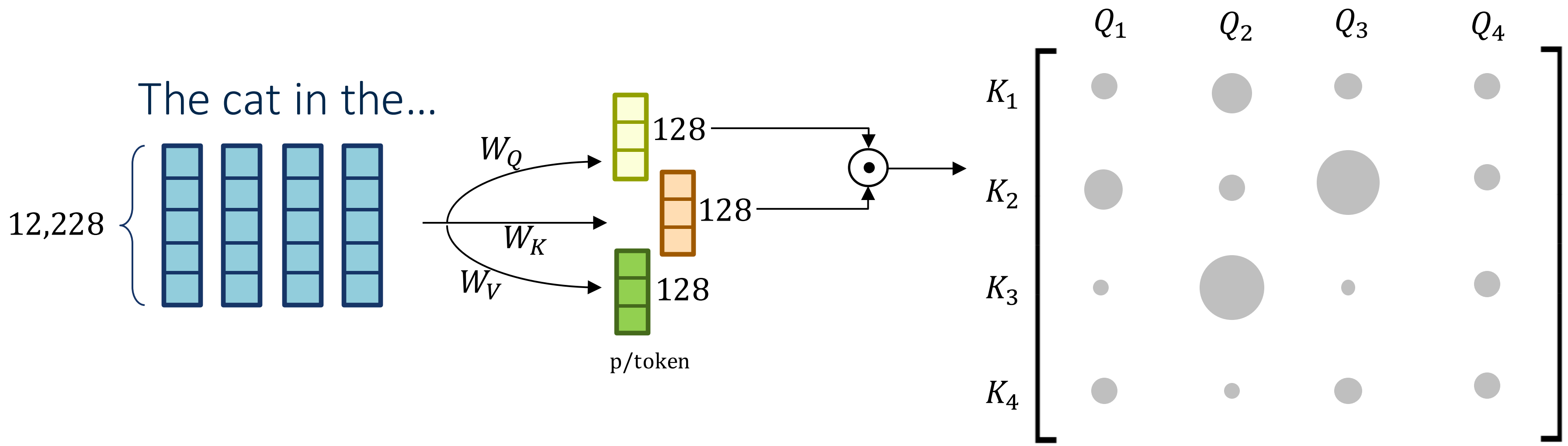
The Anatomy of a Language Model: GPT-3

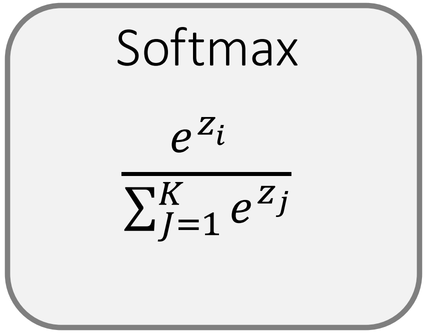


The Anatomy of a Language Model: GPT-3

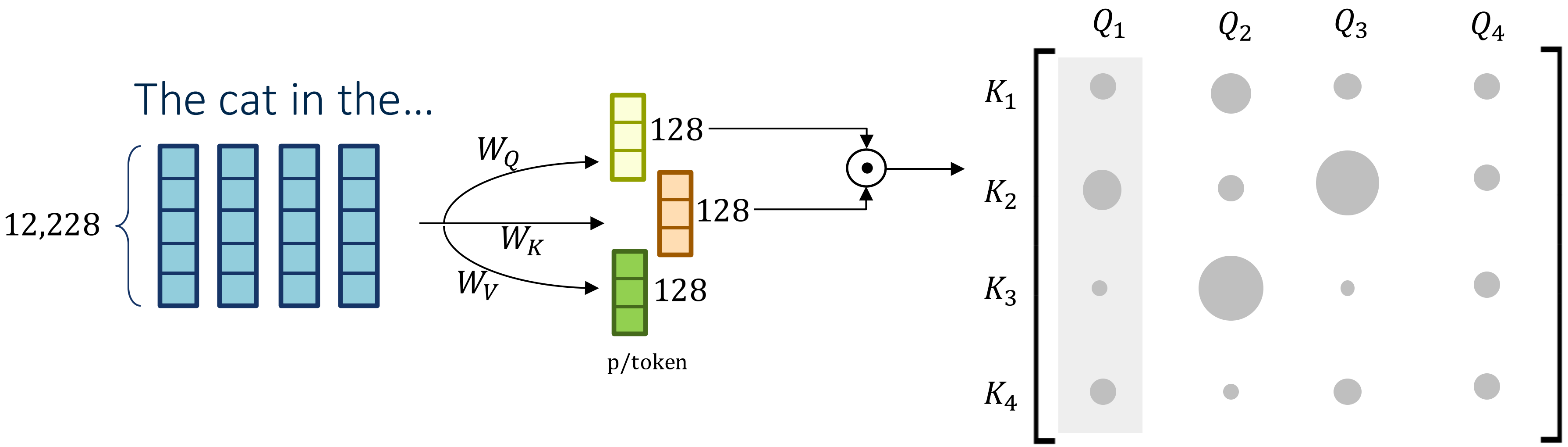


The Anatomy of a Language Model: GPT-3





The Anatomy of a Language Model: GPT-3

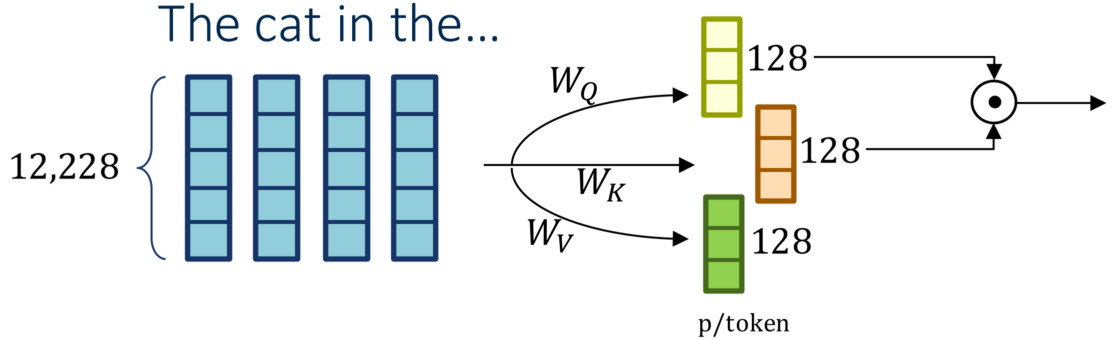


Softmax

$$\frac{e^{z_i}}{\sum_{J=1}^{K} e^{z_j}}$$

The Anatomy of a Language Model: GPT-3

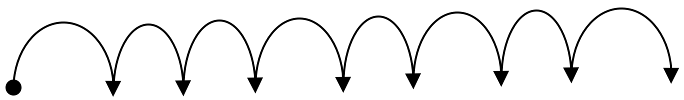
The cat in the…



$$
\begin{array}{c|cccc}
 & Q_1 & Q_2 & Q_3 & Q_4 \\
\hline
K_1 & 0.01 & 0.01 & 0.00 & 0.23 \\
K_2 & 0.97 & 0.00 & 0.99 & 0.27 \\
K_3 & 0.01 & 0.99 & 0.00 & 0.21 \\
K_4 & 0.01 & 0.00 & 0.01 & 0.29
\end{array}
$$

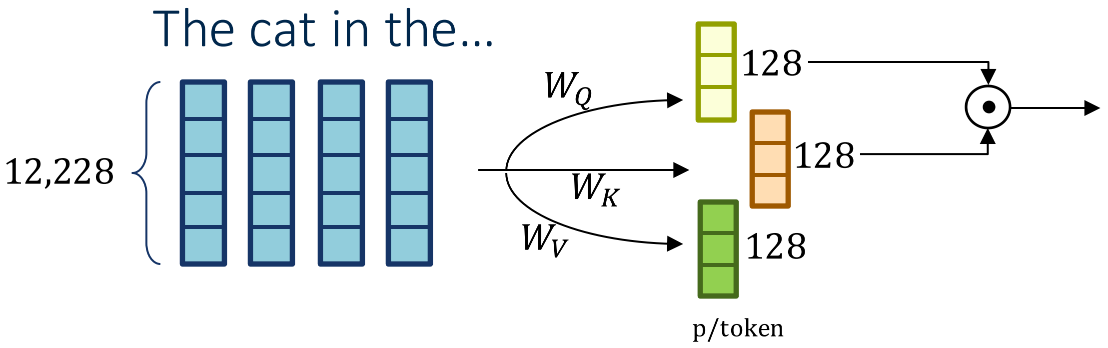
Softmax

$$\frac{e^{z_i}}{\sum_{J=1}^{K} e^{z_j}}$$



The cat in the hat sat on the mat

The Anatomy of a Language Model: GPT-3



$$
\begin{array}{c|cccc}
 & Q_1 & Q_2 & Q_3 & Q_4 \\
\hline
K_1 & 0.01 & 0.01 & 0.00 & 0.23 \\
K_2 & 0.97 & 0.00 & 0.99 & 0.27 \\
K_3 & 0.01 & 0.99 & 0.00 & 0.21 \\
K_4 & 0.01 & 0.00 & 0.01 & 0.29
\end{array}
$$

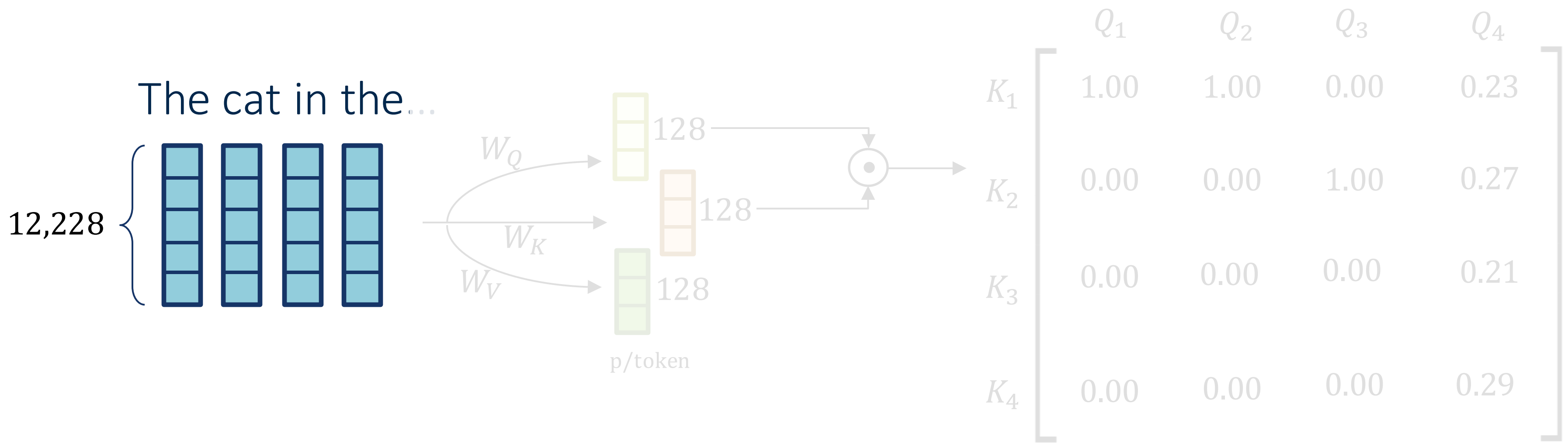
$$
\begin{bmatrix}
1.00 & 1.00 & 1.00 & 1.00 \\
-\infty & 1.00 & 1.00 & 1.00 \\
-\infty & -\infty & 1.00 & 1.00 \\
-\infty & -\infty & -\infty & 1.00
\end{bmatrix}
$$

Causal Mask

Softmax

$$\frac{e^{z_i}}{\sum_{J=1}^{K} e^{z_j}}$$

The Anatomy of a Language Model: GPT-3

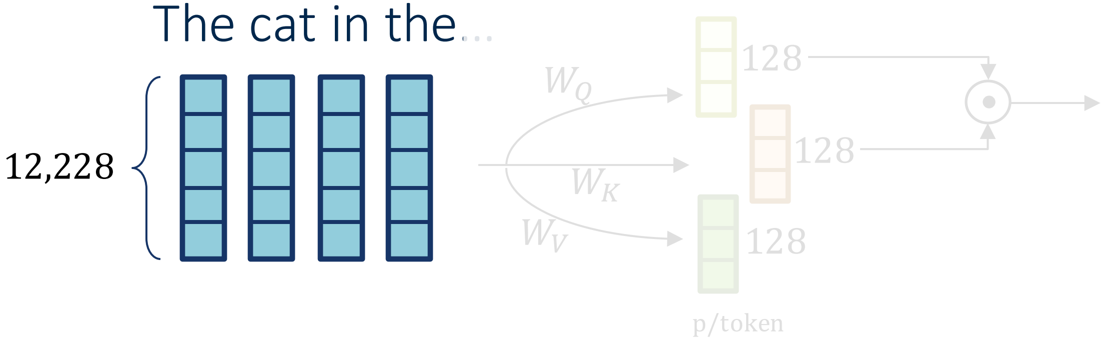


The cat in the...

haste
hasty
hat
hater
haughtiness
haughty

The Anatomy of a Language Model: GPT-3

The cat in the...

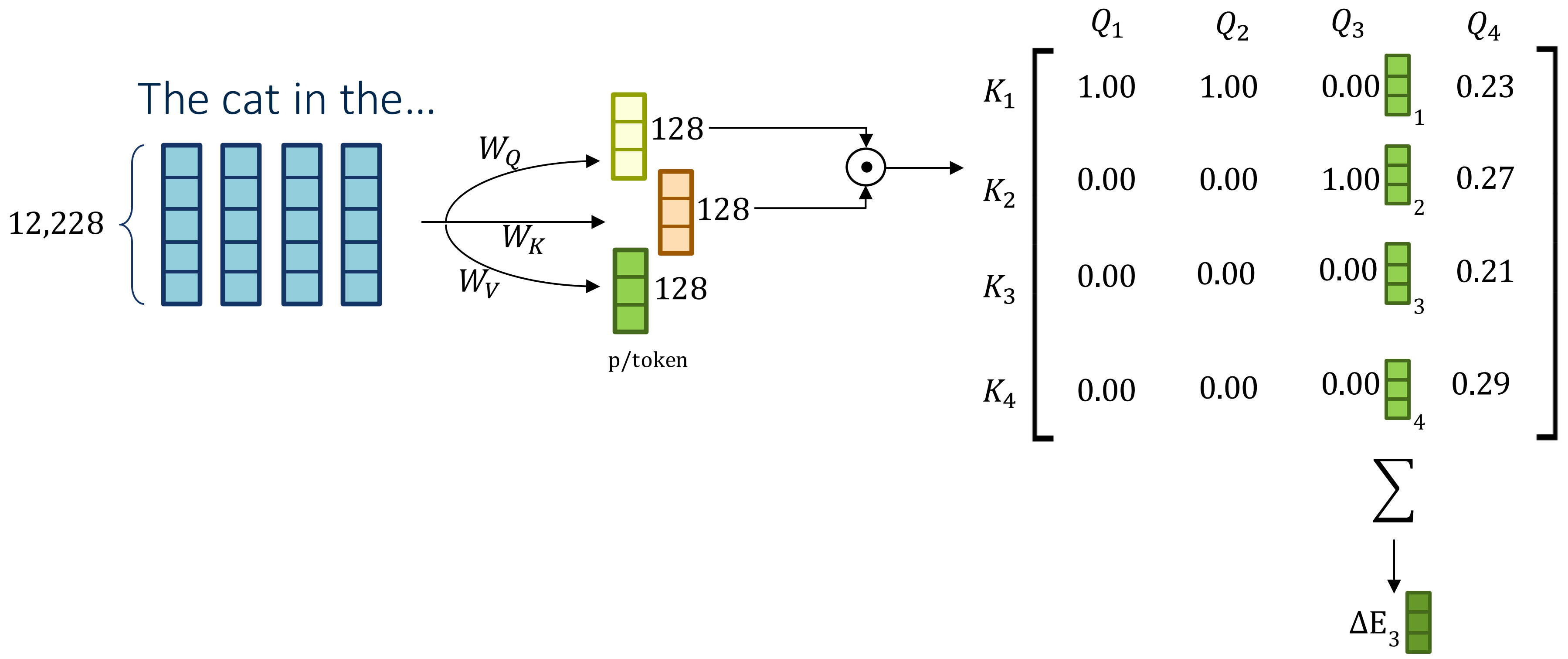


$$
\begin{array}{c|cccc}
 & Q_1 & Q_2 & Q_3 & Q_4 \\
\hline
K_1 & 1.00 & 1.00 & 0.00 & 0.23 \\
K_2 & 0.00 & 0.00 & 1.00 & 0.27 \\
K_3 & 0.00 & 0.00 & 0.00 & 0.21 \\
K_4 & 0.00 & 0.00 & 0.00 & 0.29
\end{array}
$$

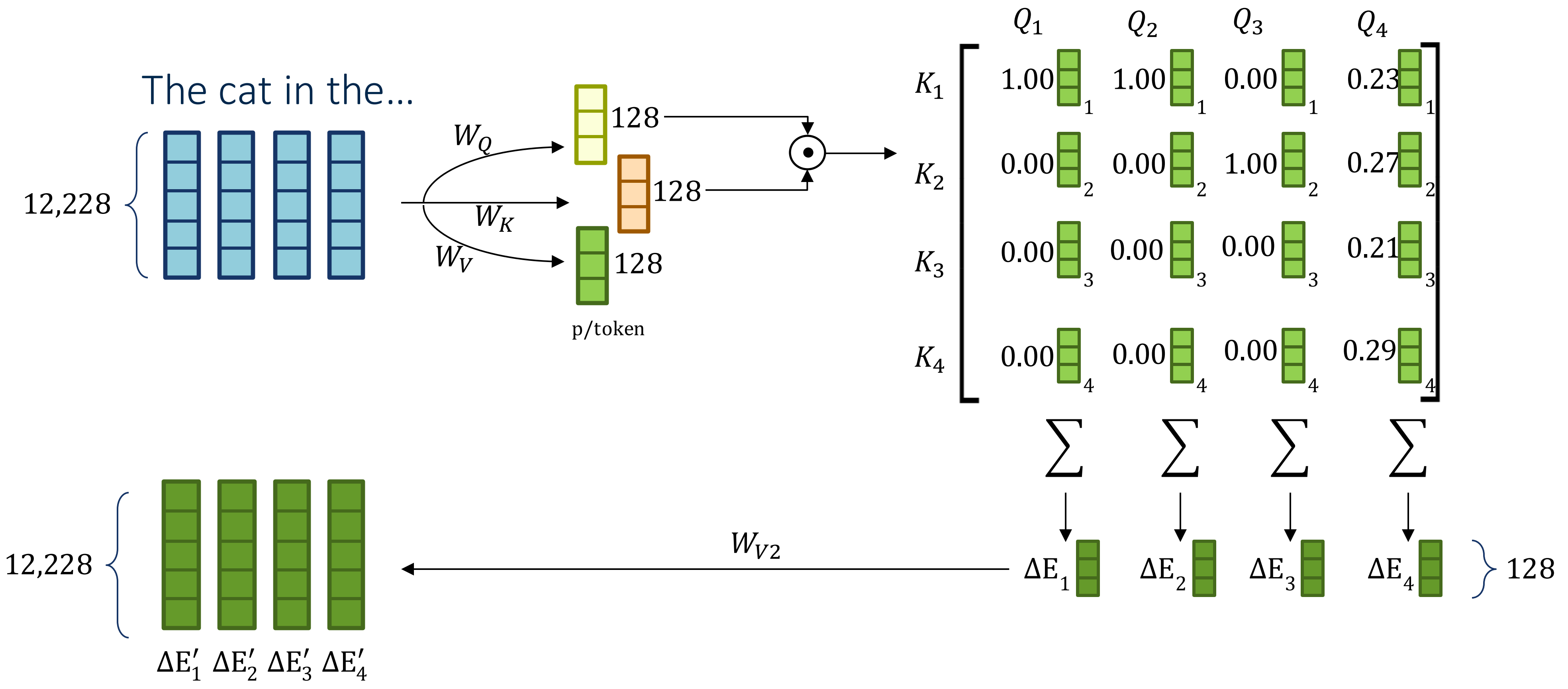
The cat in the...

haste
hasty
hat
hater
haughtiness
haughty

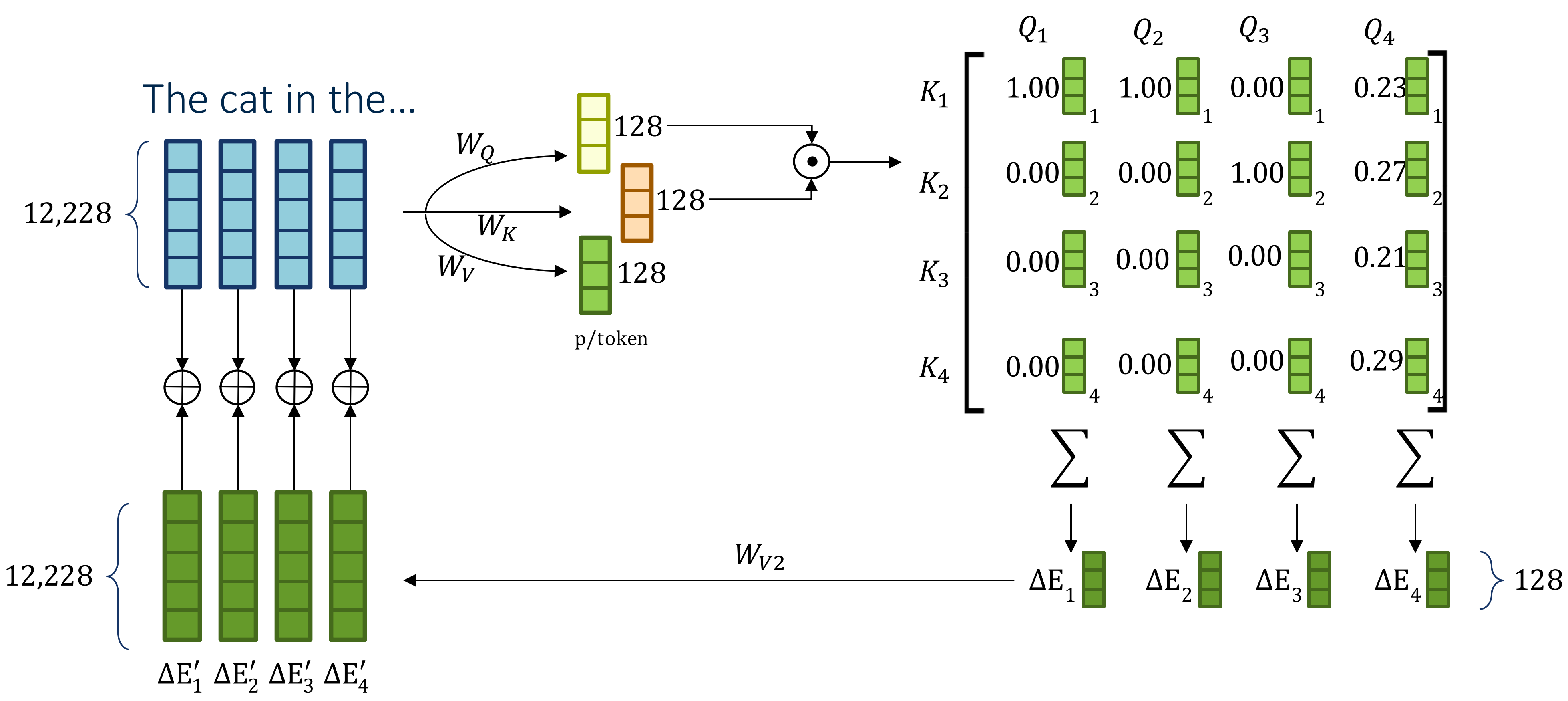
The Anatomy of a Language Model: GPT-3



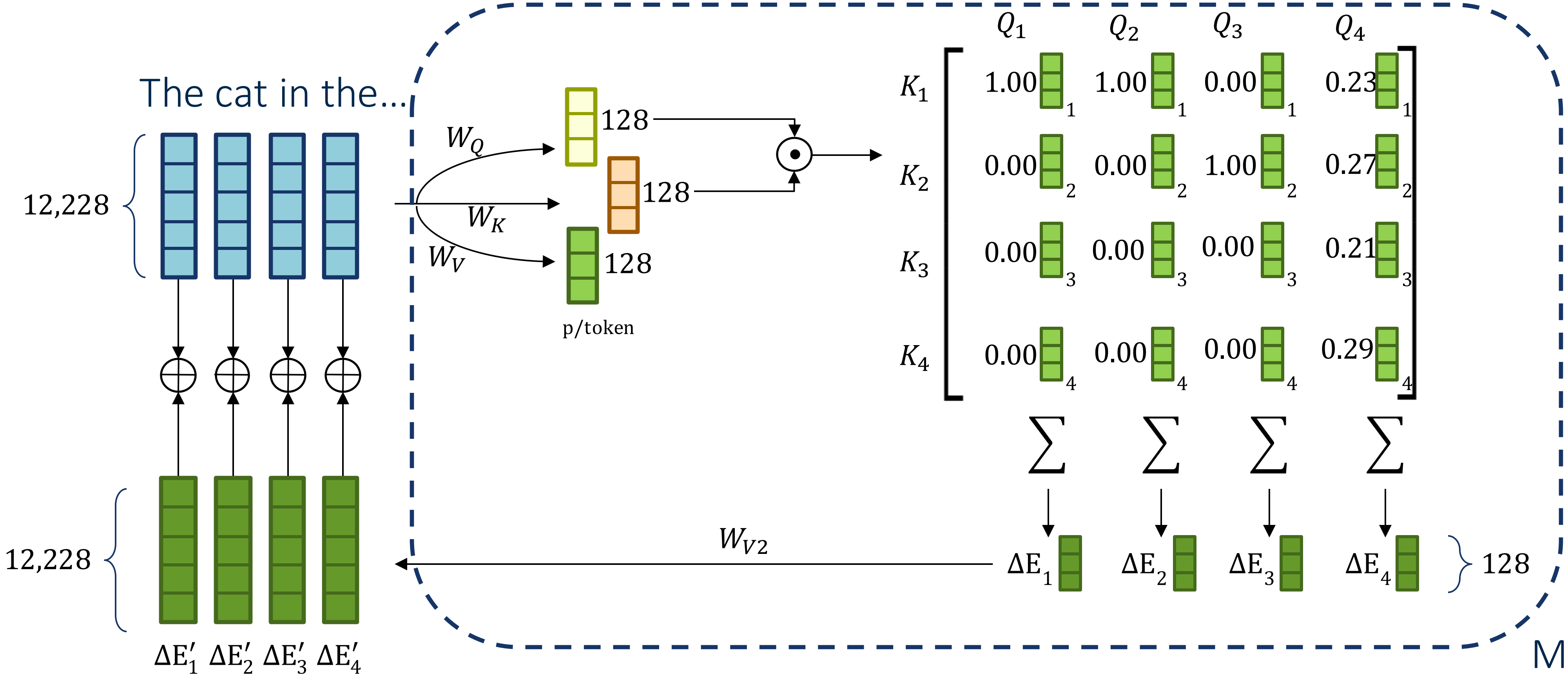
The Anatomy of a Language Model: GPT-3



The Anatomy of a Language Model: GPT-3



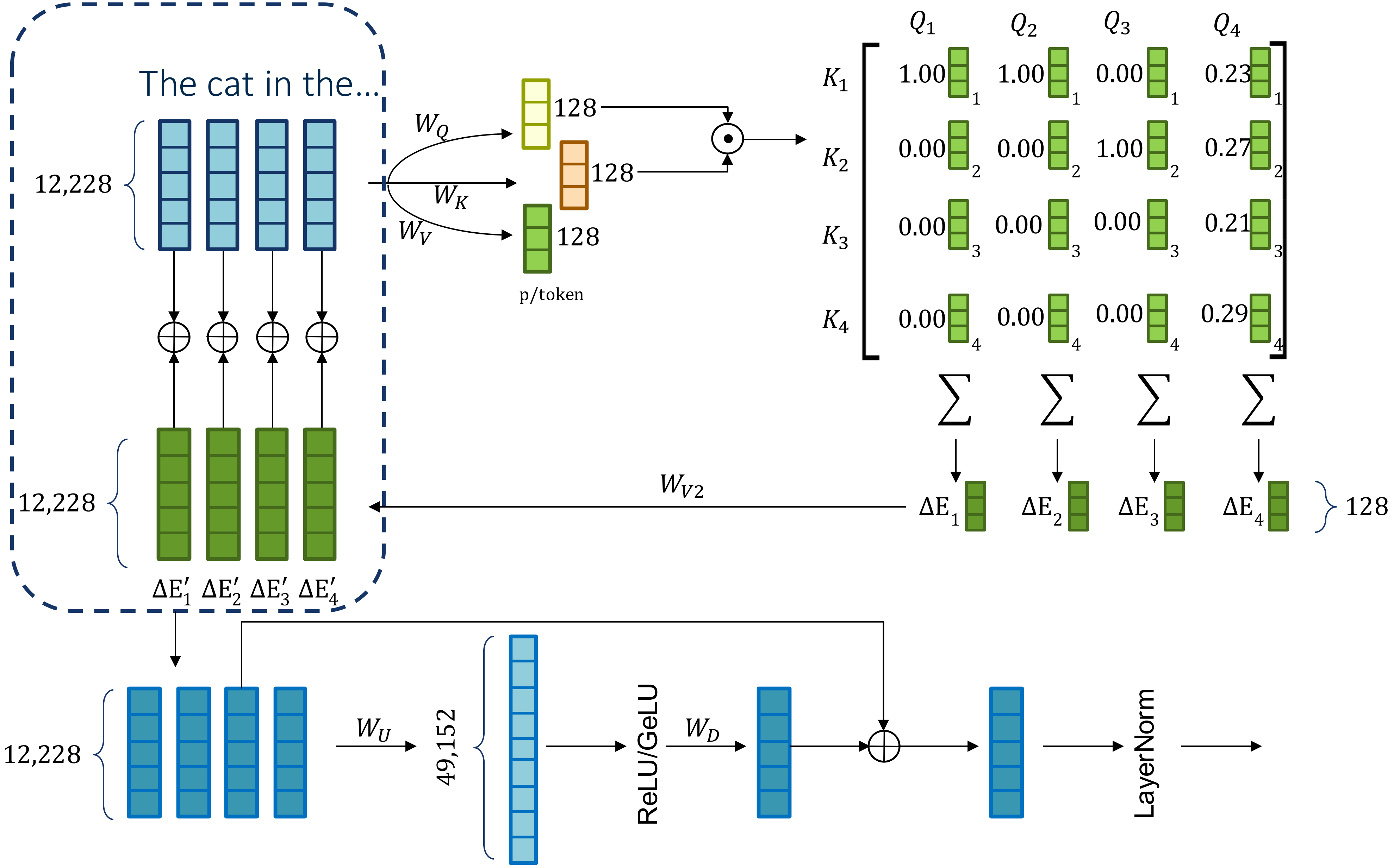
The Anatomy of a Language Model: GPT-3



Multi-head Attention
GPT-3: 96 heads

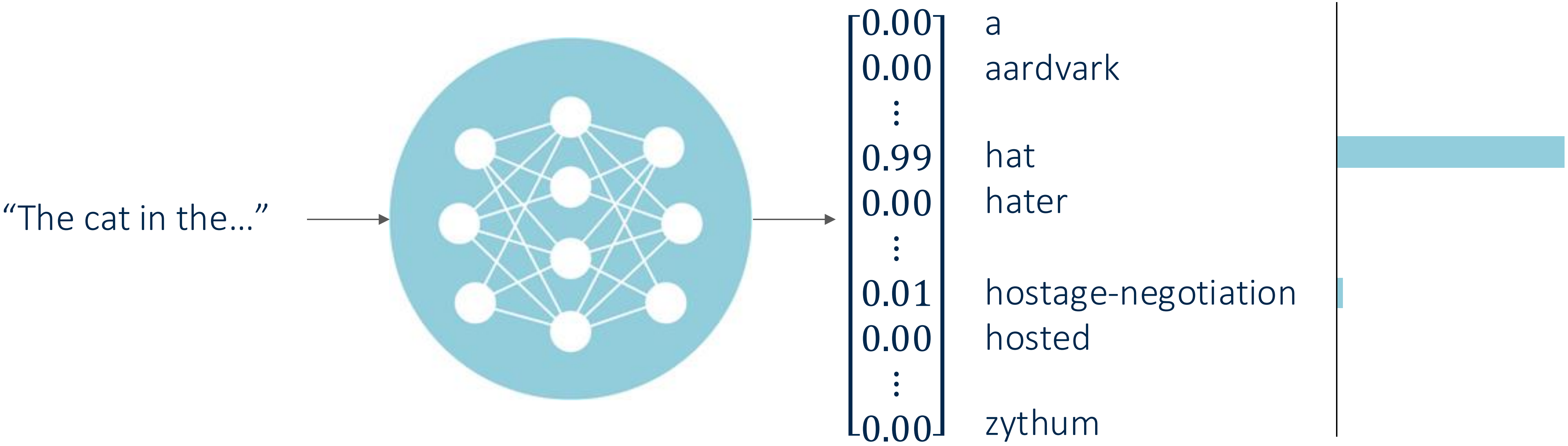
$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The Anatomy of a Language Model: GPT-3

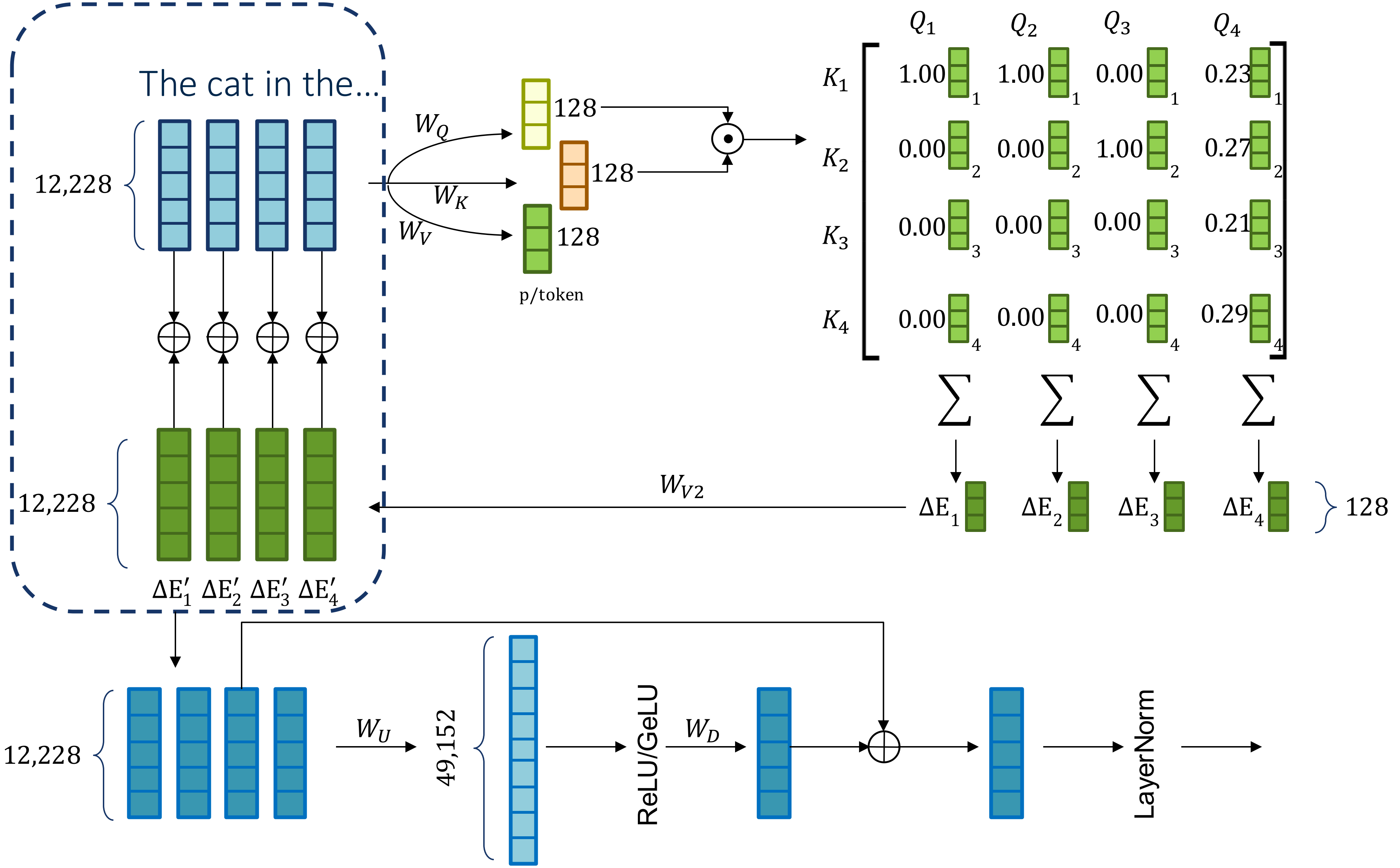


Repeat 96 times.

Unembedding

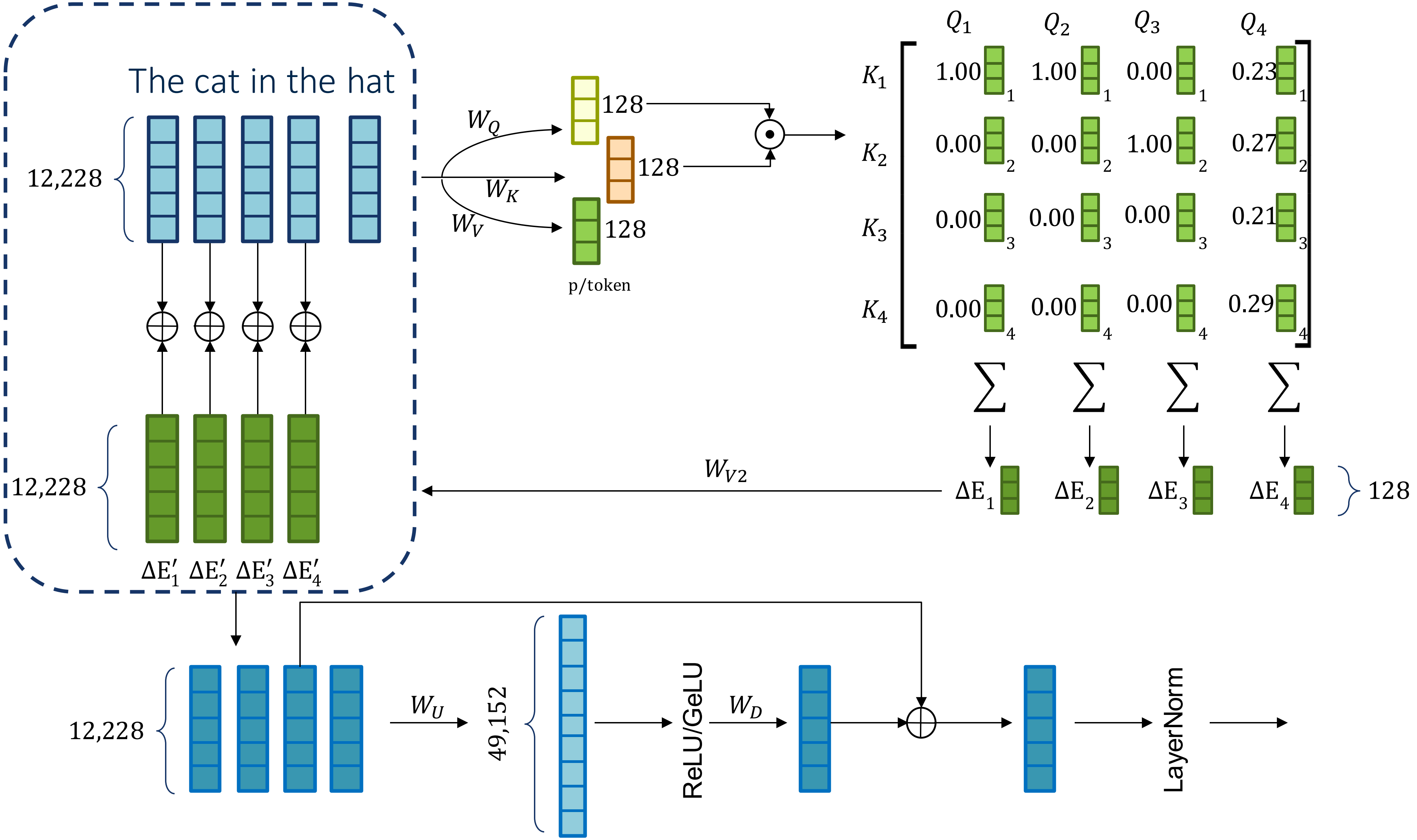


The Anatomy of a Language Model: GPT-3



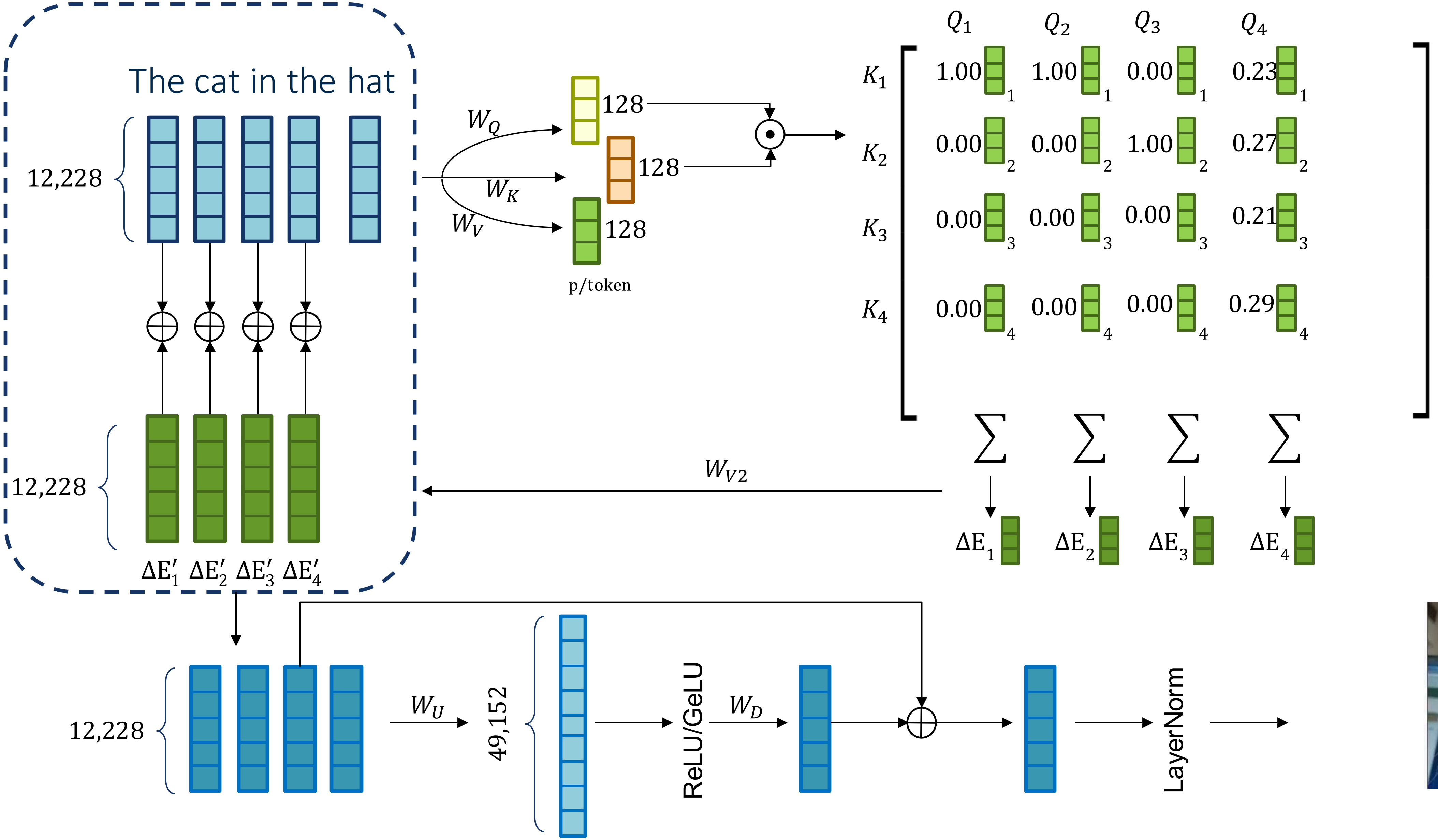
Repeat 96 times.

The Anatomy of a Language Model: GPT-3



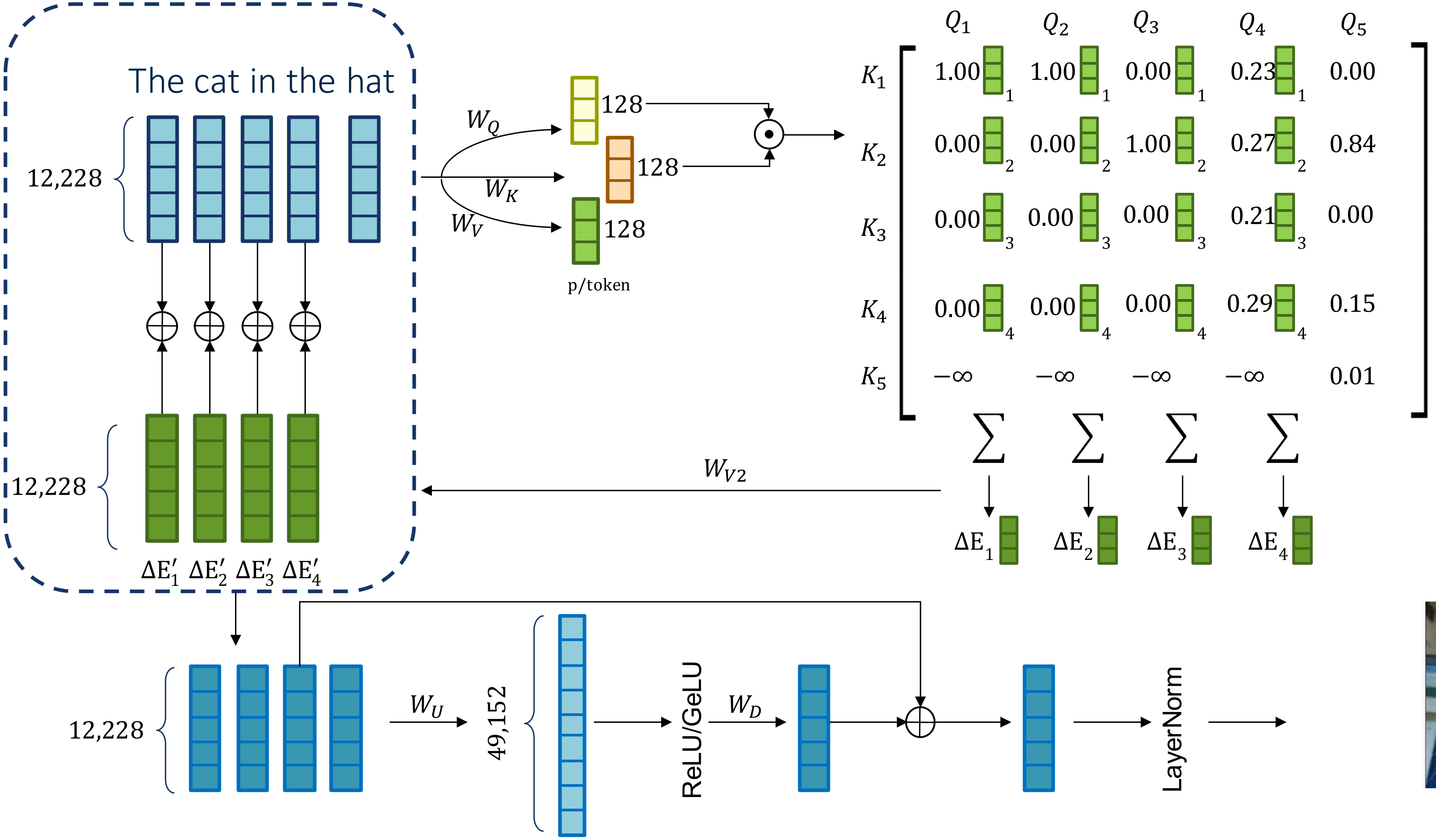
Repeat 96 times.

The Anatomy of a Language Model: GPT-3



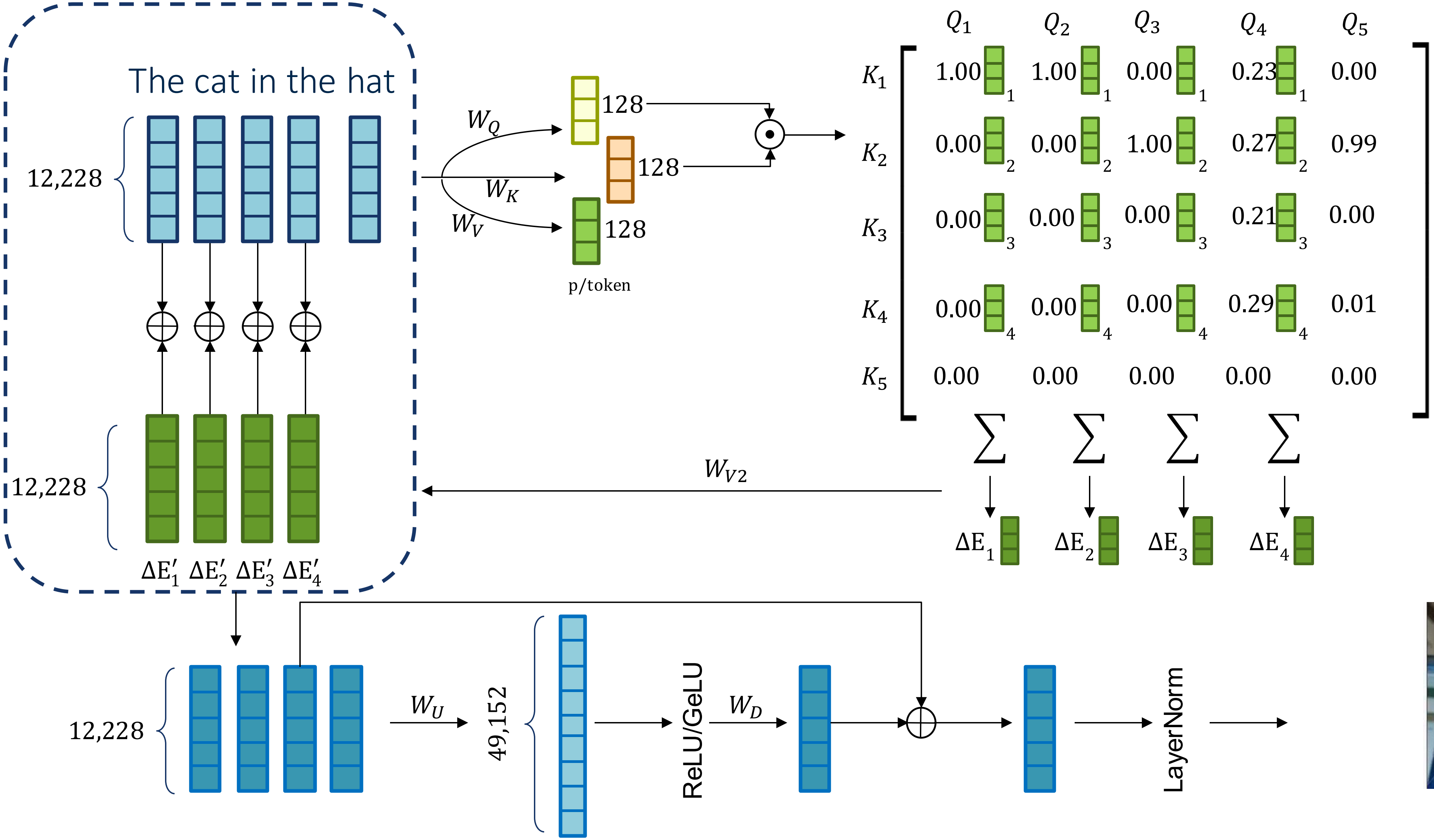
Repeat 96 times.

The Anatomy of a Language Model: GPT-3



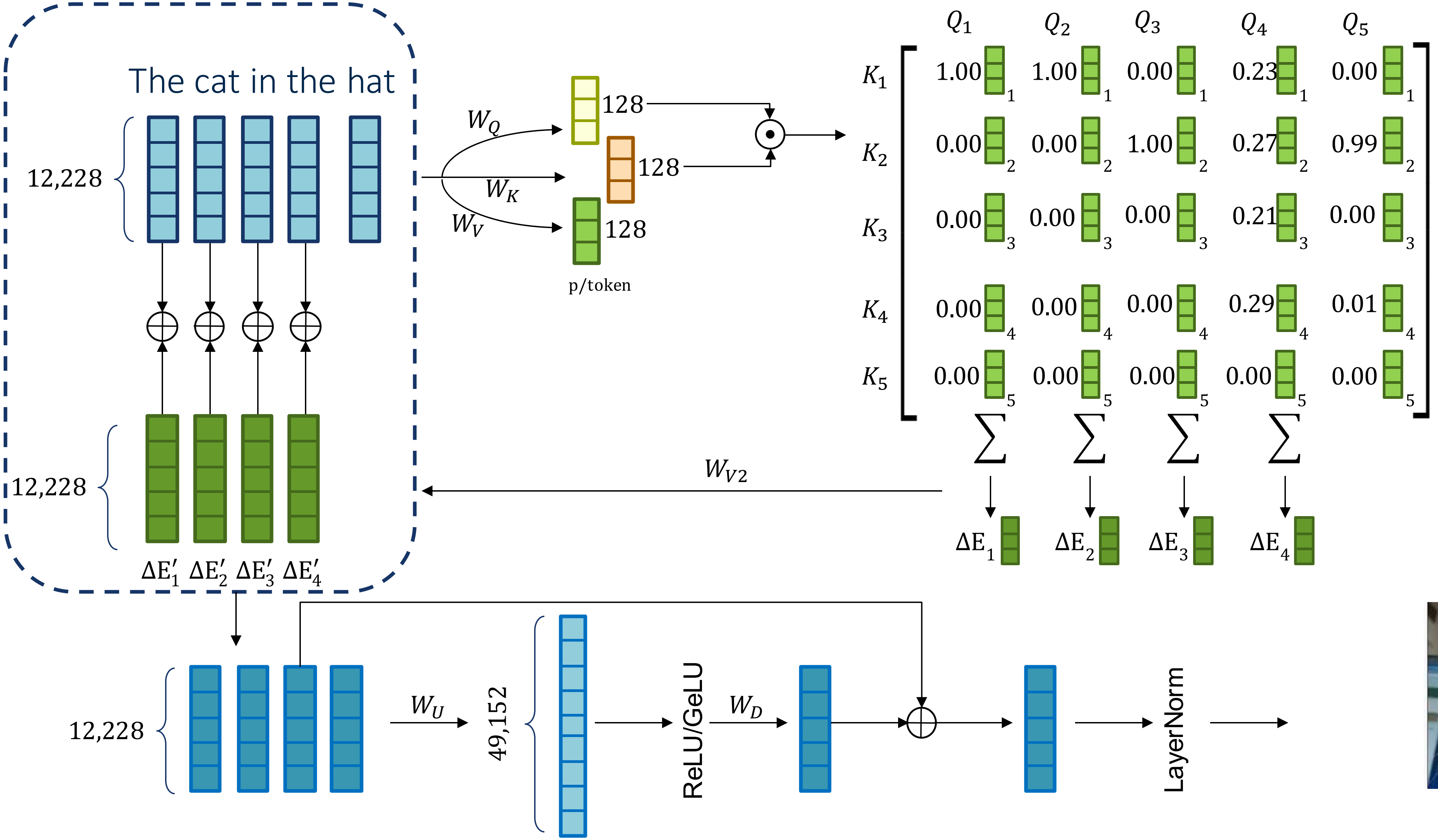
Repeat 96 times.

The Anatomy of a Language Model: GPT-3



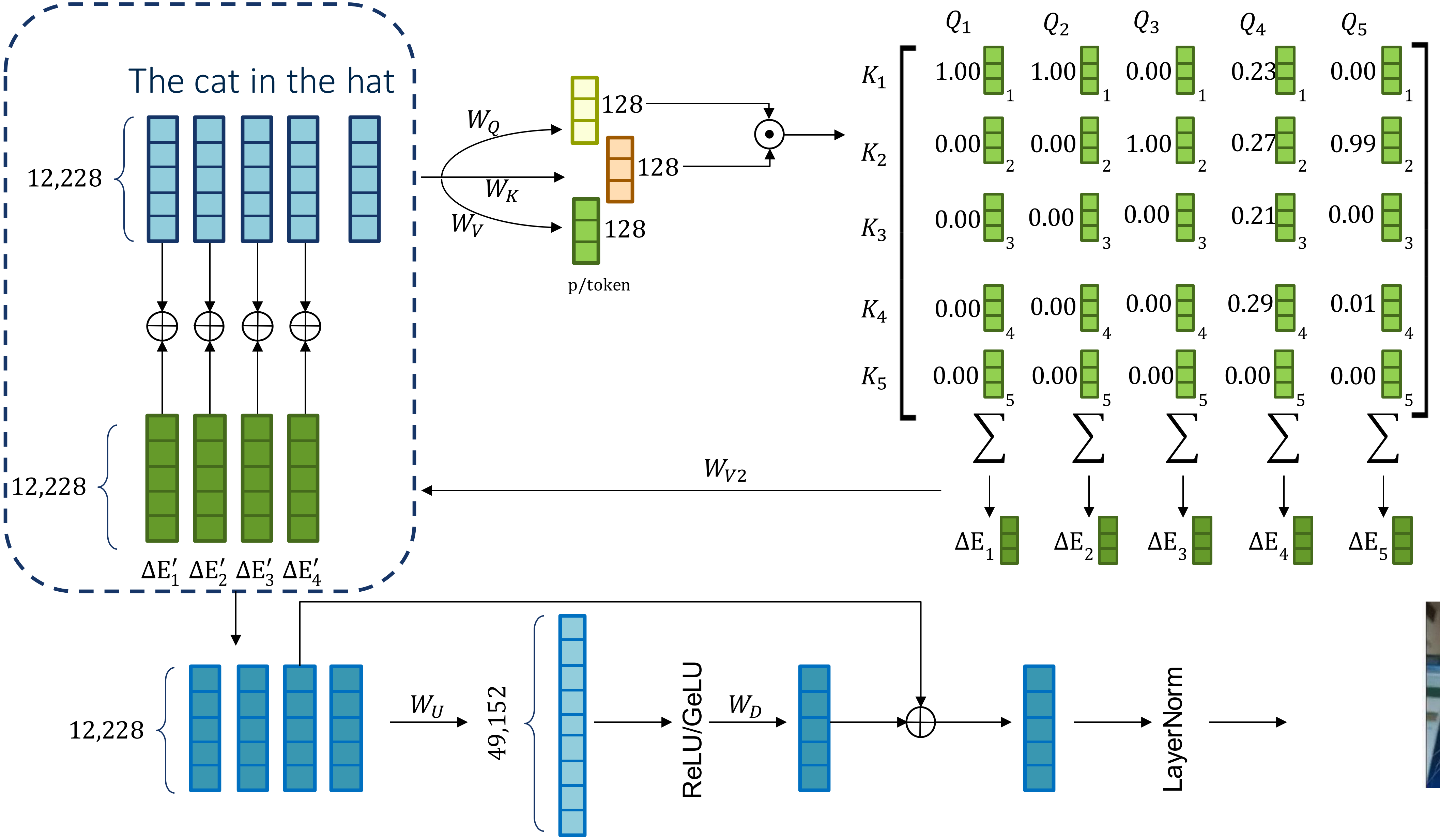
Repeat 96 times.

The Anatomy of a Language Model: GPT-3



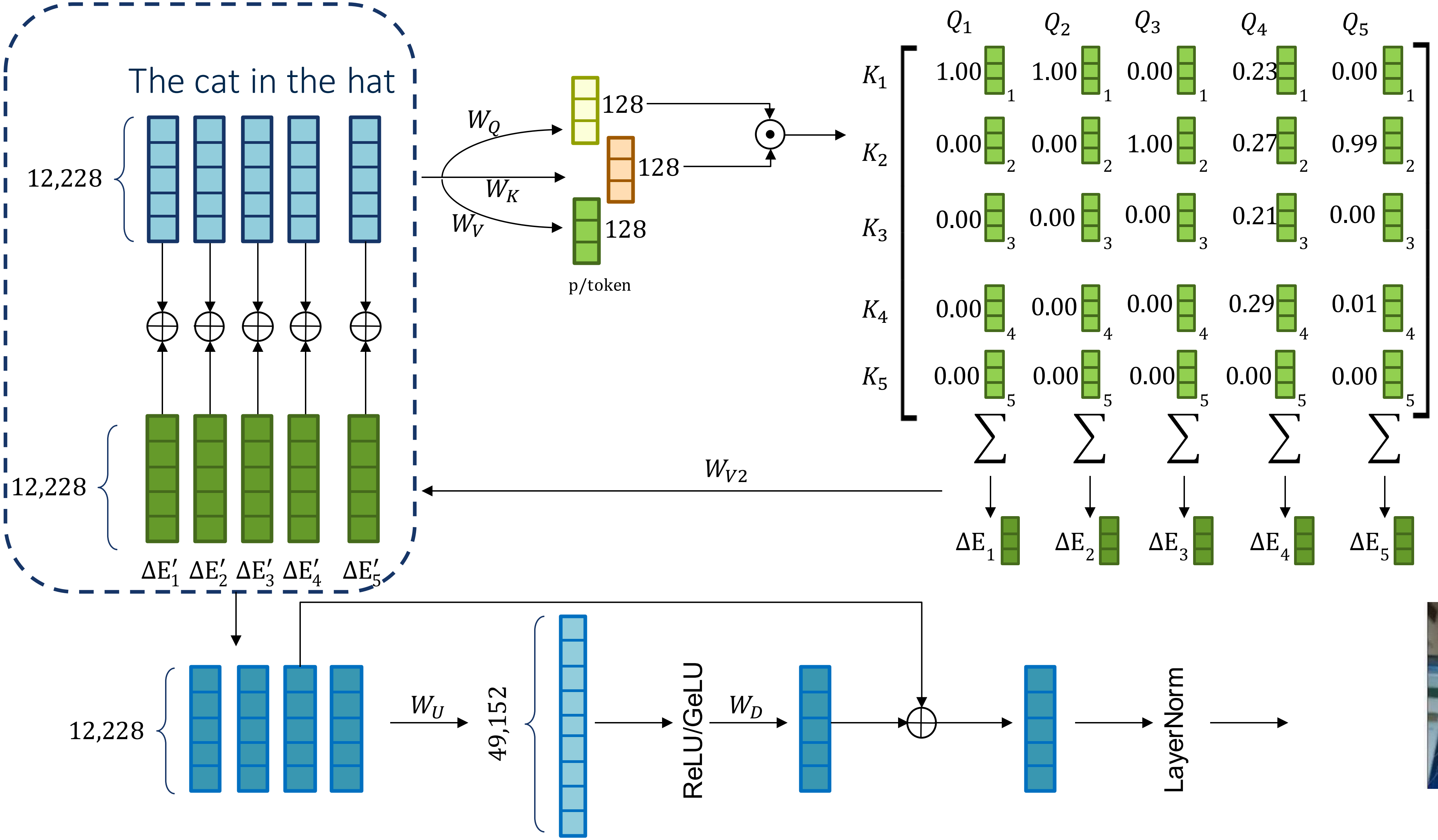
Repeat 96 times.

The Anatomy of a Language Model: GPT-3



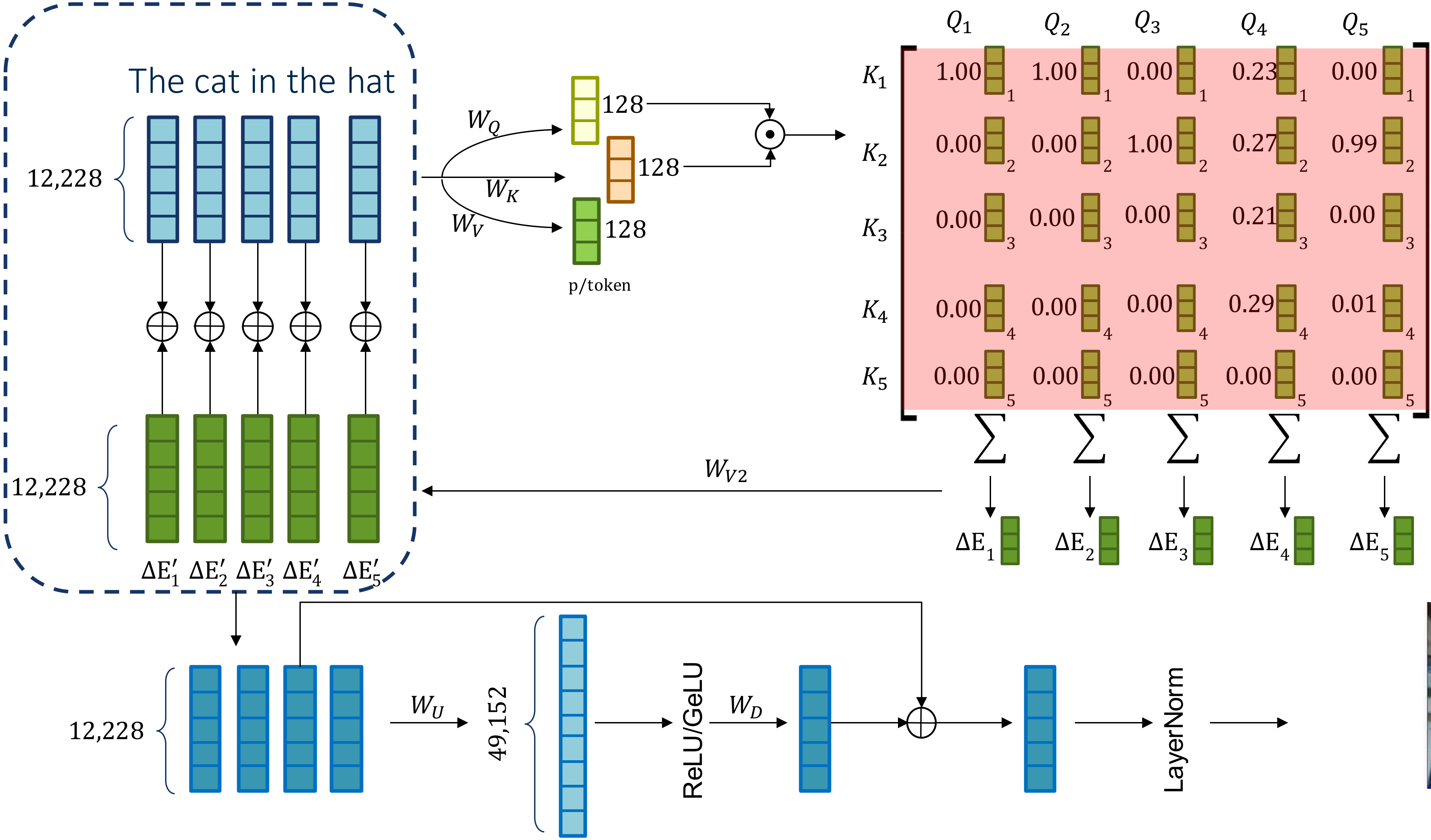
Repeat 96 times.

The Anatomy of a Language Model: GPT-3



Repeat 96 times.

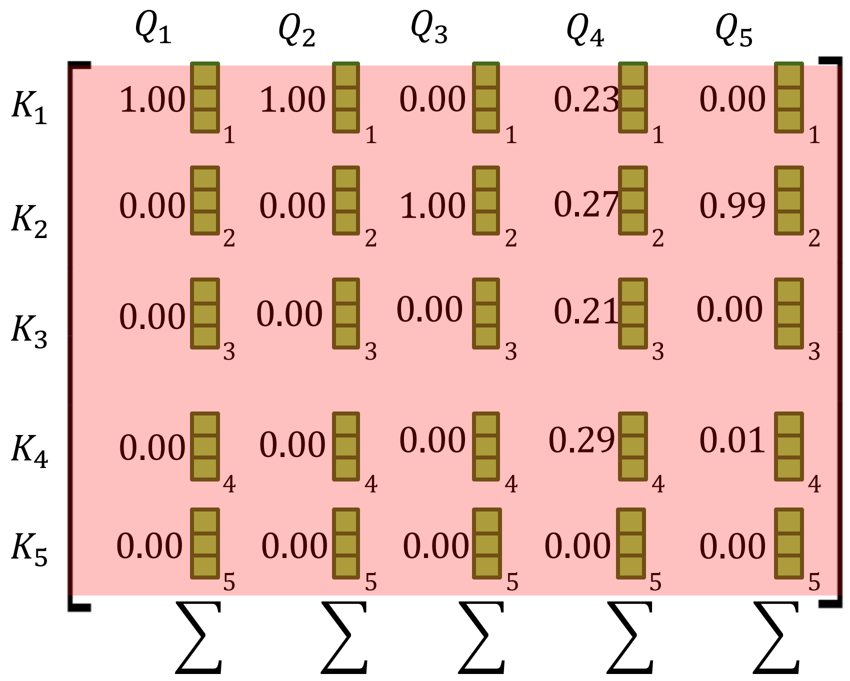
The Anatomy of a Language Model: GPT-3



Repeat 96 times.

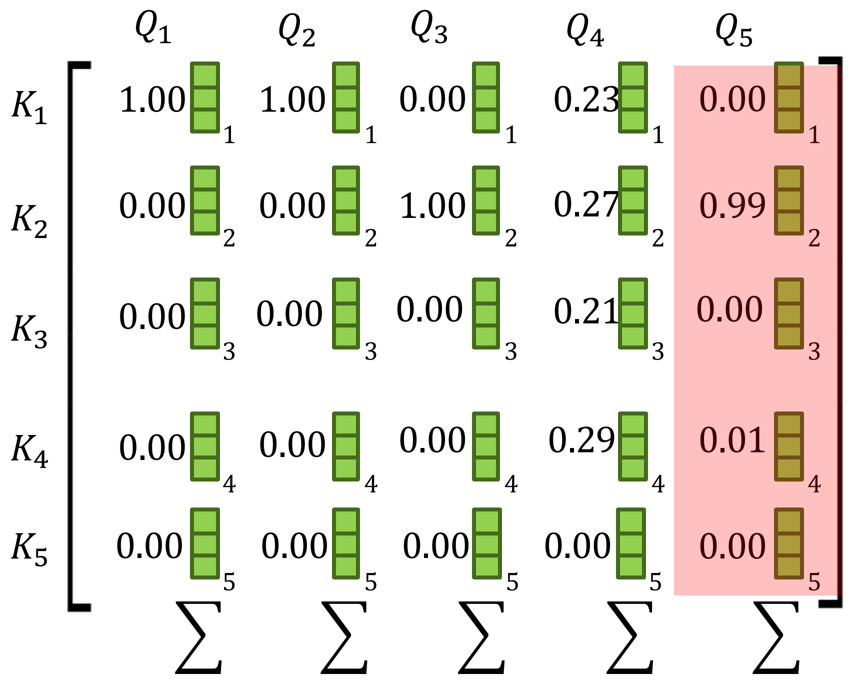
The Anatomy of a Language Model: GPT-3

Given a 1M context window, there would be 10^{12} elements in the matrix.

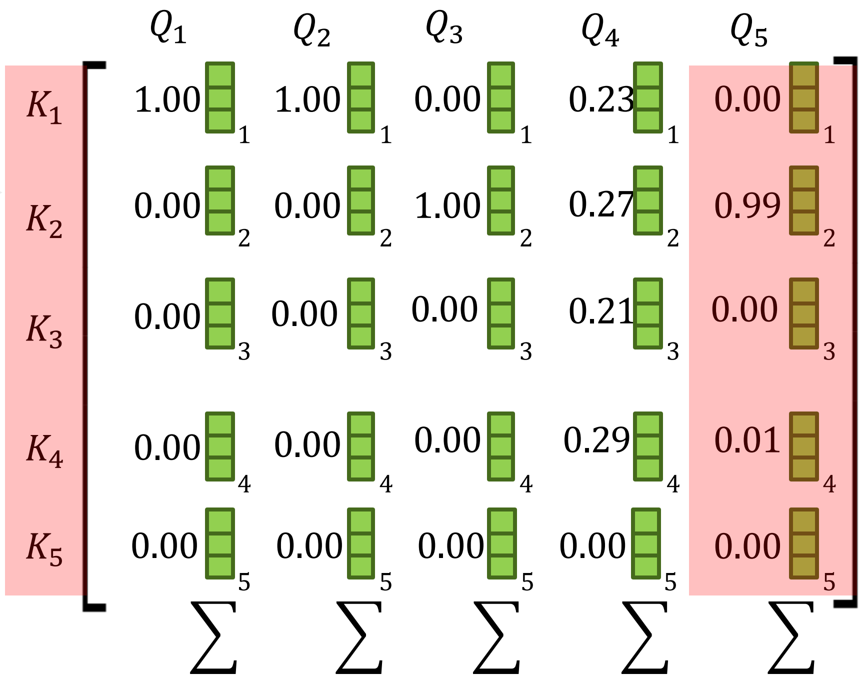


The Anatomy of a Language Model: GPT-3

Given a 1M context window, there would be 10^{12} elements in the matrix.



The Anatomy of a Language Model: GPT-3



Given a 1M context window, there would be 10^{12} elements in the matrix.

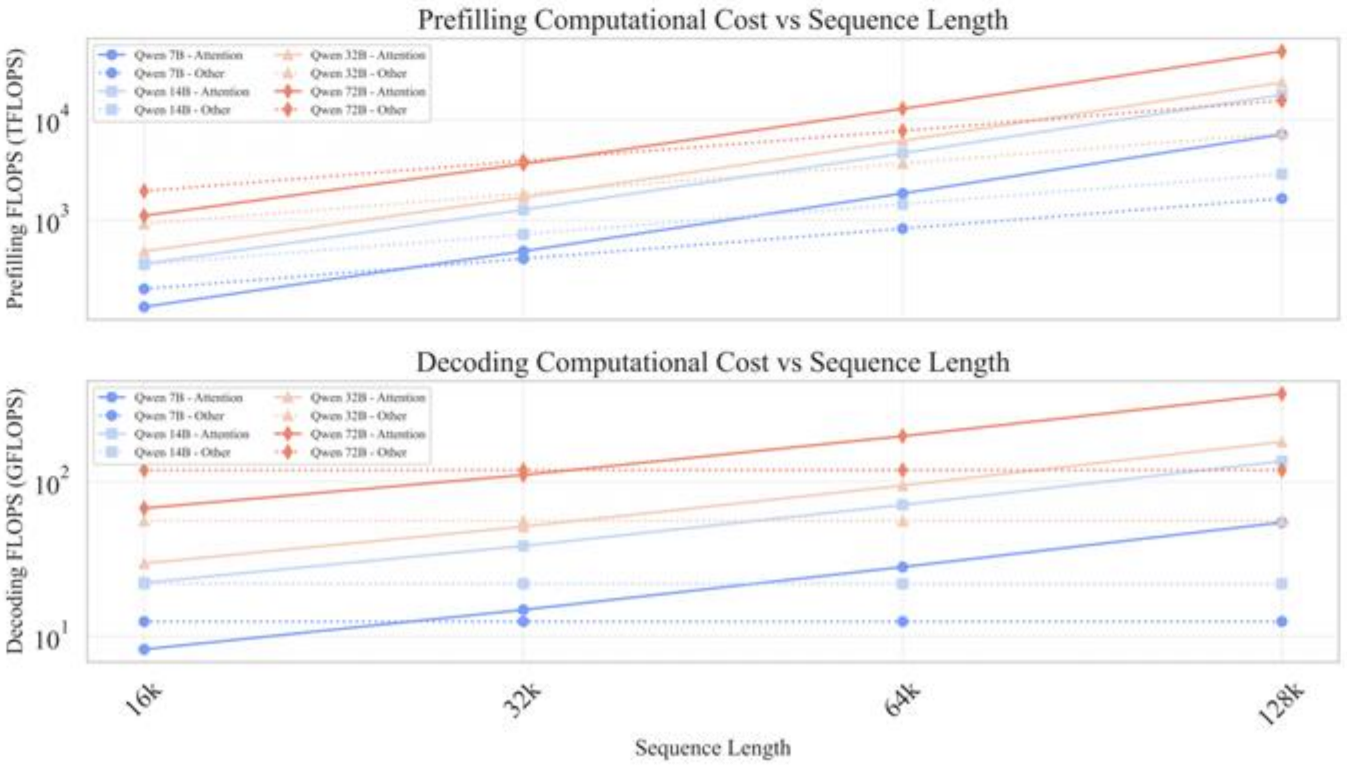
KV-Cache

Avoids recomputing the entire attention-matrix by storing prior vectors

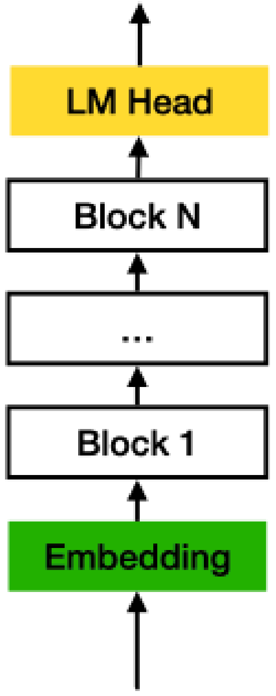
Prefill vs. auto-regressive generation

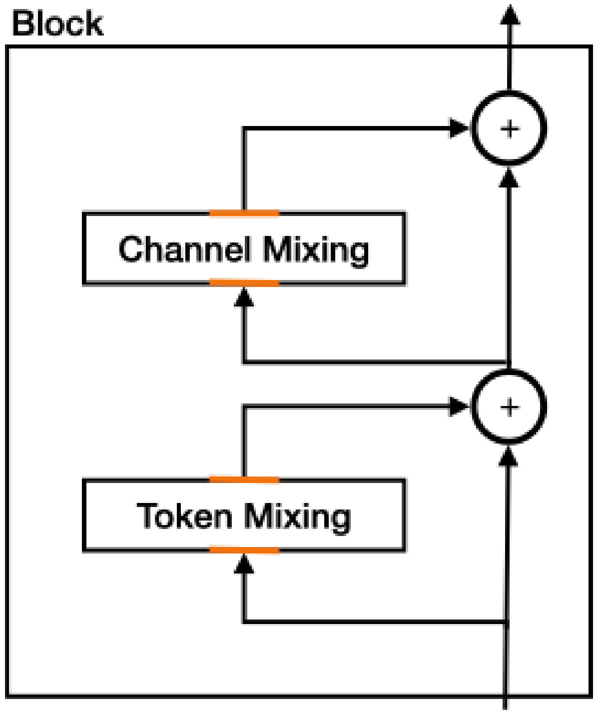
- Pre-filling: The model ingests the full prompt once to establish its internal context.
- Auto-regression/decoding: It then predicts new tokens sequentially from the context until the output is complete.

    - As sequence length increases, attention dominates.

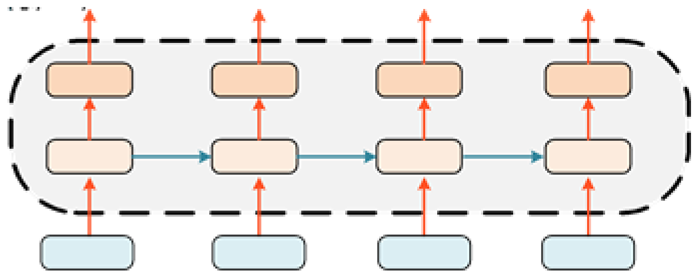


General Architecture of LLMs



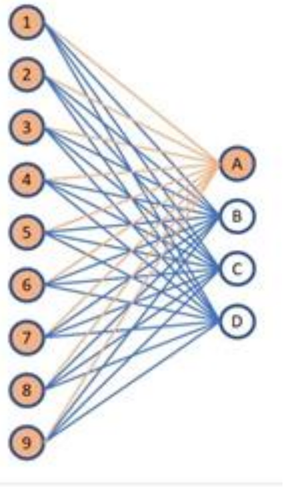


Token-Mixing

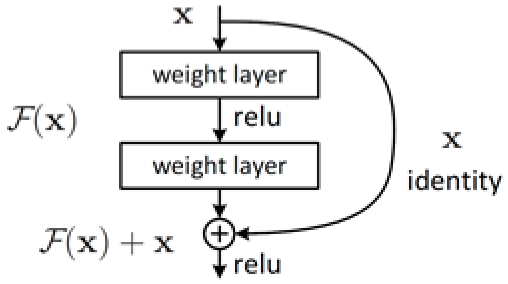


Self-Attention / Recurrent

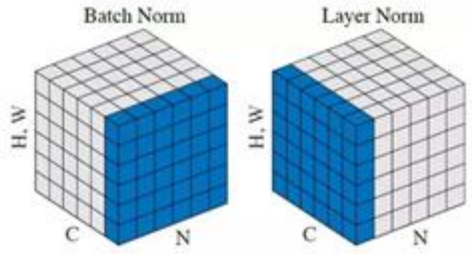
Channel-Mixing



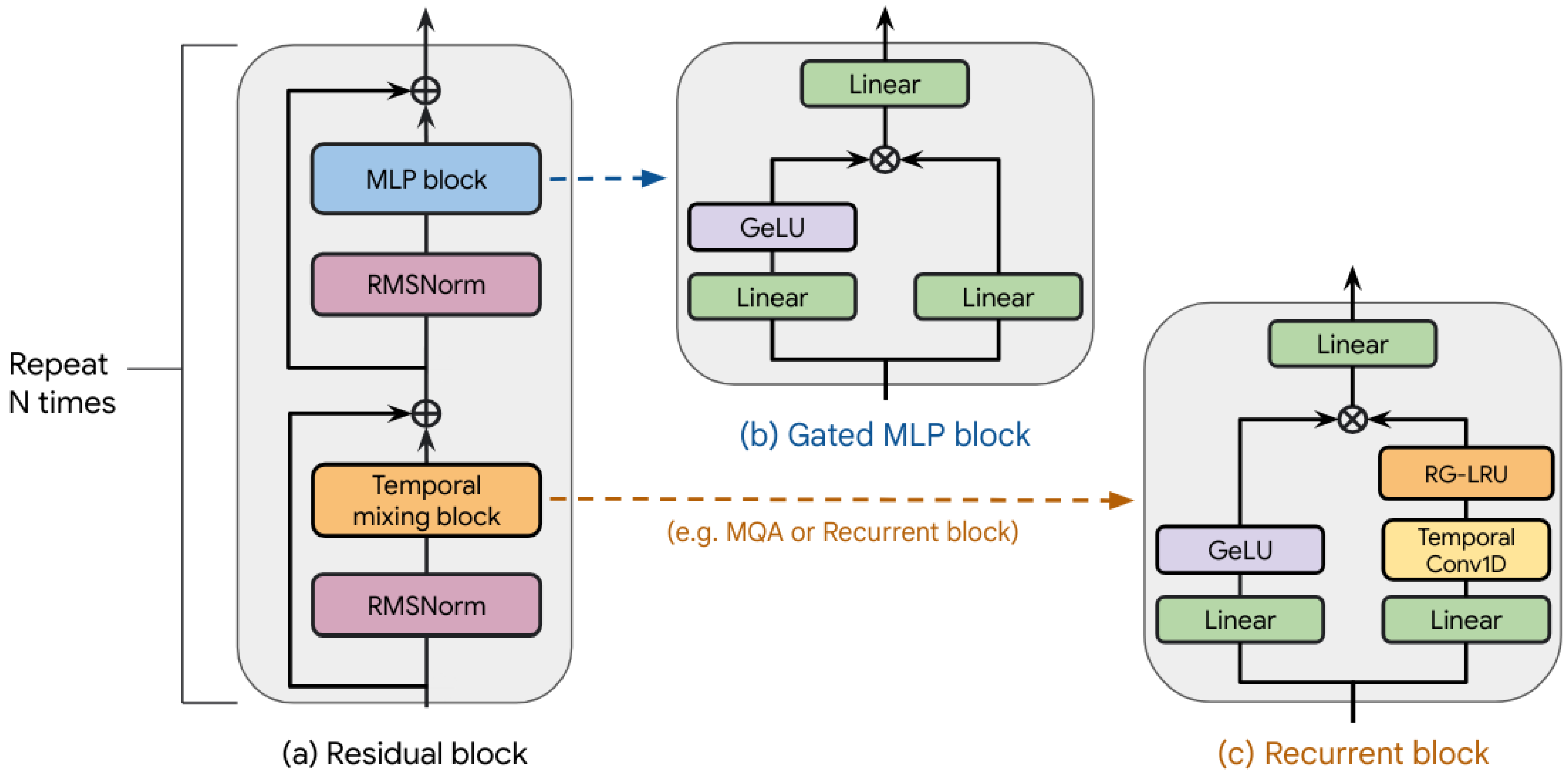
Residual Stream



(Layer) Normalization



General Architecture of LLMs



Overview

Part 1: Modern Language Models

- LLMs 101
- **Make Transformers Efficient**
  - Keeping self-attention
  - Supplementing self-attention
  - Modifying/replacing self-attention

Part 2: Next-Generation Language Models

- State-Space Models
- Neuromorphic Hardware
- MatMul-free LM on Loihi
- What’s next?

Overview

Part 1: Modern Language Models

- LLMs 101
- **Make Transformers Efficient**
  - **Keeping self-attention**
  - Supplementing self-attention
  - Modifying/replacing self-attention

Part 2: Next-Generation Language Models

- State-Space Models
- Neuromorphic Hardware
- MatMul-free LM on Loihi
- What’s next?

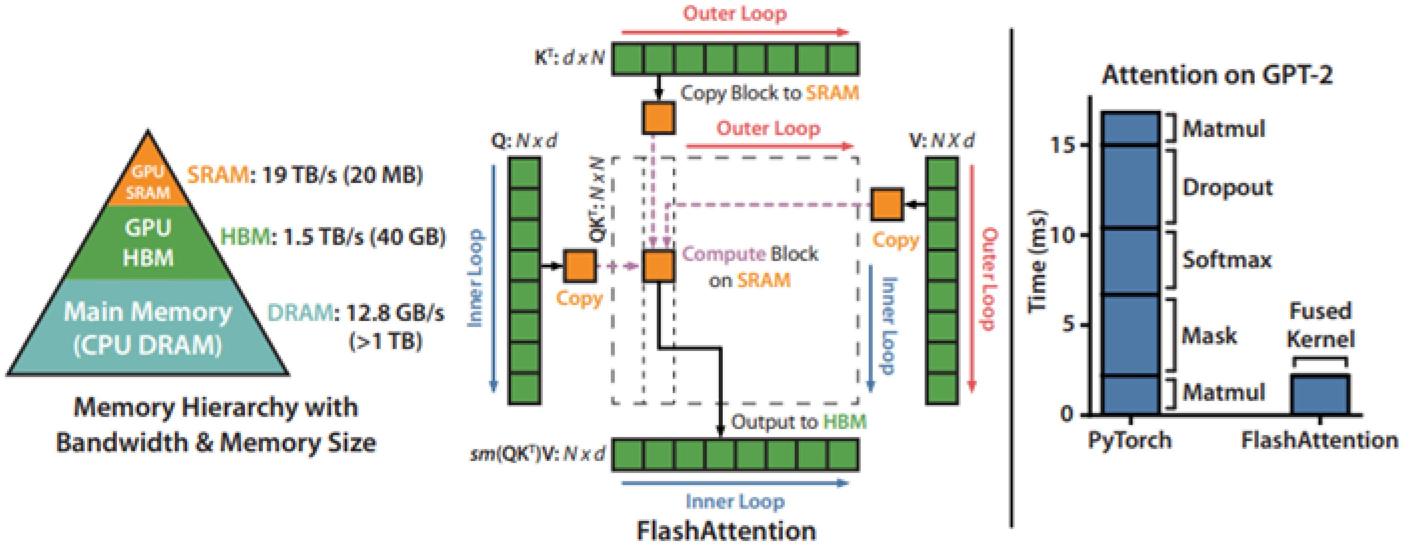
Flash Attention

Pros:

- Preserves safe and online softmax
- Block-level parallelism

Cons:

- Nothing.



Fusing kernels = minimizing memory access

Overview

Part 1: Modern Language Models

- LLMs 101
- **Make Transformers Efficient**
  - Keeping self-attention
  - **Supplementing self-attention**
  - Modifying/replacing self-attention

Part 2: Next-Generation Language Models

- State-Space Models
- Neuromorphic Hardware
- MatMul-free LM on Loihi
- What's next?

Techniques that supplement self-attention

Attention Optimizations

- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

Dense Optimizations

- Mixture of Experts

Higher-Level Optimizations

- Knowledge Distillation
- Speculative Decoding
- Low-precision
- Sparsity

Techniques that supplement self-attention

Attention Optimizations

- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

Dense Optimizations

- Mixture of Experts

Higher-Level Optimizations

- Knowledge Distillation
- Speculative Decoding
- Low-precision
- Sparsity

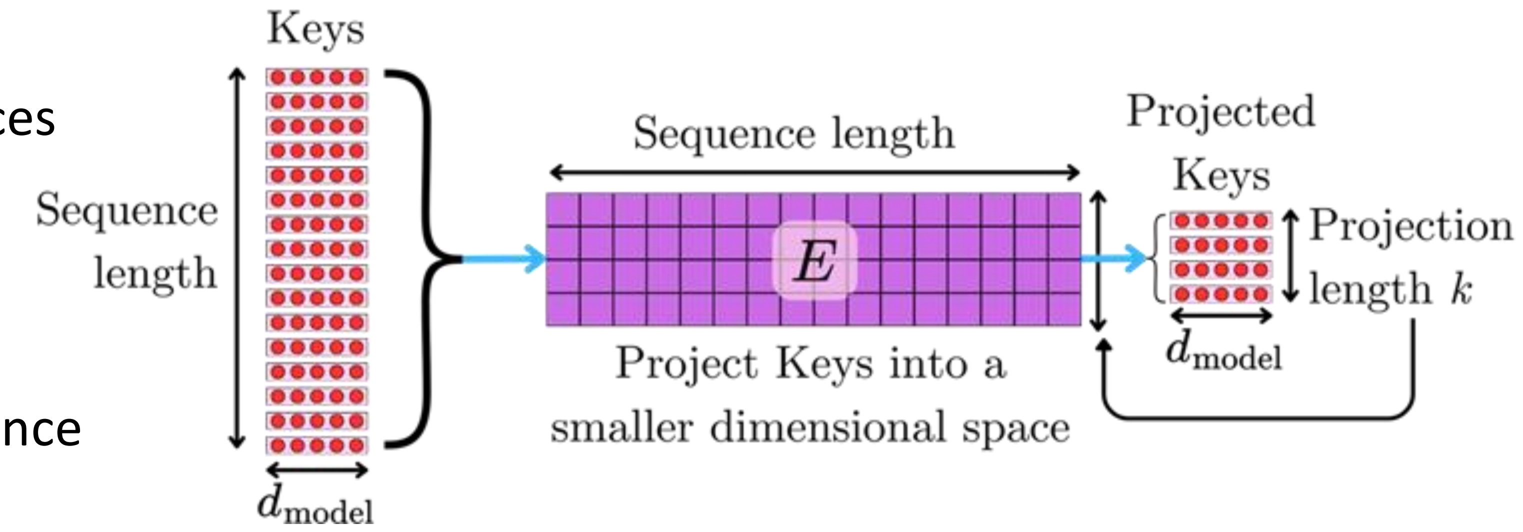
Low Rank Approximation

<u>Pros</u>

- KV Cache Compression
- Reduced FLOPs for longer sequences

<u>Cons</u>

- Training instability risk
- Poorer long-range arena performance



Techniques that supplement self-attention

Attention Optimizations

- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

Dense Optimizations

- Mixture of Experts

Higher-Level Optimizations

- Knowledge Distillation
- Speculative Decoding

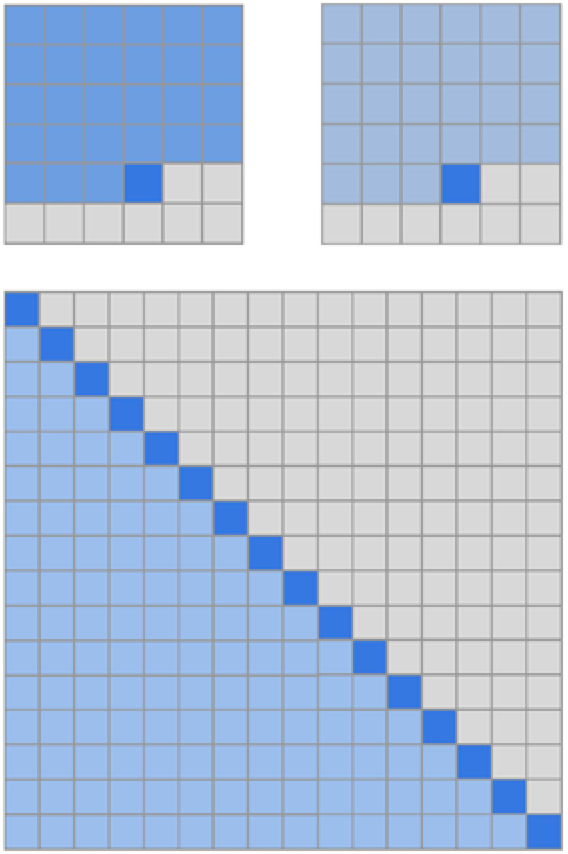
Sparse Attention

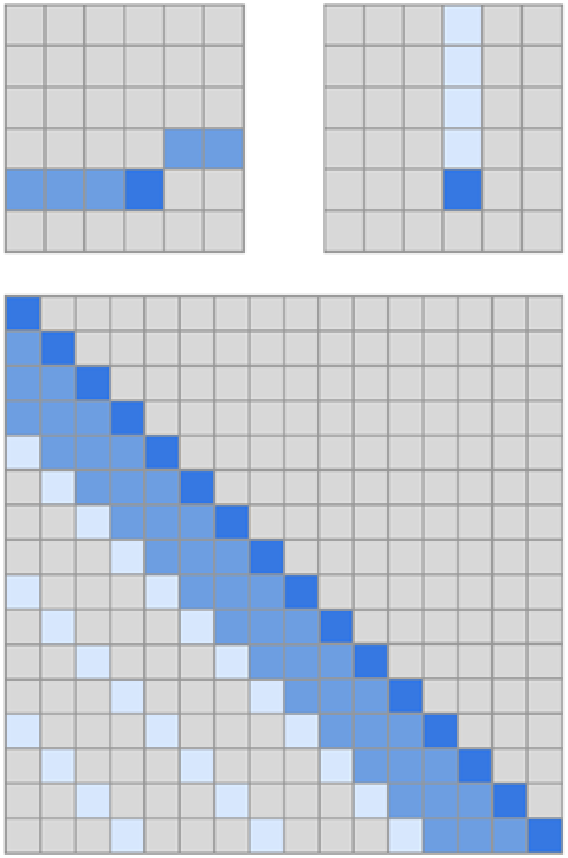
Pros

- Sparse factorizations of the attention matrix
- Huge speed up
- DeepSeek uses a form of structured sparsity during training time, as does Longformer, Big Bird, Reformer, Linformer, etc.

Cons

- Sparsity must be structured
- KV cache (typically) remains


(a) Transformer


(b) Sparse Transformer (strided)

Techniques that supplement self-attention

Attention Optimizations

- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

Dense Optimizations

- Mixture of Experts

Higher-Level Optimizations

- Knowledge Distillation
- Speculative Decoding

Multi/Group Query Attention

- Reduced KV Cache size
- Faster Attention Computation
- Lower Memory Bandwidth
- Llama-2, Mistral-7B

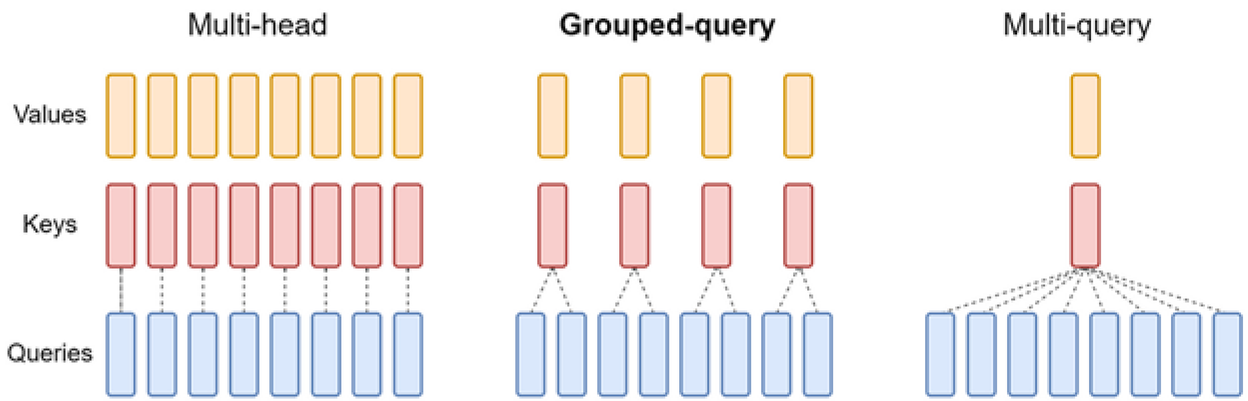


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

Techniques that supplement self-attention

Attention Optimizations

- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

Dense Optimizations

- <u>Mixture of Experts</u>

Higher-Level Optimizations

- Knowledge Distillation
- Speculative Decoding

Mixture of Experts

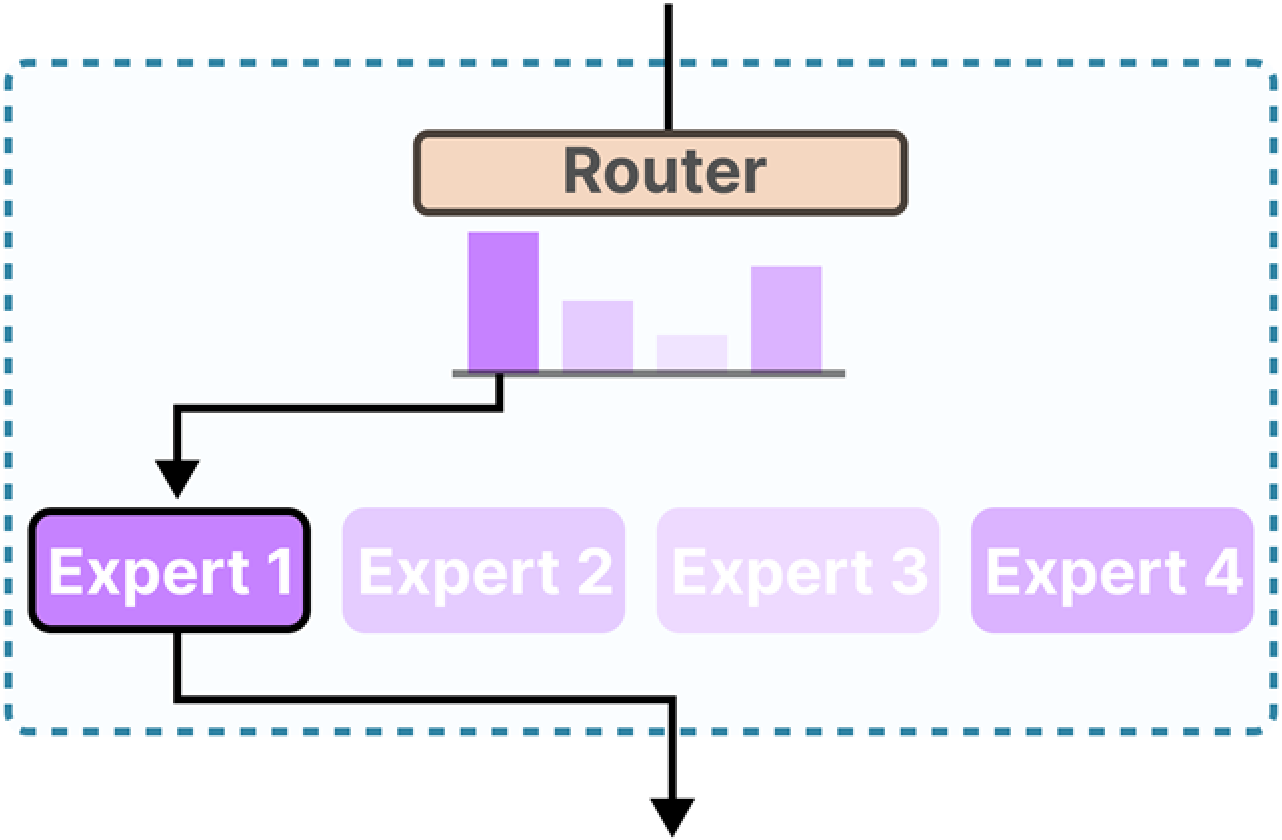
Pros

- Less FLOPs for larger models
- It's everywhere. (ChatGPT, DeepSeek, Mixtral)

Cons

- Non-uniform experts (need probabilistic sampling to fix this)

Note: this is just another form of structured sparsity. "Experts" is perhaps misleading - representations tend to still be distributed.



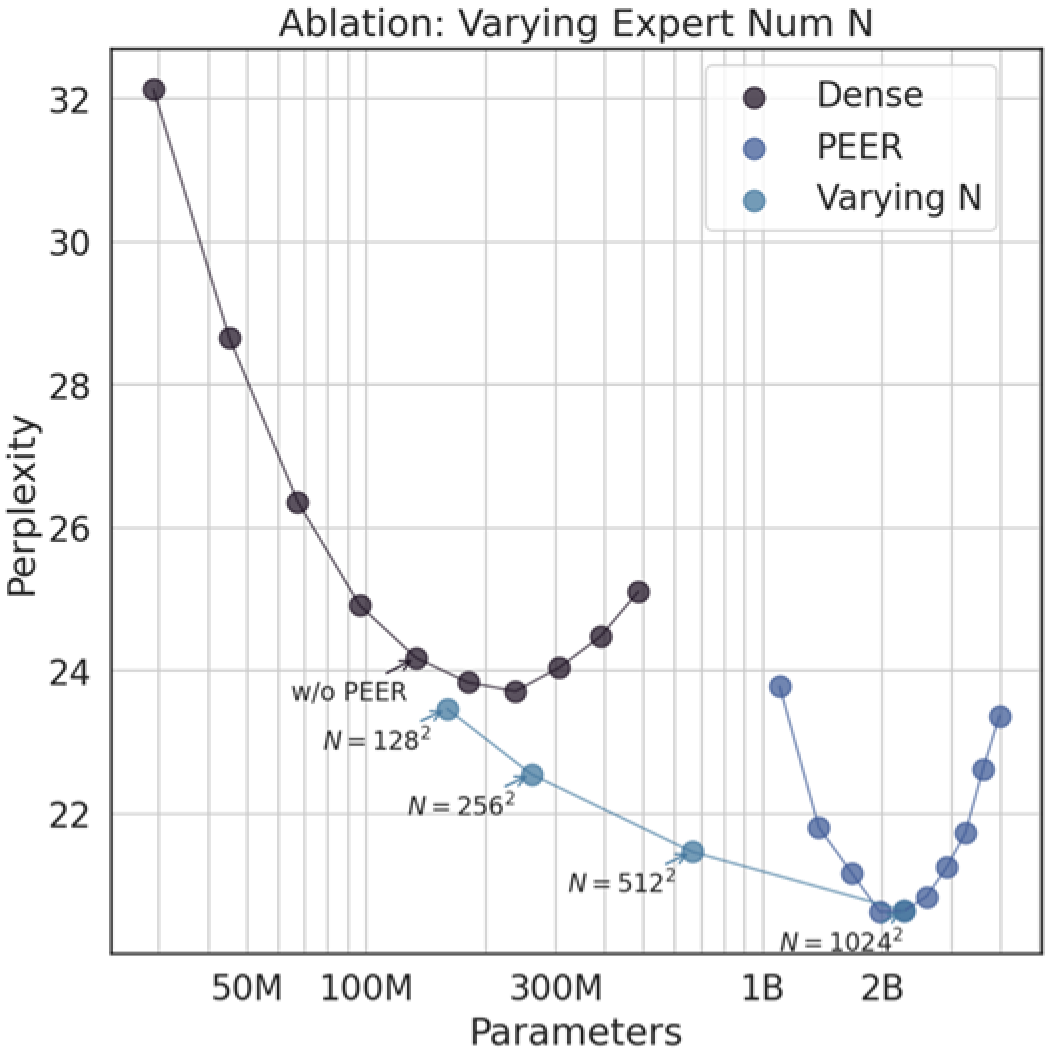
Mixture of Experts: fine-grained

Pros

- Less FLOPs for larger models
- Performance scales better with finer granularity -> more, smaller experts

Cons

- Inefficient on GPUs/TPUs (fine-grained random memory access)



He, 2024, Mixture of a Million Experts

Techniques that supplement self-attention

Attention Optimizations

- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

Dense Optimizations

- Mixture of Experts

Higher-Level Optimizations

- Knowledge Distillation
- Speculative Decoding
- Low-Precision
- Sparsity

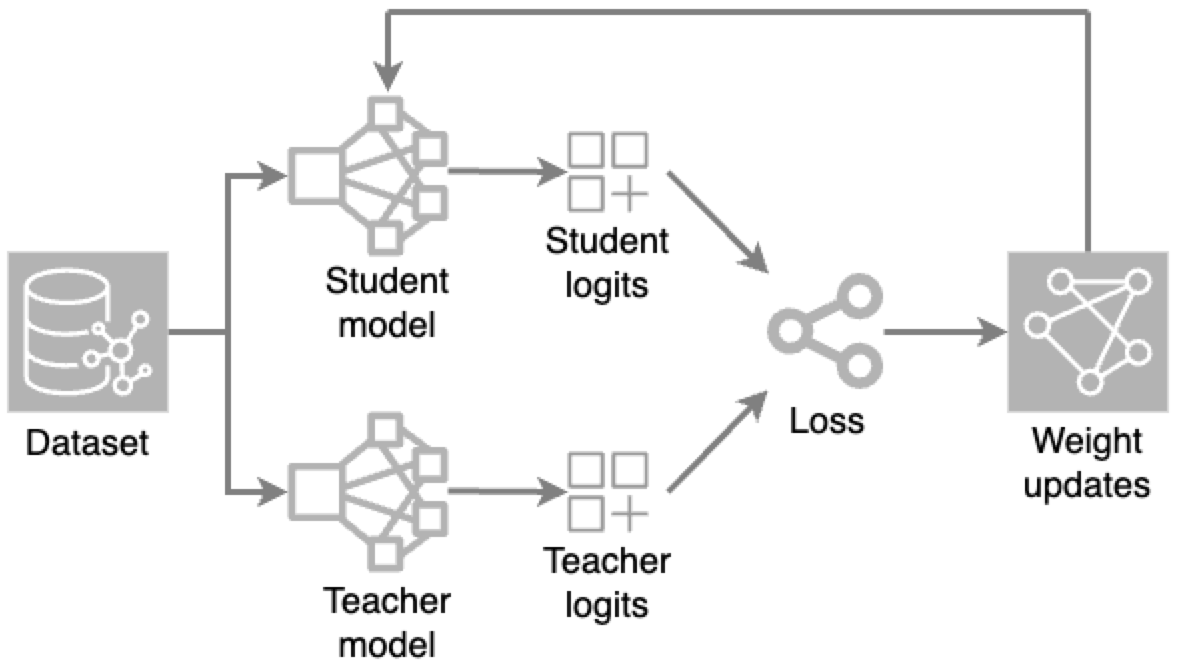
Knowledge Distillation

Pros

- Amongst the best ways to do model compression
- DeepSeek distilled ChatGPT knowledge!

Cons

- …Thus violating ChatGPTs terms of service
- But does ChatGPT have the right to "protect" a model trained on proprietary data that is not theirs?
- You also still need to have a large model trained.



Techniques that supplement self-attention

Attention Optimizations

- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

Dense Optimizations

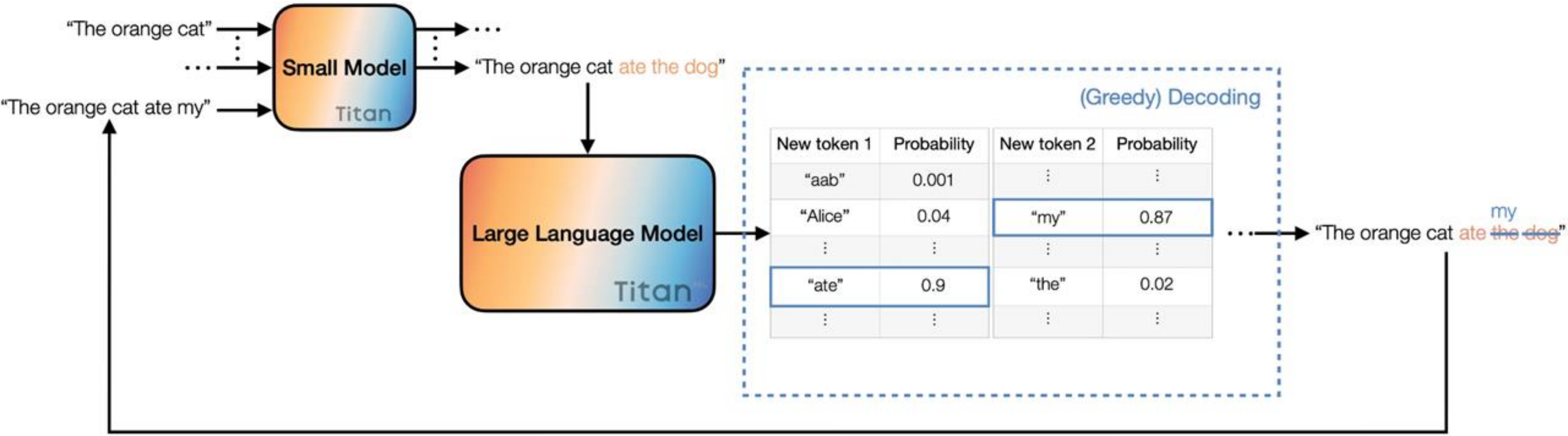
- Mixture of Experts

Higher-Level Optimizations

- Knowledge Distillation
- Speculative Decoding
- Low-Precision
- Sparsity

Speculative Decoding

- Increased throughput by multi-sampling the output
- Used widely in practice



Techniques that supplement self-attention

Attention Optimizations

- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

Dense Optimizations

- Mixture of Experts

Higher-Level Optimizations

- Knowledge Distillation
- Speculative Decoding
- Low-Precision
- Sparsity

Low-Precision Quantization

Pros

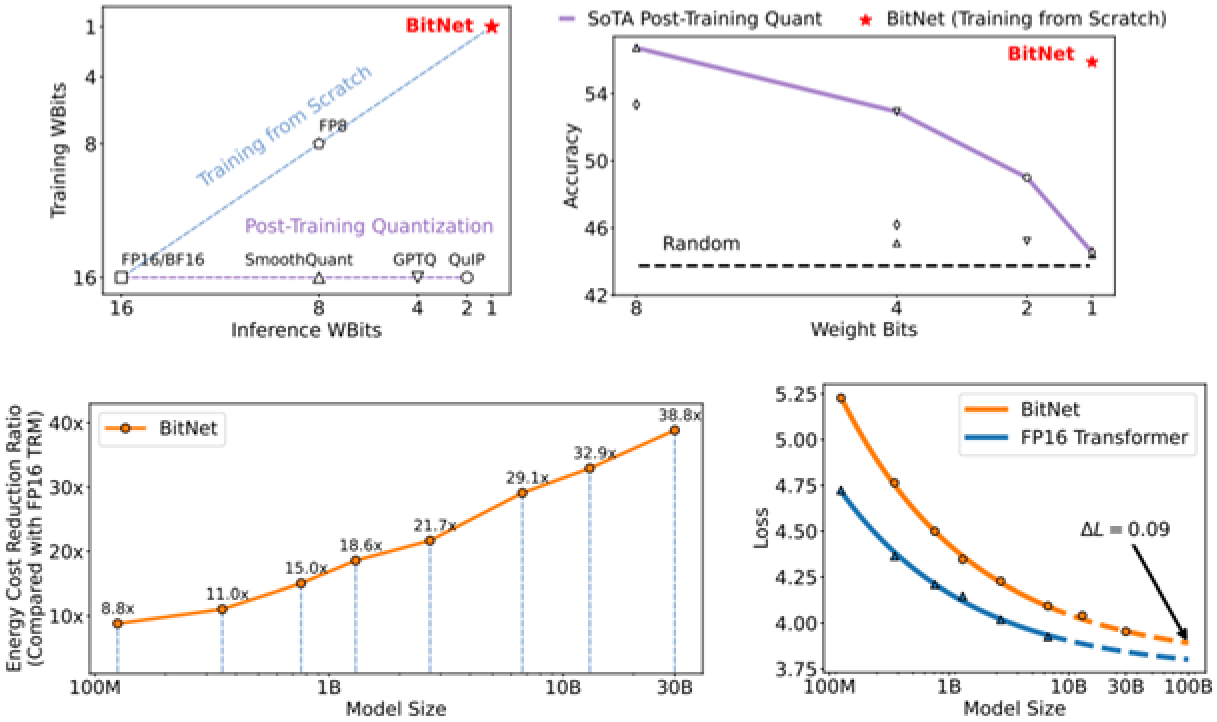
- Less memory

Cons

- Worse performance

But how much worse performance?

- Weights: very tolerant to low-precision
- Activations: less tolerant to low-precision
- Normalization ops: they hate low-precision



BitNet, Microsoft

Techniques that supplement self-attention

Attention Optimizations
- Low-Rank Approximations
- Sparse Attention Variants
- MQA/GQA

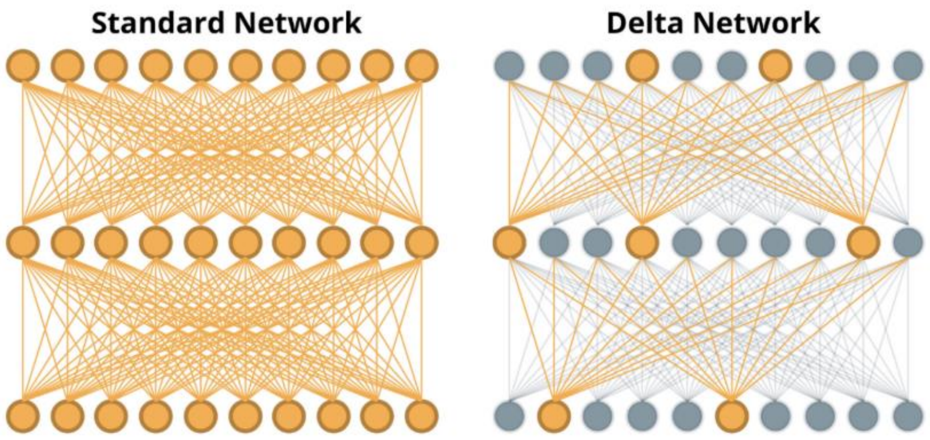
Dense Optimizations
- Mixture of Experts

Higher-Level Optimizations
- Knowledge Distillation
- Speculative Decoding
- Low-Precision
- Sparsity

Sparsity & pruning

Activation sparsity

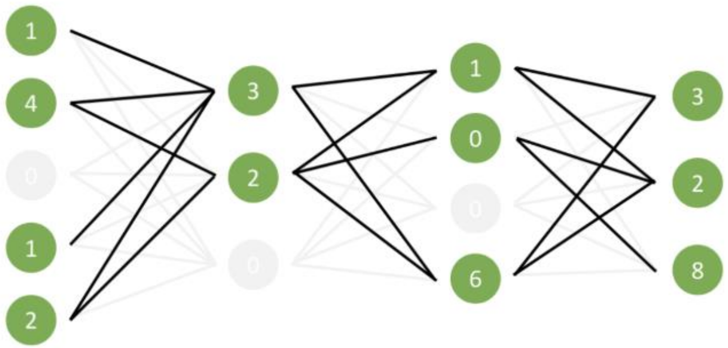
- "SNNs with graded spikes"
- Sigma-delta networks (O'Connor & Welling, 2017)



DeltaRNNs (Gao et al, 2018)

Weight sparsity

- Synaptic pruning, standard(*ish*) in DL
- During training or post-training



Hoefler et al, 2021

Overview

Part 1: Modern Language Models

- LLMs 101
- **Make Transformers Efficient**
  - Keeping self-attention
  - Supplementing self-attention
  - **Modifying/replacing self-attention**

Part 2: Next-Generation Language Models

- State-Space Models
- Neuromorphic Hardware
- MatMul-free LM on Loihi
- What’s next?

Techniques that modify self-attention

- Recurrent Neural Networks
- Linear Attention

Techniques that modify self-attention

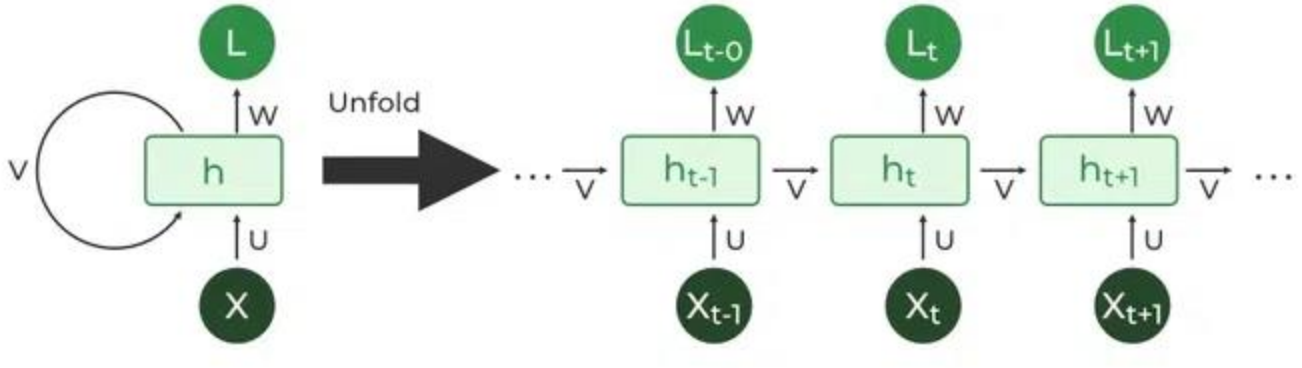
- Recurrent Neural Networks

Pros

- Constant memory usage: CHEAP

Cons

- Poor performance/long-range memory
- Poor parallelism
- Poor everything



Techniques that modify self-attention

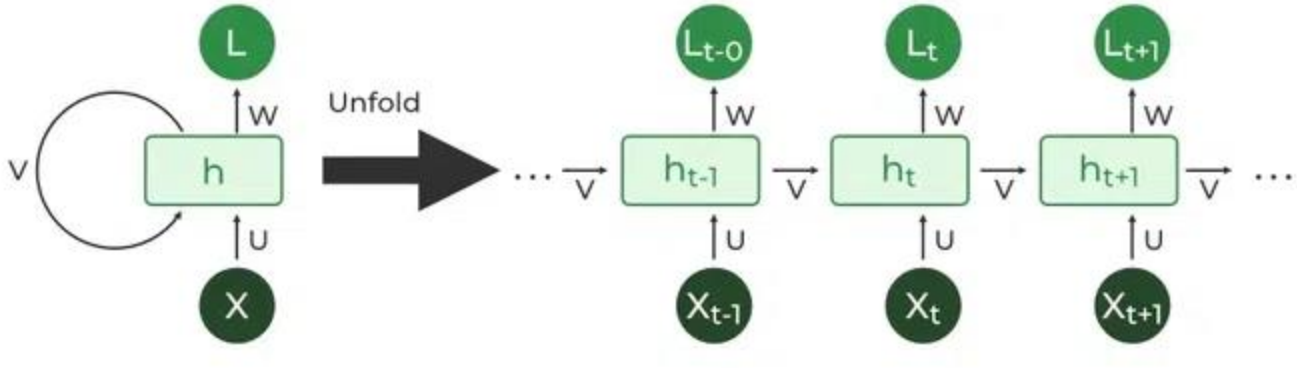
- Recurrent Neural Networks

Pros

- Constant memory usage: CHEAP

Cons

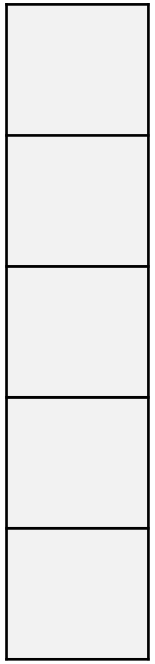
- Poor performance/long-range memory
- Poor parallelism
- Poor everything



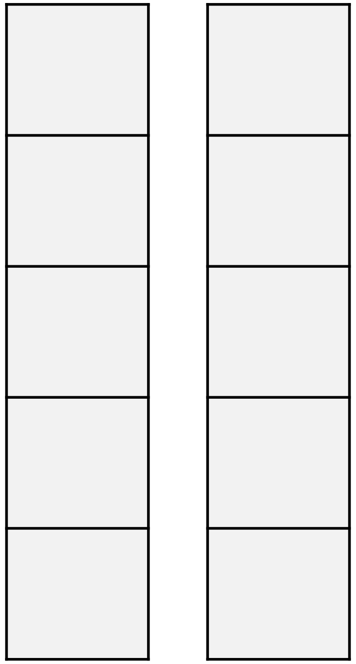
Part 2 will show how to flip the cons.

Shrek saw Taylor with binoculars

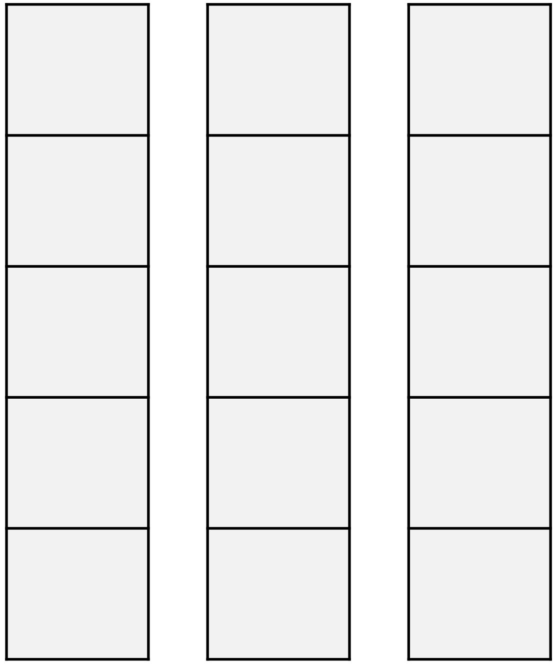
  



Shrek saw Taylor with binoculars

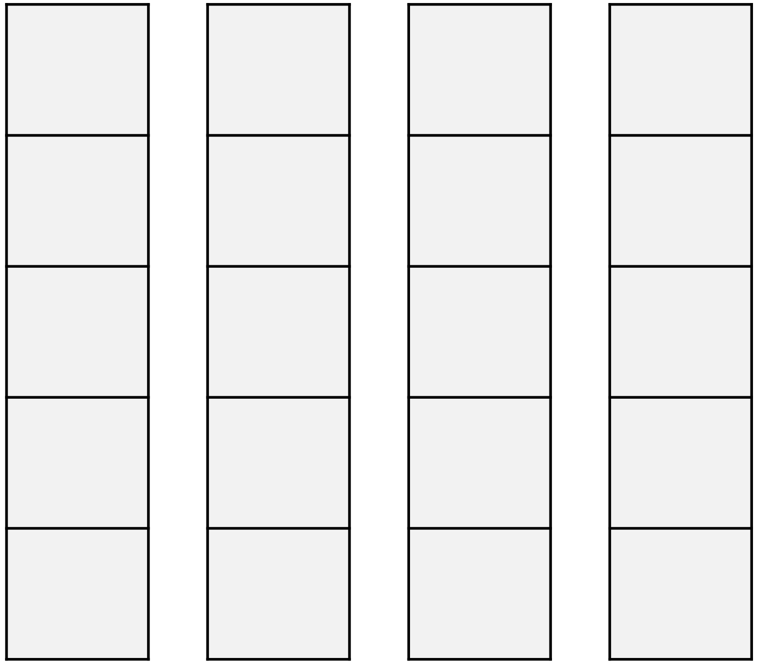


Shrek saw Taylor with binoculars

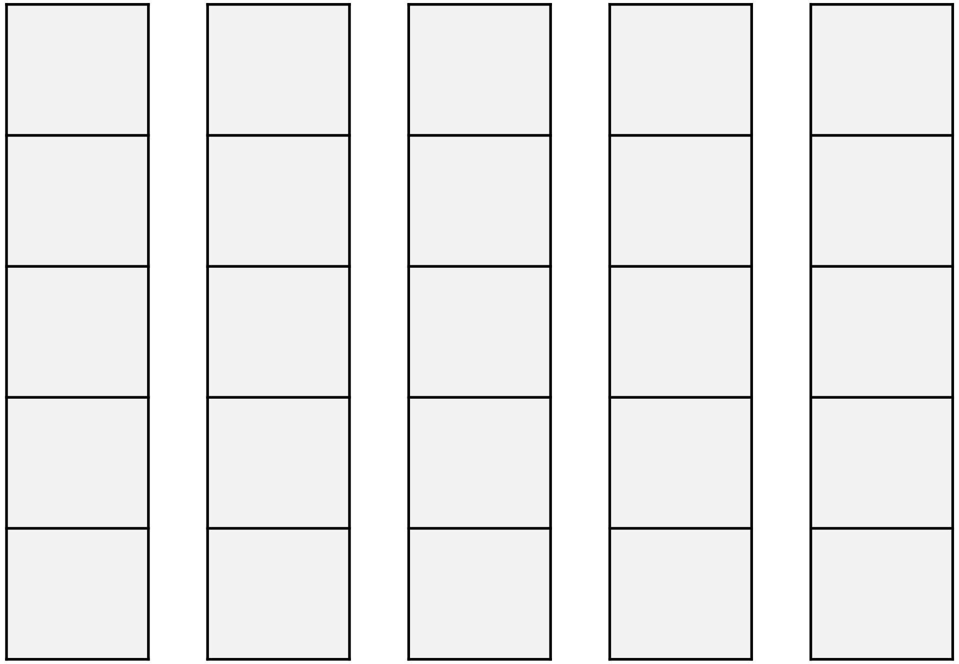


Shrek saw Taylor with binoculars





Shrek saw Taylor with binoculars



Shrek saw Taylor with binoculars

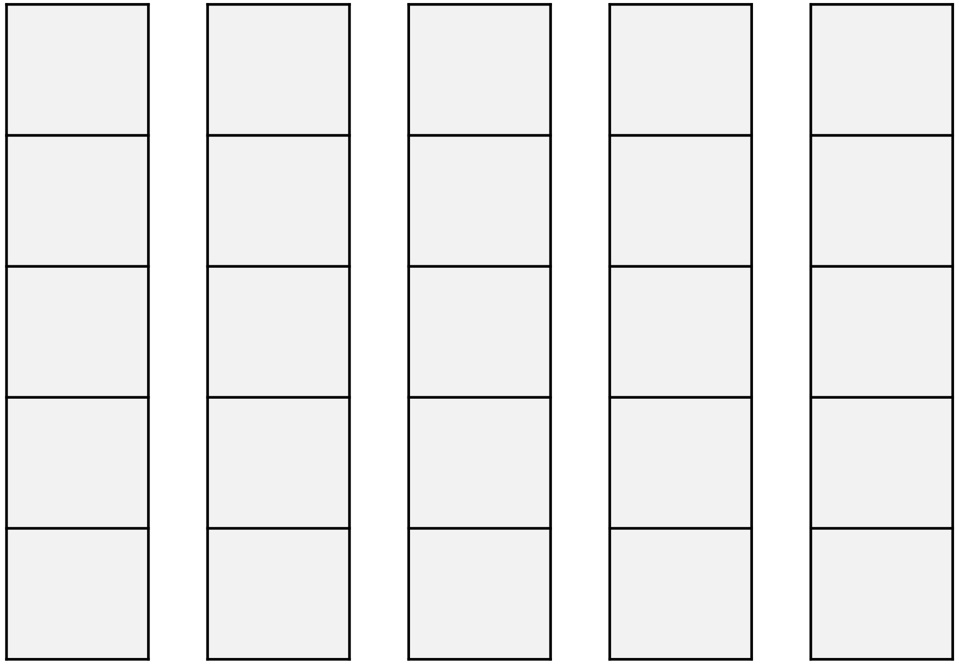
Shrek

saw

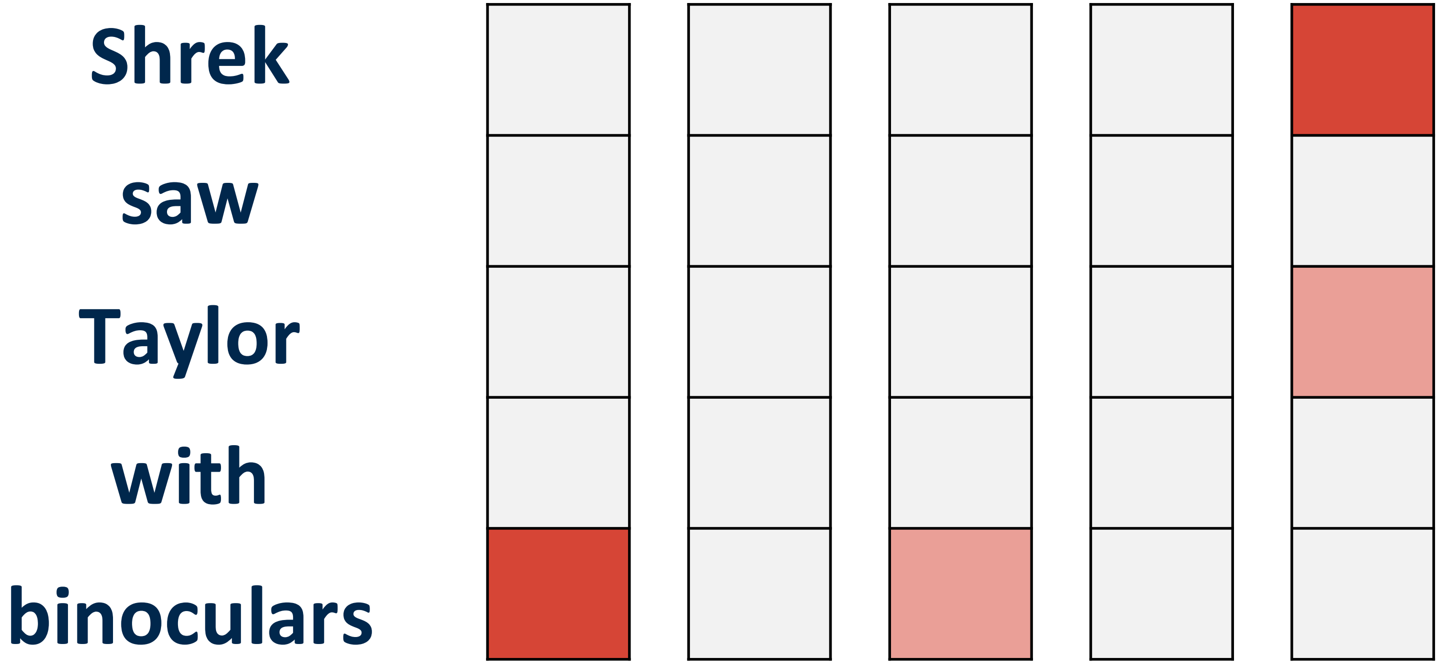
Taylor

with

binoculars

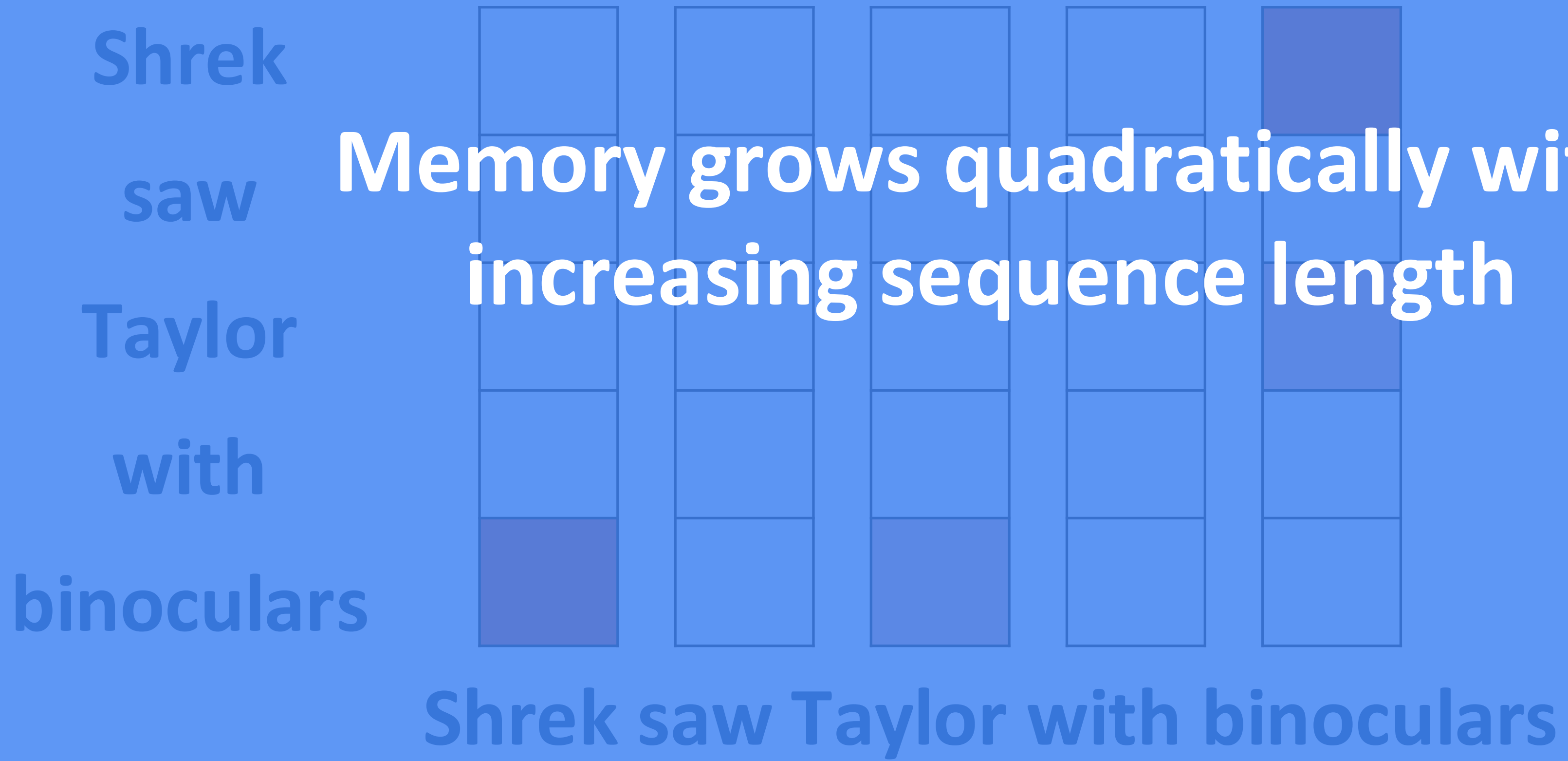


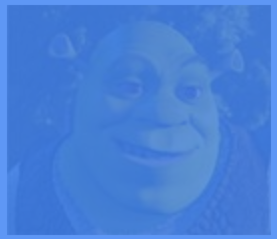
Shrek saw Taylor with binoculars

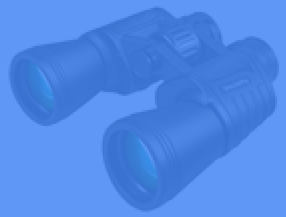


Shrek saw Taylor with binoculars

Memory grows quadratically with increasing sequence length







Memory grows quadratically with increasing sequence length

Your brains are not undergoing neurogenesis with every word I say

Shrek

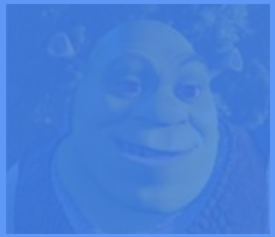
saw

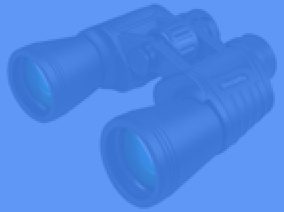
Taylor

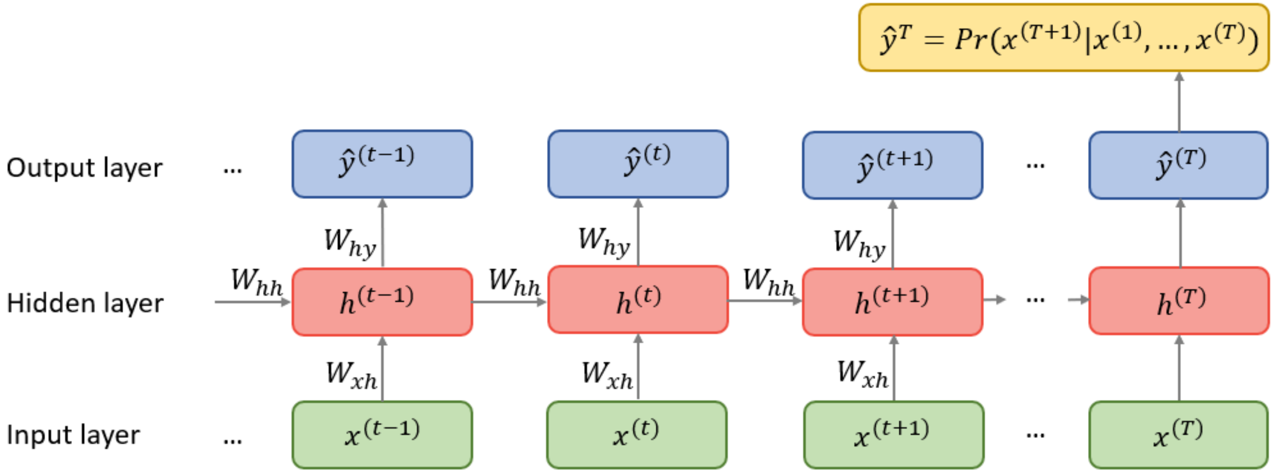
with

binoculars

Shrek saw Taylor with binoculars





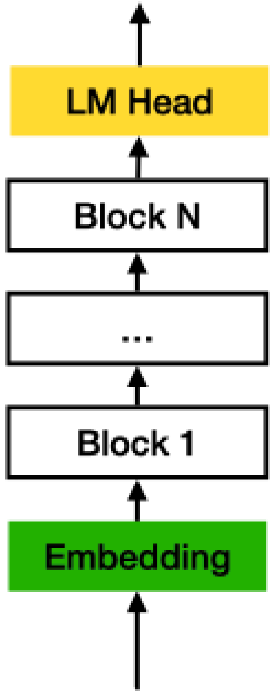


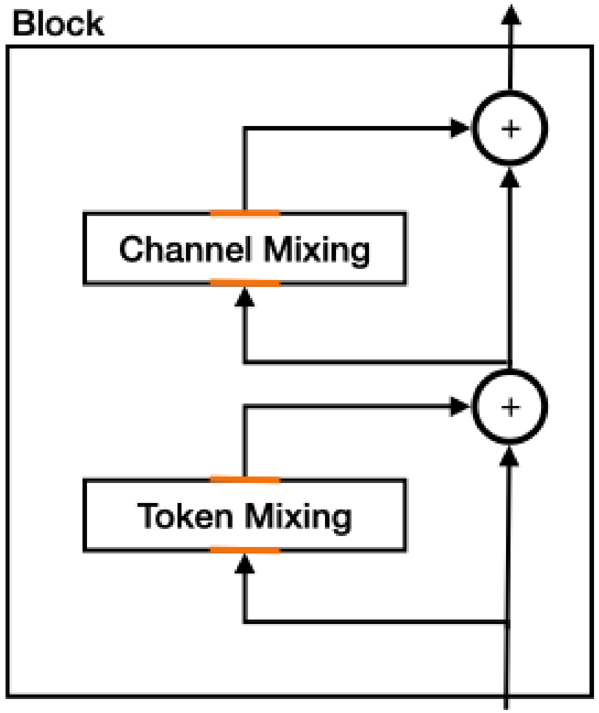
Shrek saw Taylor with binoculars

Attention: too much information

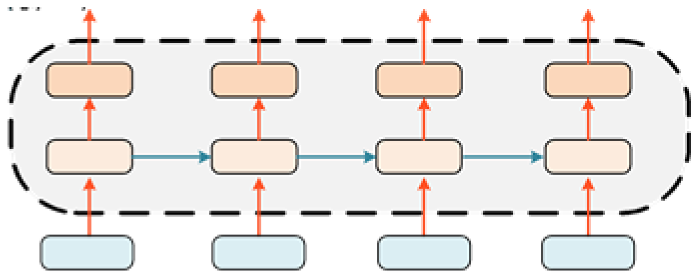
RNNs: not enough information

General Architecture of LLMs



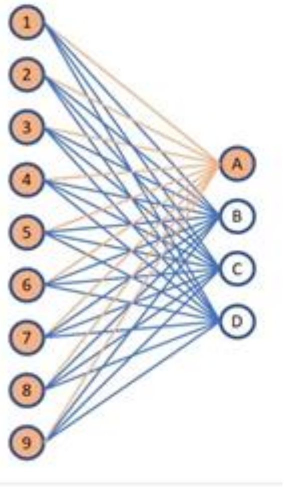


Token-Mixing

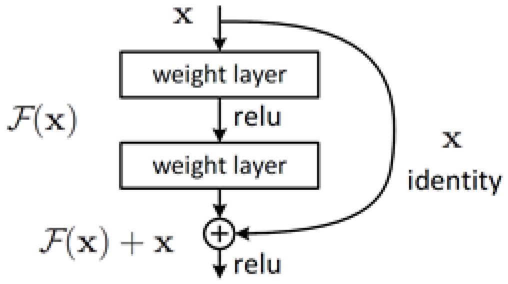


Self-Attention / Recurrent

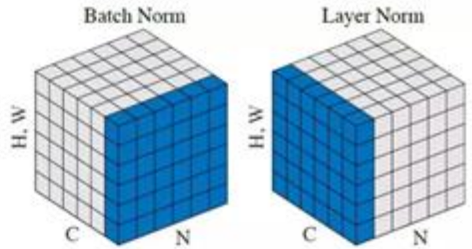
Channel-Mixing



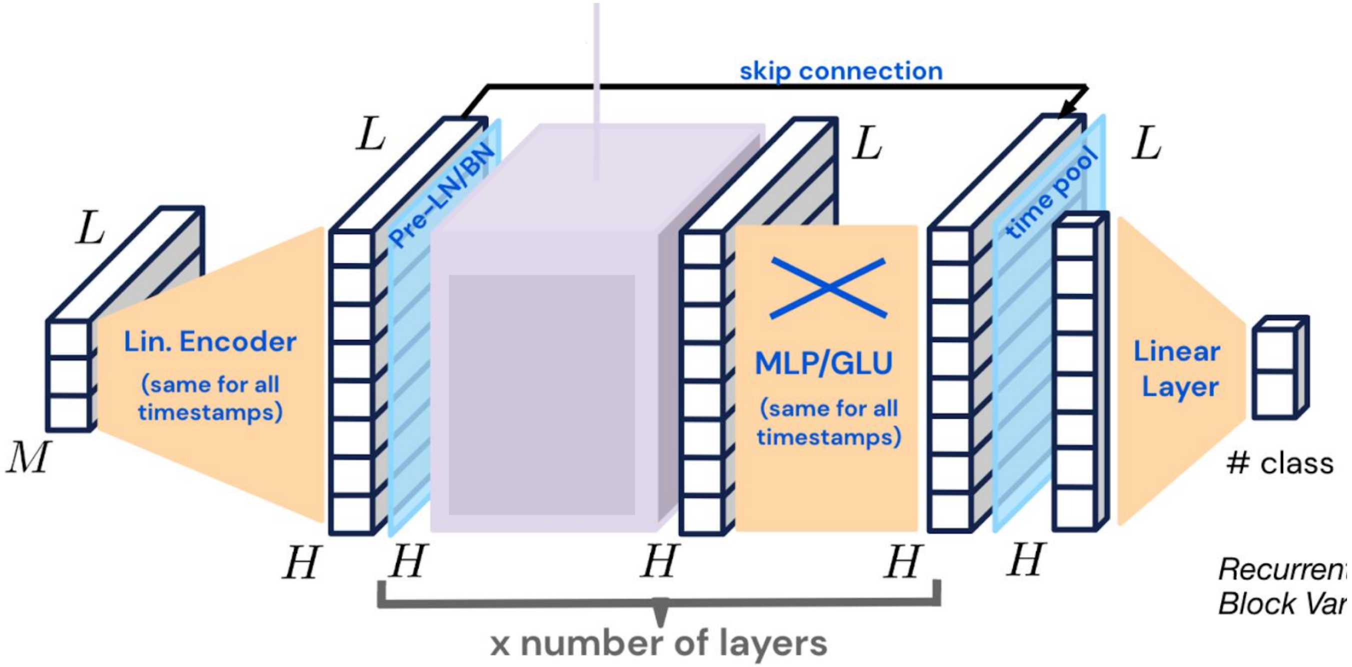
Residual Stream



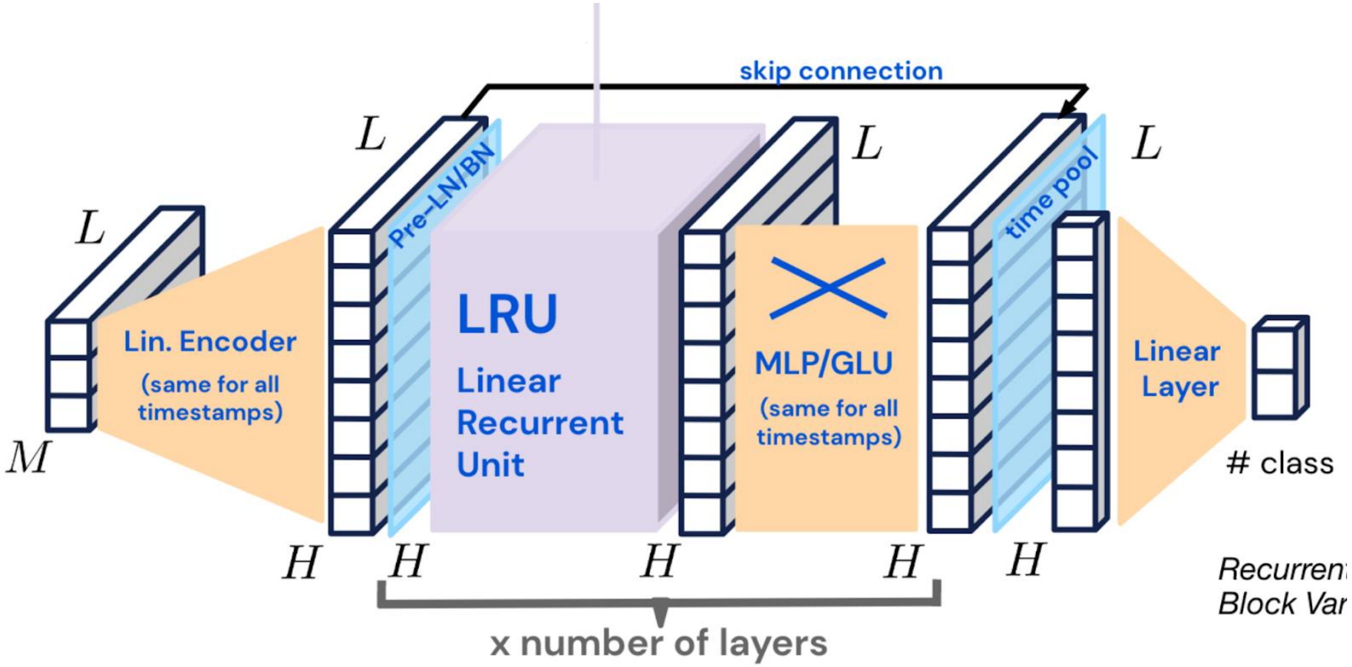
(Layer) Normalization



RNNs as an alternative to self-attention

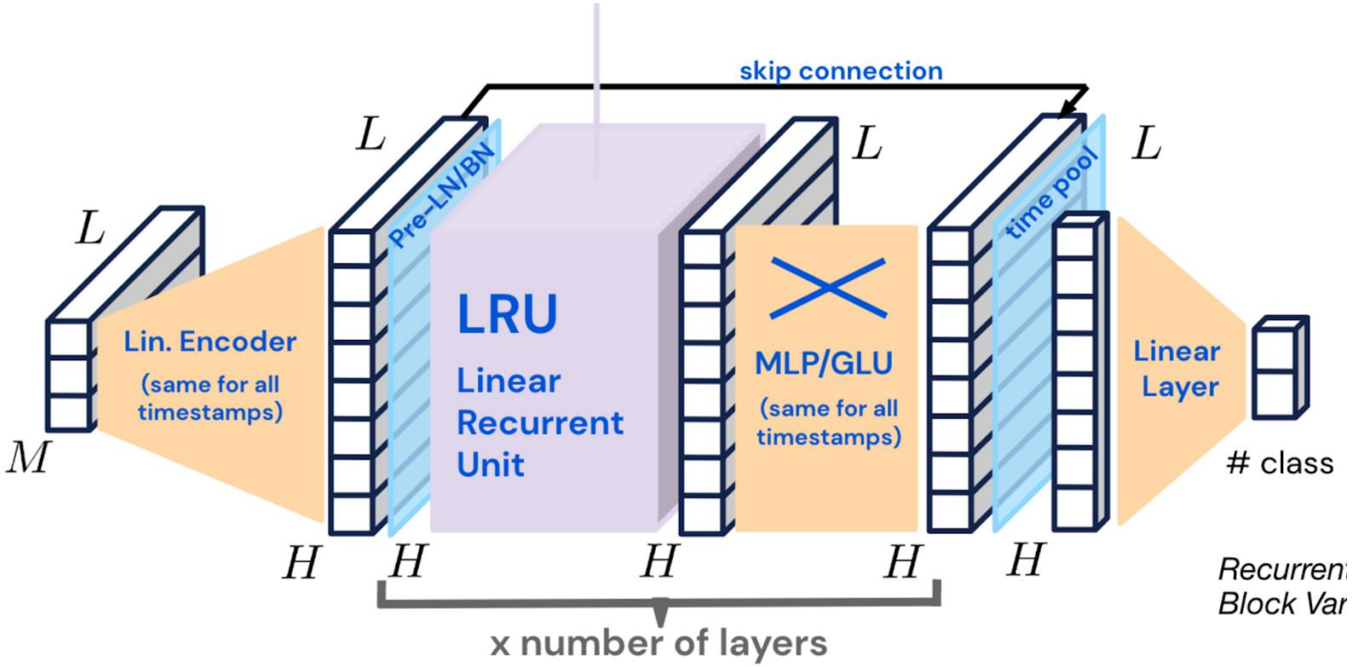


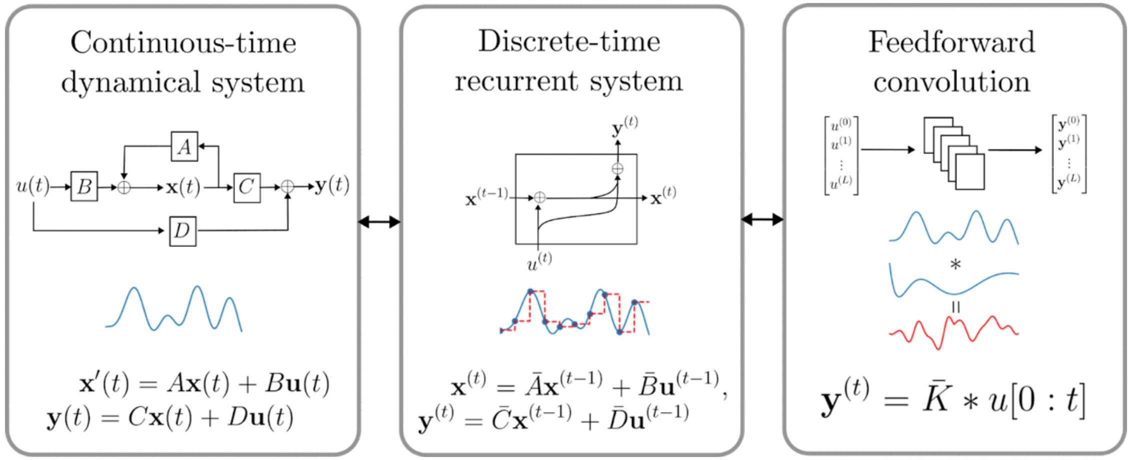
RNNs as an alternative to self-attention



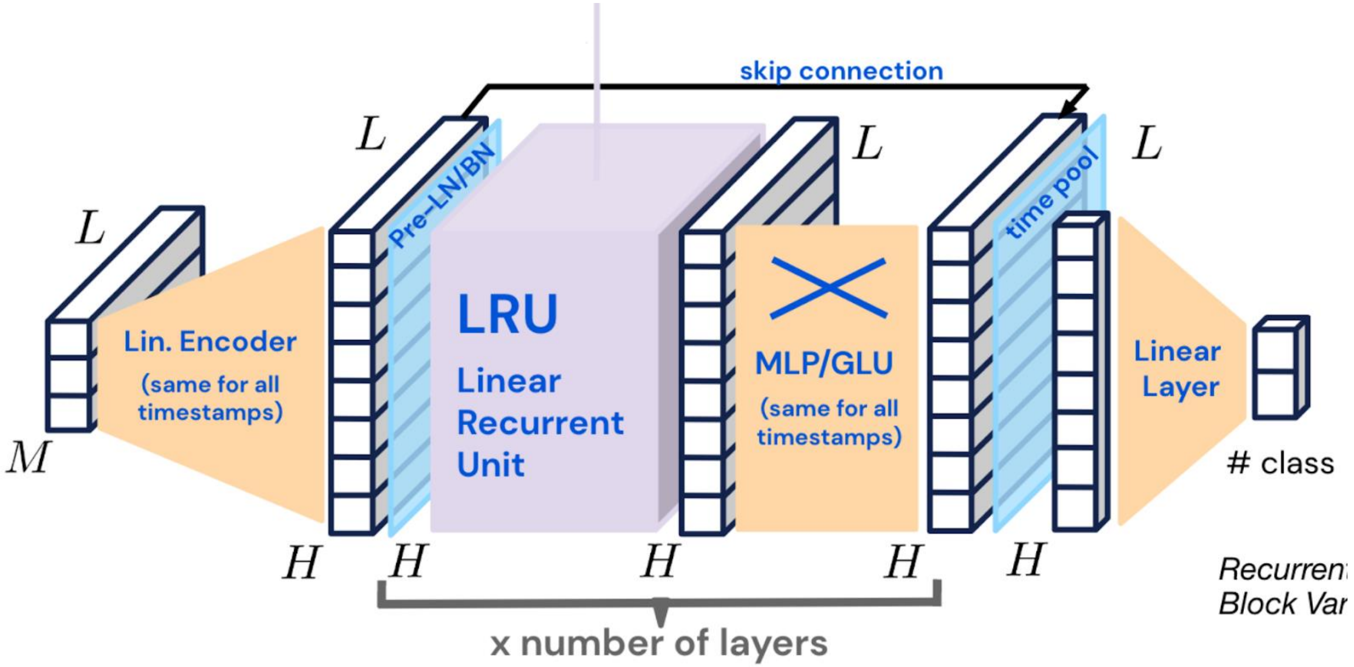
Recurren
Block Var

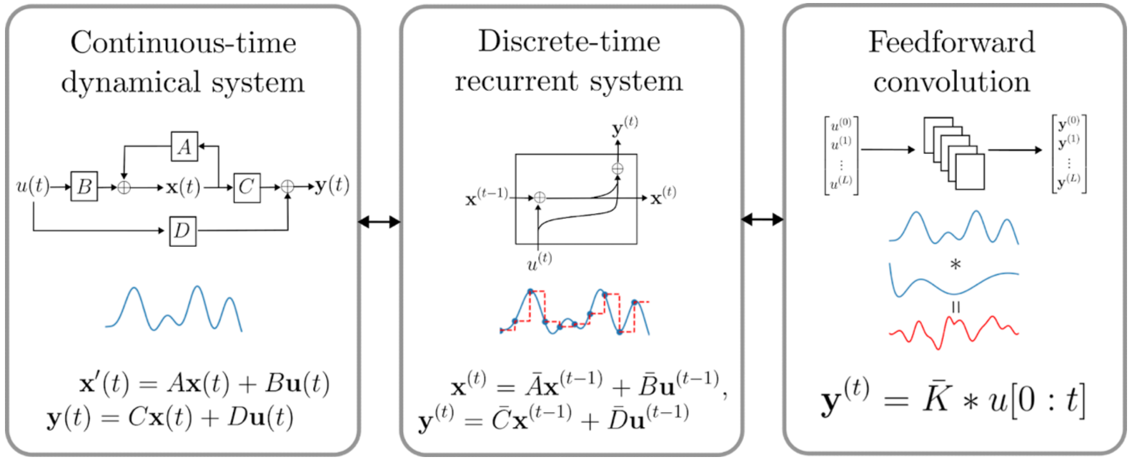
RNNs as an alternative to self-attention





RNNs as an alternative to self-attention





$$h_t = Ah_{t-1} + Bx_t$$

$$y_t = C^\top h_t$$

Eliminating Vector-Matrix Mult Through Time

Vanilla RNN

$$\begin{bmatrix} v^1 \\ v^2 \\ v^3 \end{bmatrix}_{1} = \begin{bmatrix} w_1^1 & w_1^2 & w_1^3 \\ w_2^1 & w_2^2 & w_2^3 \\ w_3^1 & w_3^2 & w_3^3 \end{bmatrix} \begin{bmatrix} v^1 \\ v^2 \\ v^3 \end{bmatrix}_{0} + \begin{bmatrix} x^1 \\ x^2 \\ x^3 \end{bmatrix}_{0}$$

$$V_1 = \boldsymbol{W} V_0 + X_0$$

$$\begin{aligned} V_2 &= \boldsymbol{W} V_1 + X_1 \\ &= \boldsymbol{W}(\boldsymbol{W} V_0 + X_0) + X_1 \\ &= \underbrace{\boldsymbol{W}^2}_{O(n^3)} V_0 + \boldsymbol{W} X_0 + X_1 \end{aligned}$$

Eliminating Vector-Matrix Mult Through Time

Element-wise Linear RNN

$$\begin{bmatrix} v^1 \\ v^2 \\ v^3 \end{bmatrix}_1 = \begin{bmatrix} w^1 \\ w^2 \\ w^3 \end{bmatrix} \odot \begin{bmatrix} v^1 \\ v^2 \\ v^3 \end{bmatrix}_0 + \begin{bmatrix} x^1 \\ x^2 \\ x^3 \end{bmatrix}_0$$

$$V_1 = W \odot V_0 + X_0$$

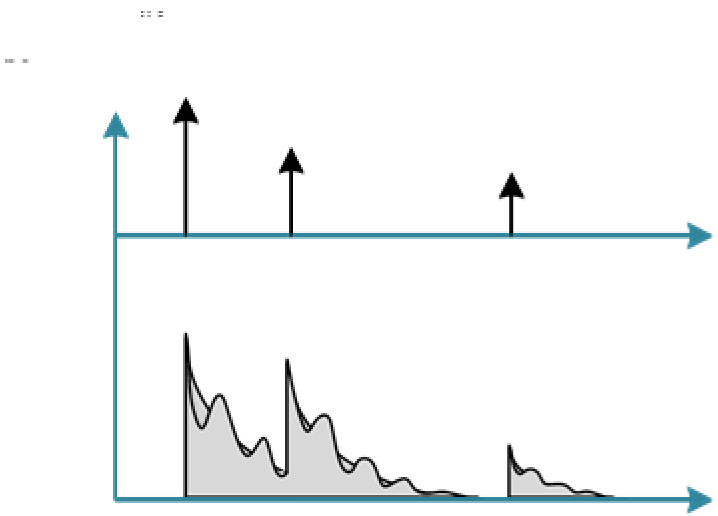
$$V_2 = W \odot V_1 + X_1$$

$$= W^2 \odot V_0 + X_0 W + X_1$$

$\mathrm{O}(n)$

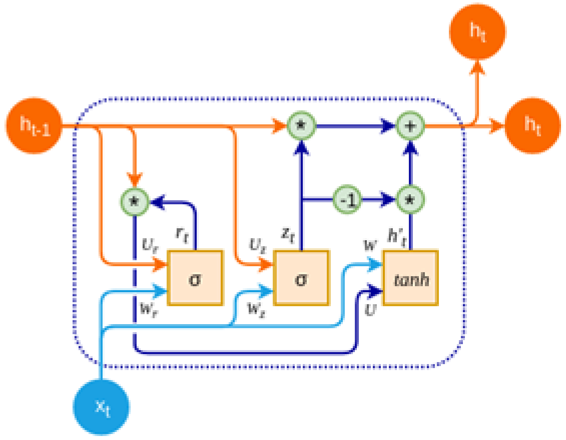
Complex eigenvalues:
Oscillatory decay

Real eigenvalues:
static decay



Addressing Long-Range Performance in RNNs

Gating Mechanisms

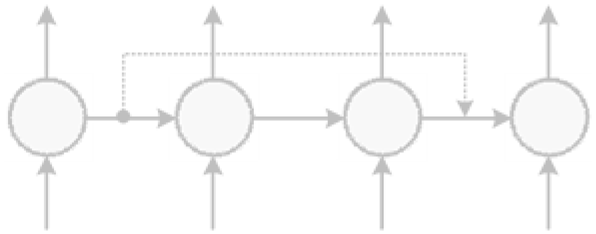


E.g., control and forget gates

Regularization

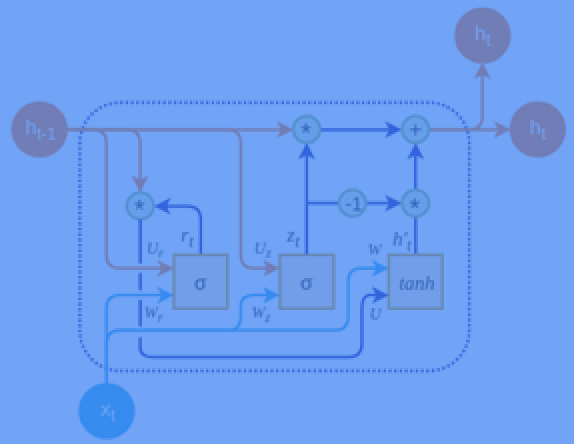
$$D = \begin{bmatrix} 0.99 & 0 \\ 0 & 1.01 \end{bmatrix}$$

Temporal Residual Connections



Addressing Long-Range Performance in RNNs

Gating Mechanisms

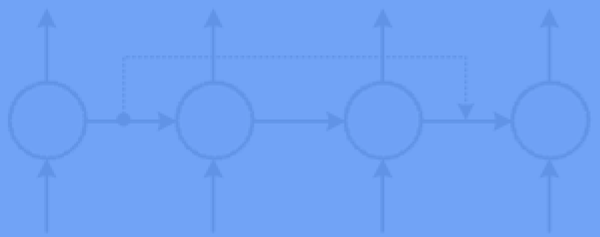


E.g., control and forget gates

Regularization

$$P = \begin{bmatrix} 0.99 & 0 \\ 0 & 1.01 \end{bmatrix}$$

Temporal Residual Connections



Attention: too much information

RNNs: the right information

An alternative derivation of attention -> RNN

We start from self-attention, and linearize the softmax

Further reading: "Transformers are RNNs" (2020), "Transformers are SSMs" (2024)

$$
\begin{aligned}
\boldsymbol{k}^{(i)}, \boldsymbol{v}^{(i)}, \boldsymbol{q}^{(i)} &= \boldsymbol{W}_k \boldsymbol{x}^{(i)}, \boldsymbol{W}_v \boldsymbol{x}^{(i)}, \boldsymbol{W}_q \boldsymbol{x}^{(i)} \\
\boldsymbol{K}^{(i)} &= \left[\boldsymbol{K}^{(i-1)}, \boldsymbol{k}^{(i)}\right] \in \mathbb{R}^{d_{\text{key}} \times i} \\
\boldsymbol{V}^{(i)} &= \left[\boldsymbol{V}^{(i-1)}, \boldsymbol{v}^{(i)}\right] \in \mathbb{R}^{d_{\text{value}} \times i} \\
\boldsymbol{y}^{(i)} &= \boldsymbol{V}^{(i)} \underline{\operatorname{softmax}((\boldsymbol{K}^{(i)})^\top \boldsymbol{q}^{(i)})}
\end{aligned}
$$

attention weights change for every new token
→ must store all previous KV

An alternative derivation of attention -> RNN

We start from self-attention, and linearize the softmax

Further reading: “Transformers are RNNs” (2020), “Transformers are SSMs” (2024)

$$
\begin{aligned}
\boldsymbol{k}^{(i)}, \boldsymbol{v}^{(i)}, \boldsymbol{q}^{(i)} &= \boldsymbol{W}_k \boldsymbol{x}^{(i)}, \boldsymbol{W}_v \boldsymbol{x}^{(i)}, \boldsymbol{W}_q \boldsymbol{x}^{(i)} \\
\boldsymbol{K}^{(i)} &= \left[\boldsymbol{K}^{(i-1)}, \boldsymbol{k}^{(i)}\right] \in \mathbb{R}^{d_{\text{key}} \times i} \\
\boldsymbol{V}^{(i)} &= \left[\boldsymbol{V}^{(i-1)}, \boldsymbol{v}^{(i)}\right] \in \mathbb{R}^{d_{\text{value}} \times i} \\
\boldsymbol{y}^{(i)} &= \boldsymbol{V}^{(i)} \underline{\operatorname{softmax}((\boldsymbol{K}^{(i)})^\top \boldsymbol{q}^{(i)})}
\end{aligned}
$$

attention weights change for every new token → must store all previous KV

write softmax as kernel

$$
\boldsymbol{y}^{(i)} = \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)} \kappa(\boldsymbol{k}^{(j)}, \boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i} \kappa(\boldsymbol{k}^{(j')}, \boldsymbol{q}^{(i)})}
$$

An alternative derivation of attention -> RNN

We start from self-attention, and linearize the softmax

Further reading: "Transformers are RNNs" (2020), "Transformers are SSMs" (2024)

$$\boldsymbol{k}^{(i)}, \boldsymbol{v}^{(i)}, \boldsymbol{q}^{(i)} = \boldsymbol{W}_k \boldsymbol{x}^{(i)}, \boldsymbol{W}_v \boldsymbol{x}^{(i)}, \boldsymbol{W}_q \boldsymbol{x}^{(i)}$$

$$\boldsymbol{K}^{(i)} = \left[\boldsymbol{K}^{(i-1)}, \boldsymbol{k}^{(i)}\right] \in \mathbb{R}^{d_{\text{key}} \times i}$$

$$\boldsymbol{V}^{(i)} = \left[\boldsymbol{V}^{(i-1)}, \boldsymbol{v}^{(i)}\right] \in \mathbb{R}^{d_{\text{value}} \times i}$$

$$\boldsymbol{y}^{(i)} = \boldsymbol{V}^{(i)} \underline{\mathrm{softmax}((\boldsymbol{K}^{(i)})^\top \boldsymbol{q}^{(i)})}$$

attention weights change for every new token
→ must store all previous KV

write softmax as kernel

$$\boldsymbol{y}^{(i)} = \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)} \kappa(\boldsymbol{k}^{(j)}, \boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i} \kappa(\boldsymbol{k}^{(j')}, \boldsymbol{q}^{(i)})}$$

use different kernel

$$\kappa'(\boldsymbol{k}, \boldsymbol{q}) = \phi(\boldsymbol{k})^\top \phi(\boldsymbol{q})$$

$$\boldsymbol{y}^{(i)} = \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)} \phi(\boldsymbol{k}^{(j)})^\top \phi(\boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i} \phi(\boldsymbol{k}^{(j')}) \cdot \phi(\boldsymbol{q}^{(i)})}$$

$$= \frac{\sum_{j=1}^{i} \left(\boldsymbol{v}^{(j)} \phi(\boldsymbol{k}^{(j)})^\top\right) \phi(\boldsymbol{q}^{(i)})}{\left(\sum_{j'=1}^{i} \phi(\boldsymbol{k}^{(j')})\right) \cdot \phi(\boldsymbol{q}^{(i)})}$$

An alternative derivation of attention -> RNN

We start from self-attention, and linearize the softmax

Further reading: "Transformers are RNNs" (2020), "Transformers are SSMs" (2024)

$$
\begin{aligned}
\boldsymbol{k}^{(i)}, \boldsymbol{v}^{(i)}, \boldsymbol{q}^{(i)} &= \boldsymbol{W}_k \boldsymbol{x}^{(i)}, \boldsymbol{W}_v \boldsymbol{x}^{(i)}, \boldsymbol{W}_q \boldsymbol{x}^{(i)} \\
\boldsymbol{K}^{(i)} &= \left[\boldsymbol{K}^{(i-1)}, \boldsymbol{k}^{(i)}\right] \in \mathbb{R}^{d_{\text{key}} \times i} \\
\boldsymbol{V}^{(i)} &= \left[\boldsymbol{V}^{(i-1)}, \boldsymbol{v}^{(i)}\right] \in \mathbb{R}^{d_{\text{value}} \times i} \\
\boldsymbol{y}^{(i)} &= \boldsymbol{V}^{(i)} \underline{\operatorname{softmax}((\boldsymbol{K}^{(i)})^\top \boldsymbol{q}^{(i)})}
\end{aligned}
$$

attention weights change for every new token $\rightarrow$ must store all previous KV

write softmax as kernel

$$
\boldsymbol{y}^{(i)} = \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)} \kappa(\boldsymbol{k}^{(j)}, \boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i} \kappa(\boldsymbol{k}^{(j')}, \boldsymbol{q}^{(i)})}
$$

use different kernel

$$
\kappa'(\boldsymbol{k}, \boldsymbol{q}) = \phi(\boldsymbol{k})^\top \phi(\boldsymbol{q})
$$

$$
\phi(x) = \operatorname{elu}(x) + 1,
$$

$$
\begin{aligned}
\boldsymbol{y}^{(i)} &= \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)} \phi(\boldsymbol{k}^{(j)})^\top \phi(\boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i} \phi(\boldsymbol{k}^{(j')}) \cdot \phi(\boldsymbol{q}^{(i)})} \\
&= \frac{\sum_{j=1}^{i} \left(\boldsymbol{v}^{(j)} \phi(\boldsymbol{k}^{(j)})^\top\right) \phi(\boldsymbol{q}^{(i)})}{\left(\sum_{j'=1}^{i} \phi(\boldsymbol{k}^{(j')})\right) \cdot \phi(\boldsymbol{q}^{(i)})}
\end{aligned}
$$

An alternative derivation of attention -> RNN

We start from self-attention, and linearize the softmax

Further reading: "Transformers are RNNs" (2020), "Transformers are SSMs" (2024)

$$\boldsymbol{k}^{(i)}, \boldsymbol{v}^{(i)}, \boldsymbol{q}^{(i)} = \boldsymbol{W}_k \boldsymbol{x}^{(i)}, \boldsymbol{W}_v \boldsymbol{x}^{(i)}, \boldsymbol{W}_q \boldsymbol{x}^{(i)}$$

$$\boldsymbol{K}^{(i)} = \left[\boldsymbol{K}^{(i-1)}, \boldsymbol{k}^{(i)}\right] \in \mathbb{R}^{d_{\text{key}} \times i}$$

$$\boldsymbol{V}^{(i)} = \left[\boldsymbol{V}^{(i-1)}, \boldsymbol{v}^{(i)}\right] \in \mathbb{R}^{d_{\text{value}} \times i}$$

$$\boldsymbol{y}^{(i)} = \boldsymbol{V}^{(i)} \underline{\mathrm{softmax}((\boldsymbol{K}^{(i)})^\top \boldsymbol{q}^{(i)})}$$

attention weights change for every new token → must store all previous KV

write softmax as kernel

$$\boldsymbol{y}^{(i)} = \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)} \kappa(\boldsymbol{k}^{(j)}, \boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i} \kappa(\boldsymbol{k}^{(j')}, \boldsymbol{q}^{(i)})}$$

use different kernel

$$\kappa'(\boldsymbol{k}, \boldsymbol{q}) = \phi(\boldsymbol{k})^\top \phi(\boldsymbol{q})$$

$$\phi(x) = \mathrm{elu}(x) + 1,$$

$$\boldsymbol{y}^{(i)} = \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)} \phi(\boldsymbol{k}^{(j)})^\top \phi(\boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i} \phi(\boldsymbol{k}^{(j')}) \cdot \phi(\boldsymbol{q}^{(i)})}$$

$$= \frac{\sum_{j=1}^{i} \boxed{\left(\boldsymbol{v}^{(j)} \phi(\boldsymbol{k}^{(j)})^\top\right)} \phi(\boldsymbol{q}^{(i)})}{\boxed{\left(\sum_{j'=1}^{i} \phi(\boldsymbol{k}^{(j')})\right)} \cdot \phi(\boldsymbol{q}^{(i)})}$$

store (1) key_sum,
(2) DxD matrix memory

Overview

Part 1: Modern Language Models

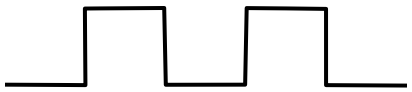
- LLMs 101
- Make Transformers Efficient
  - Keeping self-attention
  - Supplementing self-attention
  - Modifying/replacing self-attention

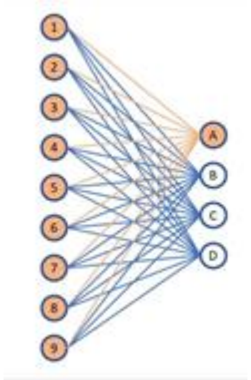
Part 2: Next-Generation Language Models

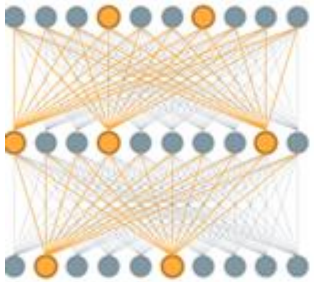
- State-Space Models
- **Neuromorphic Hardware**
- MatMul-free LM on Loihi
- What's next?

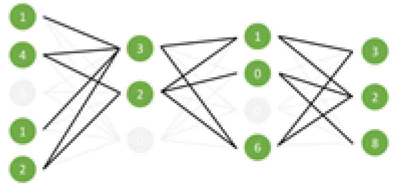
Neuromorphic Computing

CPUs	**GPUs / TPUs**	**Neuromorphic (e.g., Loihi 2)**
Synchronous clock	Synchronous clock	Asynchronous, event-based
Separate memory and processing	+ high-bandwidth memory	Memory-compute integration
Multi-core sequential processing	SIMD parallel, dense	MIMD parallel, sparse



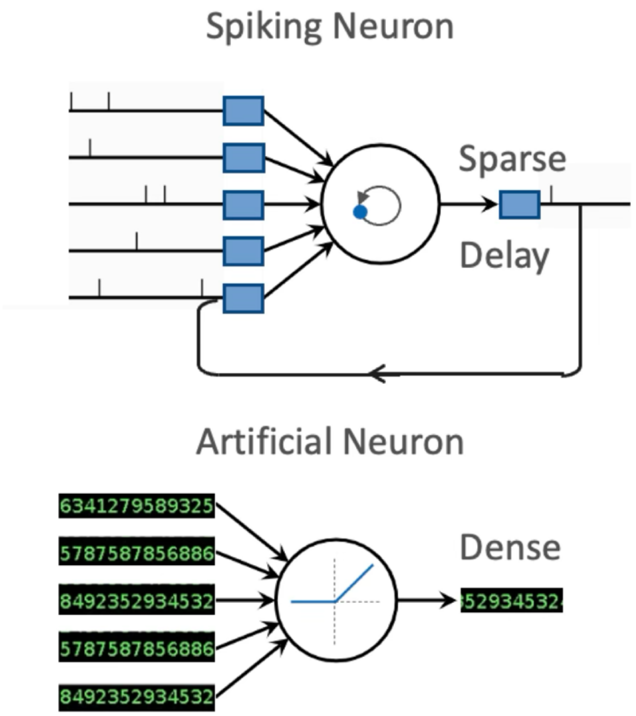






Neuromorphic Computing

Sparse Activity	Sparse Connectivity	Stateful Neurons	Neuron Dynamics	Recurrence	Delay

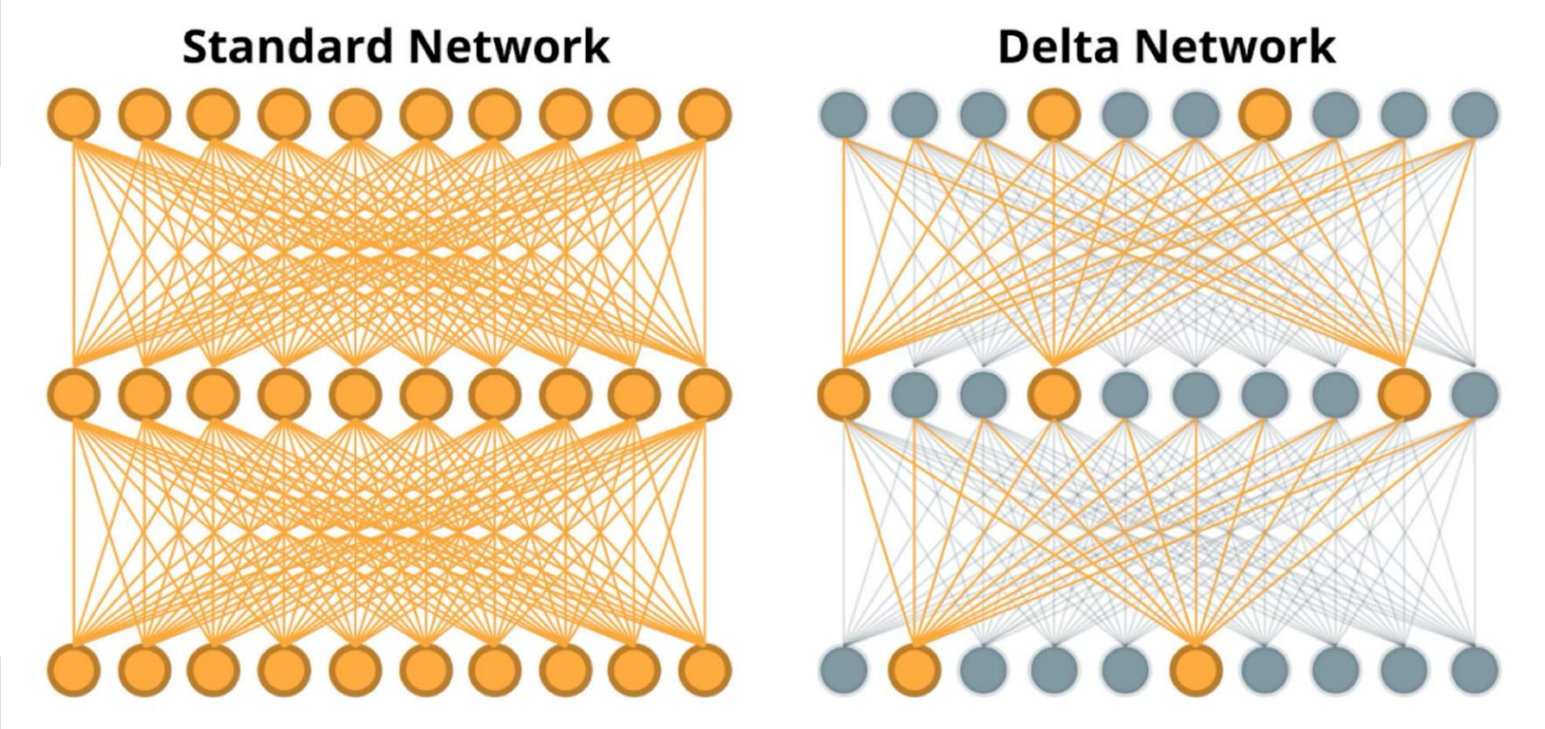


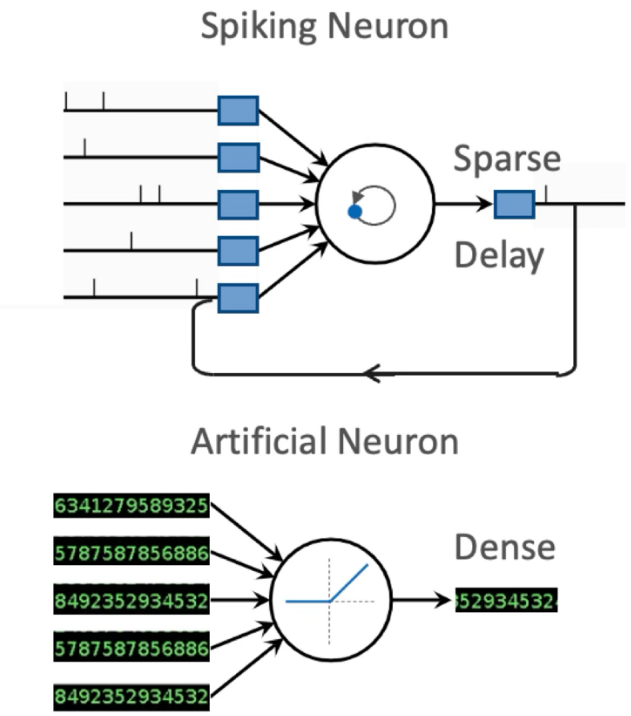
Slide credit: Philipp Stratmann (Intel Labs)

Neuromorphic Computing

Sparse Activity	Sparse Connectivity	Stateful Neurons	Neuron Dynamics	Recurrence	Delay

- Event based, sparse
- Message only when needed



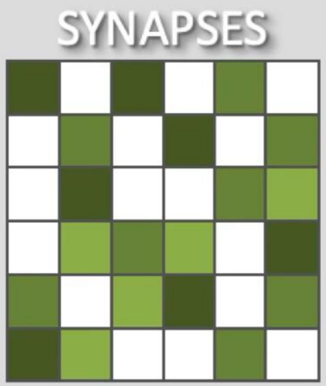


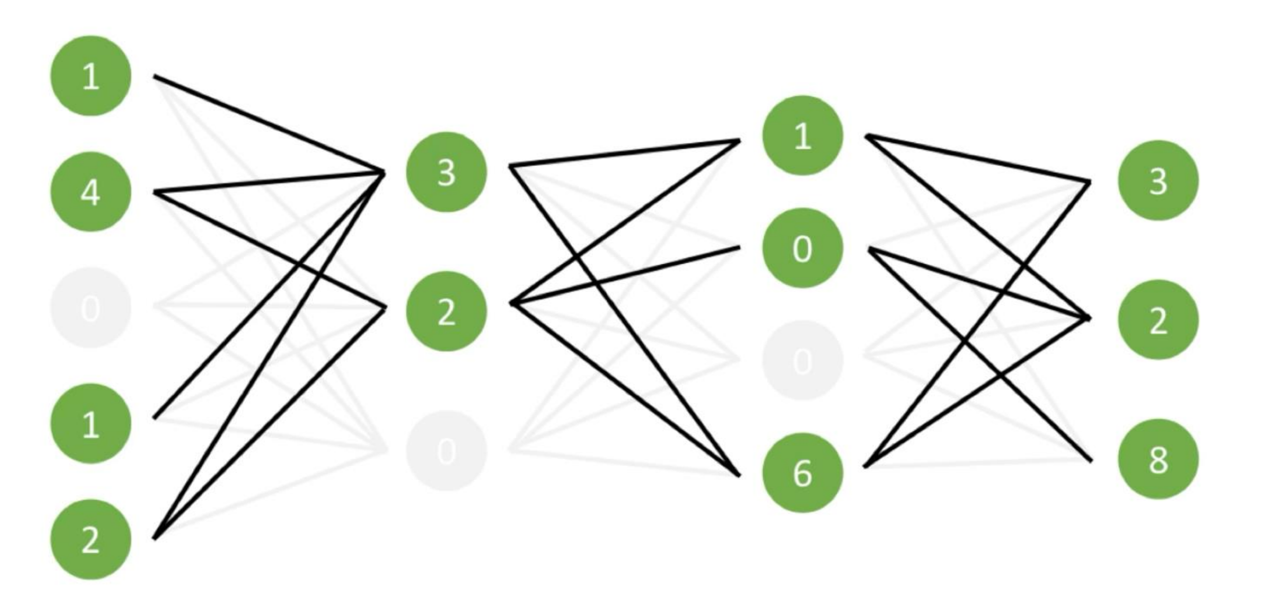
Slide credit: Philipp Stratmann (Intel Labs)

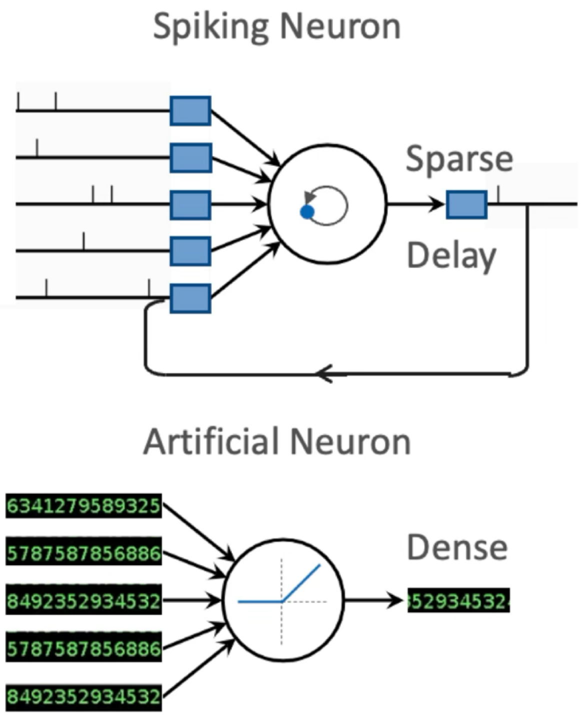
Neuromorphic Computing

Sparse Activity | **Sparse Connectivity** | Stateful Neurons | Neuron Dynamics | Recurrence | Delay

- Reduces computation per event
- Unstructured sparsity / Structured sparsity





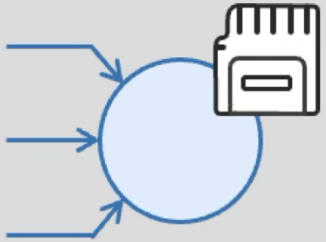


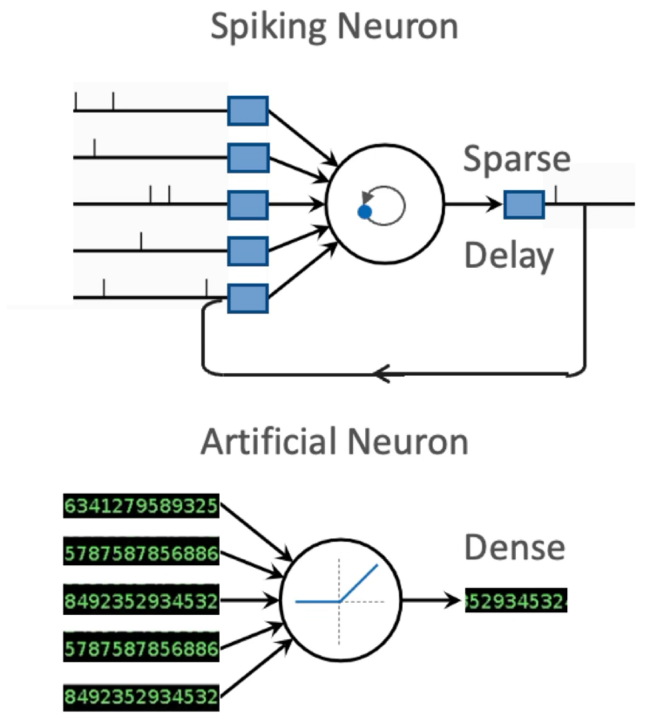
Slide credit: Philipp Stratmann (Intel Labs)

Neuromorphic Computing

Sparse Activity	Sparse Connectivity	Stateful Neurons	Neuron Dynamics	Recurrence	Delay

- Every neuron has state/memory
- Neurons in Loihi 2 use silicon real-estate, thus are expensive
- ReLU, sigmoid are stateless
  → inexpensive on CPU/GPU



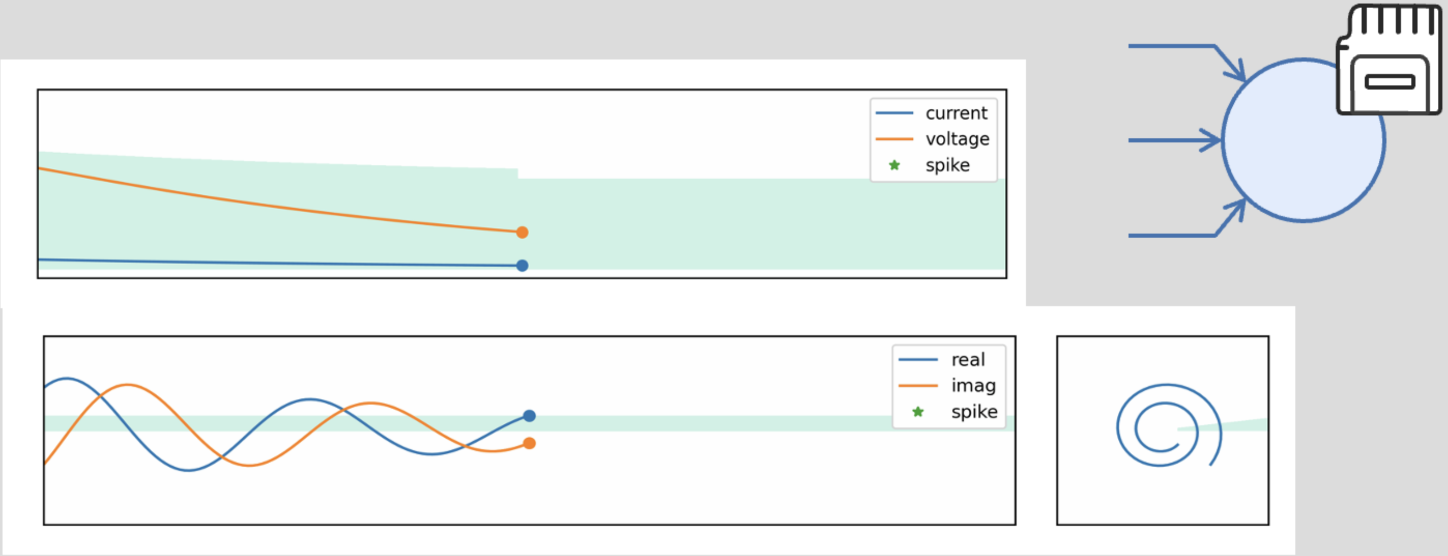


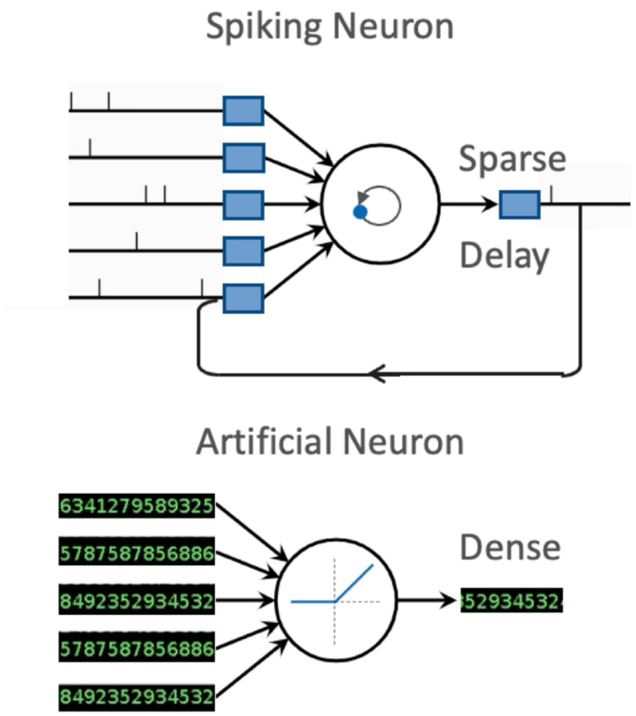
Slide credit: Philipp Stratmann (Intel Labs)

Neuromorphic Computing

Sparse Activity	Sparse Connectivity	Stateful Neurons	Neuron Dynamics	Recurrence	Delay

- Micro-code can change states
- Iterative computation of state dynamics



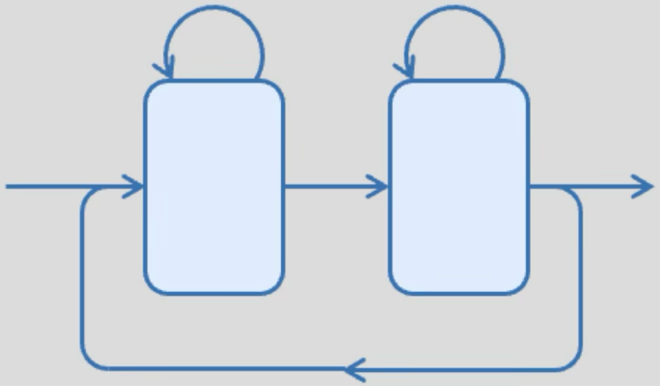


Slide credit: Philipp Stratmann (Intel Labs)

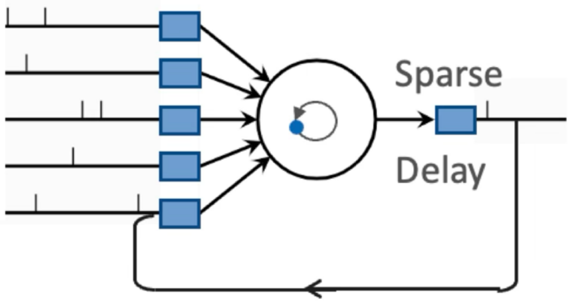
Neuromorphic Computing

Sparse Activity	Sparse Connectivity	Stateful Neurons	Neuron Dynamics	Recurrence	Delay

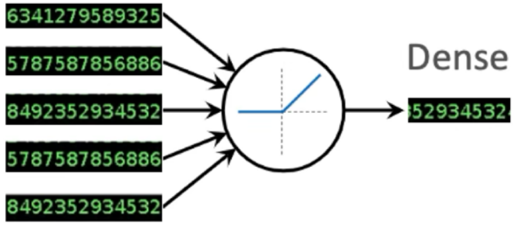
- Pros
  - Re-use states
  - Re-use neurocore resources
  - More expressive
- Cons
  - Hard to train



Spiking Neuron



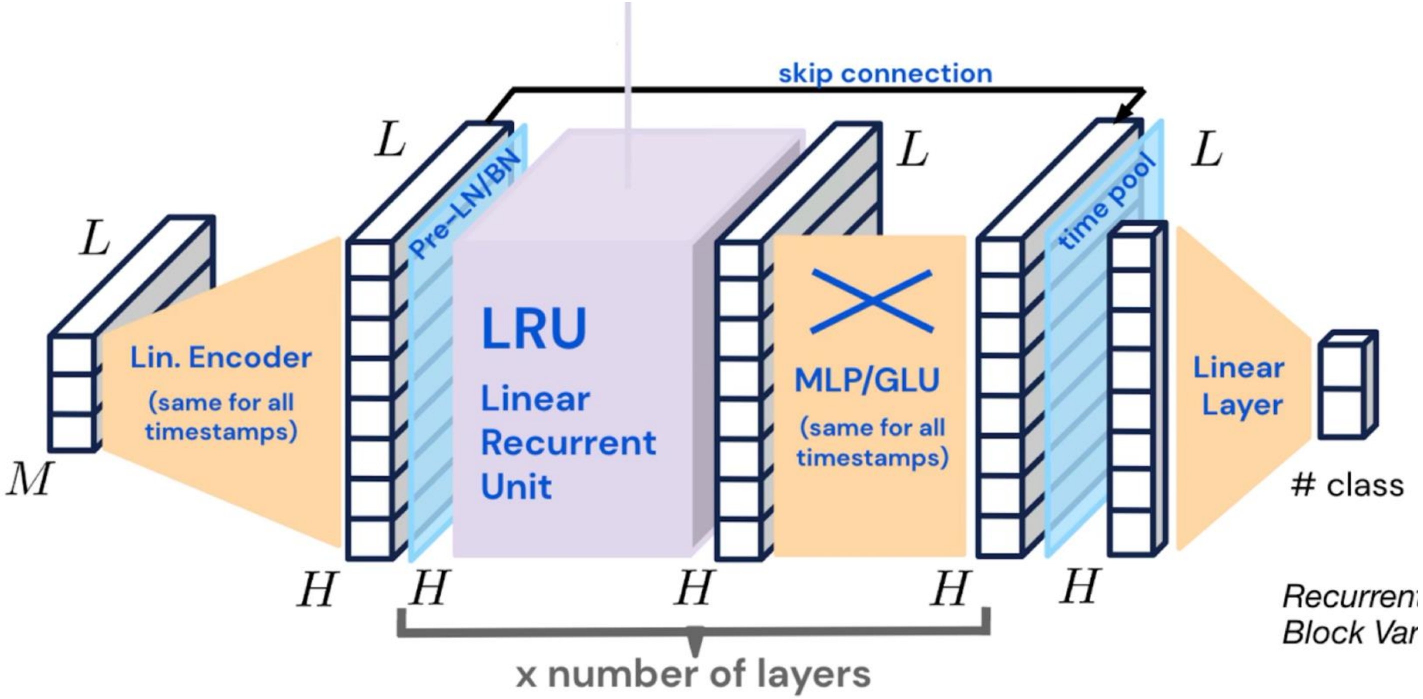
Artificial Neuron

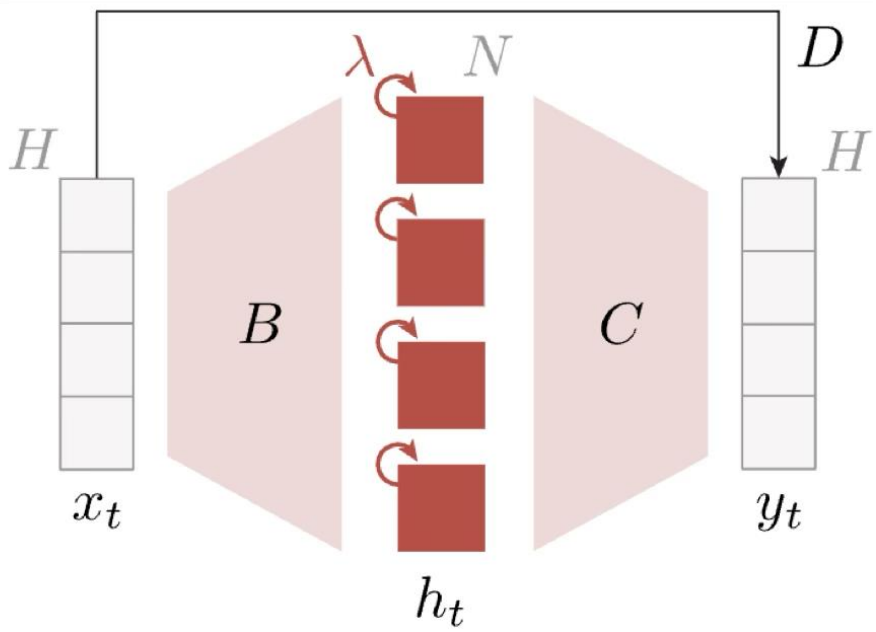


Slide credit: Philipp Stratmann (Intel Labs)

Neuromorphic Computing with State Space Models

Sparse Activity	Sparse Connectivity	Stateful Neurons	Neuron Dynamics	Recurrence	Delay





Overview

Part 1: Modern Language Models

- LLMs 101
- Make Transformers Efficient
  - Keeping self-attention
  - Supplementing self-attention
  - Modifying/replacing self-attention

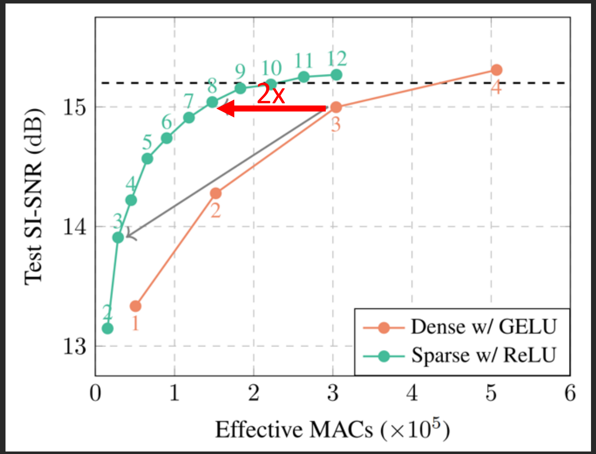
Part 2: Next-Generation Language Models

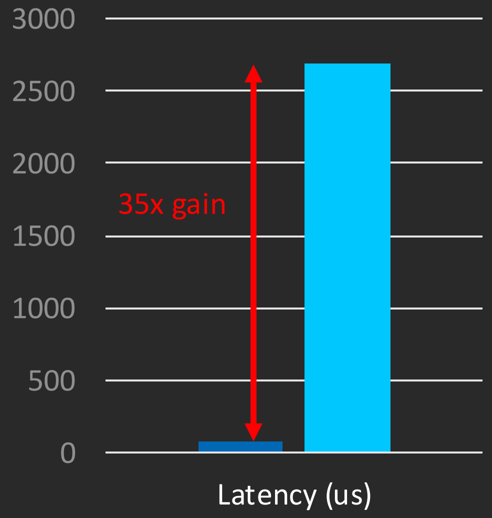
- State-Space Models
- Neuromorphic Hardware
- **MatMul-free LM on Loihi**
- What’s next?

Summary: linear RNNs enable low-latency energy-efficient language modeling on neuromorphic hardware

S5 State Space Model

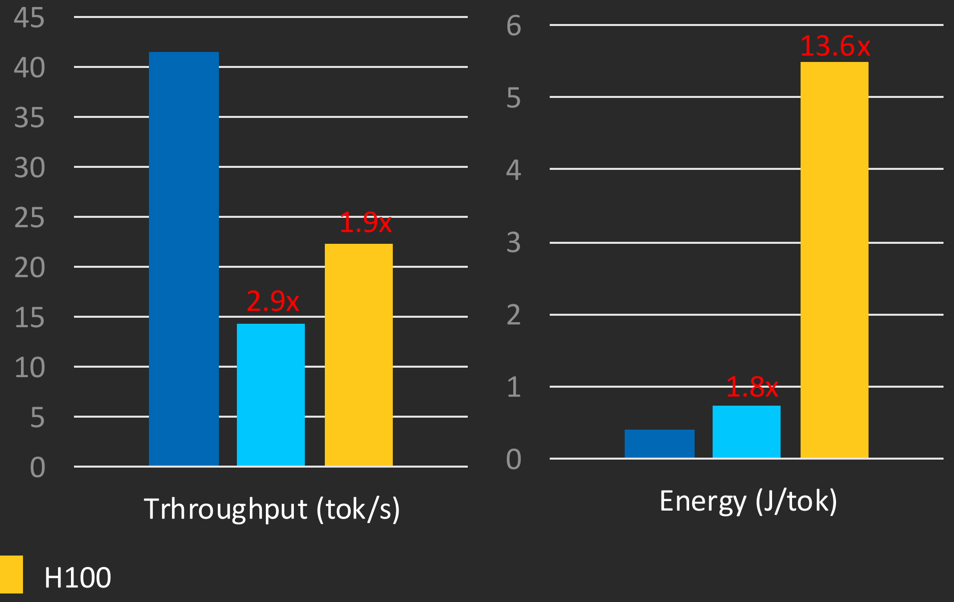
Trained 90% sparse and quantized S5 model shows **35x lower latency** and **1209x lower energy** on **real-time audio denoising**, compared to iso-accuracy dense model on a Jetson Orin Nano





Matmul-free LLM

Implemented on Loihi, enabling the deployment of 370M pre-trained model, showing 1.9x higher throughput and 13.6x lower energy in batch-1 generation, compared to an H100 and Orin Nano

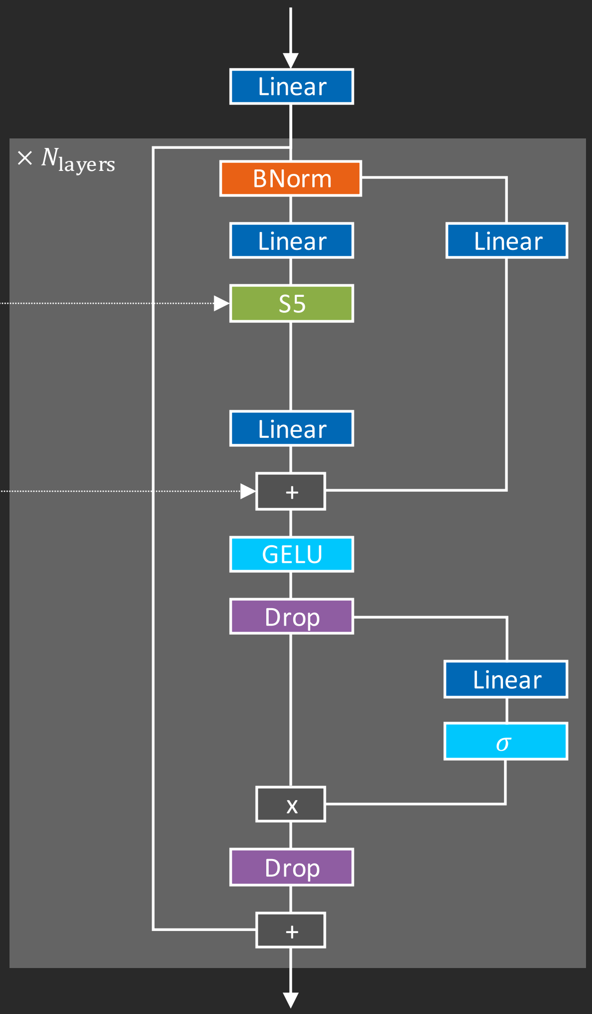


S5 State Space Model architecture

- Features:
  - Fast training with parallel scan
  - Fast inference in recurrent mode
  - Strong signal processing capabilities
  - Generalize to different inference rates

$$h_{t+1} = diag(A) \odot h_t + B^T u_{t+1}$$

$$x_{t+1} = C^T h_{t+1} + diag(D) \odot u_{t+1}$$

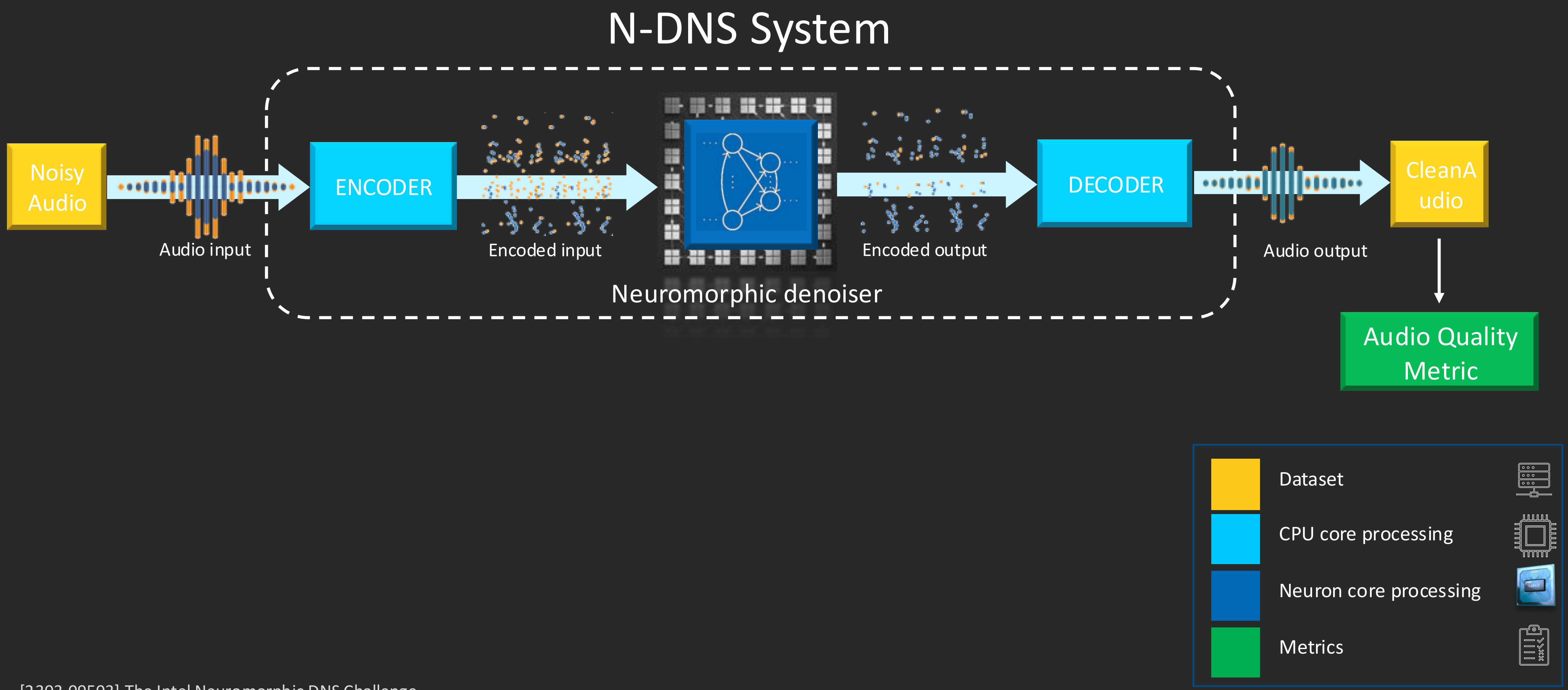


SIMPLIFIED STATE SPACE LAYERS FOR SEQUENCE MODELING

Jimmy T.H. Smith[*,1,2], **Andrew Warrington**[*,2,3], **Scott W. Linderman**[2,3]
[*]Equal contribution.
[1]Institute for Computational and Mathematical Engineering, Stanford University.
[2]Wu Tsai Neurosciences Institute, Stanford University.
[3]Department of Statistics, Stanford University.
{jsmith14,awarring,scott.linderman}@stanford.edu.

[2208.04933] Simplified State Space Layers for Sequence Modeling

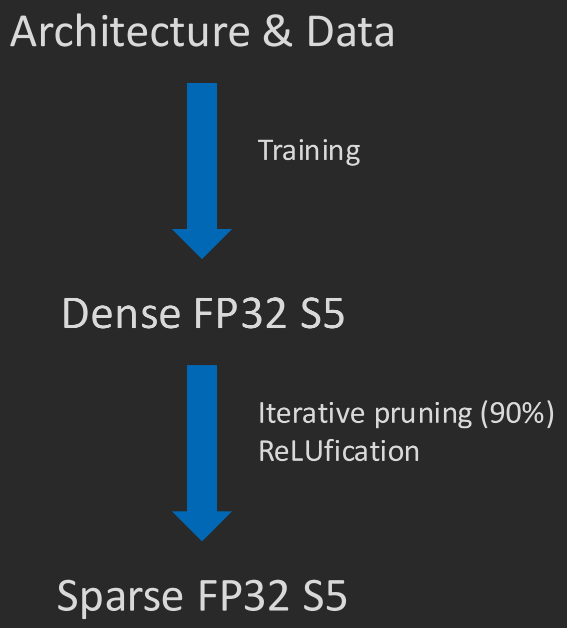
Intel N-DNS Challenge



[2303.09503] The Intel Neuromorphic DNS Challenge

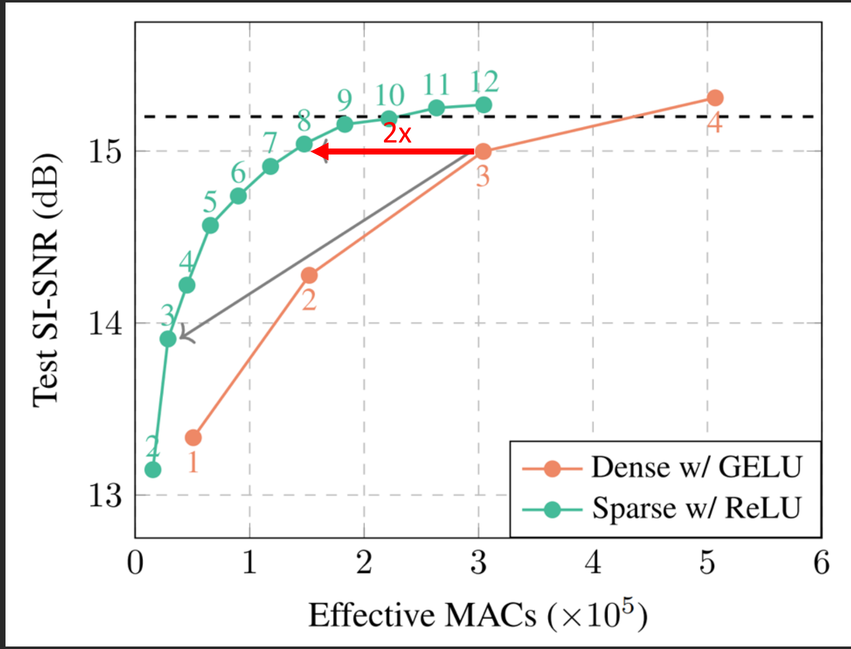
Model compression pipeline: unstructured sparsity

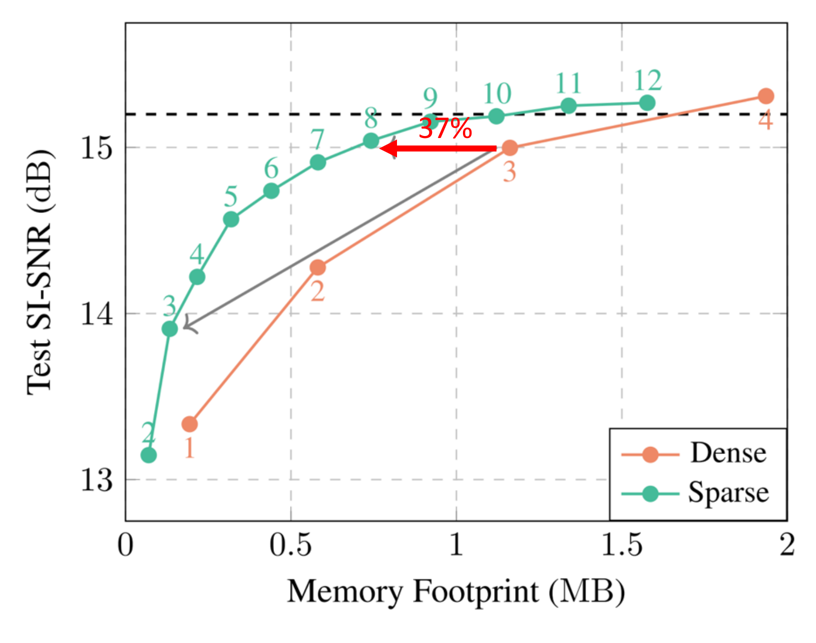
- Weight sparsity
  - **Iterative Magnitude Pruning** gradually updates the sparsity masks during training to reach a target sparsity
  - It works better than one-shot pruning at high sparsity levels
- Activation sparsity
  - **ReLUfication**: replaced GELU non-linaerity with ReLU and introduced additional ReLUs before key linear layers
- Both interventions are applied with a single fine-tuning run, starting from a pre-trained dense model



[2310.04564] ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models
[2304.14082] JaxPruner: A concise library for sparsity research
[1710.01878] To prune, or not to prune: exploring the efficacy of pruning for model compression

Key Result 1: unstructured sparse models are at the efficiency-performance Pareto front





Pareto fronts for S5 network audio denoising quality (SI-SNR) as a function of effective compute (left) and memory footprint (right) on the Intel N-DNS test set. S5 networks with weight and activation sparsity (green) exhibit a large domain of Pareto optimality versus dense S5 networks (orange). Number annotations enumerate increasing S5 dimensionality configurations, from 500 k to 4 M parameters. Dashed horizontal like marks SI-SNR of Spiking-FullSubNet XL, the previous state-of-the-art model. The horizontal arrows highlight models used for hardware deployment, the diagonal arrows highlight models of the same width.

Model compression pipeline: quantization

- Loihi requires fully quantized computation
- Precision: 8bit for weights, 16bit for activations and diagonal components
- Three steps
  - Quantization-aware training (optional)
  - Conversion to static quantization
  - Fixed-point arithmetic (simulates execution on the chip)

Architecture & Data

↓ Training

Dense FP32 S5

↓ Iterative pruning (90%)
ReLUfication
Quantization-aware training

Sparse Quantized S5

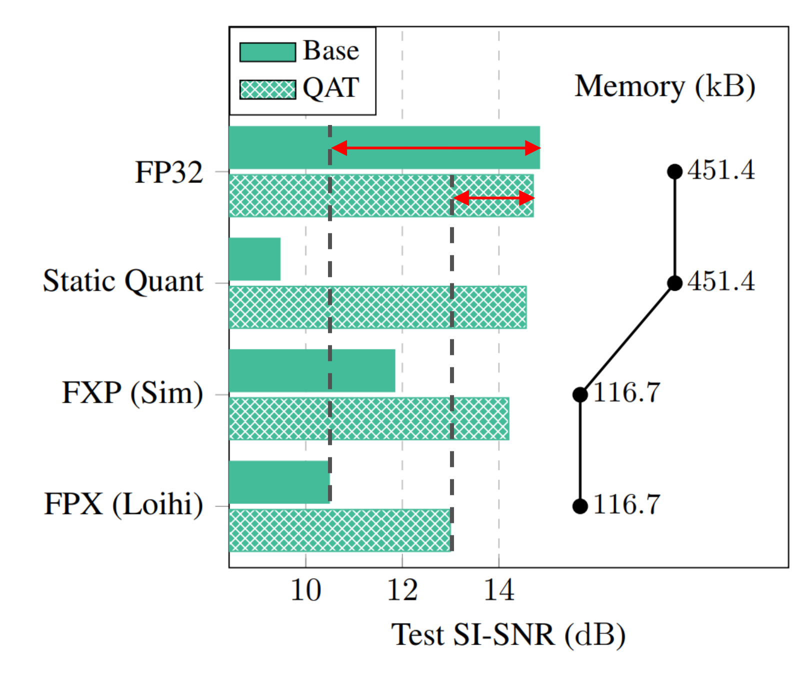
↓ Static quantization
Fixed-point conversion

Sparse FXP S5

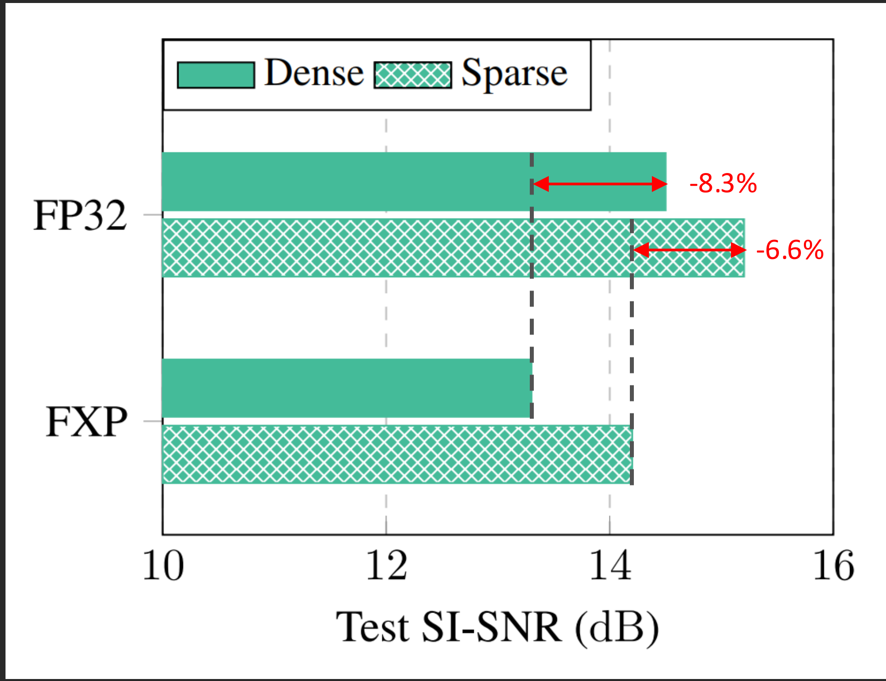
[2406.09477] Q-S5: Towards Quantized State Space Models

Key Result 2: sparse models can be converted to fixed-point with small accuracy degradation

QAT significantly reduces the FP to FXP gap



Sparse wider models are more resilient to quantization



[WIP] close the gap between FXP simulation and Loihi

Impact of quantization interventions on Test SI-SNR and memory footprint, with and without quantization-aware training, for model variant sparse-6.

Model compression pipeline: hardware acceleration

- Real-time audio de-noising requires each token to be processed within 8ms
  - Parallelization on sequence length not possible!
- We implement the sparse S5 model on Loihi using the new NxKernel API
- Benchmark latency, energy, and throughput against a dense FP32 JAX implementation on a Jetson Orin Nano
  - The quantized version didn't provide a speedup on the Jetson

Architecture & Data

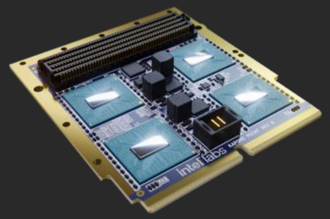
Training

Dense FP32 S5 → Deploy → Jetson Orin Nano

Iterative pruning (90%)
ReLUfication
Quantization-aware training

Sparse Quantized S5

Static quantization
Fixed-point conversion



Sparse FXP S5 → Deploy → Intel Loihi 2

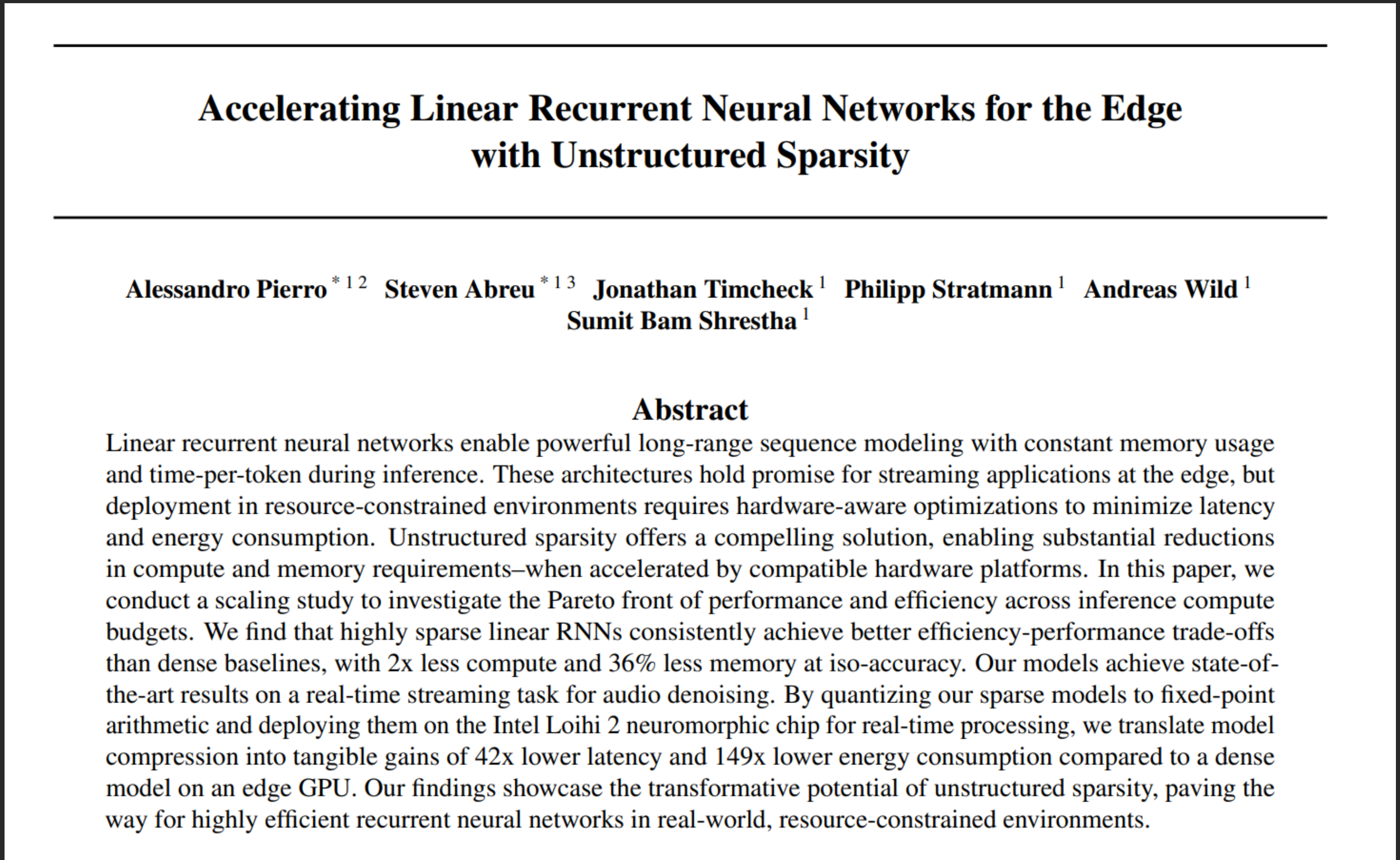
Key Result 3: hardware results

	Mode	Latency (↓)	Energy (↓)	Throughput (↑)
Token-by-token		35x	1209x	35x
Intel Loihi 2†	Fall-Through	76 µs	13 µJ/tok	13 178 tok/s
Jetson Orin Nano‡	Recurrent 1-step ($b = 1$)	2 688 µs	15 724 µJ/tok	372 tok/s
Jetson Orin Nano‡	Recurrent 10-step ($b = 1$)	3 224 µs	1 936 µJ/tok	3 103 tok/s
Jetson Orin Nano‡	Recurrent 100-step ($b = 1$)	10 653 µs	626 µJ/tok	9 516 tok/s
Jetson Orin Nano‡	Recurrent scan ($b = 1$)	236 717 µs	404 µJ/tok	15 845 tok/s
Sample-by-sample				
Intel Loihi 2†	Pipeline	60.58 ms	185.80 mJ/sam	16.58 sam/s
Jetson Orin Nano‡	Scan ($b = 1$)	233.48 ms	1 512.60 mJ/sam	4.28 sam/s
Jetson Orin Nano‡	Scan ($b = b_{max}$)	226.53 ms	5.89 mJ/sam	1 130.09 sam/s

Power and performance results∗ . The Loihi 2 is running a sparse and quantized S5 model, while the Jetson Orin Nano is running a smaller dense S5 model that reaches similar test performance. All measurements are averaged over 8 random samples from the test set, each containing 3750 steps.

† Loihi 2 workloads were characterized on an Oheo Gulch system with N3C1-revision Loihi 2 chips running NxCore 2.5.8 and NxKernel 0.2.0 with on-chip IO unthrottled sequencing of inputs. Researchers interested to run S5 on Loihi 2 can gain access to the software and systems by joining Intel's Neuromorphic Research Community. ‡ Jetson workloads were characterized on an NVIDIA Jetson Orin Nano 8GB running Jetpack 6.2, CUDA 12.4, JAX 0.4.32, using the MAXN SUPER power mode; energy values are computed based on the TOT power as reported by jtop 4.3.0. The batch size bmax = 256 was chosen to be the largest that fits into memory. ∗Performance results are based on testing as of January 2025 and may not reflect all publicly available security updates; results may vary

Accepted to ICML 2025 and available on arXiv



Accelerating Linear Recurrent Neural Networks for the Edge with Unstructured Sparsity

Alessandro Pierro [*,1,2] **Steven Abreu** [*,1,3] **Jonathan Timcheck** [1] **Philipp Stratmann** [1] **Andreas Wild** [1]
Sumit Bam Shrestha [1]

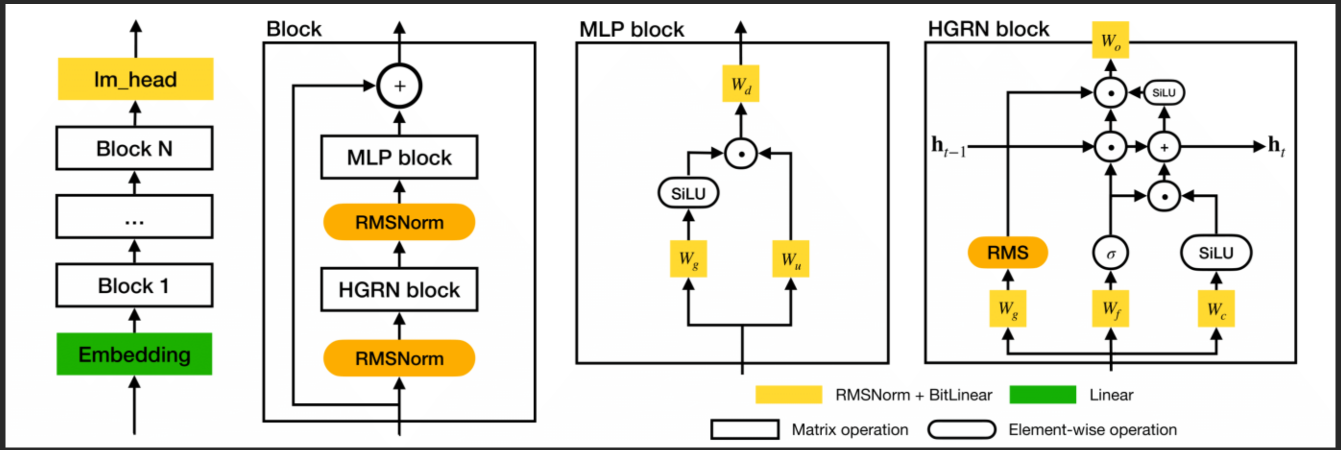
Abstract

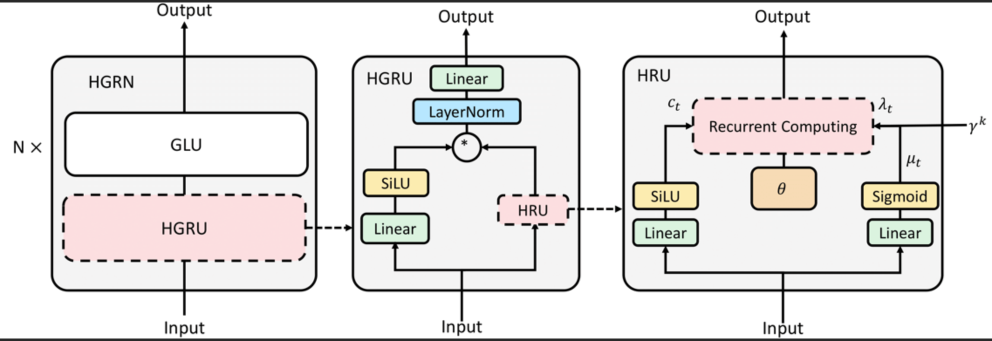
Linear recurrent neural networks enable powerful long-range sequence modeling with constant memory usage and time-per-token during inference. These architectures hold promise for streaming applications at the edge, but deployment in resource-constrained environments requires hardware-aware optimizations to minimize latency and energy consumption. Unstructured sparsity offers a compelling solution, enabling substantial reductions in compute and memory requirements–when accelerated by compatible hardware platforms. In this paper, we conduct a scaling study to investigate the Pareto front of performance and efficiency across inference compute budgets. We find that highly sparse linear RNNs consistently achieve better efficiency-performance trade-offs than dense baselines, with 2x less compute and 36% less memory at iso-accuracy. Our models achieve state-of-the-art results on a real-time streaming task for audio denoising. By quantizing our sparse models to fixed-point arithmetic and deploying them on the Intel Loihi 2 neuromorphic chip for real-time processing, we translate model compression into tangible gains of 42x lower latency and 149x lower energy consumption compared to a dense model on an edge GPU. Our findings showcase the transformative potential of unstructured sparsity, paving the way for highly efficient recurrent neural networks in real-world, resource-constrained environments.

Language modeling on Loihi 2

- Want fully recurrent LLM
  - We use an HGRN-based LLM with ternary weights – 370M params

$$
\begin{aligned}
f_t &= \sigma\left(\mathrm{BitLinear}(x_t; \mathbf{W}_f, g_f, \epsilon)\right), \\
c_t &= \tau\left(\mathrm{BitLinear}(x_t; \mathbf{W}_c, g_c, \epsilon)\right), \\
h_t &= f_t \odot h_{t-1} + (1 - f_t) \odot c_t, \\
g_t &= \mathrm{RMSNorm}\left(\mathrm{BitLinear}(x_t; \mathbf{W}_g, g_g, \epsilon); g_{g'}, \epsilon\right), \\
o_t' &= g_t \odot \tau(h_t), \\
o_t &= \mathrm{BitLinear}(o_t'; \mathbf{W}_o, g_o, \epsilon)
\end{aligned}
$$





Scalable MatMul-free Language Modeling

Rui-Jie Zhu[1], Yu Zhang[2], Ethan Sifferman[1], Tyler Sheaves[3], Yiqiao Wang[4], Dustin Richmond[1], Peng Zhou[1,4], Jason K. Eshraghian[1*]

[1]University of California, Santa Cruz [2]Soochow University
[3]University of California, Davis [4]LuxiTech

[2406.02528] Scalable MatMul-free Language Modeling, [2311.04823] Hierarchically Gated Recurrent Neural Network for Sequence Modeling (NeurIPS 2023 Spotlight)

Quantization of the model (simulation)

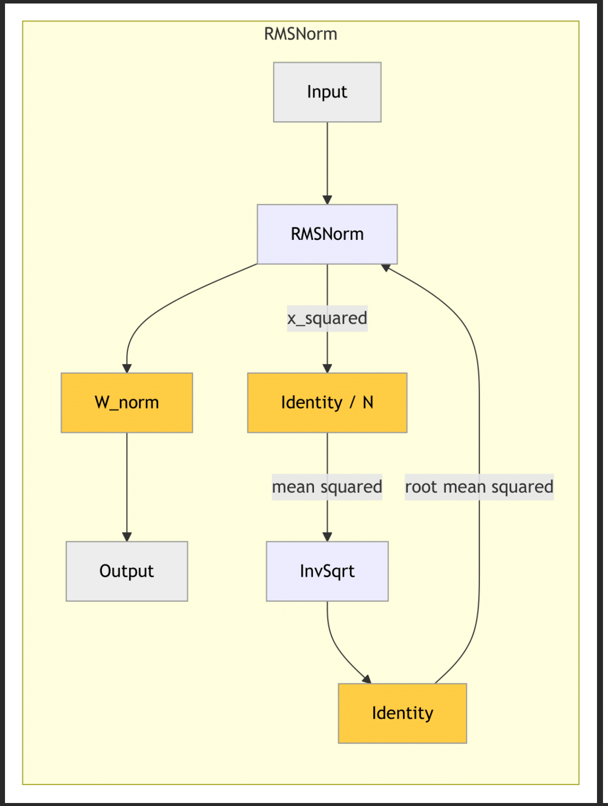
- No accuracy loss
  - Only quantizing element-wise weights (matrices are ternary)

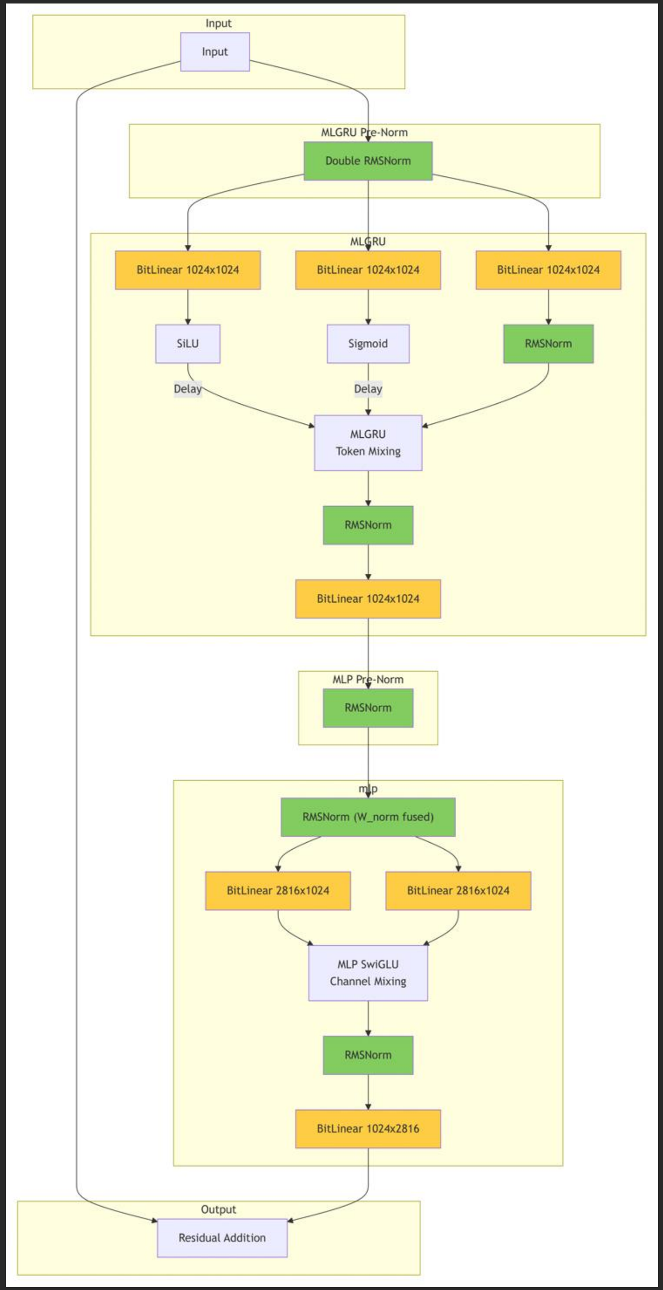
Configuration	ARCc	ARCe	HS	OQA	PQ	WG	Avg	Diff.†
MatMul-free baseline	22.8	42.1	32.4	28.4	62.6	49.4	39.6	(0.0%)
Transformer baseline	24.0	45.0	34.3	29.2	64.0	49.9	41.1	(3.8%)
Qwen2-500M	31.0	64.6	49.1	35.2	70.3	56.5	51.1	(29.0%)
PT	22.7	42.2	32.5	28.4	62.4	48.5	39.4	-0.4%
PT + W8	23.2	41.8	32.4	28.0	62.4	49.5	39.5	-0.2%
PT + A8	22.7	40.0	31.5	27.6	61.0	50.0	38.8	-2.0%
PT + A16	22.7	42.5	32.5	29.0	63.2	49.9	40.0	**0.9%**
PT + W8A8	22.3	40.3	31.9	27.2	59.9	49.1	38.5	-2.9%
PT + W8A16	22.7	42.3	32.3	28.0	63.1	49.3	39.6	0.0%
PT + W8A8 + $\epsilon_{\text{rms}} \uparrow$	28.3	26.8	26.1	27.0	52.7	51.5	35.4	-10.7%
PT + W8A16 + $\epsilon_{\text{rms}} \uparrow$	23.0	42.4	32.4	27.8	63.0	50.1	39.8%	**0.4%**

Results from quantization of the 370M MatMul-free language model on GPU. Baseline: optimized models from Zhu et al. (2024) and Qwen Team (2024). PT: PyTorch-only implementation. Ax / Wx: activations / RMSNorm weights quantized to x-bit integers. ϵ_rms↑: setting the value for ϵ_rms to 10−3 from previously ϵ_rms = 10−6.
†: difference relative to MatMul-free baseline.

Other modifications

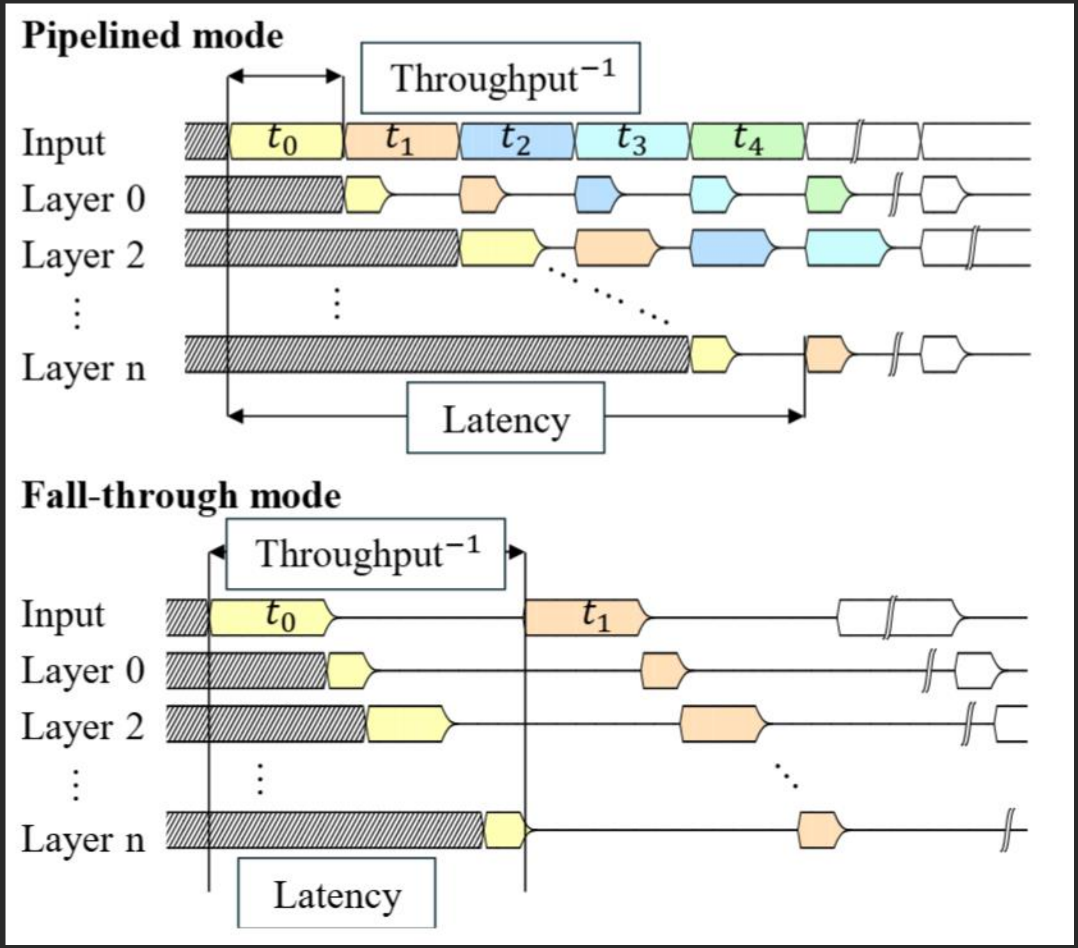
- Fixed-point implementation of the RMSNorm (incl. InvSqrt), Sigmoid function, etc.
- Mapping the model to Loihi
- Operator fusion where possible





Execution modes on Loihi 2

- Pipelined mode: high throughput -> prefill
- Fall-though mode: low latency -> autoregressive generation



PnP results: batch-1 processing

			Throughput (↑ tokens/sec)					Efficiency (↓ mJ/token)				
	Sequence length		500	1000	4000	8000	16000	500	1000	4000	8000	16000
Generate	**MMF (370M)**	**Loihi 2**	**41.5**	**41.5**	**41.5**	**41.5**	**41.5**	**405**	**405**	**405**	**405**	**405**
	MMF (370M)	H100	13.4	13.3	13.5	13.2	13.5	10.1k	10.1k	10.0k	9.9k	9.8k
	TF++ (370M)	H100	22.4	22.9	21.7	21.3	20.9	5.5k	5.6k	6.2k	6.8k	8.2k
	Llama* (400M)	Jetson	14.3	14.9	14.7	15.2	12.8	723	719	853	812	1.2k
	Qwen2 (500M)	Jetson	13.4	14.0	14.1	15.4	12.6	791	785	912	839	1.2k
Prefill	**MMF (370M)**	**Loihi 2**	**6632**	**6632**	**6632**	**6632**	**6632**	**3.7**	**3.7**	**3.7**	**3.7**	**3.7**
	MMF (370M)	H100	11.4k	13.1k	30.6k	51.6k	84.6k	6.1	5.3	2.5	1.4	0.9
	TF++ (370M)	H100	21.6k	32.7k	44.3k	55.4k	60.5k	11.3	7.3	5.4	4.3	3.8
	Llama* (400M)	Jetson	849	1620	3153	2258	1440	11.7	7.8	6.8	7.6	11.5
	Qwen2 (500M)	Jetson	627	909	2639	3861	3617	17.9	13.9	6.7	4.4	5.3

Throughput and energy efficiency for two transformer-based language models running on the NVIDIA Jetson Orin Nano compared to our MatMul-free LM running on Intel's Loihi 2, across different sequence lengths for prefill and generation. The best-performing sequence length for each model and metric is underlined. Metrics for Loihi 2 are based on preliminary experiments and subject to further performance optimization. Gen: autoregressive generation, Prefill: prefill mode. ∗ Llama representative model from Montebovi (2024).

Overview

Part 1: Modern Language Models

- LLMs 101
- Make Transformers Efficient
  - Keeping self-attention
  - Supplementing self-attention
  - Modifying/replacing self-attention

Part 2: Next-Generation Language Models

- State-Space Models
- Neuromorphic Hardware
- MatMul-free LM on Loihi
- **What's next?**

What’s next?

- Scaling up to larger models – (far) beyond 1B parameters
  - Current largest LLM on Loihi 2 is ~500M parameters
- Hardware-model co-design from scratch
  - Instead of fitting a model to some hardware, can we design optimal models for given hardware?
  - How far can we push the advantages of unstructured sparsity in weights and activations?
- State-of-the-art LLMs
  - Recurrent LLMs are limited, the best models might be hybrid attention-recurrent
    (e.g. RecurrentGemma, Jamba, Hymba)

Thank you