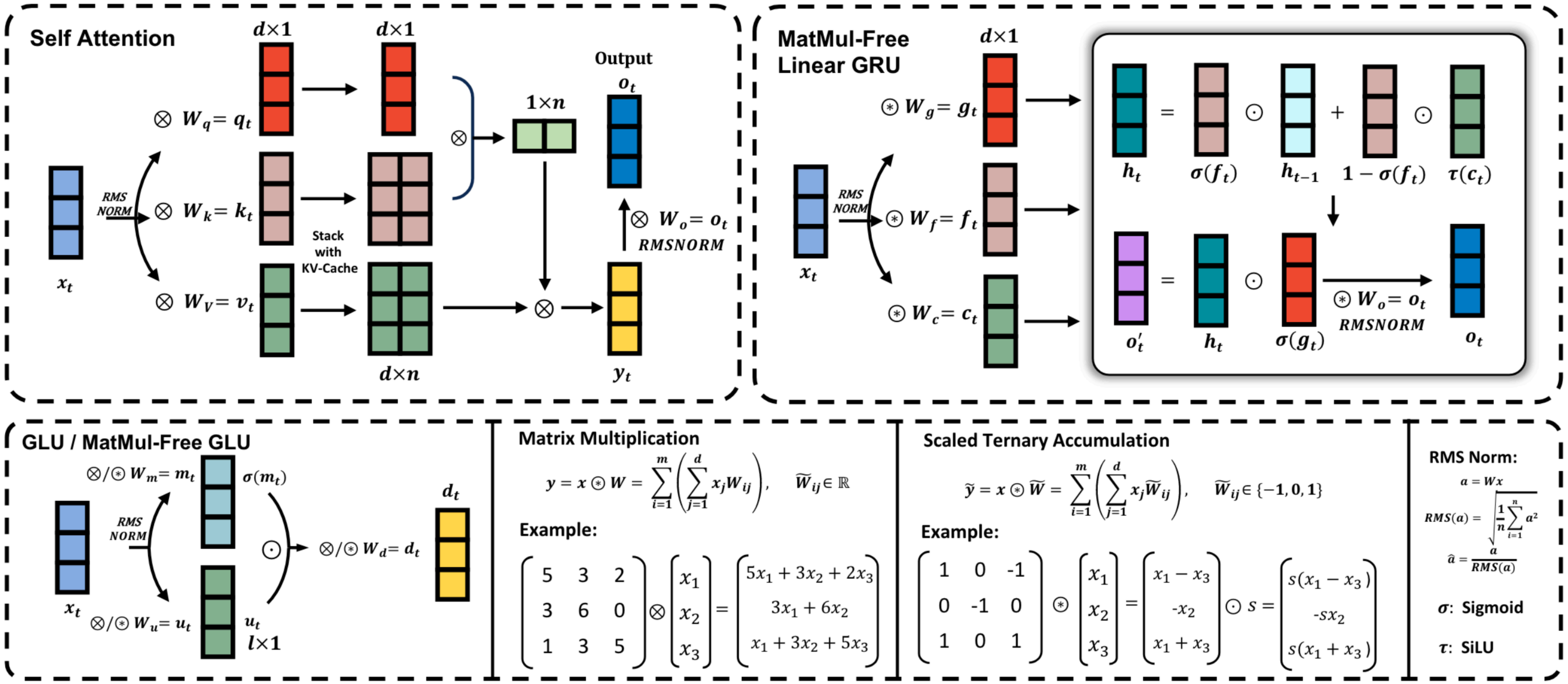


Scalable MatMul-Free Language Modelling

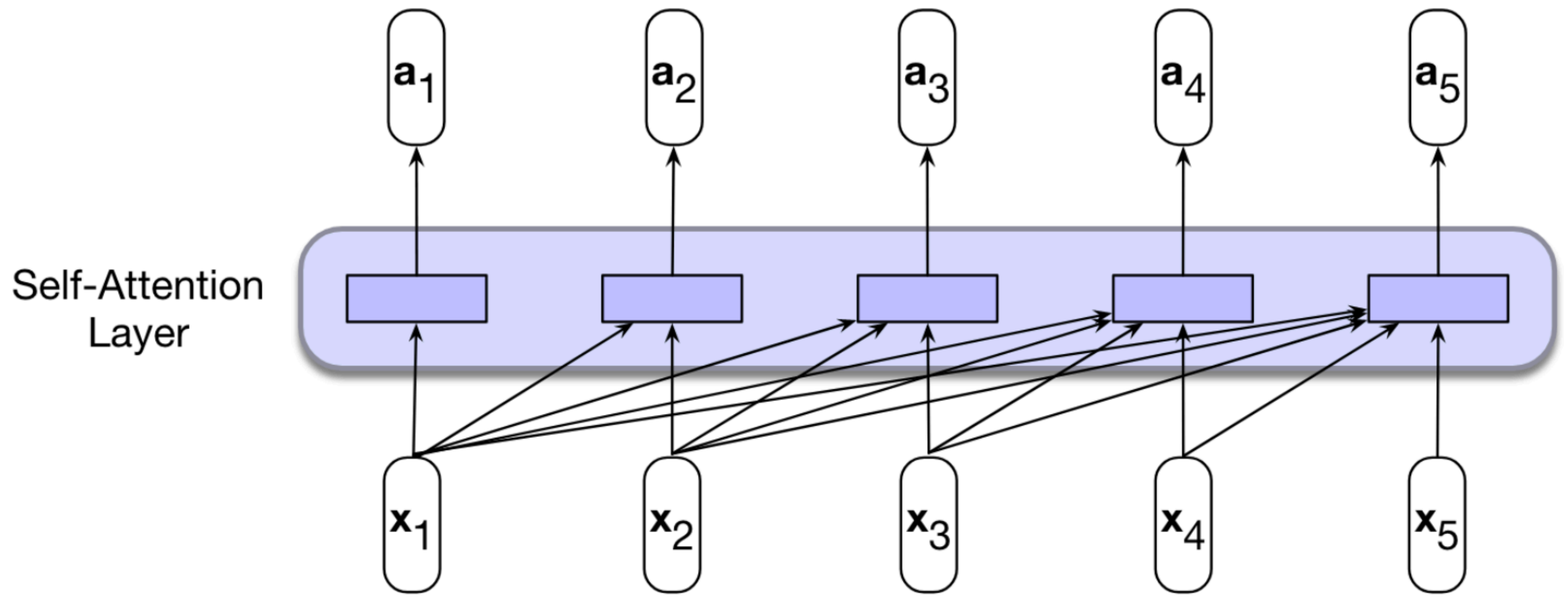
A Brief Overview

Sagar Prakash Barad
15 July 2024

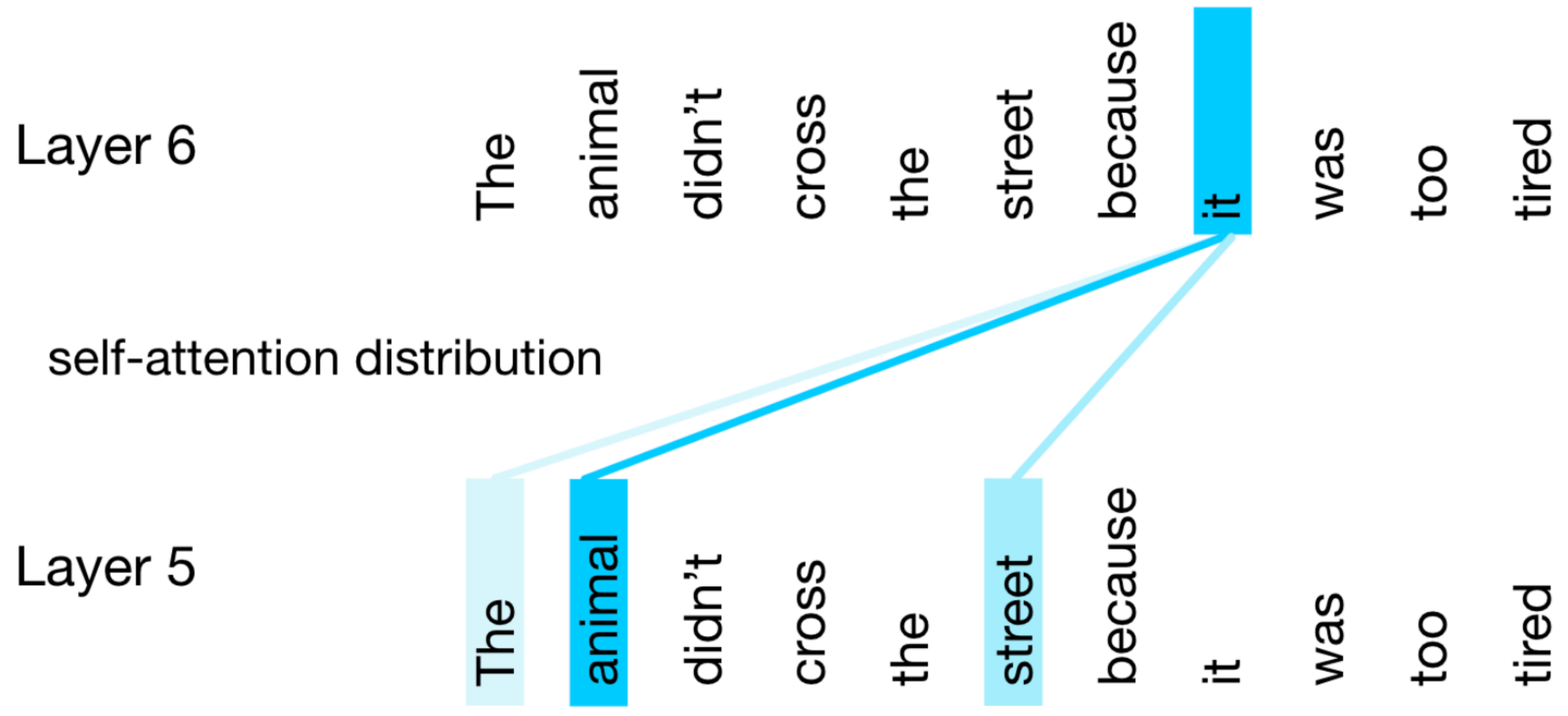
Transformers and Large Language Model

Pretraining—learning knowledge about pre-training language and the world from vast amounts of text—and call the resulting pre-trained language models **large language models**.

Transformers-based large language models (LLMs) uses a novel mechanism called **Self Attention,** where compares item/words/tokens of interest to a collection of other item/words/tokens in a way that reveals their relevance in the current context.



causal (or masked) self-attention model



The self-attention weight distribution α that is part of the computation of the representation for the word it at layer 6. In computing the representation for it, we attend differently to the various words at layer 5, with darker shades indicating higher self-attention values.

Transformers and Large Language Model

$$q_i = x_i W_Q; k_i = x_i W_K; v_i = x_i W_V$$

$$score(x_i, x_j) = \frac{q_i \,.\, k_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = softmax(score(x_i, x_j)) \forall j \le i$$

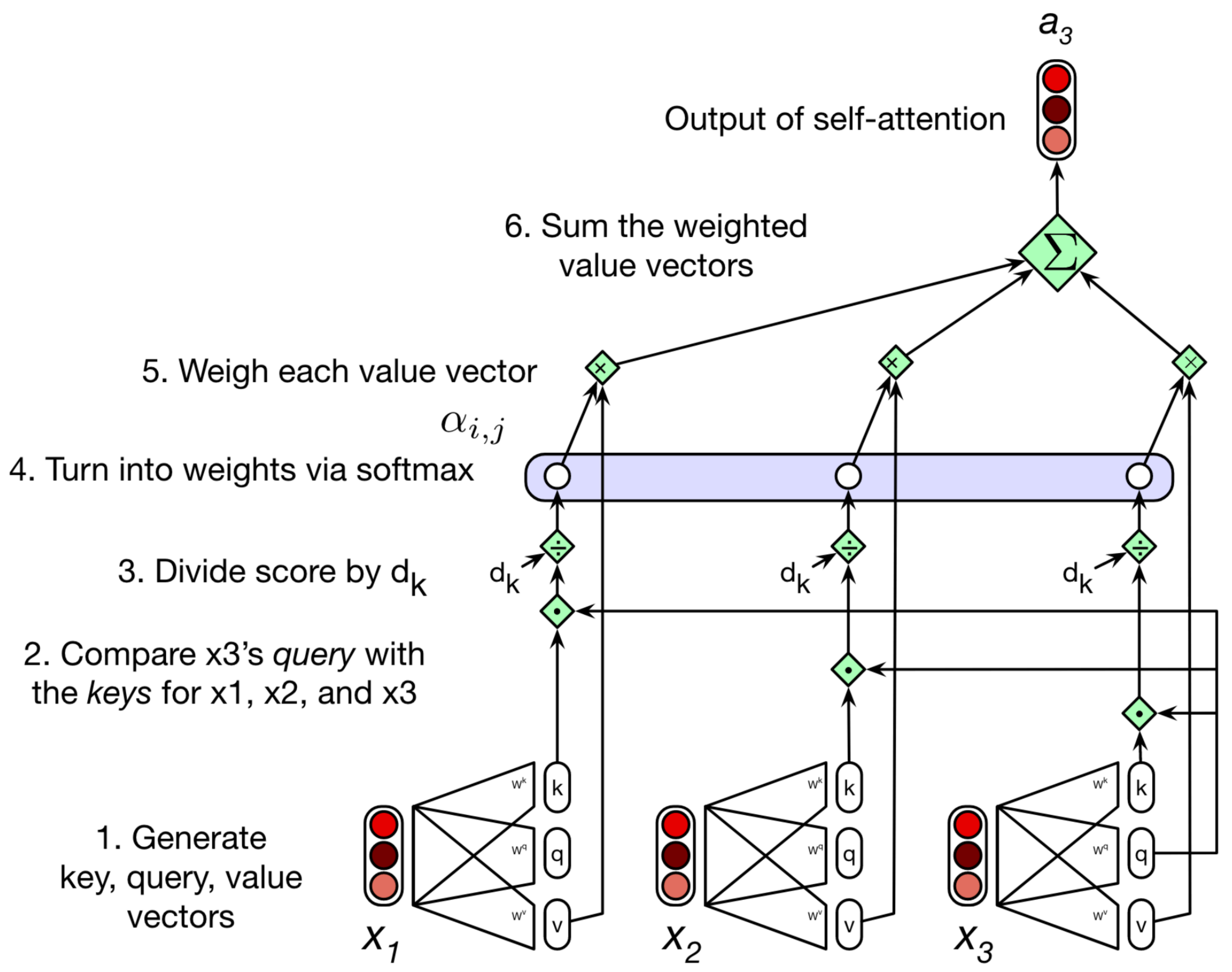
$$a_i = \sum_{j \le i} \alpha_{ij} v_j$$

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

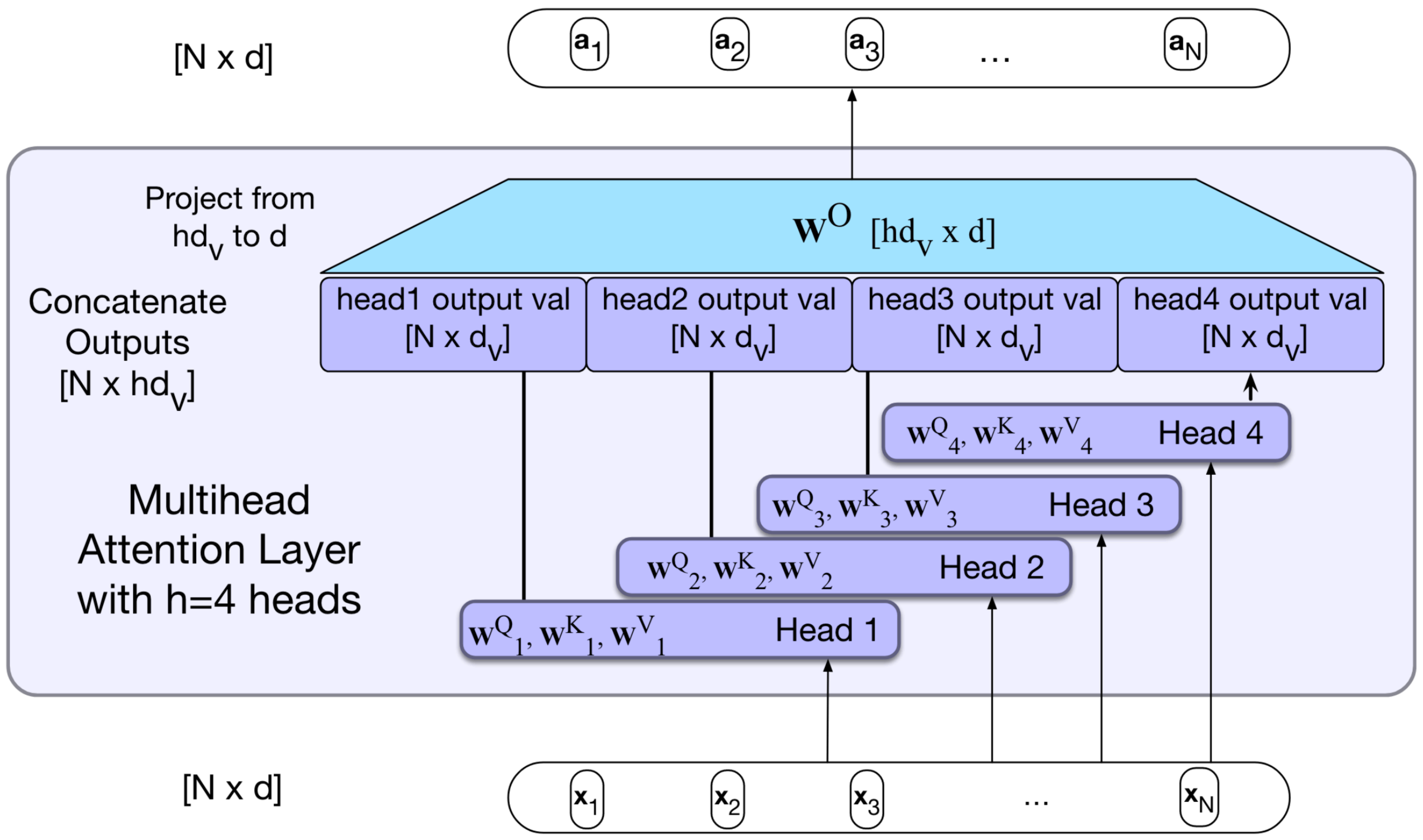
N

The N × N $\mathbf{QK}^\intercal$ matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to −∞, which the softmax will turn to zero).

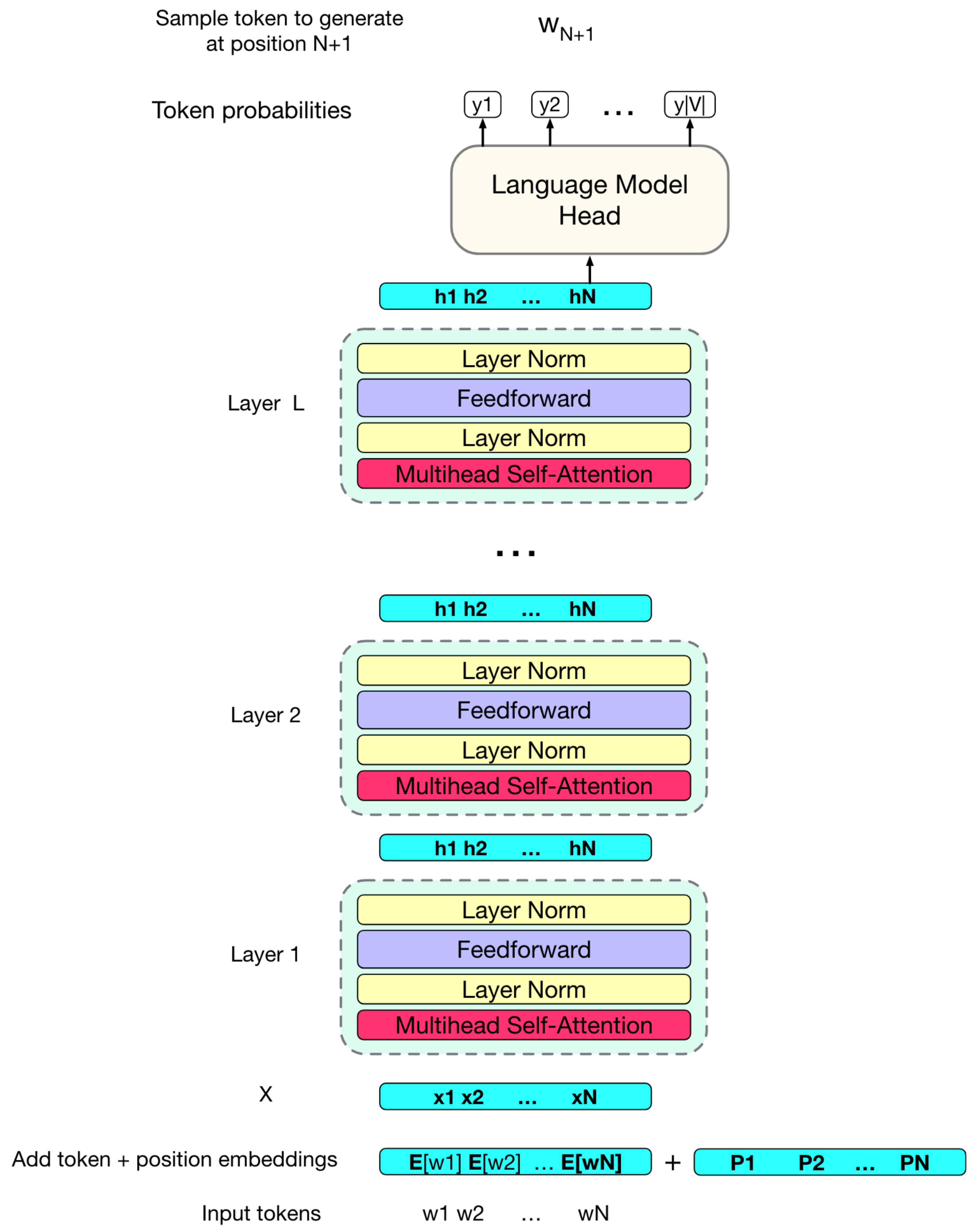

a_3
Output of self-attention
6. Sum the weighted value vectors
5. Weigh each value vector
$\alpha_{i,j}$
4. Turn into weights via softmax
3. Divide score by d_k
d_k
d_k
d_k
2. Compare x3's *query* with the *keys* for x1, x2, and x3
1. Generate key, query, value vectors
k
q
v
x_1
x_2
x_3

Calculating the value of a_3, the third element of a sequence using causal (left-to-right) self-attention.

Language Modeling Head


[N x d]
a_1
a_2
a_3
…
a_N
Project from
hd_V to d
W^O $[hd_V \times d]$
Concatenate
Outputs
$[N \times hd_V]$
head1 output val $[N \times d_V]$
head2 output val $[N \times d_V]$
head3 output val $[N \times d_V]$
head4 output val $[N \times d_V]$
W^Q_4, W^K_4, W^V_4 Head 4
W^Q_3, W^K_3, W^V_3 Head 3
W^Q_2, W^K_2, W^V_2 Head 2
W^Q_1, W^K_1, W^V_1 Head 1
Multihead
Attention Layer
with h=4 heads
[N x d]
x_1
x_2
x_3
…
x_N

Multihead self-attention

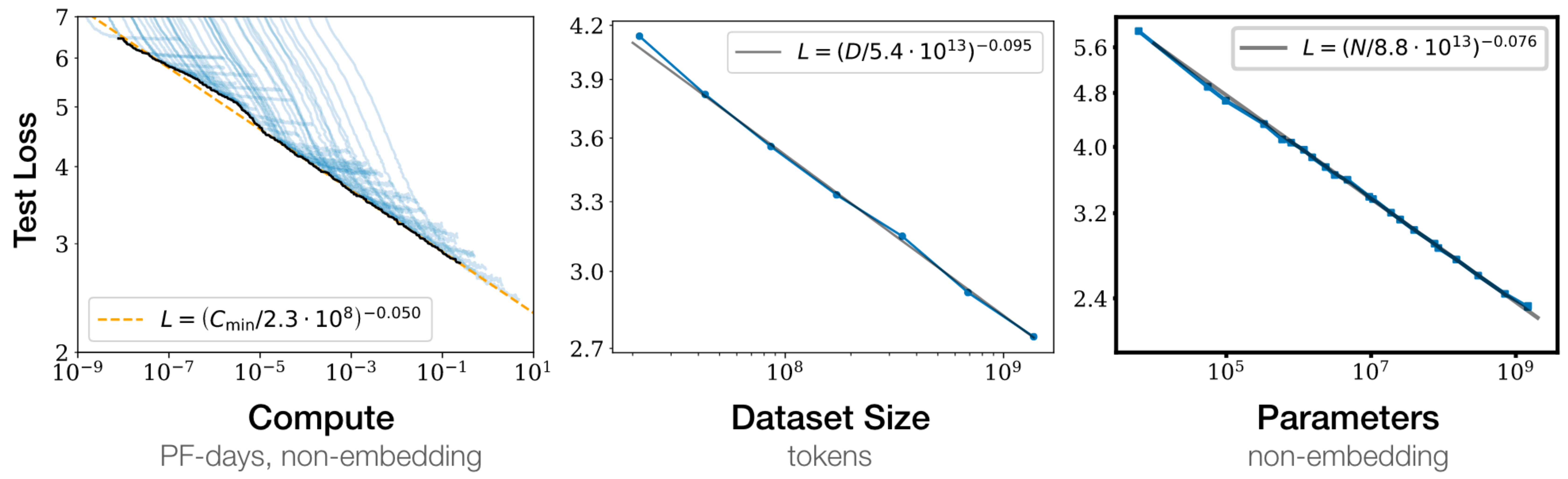


A final transformer decoder-only model, stacking post-norm transformer blocks and mapping from a set of input tokens $\mathbf{w_1}$ to $\mathbf{w_N}$ to a predicted next word $\mathbf{w_{N+1}}$.

Scaling Laws

loss **L** as a function of the number of non-embedding parameters **N**, the dataset size **D**, and the compute budget **C**, for models training with limited parameters, dataset, or compute budget,

$$L(N) = (N_c/N)^{\alpha_N}; L(D) = (D_c/D)^{\alpha_D}; L(N) = (C_c/C)^{\alpha_C}$$



Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute used for training.

- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling Laws for Neural Language Models* (arXiv:2001.08361). arXiv. http://arxiv.org/abs/2001.08361

MatMul Free Language Modeling

Goal: To eliminate MatMul operations from LLMs while maintaining strong performance at billion-parameter scales.

Reason: MatMul operations account for the dominant portion of computational expense, often consuming the majority of the execution time and memory access during both training and inference phases.

But GPUs are optimised for MatMul operations and further models are optimised to run on GPUs. So why even this?

Potential use cases : Running on edge devices or less powerful hardware optimised for simpler operations.

Inspirations : AdderNet (MatMul reformulation) and BitNet (ternary weight quantisation) models, MatMul operations where values are either flipped or zeroed out before accumulation. Quantization can be applied to either activations or weights: spiking neural networks (SNNs) use binarized activations

MatMul-free Dense Layers with Ternary Weights

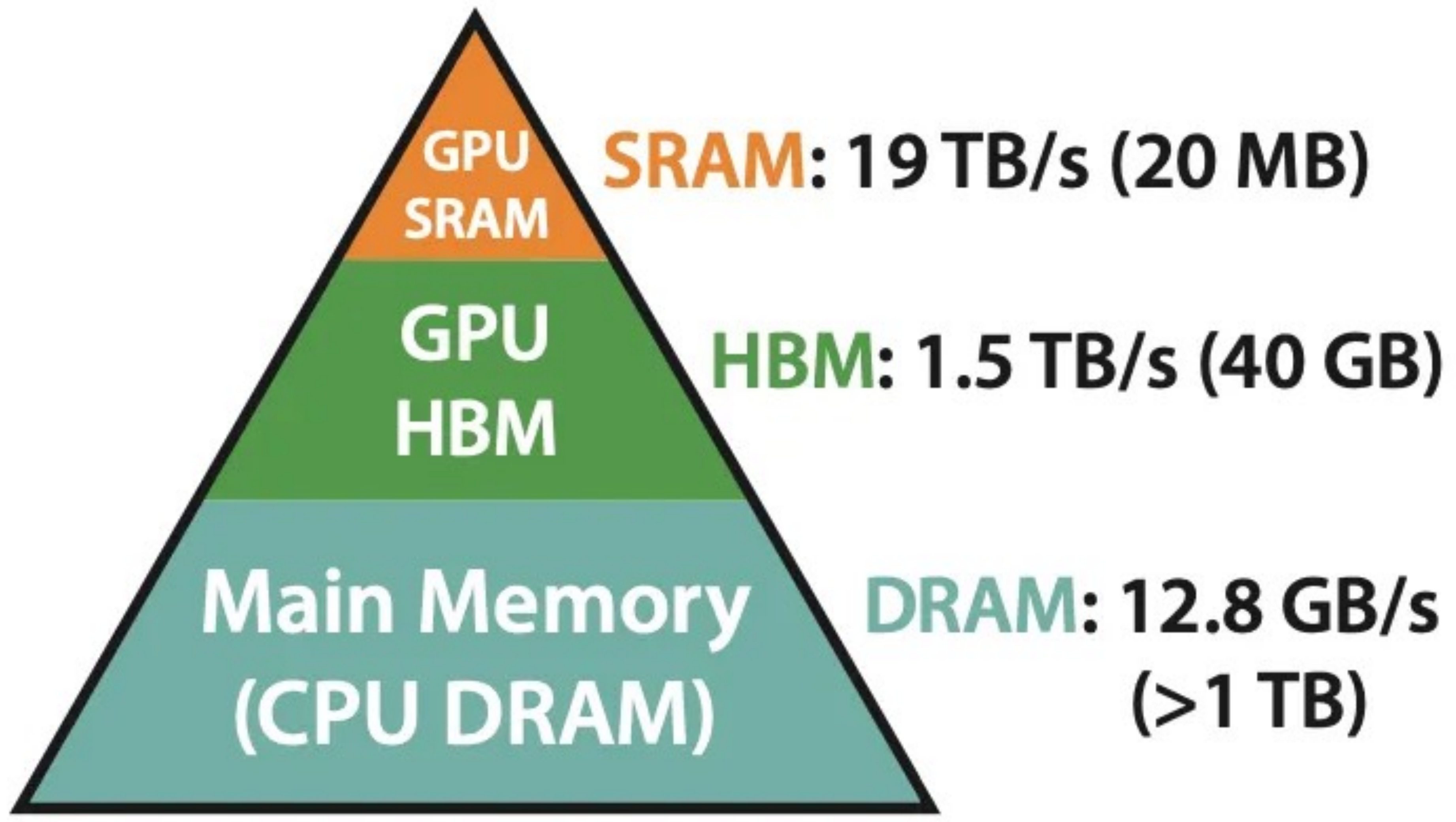
$$y = xW = \sum_{j=1}^{d} x_j W_{ij} \quad \text{for } i = 1,2,\ldots,m \longrightarrow \tilde{Y} = x \odot \tilde{W} = \sum_{j=1}^{d} x_j \tilde{W}_{ij}, \quad \tilde{W}_{ij} \in \{-1, 0, +1\}, \quad \text{for } i = 1,2,\ldots,m$$

$$x_j \tilde{W}_{ij} = \begin{cases} x_j, & \text{if } \tilde{W}_{ij} = 1, \\ 0, & \text{if } \tilde{W}_{ij} = 0, \\ -x_j, & \text{if } \tilde{W}_{ij} = -1. \end{cases}$$

Will attention with ternary Weights work?

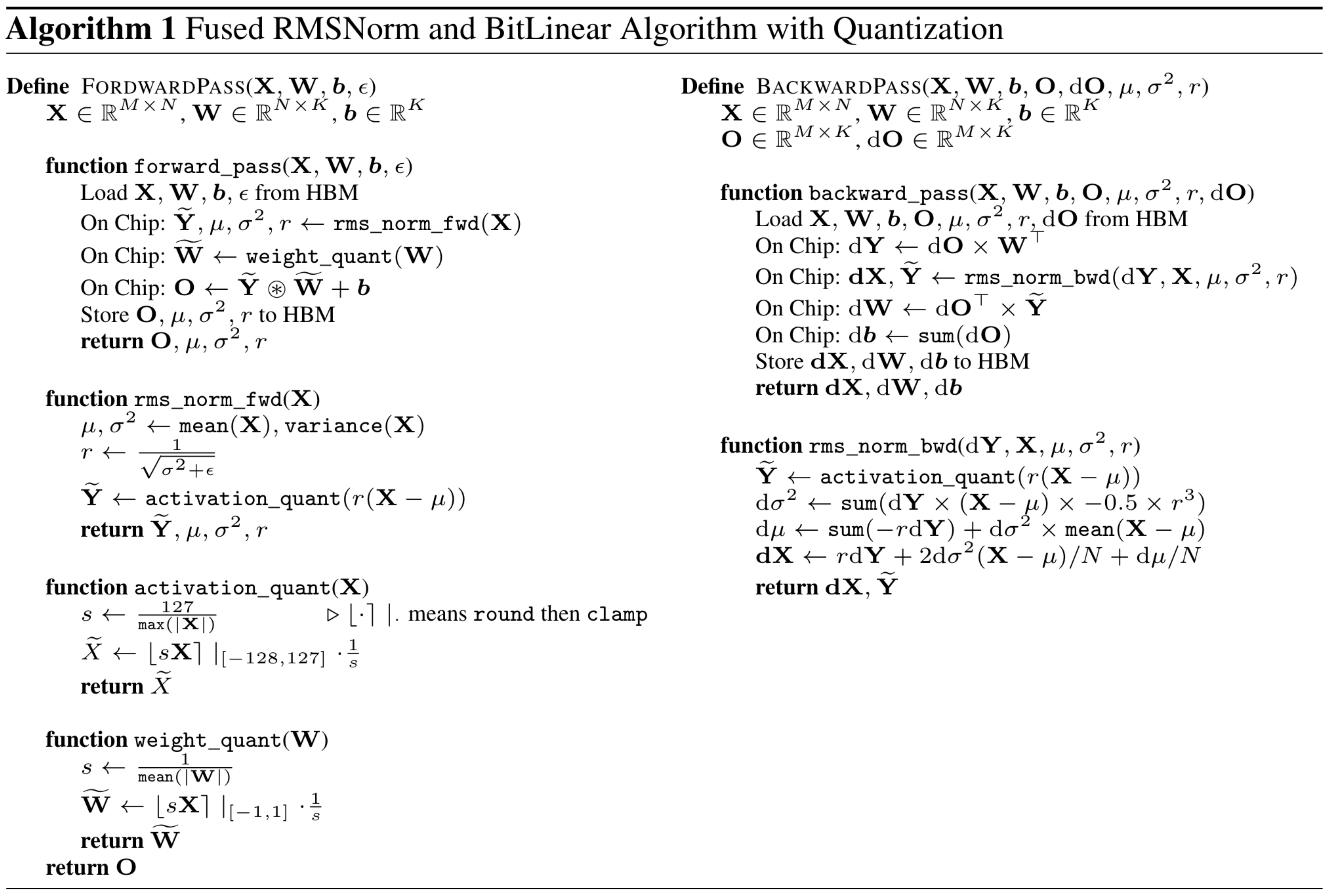
$$\tilde{Y}_i = \sum_{j=1}^{d} x_j \tilde{W}_{ij} = \sum_{j:\tilde{W}_{ij}=1} x_j - \sum_{j:\tilde{W}_{ij}=-1} x_j, \quad \text{for } i = 1,2,\ldots,m$$

Hardware-efficient Fused BitLinear Layer



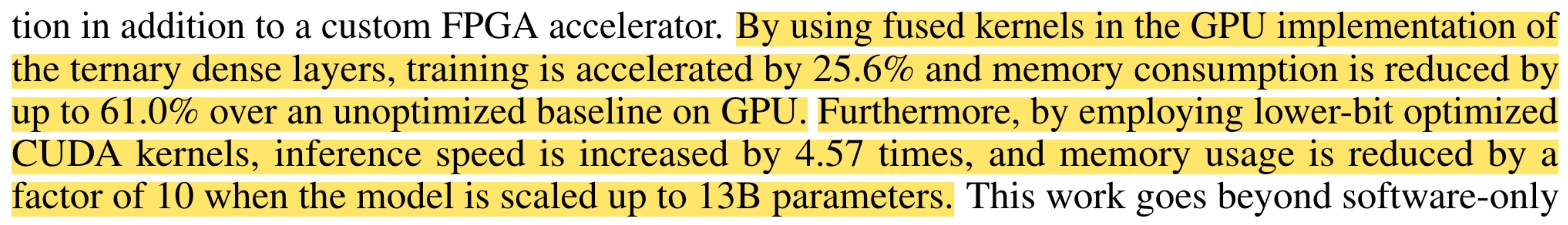
Compute speed of parts of GPUs

On Chip = SRAM



Algorithm 1 Fused RMSNorm and BitLinear Algorithm with Quantization

```
Define FORWARDPASS($\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \epsilon$)
    $\mathbf{X} \in \mathbb{R}^{M \times N}, \mathbf{W} \in \mathbb{R}^{N \times K}, \boldsymbol{b} \in \mathbb{R}^{K}$

    function forward_pass($\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \epsilon$)
        Load $\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \epsilon$ from HBM
        On Chip: $\widetilde{\mathbf{Y}}, \mu, \sigma^2, r \leftarrow$ rms_norm_fwd($\mathbf{X}$)
        On Chip: $\widetilde{\mathbf{W}} \leftarrow$ weight_quant($\mathbf{W}$)
        On Chip: $\mathbf{O} \leftarrow \widetilde{\mathbf{Y}} \circledast \widetilde{\mathbf{W}} + \boldsymbol{b}$
        Store $\mathbf{O}, \mu, \sigma^2, r$ to HBM
        return $\mathbf{O}, \mu, \sigma^2, r$

    function rms_norm_fwd($\mathbf{X}$)
        $\mu, \sigma^2 \leftarrow$ mean($\mathbf{X}$), variance($\mathbf{X}$)
        $r \leftarrow \frac{1}{\sqrt{\sigma^2+\epsilon}}$
        $\widetilde{\mathbf{Y}} \leftarrow$ activation_quant($r(\mathbf{X} - \mu)$)
        return $\widetilde{\mathbf{Y}}, \mu, \sigma^2, r$

    function activation_quant($\mathbf{X}$)
        $s \leftarrow \frac{127}{\max(|\mathbf{X}|)}$            ▷ $\lfloor \cdot \rceil \,|$. means round then clamp
        $\widetilde{X} \leftarrow \lfloor s\mathbf{X} \rceil \,|_{[-128,127]} \cdot \frac{1}{s}$
        return $\widetilde{X}$

    function weight_quant($\mathbf{W}$)
        $s \leftarrow \frac{1}{\mathrm{mean}(|\mathbf{W}|)}$
        $\widetilde{\mathbf{W}} \leftarrow \lfloor s\mathbf{X} \rceil \,|_{[-1,1]} \cdot \frac{1}{s}$
        return $\widetilde{\mathbf{W}}$
    return $\mathbf{O}$

Define BACKWARDPASS($\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \mathbf{O}, \mathrm{d}\mathbf{O}, \mu, \sigma^2, r$)
    $\mathbf{X} \in \mathbb{R}^{M \times N}, \mathbf{W} \in \mathbb{R}^{N \times K}, \boldsymbol{b} \in \mathbb{R}^{K}$
    $\mathbf{O} \in \mathbb{R}^{M \times K}, \mathrm{d}\mathbf{O} \in \mathbb{R}^{M \times K}$

    function backward_pass($\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \mathbf{O}, \mu, \sigma^2, r, \mathrm{d}\mathbf{O}$)
        Load $\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \mathbf{O}, \mu, \sigma^2, r, \mathrm{d}\mathbf{O}$ from HBM
        On Chip: $\mathrm{d}\mathbf{Y} \leftarrow \mathrm{d}\mathbf{O} \times \mathbf{W}^\top$
        On Chip: $\mathbf{dX}, \widetilde{\mathbf{Y}} \leftarrow$ rms_norm_bwd($\mathrm{d}\mathbf{Y}, \mathbf{X}, \mu, \sigma^2, r$)
        On Chip: $\mathrm{d}\mathbf{W} \leftarrow \mathrm{d}\mathbf{O}^\top \times \widetilde{\mathbf{Y}}$
        On Chip: $\mathrm{d}\boldsymbol{b} \leftarrow$ sum($\mathrm{d}\mathbf{O}$)
        Store $\mathbf{dX}, \mathrm{d}\mathbf{W}, \mathrm{d}\boldsymbol{b}$ to HBM
        return $\mathbf{dX}, \mathrm{d}\mathbf{W}, \mathrm{d}\boldsymbol{b}$

    function rms_norm_bwd($\mathrm{d}\mathbf{Y}, \mathbf{X}, \mu, \sigma^2, r$)
        $\widetilde{\mathbf{Y}} \leftarrow$ activation_quant($r(\mathbf{X} - \mu)$)
        $\mathrm{d}\sigma^2 \leftarrow$ sum($\mathrm{d}\mathbf{Y} \times (\mathbf{X} - \mu) \times -0.5 \times r^3$)
        $\mathrm{d}\mu \leftarrow$ sum($-r\mathrm{d}\mathbf{Y}$) $+ \mathrm{d}\sigma^2 \times$ mean($\mathbf{X} - \mu$)
        $\mathbf{dX} \leftarrow r\mathrm{d}\mathbf{Y} + 2\mathrm{d}\sigma^2(\mathbf{X} - \mu)/N + \mathrm{d}\mu/N$
        return $\mathbf{dX}, \widetilde{\mathbf{Y}}$
```

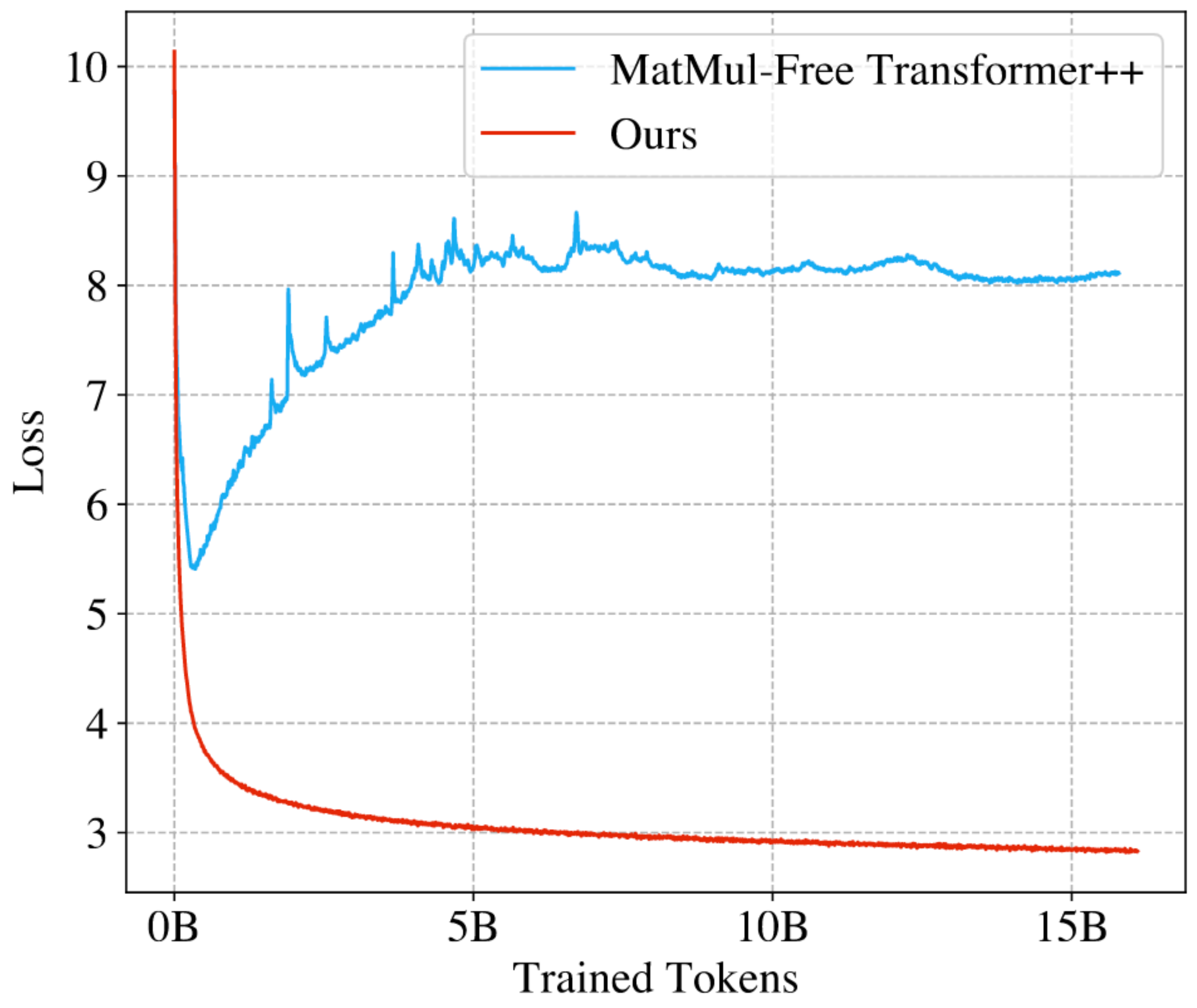


tion in addition to a custom FPGA accelerator. By using fused kernels in the GPU implementation of the ternary dense layers, training is accelerated by 25.6% and memory consumption is reduced by up to 61.0% over an unoptimized baseline on GPU. Furthermore, by employing lower-bit optimized CUDA kernels, inference speed is increased by 4.57 times, and memory usage is reduced by a factor of 10 when the model is scaled up to 13B parameters. This work goes beyond software-only

Ternary weights Self Attention

Losses does not converge. But Why?

Hint: Structure of Attention Matrix.



Training loss over steps

MatMul-free Language Model Architecture

MatMul-free Token Mixer

mixing temporal information

Gated Recurrent Unit

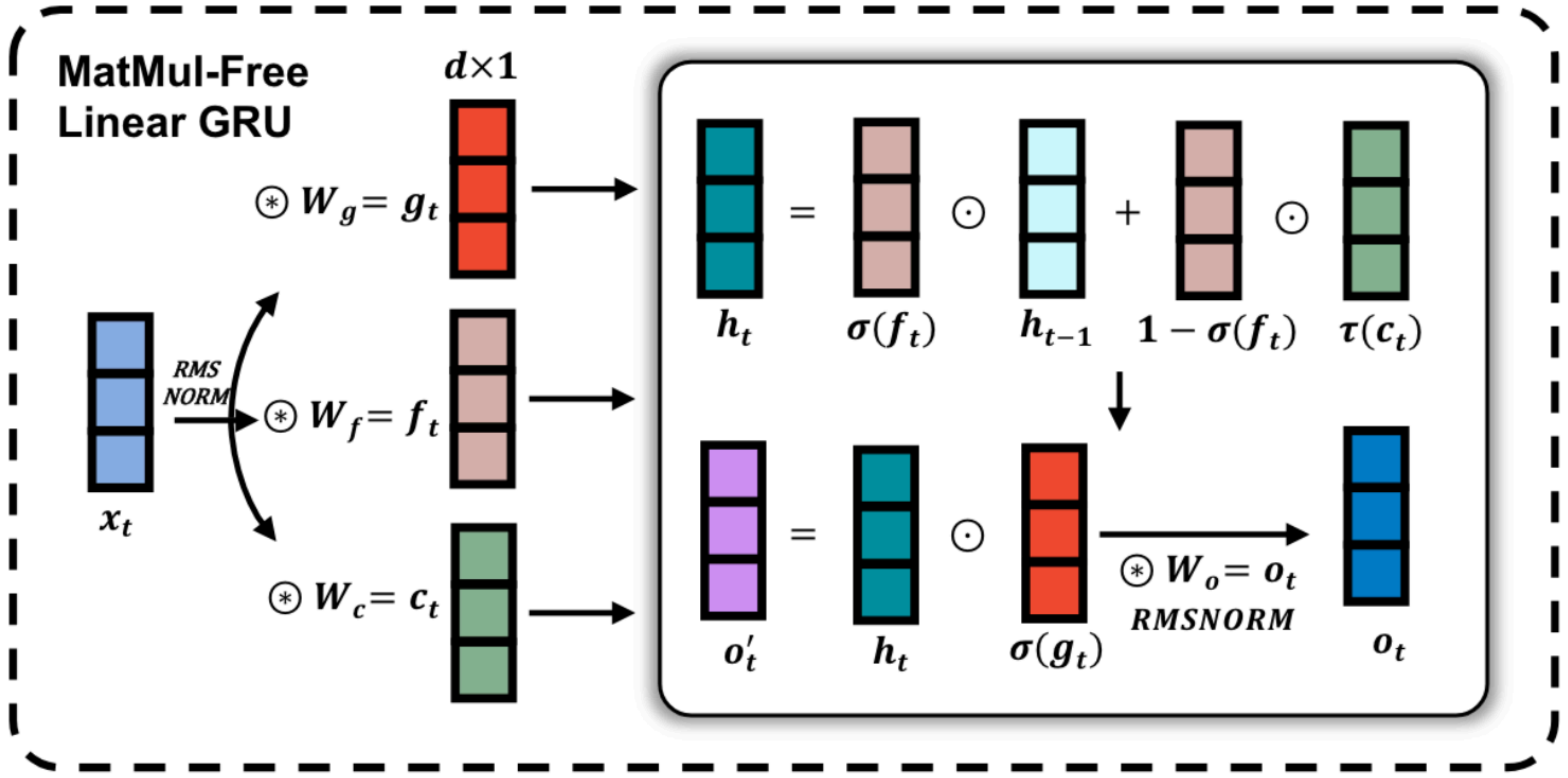
Last hidden state dependent & Non parallelizable

$$
\begin{aligned}
\boldsymbol{r}_t &= \sigma\left(\boldsymbol{x}_t \mathbf{W}_{xr} + \boldsymbol{h}_{t-1}\mathbf{W}_{hr} + \mathbf{b}_r\right) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{f}_t &= \sigma\left(\boldsymbol{x}_t \mathbf{W}_{xf} + \boldsymbol{h}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f\right) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{c}_t &= \tanh\left(\boldsymbol{x}_t \mathbf{W}_{xc} + (\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})\mathbf{W}_{cc} + \mathbf{b}_c\right) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{h}_t &= \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + (1-\boldsymbol{f}_t) \odot \boldsymbol{c}_t \in \mathbb{R}^{1\times d}, \\
\boldsymbol{o}_t &= \boldsymbol{h}_t
\end{aligned}
$$

MatMul-free Linear Gated Recurrent Unit

Last hidden state independent & parallelizable

$$
\begin{aligned}
\boldsymbol{f}_t &= \sigma\left(\boldsymbol{x}_t \circledast \mathbf{W}_f + \mathbf{b}_f\right) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{c}_t &= \tau\left(\boldsymbol{x}_t \circledast \mathbf{W}_c + \mathbf{b}_c\right) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{h}_t &= \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + (1-\boldsymbol{f}_t) \odot \boldsymbol{c}_t \in \mathbb{R}^{1\times d}, \\
\boldsymbol{g}_t &= \sigma(\boldsymbol{x}_t \circledast \mathbf{W}_g + \mathbf{b}_g) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{o}'_t &= \boldsymbol{g}_t \odot \boldsymbol{h}_t \in \mathbb{R}^{1\times d}, \\
\boldsymbol{o}_t &= \boldsymbol{o}'_t \circledast \mathbf{W}_o + \mathbf{b}_o \in \mathbb{R}^{1\times d}.
\end{aligned}
$$



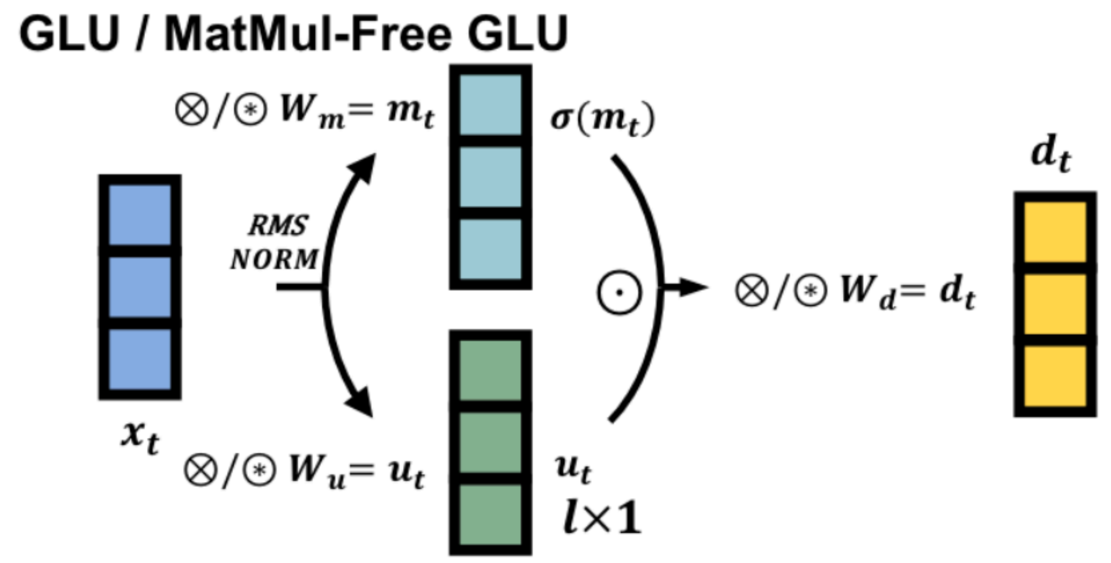
Demo: https://github.com/jessevig/bertviz?tab=readme-ov-file

MatMul-free Language Model Architecture

MatMul-free Channel Mixer

mixing spatial/embedding information

MatMul-free Gated Linear Unit


GLU / MatMul-Free GLU
$\otimes/\circledast\ W_m = m_t$
$\sigma(m_t)$
d_t
RMS NORM
$\odot$
$\otimes/\circledast\ W_d = d_t$
x_t
$\otimes/\circledast\ W_u = u_t$
u_t
$l\times 1$

$$
\begin{aligned}
\boldsymbol{g}_t &= \boldsymbol{x}_t \circledast \boldsymbol{W}_g \in \mathbb{R}^{1\times l},\\
\boldsymbol{u}_t &= \boldsymbol{x}_t \circledast \mathbf{W}_u \in \mathbb{R}^{1\times l},\\
\boldsymbol{p}_t &= \tau(\boldsymbol{g}_t) \odot \boldsymbol{u}_t \in \mathbb{R}^{1\times l},\\
\boldsymbol{d}_t &= \boldsymbol{p}_t \circledast \mathbf{W}_d \in \mathbb{R}^{1\times d},
\end{aligned}
$$

Quantisation

- Quantised Activation Functions
- RSMNorm

Training Details

- ## Surrogate Gradient:

Ternary weights and quantization introduce non-differentiable operations, so the model uses a surrogate gradient method, specifically the straight-through estimator, to enable backpropagation.

- ## Larger Learning Rates:

Ternary weights produce smaller gradients compared to full-precision weights, which can hinder effective weight updates.Thus using larger learning rates helps counteract slow convergence, facilitating faster updates and allowing the model to escape local minima more efficiently.

- ## Learning Rate Scheduler:

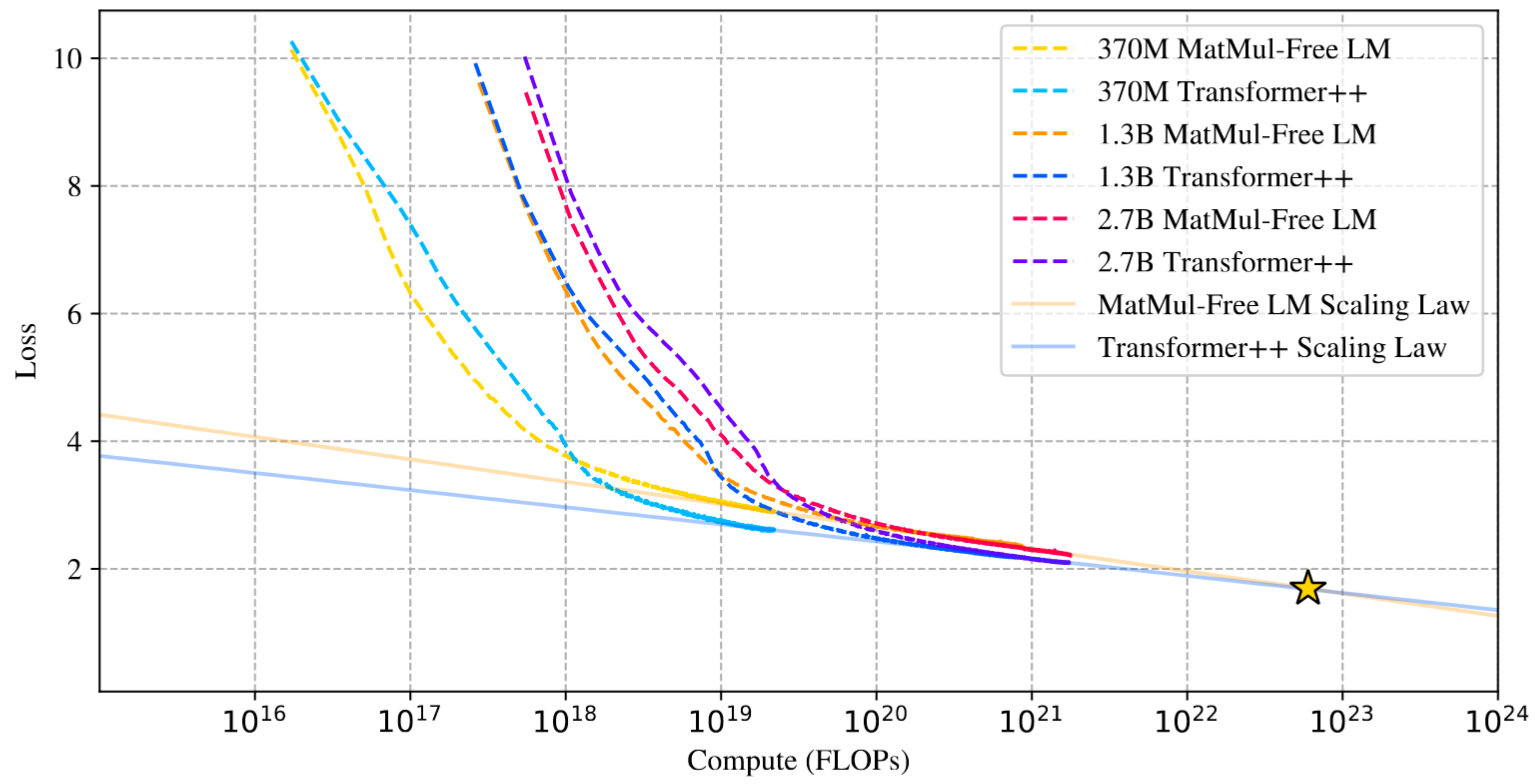
The learning rate is initially maintained using a cosine scheduler and then halved midway through the training process.

- ## Training Efficiency:

When the batch size is 2^8, the training speed of the 1.3B parameter model improves from 1.52s to 1.21s per iteration, a 25.6% speedup over the vanilla implementation. Additionally, memory consumption decreases from 82GB to 32GB, a 61.0% reduction in memory usage.

Experiments

Scaling Law of MatMul-free LM



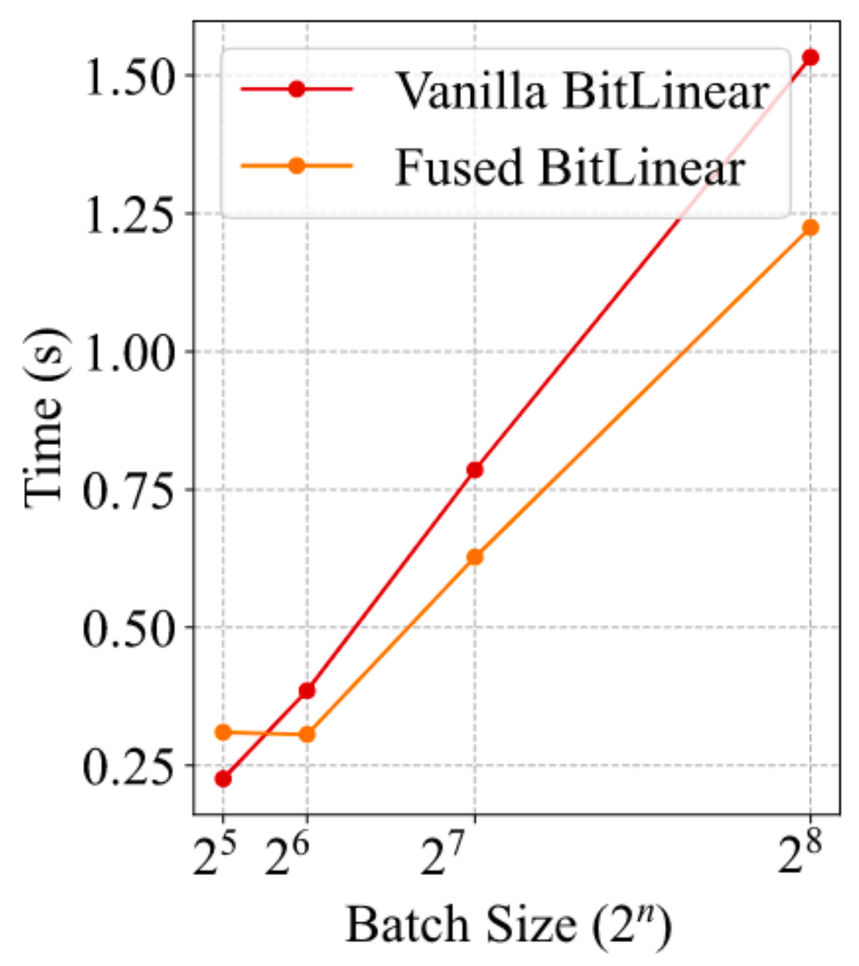
Problems:

- Having only three experiments and then extrapolating for much larger compute.
- Losses can't decrease infinitely, thus optimal compute point may or may come if we hit the ceiling for losses.
- MatMul-Free models decreases steeper but get overtaken by transformer at larger compute.

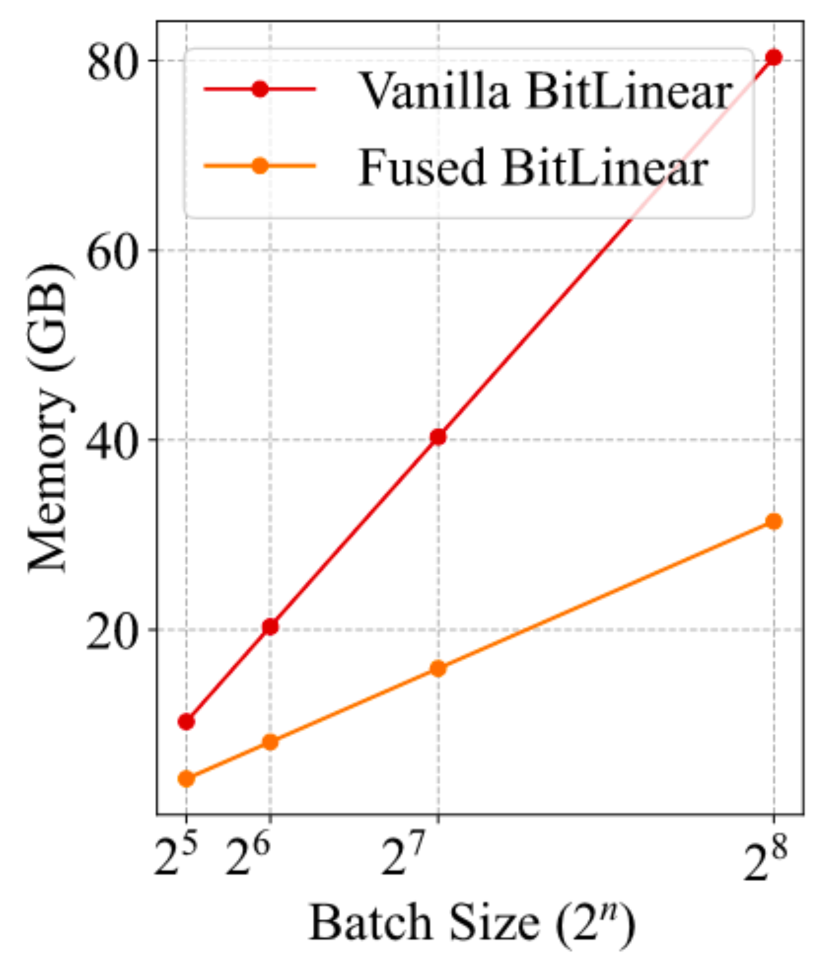
But still close enough to be useful.

Experiments

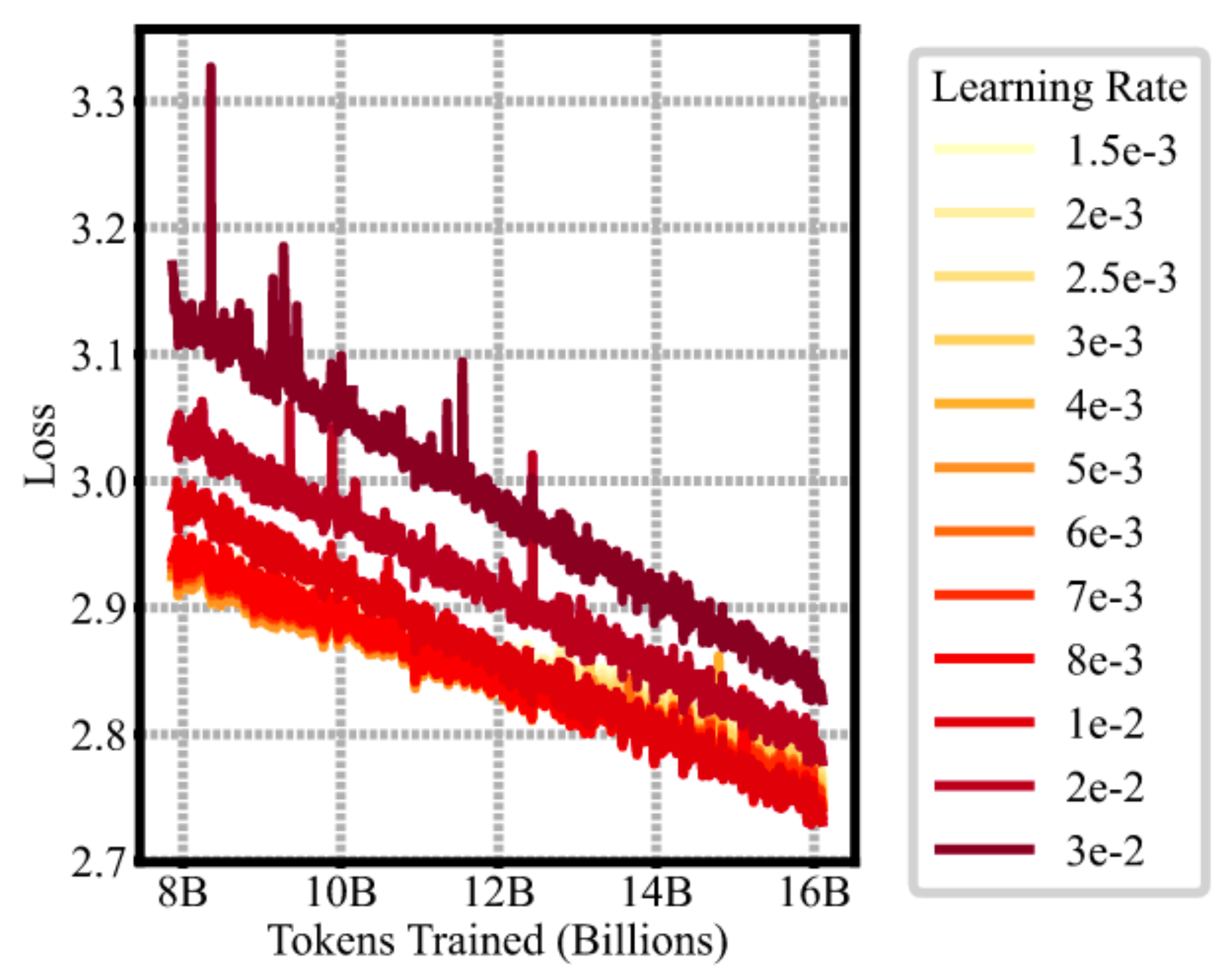
Memory Bottlenecks



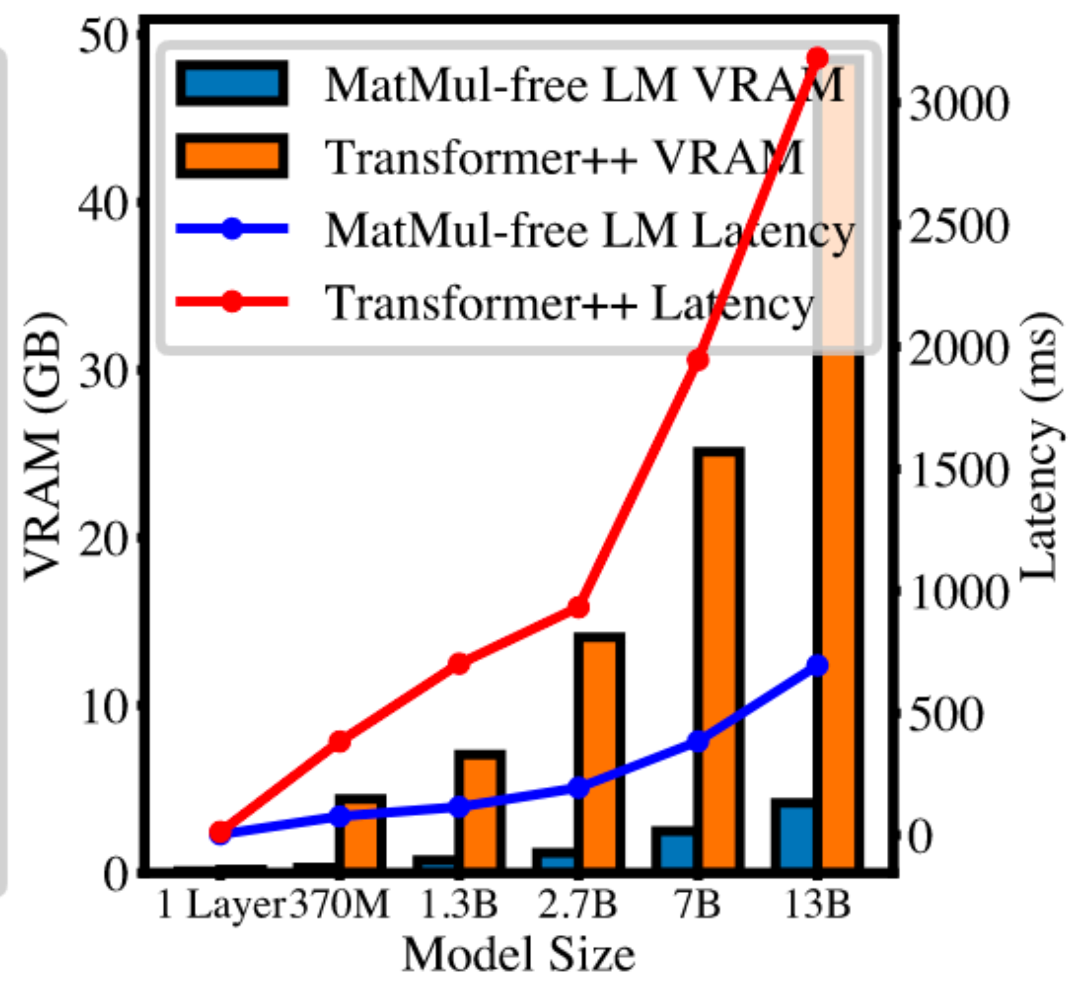
(a) Time vs Batch Size: Fused BitLinear vs. Vanilla BitLinear.



(b) Memory vs Batch Size: Fused BitLinear vs. Vanilla BitLinear.



(c) Effect of learning rate on training loss For MatMul-free LM.



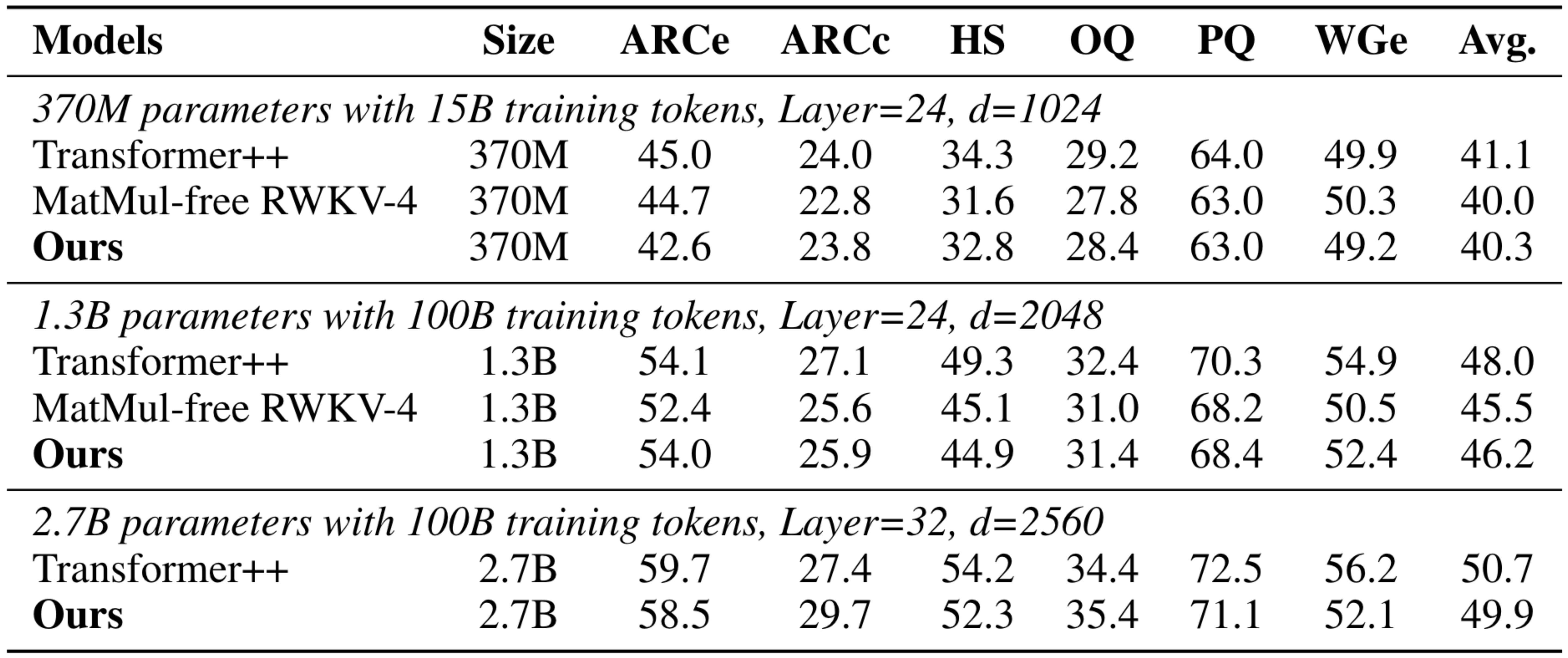
(d) Comparison of GPU Memory and Latency in inference between MatMul-free LM and Transformer++.

This approach combines the operations of RMSNorm and quantization into a single, fused operation executed directly in the GPU's SRAM. By performing these steps in the faster SRAM, the need for multiple data transfers between memory levels is eliminated, significantly reducing overhead.

Experiments

Downstream Tasks

Table 1: Zero-shot accuracy of MatMul-free LM and Transformer++ on benchmark datasets.



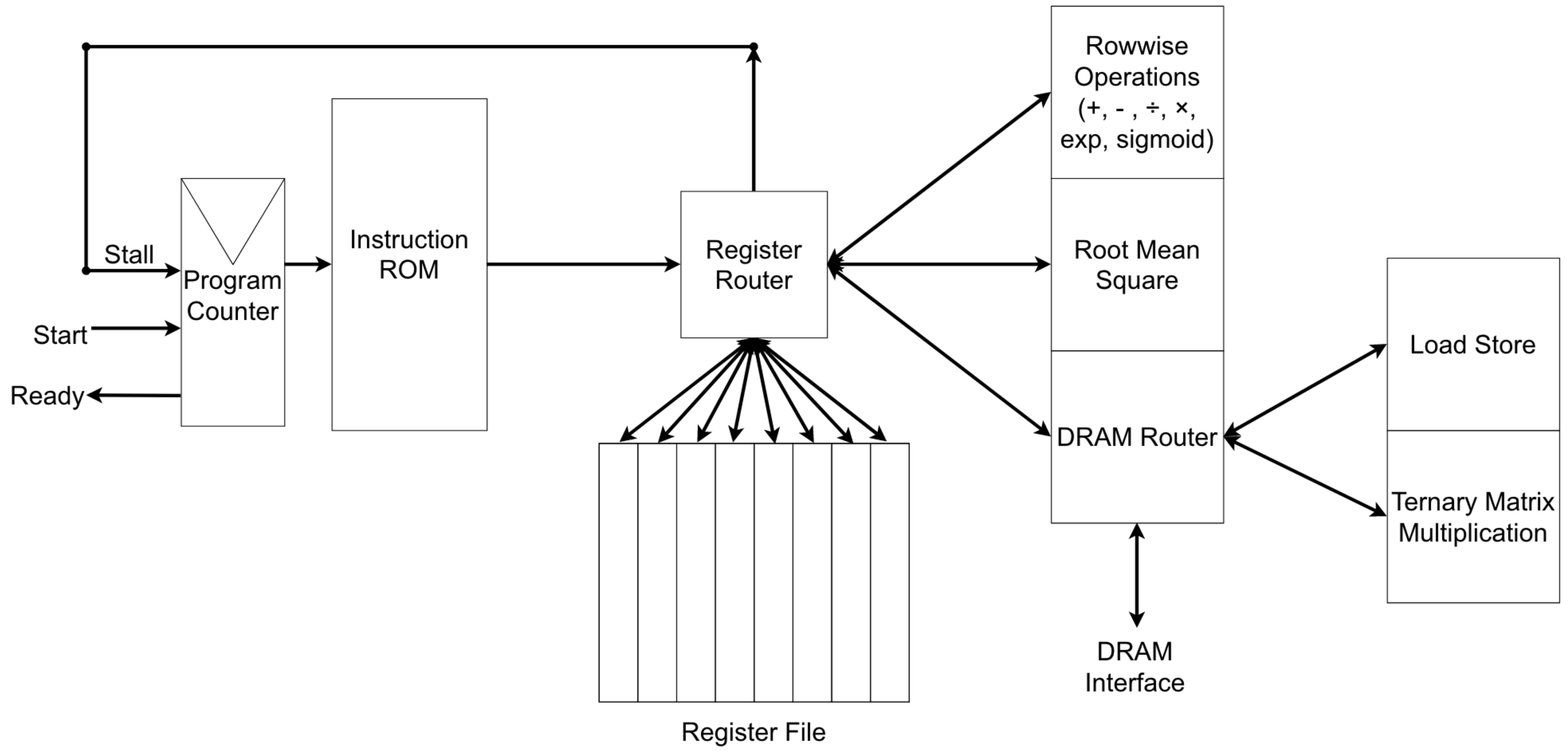
Models	Size	ARCe	ARCc	HS	OQ	PQ	WGe	Avg.
370M parameters with 15B training tokens, Layer=24, d=1024								
Transformer++	370M	45.0	24.0	34.3	29.2	64.0	49.9	41.1
MatMul-free RWKV-4	370M	44.7	22.8	31.6	27.8	63.0	50.3	40.0
Ours	370M	42.6	23.8	32.8	28.4	63.0	49.2	40.3
1.3B parameters with 100B training tokens, Layer=24, d=2048								
Transformer++	1.3B	54.1	27.1	49.3	32.4	70.3	54.9	48.0
MatMul-free RWKV-4	1.3B	52.4	25.6	45.1	31.0	68.2	50.5	45.5
Ours	1.3B	54.0	25.9	44.9	31.4	68.4	52.4	46.2
2.7B parameters with 100B training tokens, Layer=32, d=2560								
Transformer++	2.7B	59.7	27.4	54.2	34.4	72.5	56.2	50.7
Ours	2.7B	58.5	29.7	52.3	35.4	71.1	52.1	49.9

Problems:

- Since the updates are not dependent on current token and not the previous state, the model will struggle to do complex inferences.
- Need results on more complex tasks.

FPGA Implementation

With specialised operation, “Row-wise Operation,” “Root Mean Square,” “Load Store,” and “Ternary Matrix Multiplication,” and they each allow for simple out-of-order execution.



Results not that clear. Requires much more involved knowledge of hardware.

References

- Zhu, R.-J., Zhang, Y., Sifferman, E., Sheaves, T., Wang, Y., Richmond, D., Zhou, P., & Eshraghian, J. K. (2024). *Scalable MatMul-free language modeling*. arXiv. https://arxiv.org/abs/2406.02528
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling Laws for Neural Language Models* (arXiv:2001.08361). arXiv. http://arxiv.org/abs/2001.08361
- Yannic Kilcher. (2024, July 8). *Scalable MaTMUL-free language modeling (Paper explained)* [Video]. YouTube. https://www.youtube.com/watch?v=B45FlSQ8ITo
- Jurafsky, D., & Martin, J. H. (2023). *Speech and language processing* (Draft of February 3, 2024). In *Transformers and large language models* (Chap. 10).

Thank You for Listening!!!