Efficient LLM Inference

Saley Vishal Vivek

PhD Student

Department of CSE, IIT Delhi

LLMs so far

- Transformers, pre-training, zero-shot,……
- Training models on small GPU(s)
  - Adapters
  - LORA
  - Mixed Precision Training
- Training models at scale
  - DDP
  - DeepSpeed

Do we really need efficient LLM inference?

- Complexity – Quadratic due to Multi-head Attention
- LLM sizes have increased rapidly. Not everyone can afford GPUs needed to run larger (and most capable) models.
- GPUs have considerable environmental impact. Achieve similar inference with a less number of GPUs.
- Deployment Concerns – Inference latency, Inferences per second (Throughput), Cost, etc.

- Quantization

Convert LLMs to use simpler data types.

- Pruning

Remove un-important weights from LLMs

- Hardware Aware Optimizations

Code-up LLMs to improve hardware utilization

Simple Idea: Reduce model size to fit the GPU(s)

But maintain the model performance as much as possible!

How?

Quantization

After the training…

FP32 data type

Computers use 4 bytes to represent floating point numbers

23 bits for mantissa

8 bits for exponent

All parameters of a (pre-trained) LLMs are in FP32 (sometimes called full precision).

Credits: https://huggingface.co/blog/hf-bitsand

Can we use a different data type?

Another floating-point representation

10 bits for mantissa

5 bits for exponent

Credits: https://huggingface.co/blog/hf-bitsandb

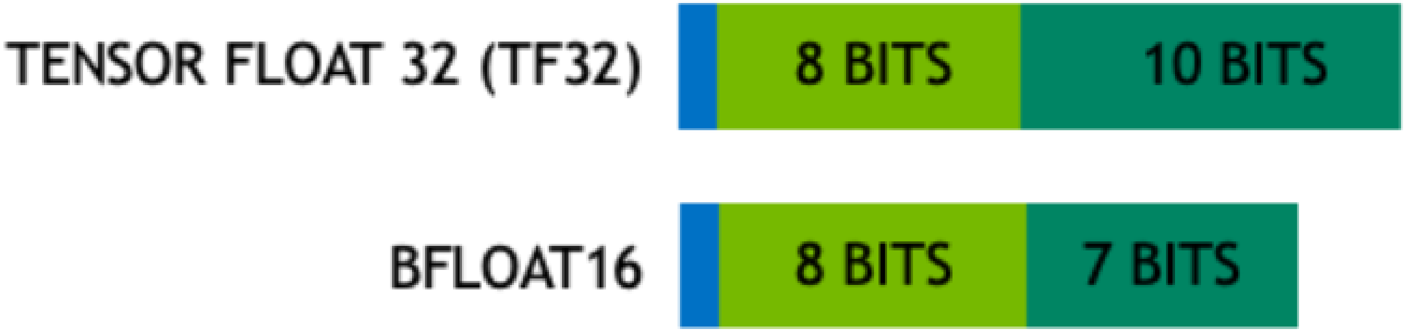
Use FP16 instead for FP32

- Pros
  - Reduced memory usage
  - Faster compute

- Cons
  - Converting all LLM weights may not be straight forward
  - Over/underflows during computation (why?)

Inference is boost is almost 2x.

Other possible data types



BF16 has range similar to FP32 but less precision -> Overflow/underflow handled.
TF32 - nvidia's own TensorFloat 32 alternate to FP32

These data types require hardware support.

Credits: https://huggingface.co/blog/hf-bitsandb

Can we do better?

Int8 Quantization

1. Convert FP16/32 tensors to Int8 tensors
2. Perform Int8 tensor operations
3. Convert results back to FP16/32

But why would this approach provide benefit?

- (Most) C/GPUs can perform integer operations faster than FPs
- Lower memory utilization

Conversion to Int8

Let A be a FP32 matrix where values in A are in range $[-\alpha, \alpha]$

- Quantize

$$a = round(A * s_a)$$

- De-quantize

$$\tilde{A} = \frac{a}{s_a}$$

where s_a is quantization parameter depending on b and α.

Typical values are $s_a = \frac{2^{b-1}-1}{\alpha}$ and $\alpha = \max(abs(A))$.

Credits: https://arxiv.org/pdf/20

Int8 Matrix Multiplication

Computing $Y = XW$ using Int8 quantization.

$$Y = XW$$

$$\approx \tilde{X}\widetilde{W} = \frac{x}{s_x}\frac{w}{s_w} = \frac{1}{s_x s_w}(xw)$$

where (xw) is now an integer matrix multiplication.

Credits: https://arxiv.org/pdf/20

Vector-wise Quantization

Compute $Y_{ij} = X_{i:}W_{:j} \approx \frac{1}{s_{x_{i:}} s_{w_{:j}}} x_{i:} w_{:j}$

Impact of large magnitude is not contained.

Credits: https://arxiv.org/pdf/20

Calibration

Fixing the values of the quantization constants $s_{x_{i:}}, s_{w_{:j}}$. Well, $s_{w_{:j}}$ is no problem. Just take $\max(abs(W_{:j}))$. What about the activations?

- Run the model in FP32 on a calibration dataset.
- Record all the activations from each layer (i.e. $X_{i:}$).
- Decide on $s_{x_{i:}}$ such that the loss of information between $X_{i:}$ and $\tilde{X}_{i:}$ is minimum. Criteria – Entropy, Percentile, etc.

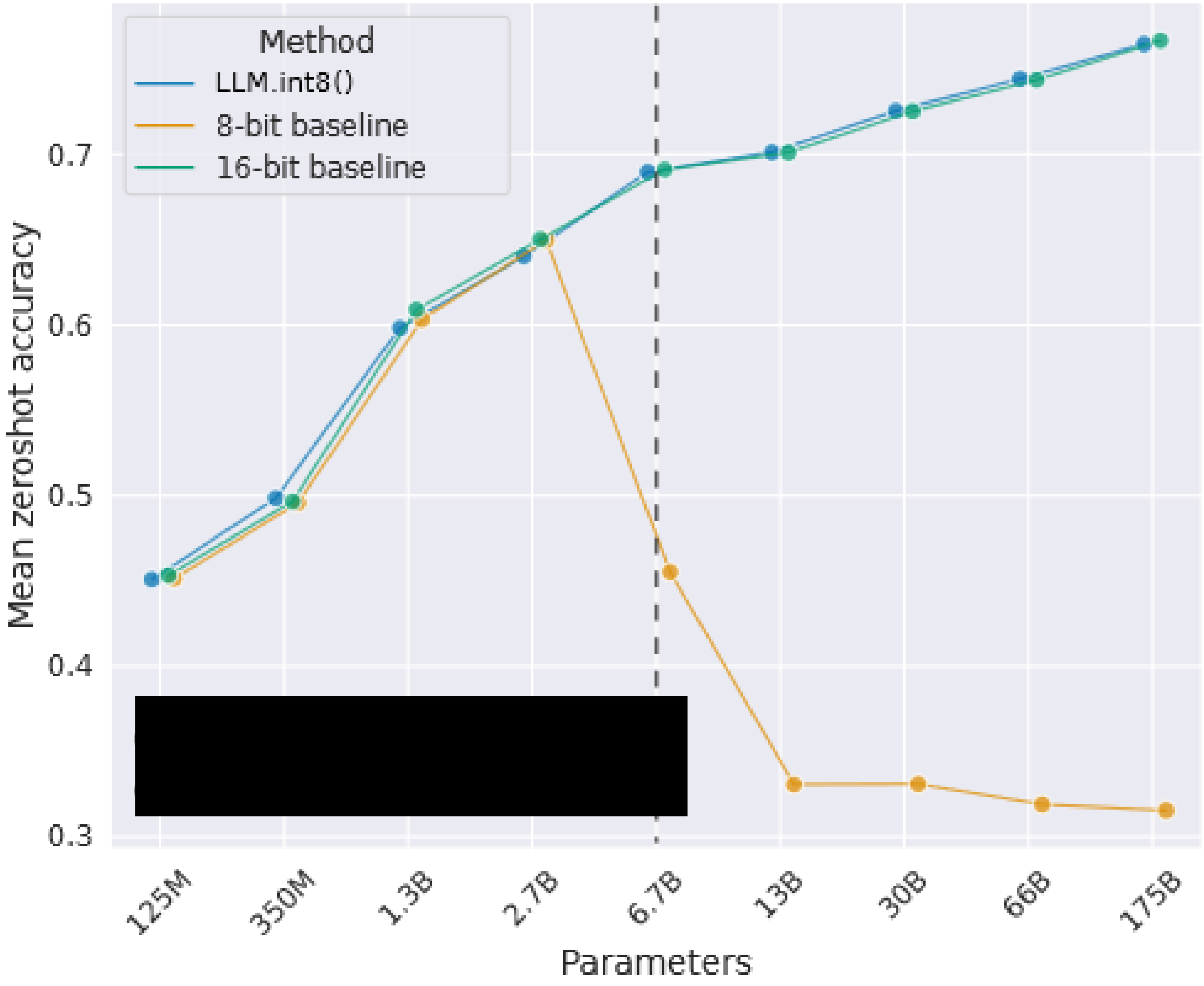
Credits: https://arxiv.org/pdf/20

Performance, Performance…

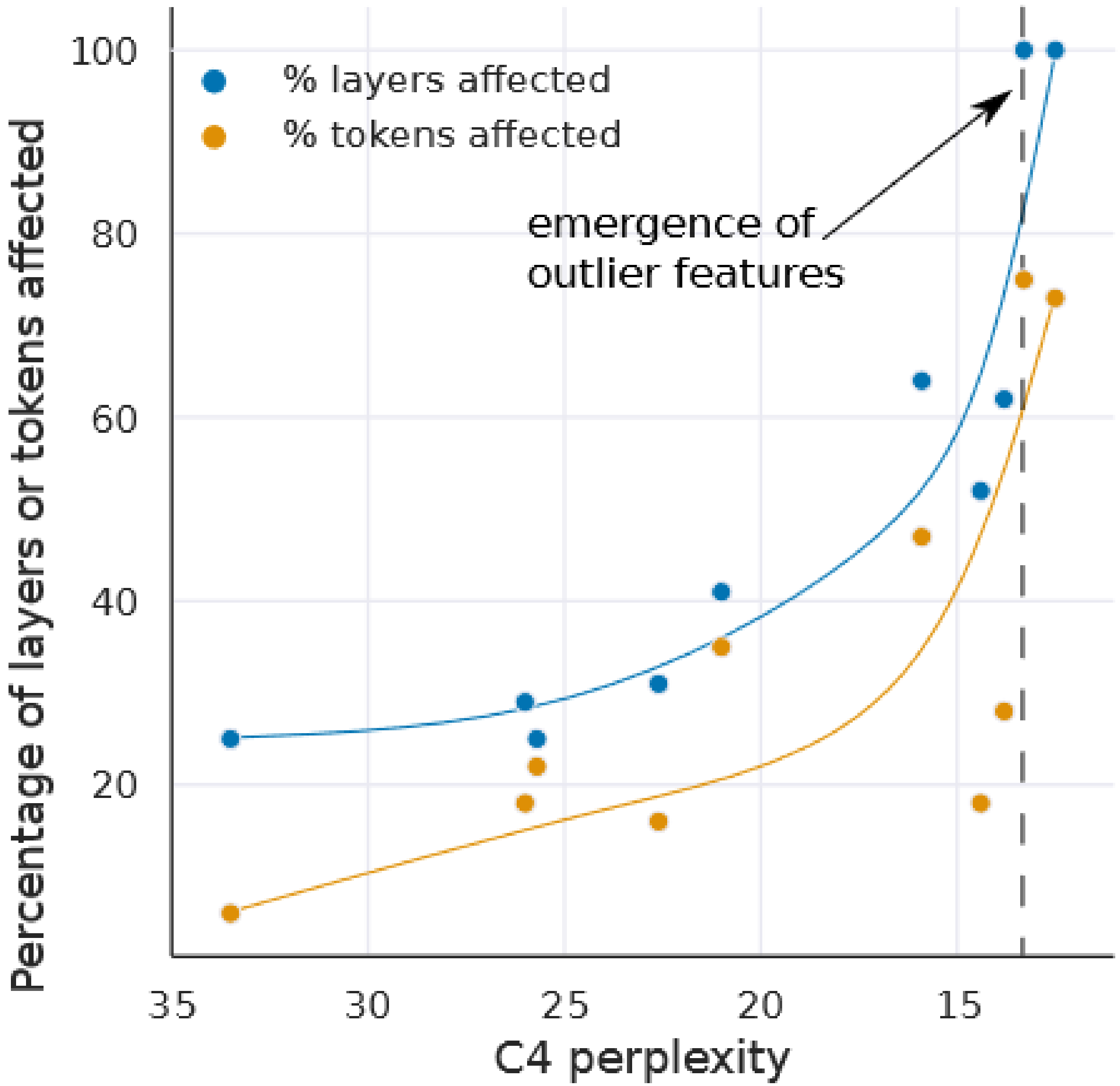
Input Data type	Accumulation Data type	Math Throughput	Bandwidth Reduction
FP32	FP32	1x	1x
FP16	FP16	8x	2x
INT8	INT32	16x	4x
INT4	INT32	32x	8x
INT1	INT32	128x	32x

Credits: https://arxiv.org/pdf/20

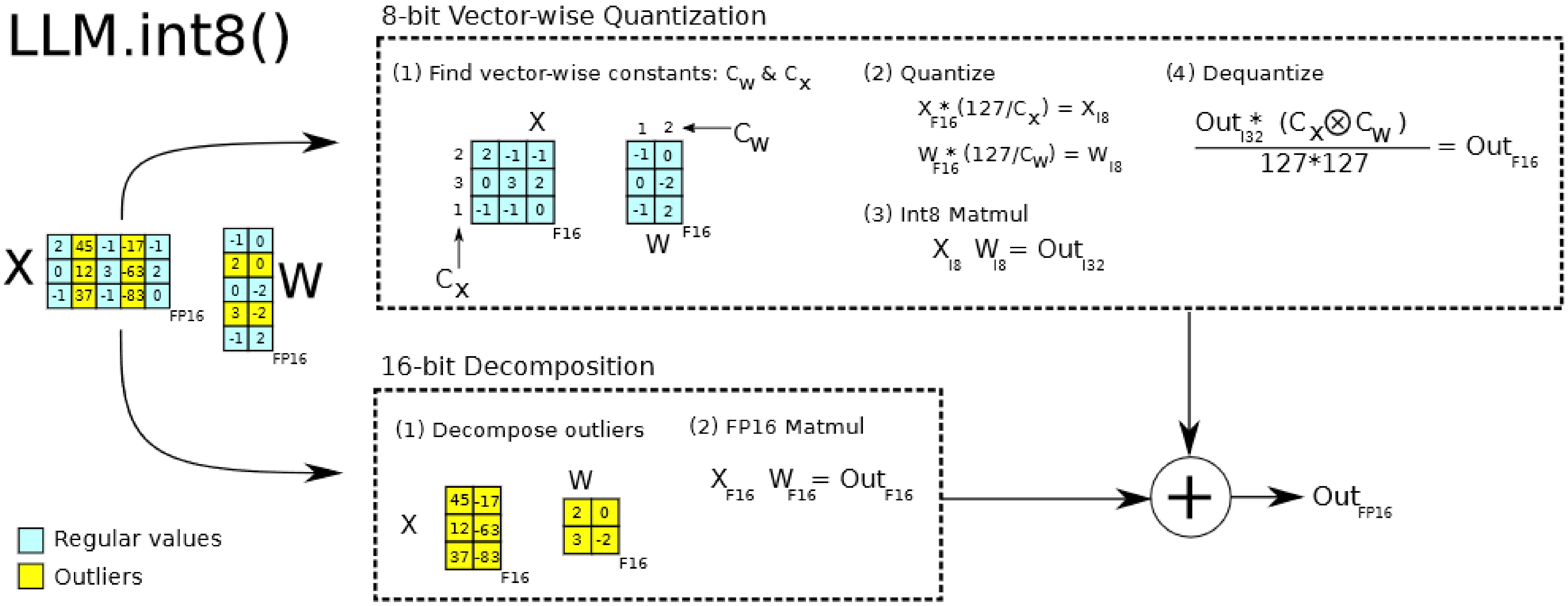
Unfortunately, it does not scale….



Emergence of Outlier Features



LLM.int8()



Credits: https://arxiv.org/pdf/22

Results

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47
Absmax LLM.int8() (vector-wise + decomp)	25.83	15.93	14.44	**13.24**	**12.45**
Zeropoint LLM.int8() (vector-wise + decomp)	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

Perplexity on C4 dataset.
Lower is better.

Pruning

After training…

Idea: Zero-out some weights in LLM

- Some weights in LLMs are important than others
- Select the weights that are important based on a certain criteria (saliency score) and zero-out all other weights.

Constraints – LLM performance is maintained.

We perform pruning post-training (Why?).

But does pruning really provide improved inference speed?

- Well depends on the hardware.

A General Framework

Let w be the neural network weights. And $\mathrm{L}(\mathrm{w})$ be the loss of the model on some calibration data.

We want to find an update $\Delta\mathrm{w}$ such that $\mathrm{L}(\mathrm{w}+\Delta\mathrm{w})$ is close to $L(\mathrm{w})$ and $\mathrm{w}+\Delta\mathrm{w}$ is sparse.

Using second order Taylor expansion

$$|L(\mathrm{w}+\Delta\mathrm{w})-L(\mathrm{w})| \approx \left|\Delta\mathrm{w}^T\nabla L+\frac{1}{2}\Delta\mathrm{w}^T H\Delta\mathrm{w}\right|$$

Here H is second-order derivative (hessian) with respect to parameters. Our goal is to find $\Delta\mathrm{w}$ that minimizes above term and introduces sparsity in w.

Credits: Laurent, César et al. "Revisiting Loss Modelling fo
Unstructured Pruning."

Magnitude Pruning

- Assume that the network has converged. Then $\nabla \mathrm{w}$ is zero.
- Assume that the hessian is identity matrix.

Thus,

$$|L(\mathrm{w}+\Delta \mathrm{w})-L(\mathrm{w})| \approx 0.5 * \left|\sum_k \mathrm{w}_k^2\right|$$

Pruning – To minimize the damage, prune p% weight with small magnitudes.

Credits: Laurent, César et al. "Revisiting Loss Modelling fo
Unstructured Pruning."

Alternate, Use first-order approximation

Then,

$$|L(\mathrm{w} + \Delta\mathrm{w}) - L(\mathrm{w})| \approx |\Delta\mathrm{w}^T \nabla L| \leq \sum_k |\mathrm{w}_k| |\nabla L_k|$$

Intuitively, weights with large magnitude and large gradients are important.

Optimal Brain Surgeon

Assume that network has converged, i.e., gradient is zero.

Let δw be the update such that a weight w_q is pruned by update $w + \delta w$, i.e., $e_q^T \delta w + w_q = 0$. e_q^T is just a basis vector corresponding to dimension q.

Now find δw such that above condition is met and change in the loss is minimized.

$$\delta w = -\frac{w_q}{[H^{-1}]_{qq}} H^{-1} e_q$$

$$L_q = \frac{1}{2} \frac{w_q^2}{[H^{-1}]_{qq}}$$

Updating $w + \delta w$ is called weight reconstruction.

Credits: Hassibi, Babak et al. “Optimal Brain Surgeon and

Why would this not scale for LLMs?

- Benefit Magnitude pruning depends upon hardware support. Loss of performance can be too much at high sparsities.

- Using second order approaches such as OBS do not scale well. Computing (approximate) Hessian matrix is daunting task – large memory and compute.

SparseGPT (Frantar, et.al., 2023)

Localized pruning - Use layer-wise reconstruction loss for pruning

$$L(M, \widetilde{W}) = \left\| WX - (M \odot \widetilde{W})X \right\|_2^2$$

Here, X – inputs (d_{col}, bs), W – weights (d_{row}, d_{col})

M is sparse mask and $\widetilde{W}$ is reconstructed weights.

Note that $\mathrm{W} + \delta W = M \odot \widetilde{W}$ in this case.

We need to find both $M, \widetilde{W}$.

SparseGPT solution

Let mask M is known. Then, we have closed form solution f

$$\mathbf{w^i}_{\mathbf{M_i}} = (\mathbf{X}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}}^\top)^{-1}\mathbf{X}_{\mathbf{M_i}}(\mathbf{w}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}})^\top$$

where, i is a row number

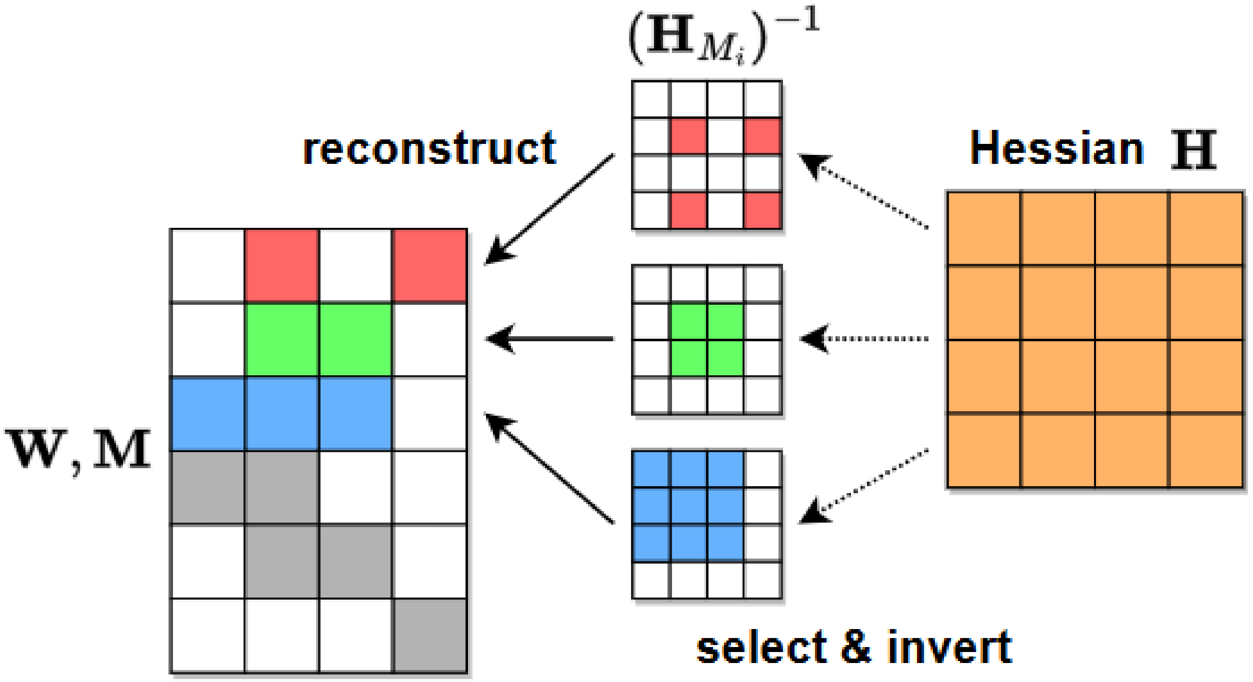
M_i are columns in row i that are not pruned

X_{M_i} is input matrix with columns in $\mathrm{M_i}$

$X_{M_i}X_{M_i}^T$ is the hessian.

Different Row-Hessian Challenge

Each row requires inverting differ sections of the hessian depending upon the masks. Computationally expensive

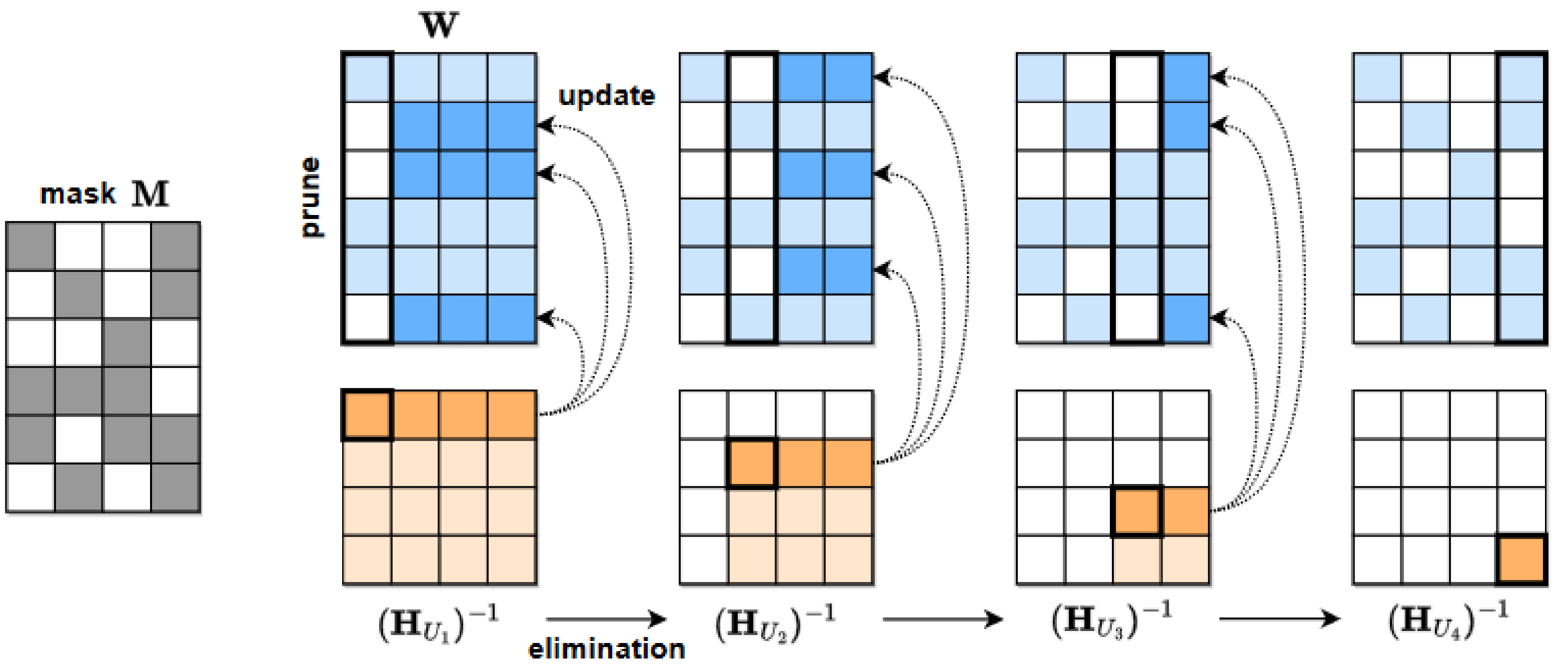


Credits: Frantar,

Optimal Partial Updates

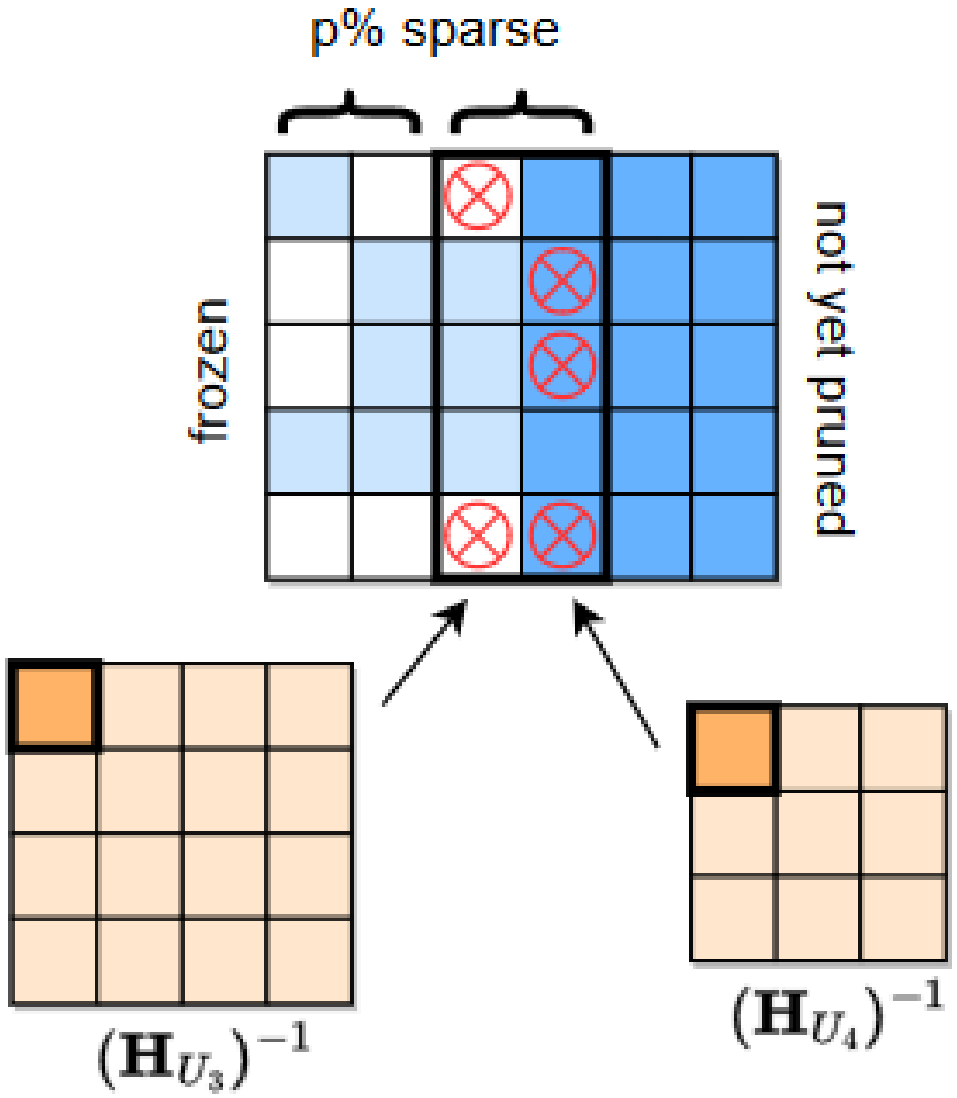
Only update the columns that are not pruned yet. This is approximation of true updates.

Advantages – Hessian computation becomes easier.



Selecting the Mask

- Use Magnitude pruning as the mask. Use SparseGPT for reconstruction. Is it okay?
- But, what about outlier features?

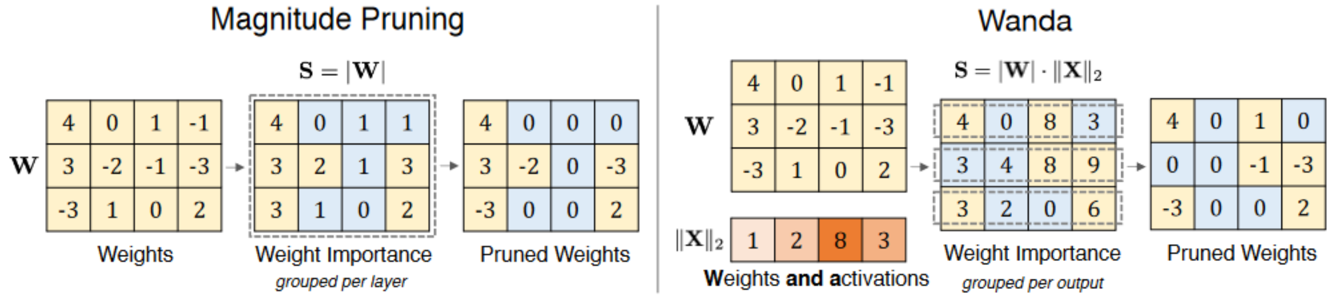


Wanda

Solving reconstruction problem (as in SparseGPT) is still expensive.

Can we get rid of reconstruction all together?

Enter Pruning by **W**eights **and A**ctivation (Wanda)



Credits:

Approach

1. Use a simple saliency metric $S_{ij} = \left|W_{ij}\right| \left\|X_j\right\|_2$

   Rationale – In LLMs, high magnitude features emerge because of inputs and weights combined.

2. Prune at each neuron level – Output of each neuron is sum of all its weight.

Comparison

Method	Weight Update	Calibration Data	Pruning Metric $\mathbf{S}_{ij}$	Complexity
Magnitude	✗	✗	$\lvert \mathbf{W}_{ij} \rvert$	$O(1)$
SparseGPT	✓	✓	$\left[\lvert \mathbf{W} \rvert^2 / \mathrm{diag}\left[(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1} \right] \right]_{ij}$	$O(d^3_{\mathtt{hidden}})$
Wanda	✗	✓	$\lvert \mathbf{W}_{ij} \rvert \cdot \lVert \mathbf{X}_j \rVert_2$	$O(d^2_{\mathtt{hidden}})$

Credits:

Speedups

Weight	Q/K/V/Out	FC1	FC2
Dense	2.84ms	10.26ms	10.23ms
2:4 Sparse	1.59ms	6.15ms	6.64ms
Speedup	1.79×	1.67×	1.54×

SparseGPT
Credits:
https://proceedings.mlr.press/v202/frantar23a/frantar23a.pdf

LLaMA Layer	Dense	2:4	Speedup
q/k/v/o_proj	3.49	2.14	1.63×
up/gate_proj	9.82	6.10	1.61×
down_proj	9.92	6.45	1.54×

Wanda
Credits:
https://arxiv.org/pdf/2306.11695.pdf

Performance, Performance

Method	Weight Update	Sparsity	LLaMA 7B	LLaMA 13B	LLaMA 30B	LLaMA 65B	LLaMA-2 7B	LLaMA-2 13B	LLaMA-2 70B
Dense	-	0%	5.68	5.09	4.77	3.56	5.12	4.57	3.12
Magnitude	✗	50%	17.29	20.21	7.54	5.90	14.89	6.37	4.98
SparseGPT	✓	50%	**7.22**	6.21	5.31	**4.57**	6.51	5.63	**3.98**
Wanda	✗	50%	7.26	**6.15**	**5.24**	**4.57**	**6.42**	**5.56**	**3.98**
Magnitude	✗	4:8	16.84	13.84	7.62	6.36	16.48	6.76	5.54
SparseGPT	✓	4:8	8.61	**7.40**	6.17	5.38	8.12	6.60	4.59
Wanda	✗	4:8	**8.57**	**7.40**	**5.97**	**5.30**	**7.97**	**6.55**	**4.47**
Magnitude	✗	2:4	42.13	18.37	9.10	7.11	54.59	8.33	6.33
SparseGPT	✓	2:4	**11.00**	**9.11**	7.16	6.28	**10.17**	8.32	5.40
Wanda	✗	2:4	11.53	9.58	**6.90**	**6.25**	11.02	**8.27**	**5.16**

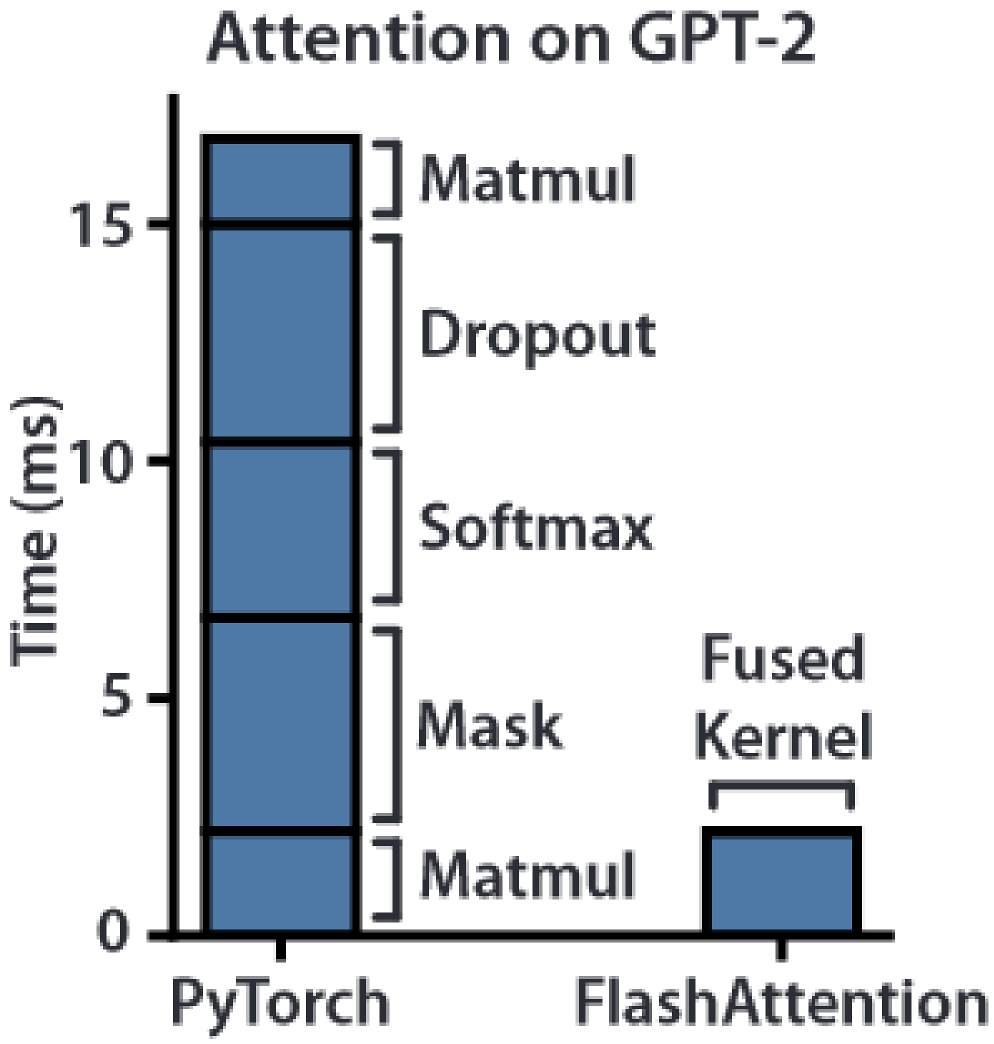
Table 3: WikiText perplexity of pruned LLaMA and LLaMA-2 models. Wanda performs competitively against prior best method SparseGPT, without introducing any weight update.

Credits:

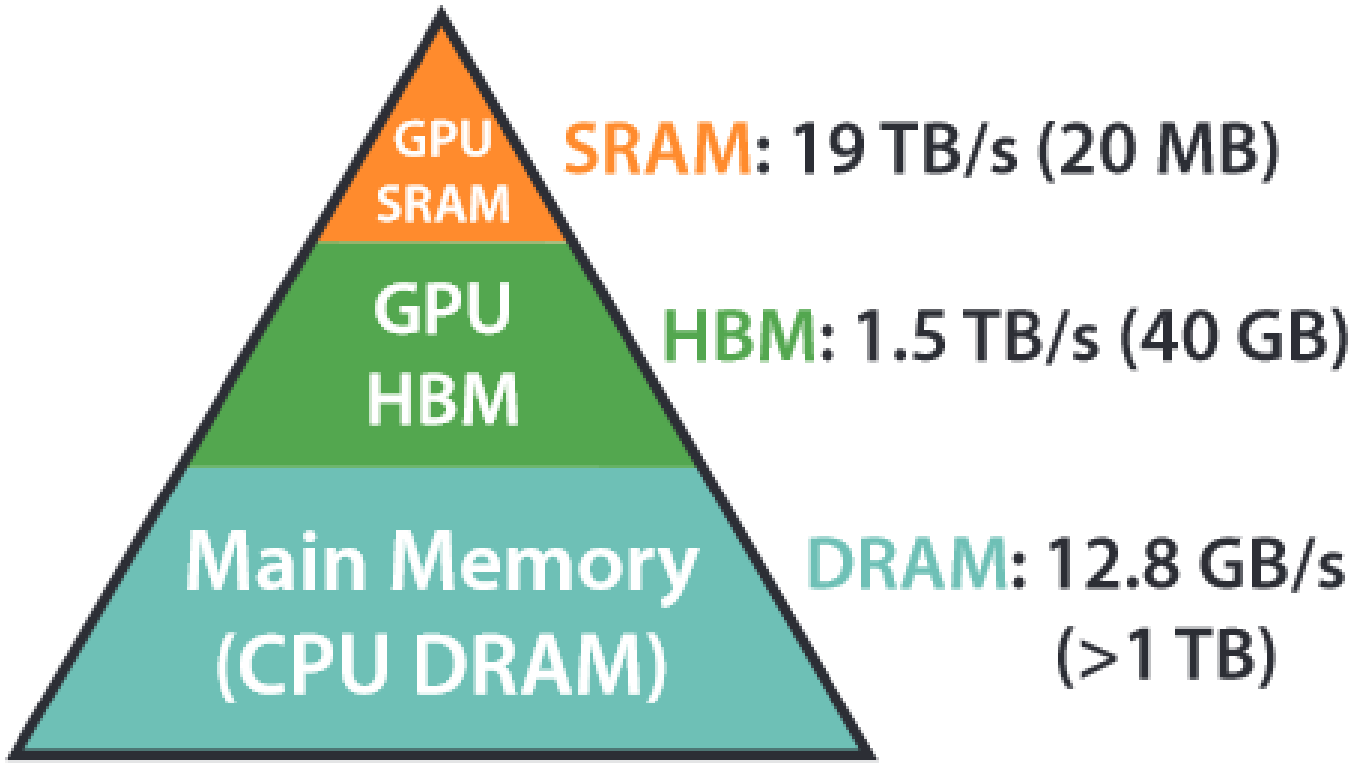
Flash Attention

Making attention GPU aware
Credits: https://arxiv.org/pdf/2205.14135.pdf

Operation-wise split-up



GPU Memory Hierarchy



Memory Hierarchy with Bandwidth & Memory Size

Standard Attention Mechanism

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Tiling - Perform SoftMax in blocks

Regular SoftMax

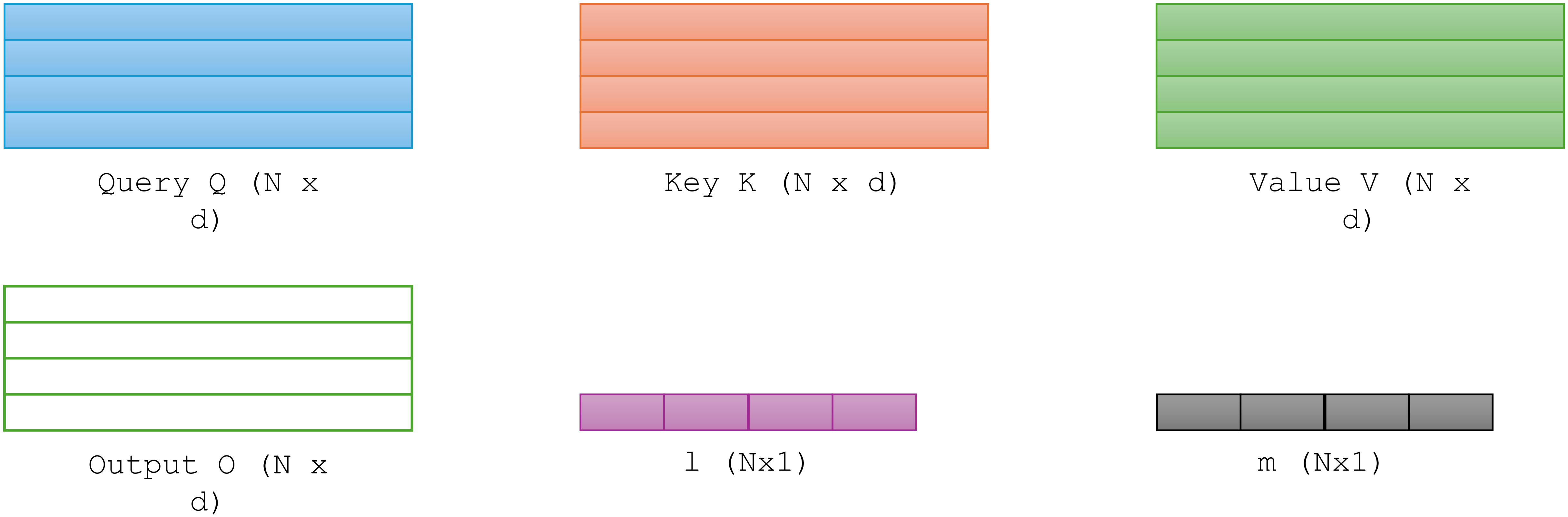
$$m(x) := \max_i \; x_i, \quad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \dots & e^{x_B - m(x)} \end{bmatrix}, \quad \ell(x) := \sum_i f(x)_i, \quad \mathrm{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

SoftMax over concatenation of two vectors

$$m(x) = m(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \begin{bmatrix} e^{m(x^{(1)}) - m(x)} f(x^{(1)}) & e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \end{bmatrix},$$

$$\ell(x) = \ell(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \mathrm{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

Let HBM is large enough to store Q, K, V
Let SRAM size be M.
Divide Q, K, V into blocks of size (B x d) as shown below.
Here $B = \lceil {}^{M}/_{4d} \rceil$.
Initialise tensors $O\,(N\,x\,d),\ l(0)_N\ and\ l(-inf)_N$
Also divide tensors into B blocks



Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.

Initialize $\mathbf{O} = (0)_{N\times d} \in \mathbb{R}^{N\times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.

Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \dots, m_{T_r}$ of size B_r each.

for $1 \le j \le T_c$ **do**

  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

  for $1 \le i \le T_r$ **do**

    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.

    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r\times B_c}$.

    On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r\times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

    On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.

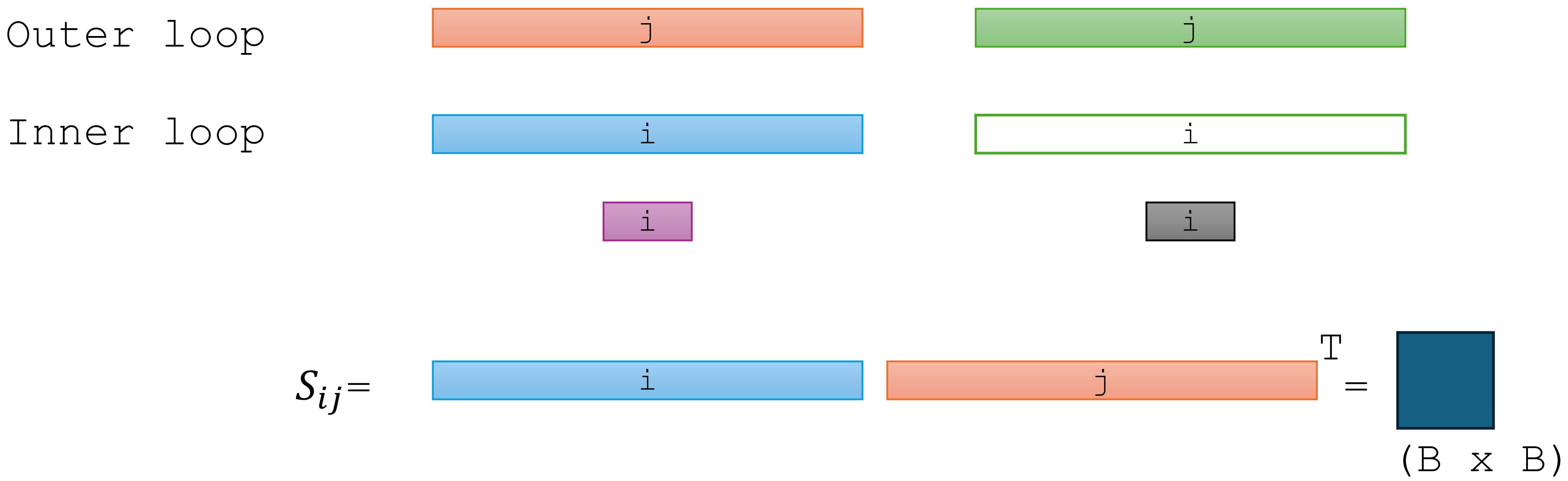
    Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.

    Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.

  end for

end for

Return $\mathbf{O}$.



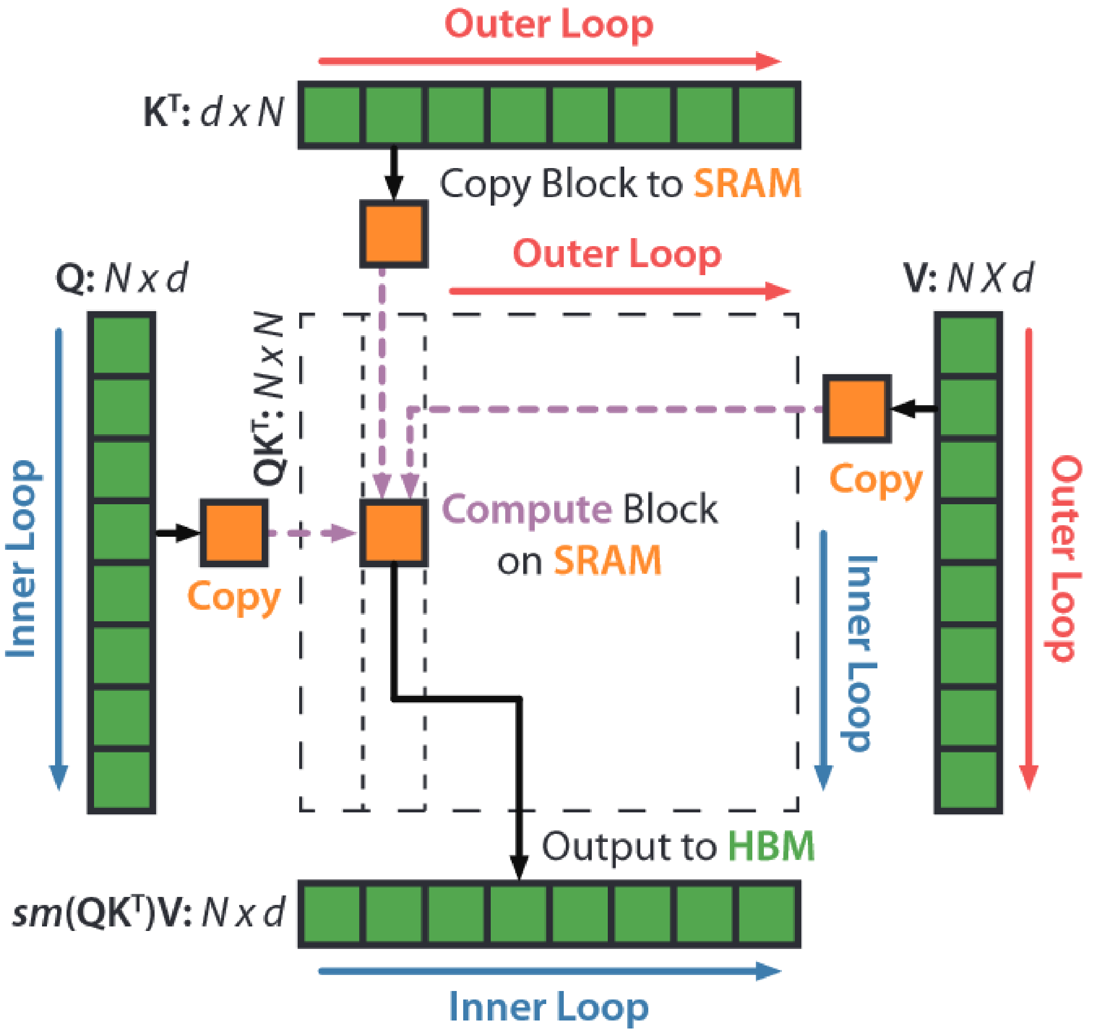
$\tilde{m}_{ij} = \text{rowmax}(S_{ij})$ $\tilde{P}_{ij} = \exp(S_{ij}\ -$ $\tilde{l}_{ij} = \text{rowsum}(\tilde{P}_{ij})$

On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.

Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1}(\text{diag}(\ell_i)e^{m_i - m_i^{\text{new}}}\mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}}\tilde{\mathbf{P}}_{ij}\mathbf{V}_j)$ to HBM.

Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.



FlashAttention

Performance

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

Questions?