Optimizing Attention

CSE 493G1, Winter 2026

Tanush Yadav

Attention: $O(n^2)$

Background

LLM Inference

Two Distinct Phases

how can we make decode more compute-efficient?

- **Decode**
  - generates output tokens one-at-a-time
  - passes generated token through model when generating new token
  - terminates when end-of-sequence token is generated

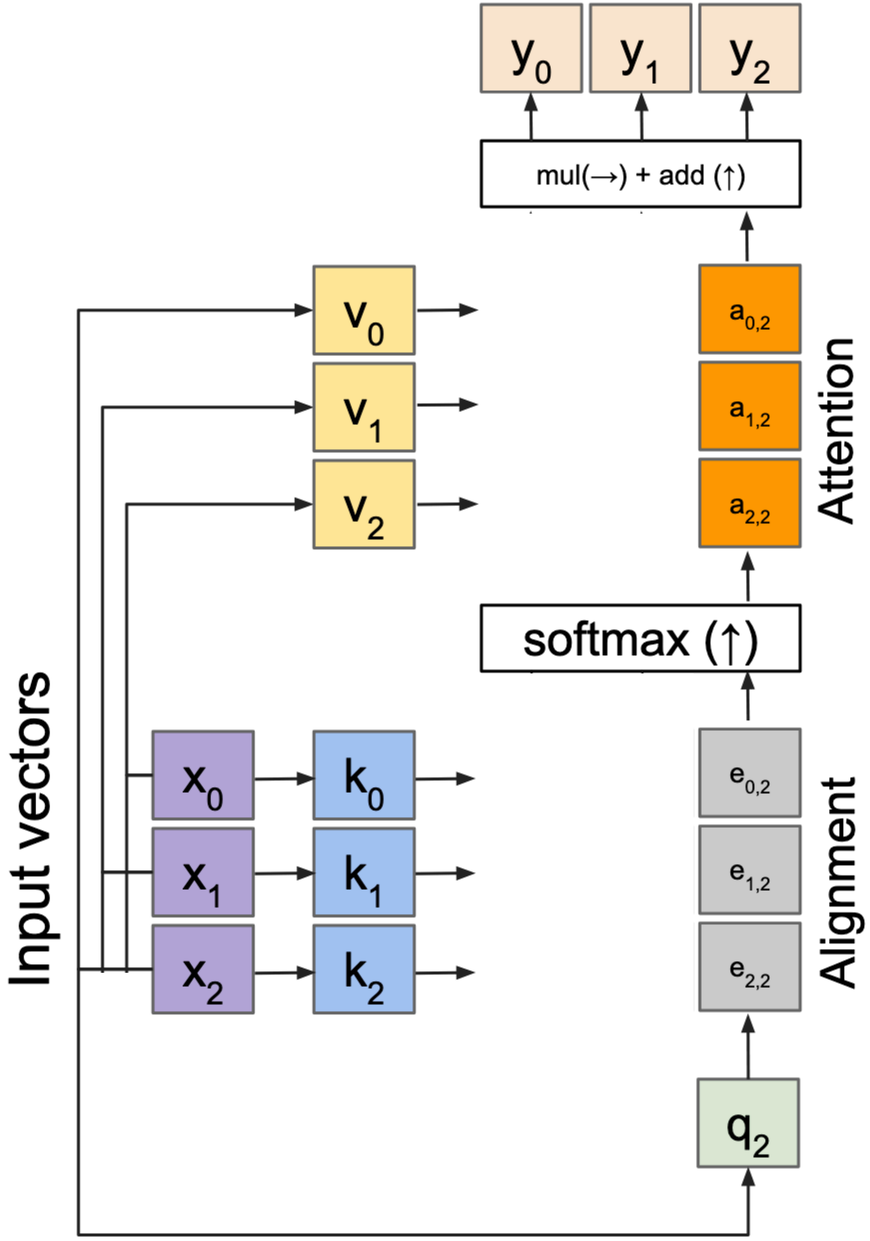
Agrawal et al. 2024, “Sarathi-Serve”

LLM Inference

Standard Attention

- Attending over **all** queries makes sense for prefill, since we need to compute attention outputs for all available tokens
- During decode, we only need the attention outputs for the last token

$$O(n^2) \rightarrow O(n)$$



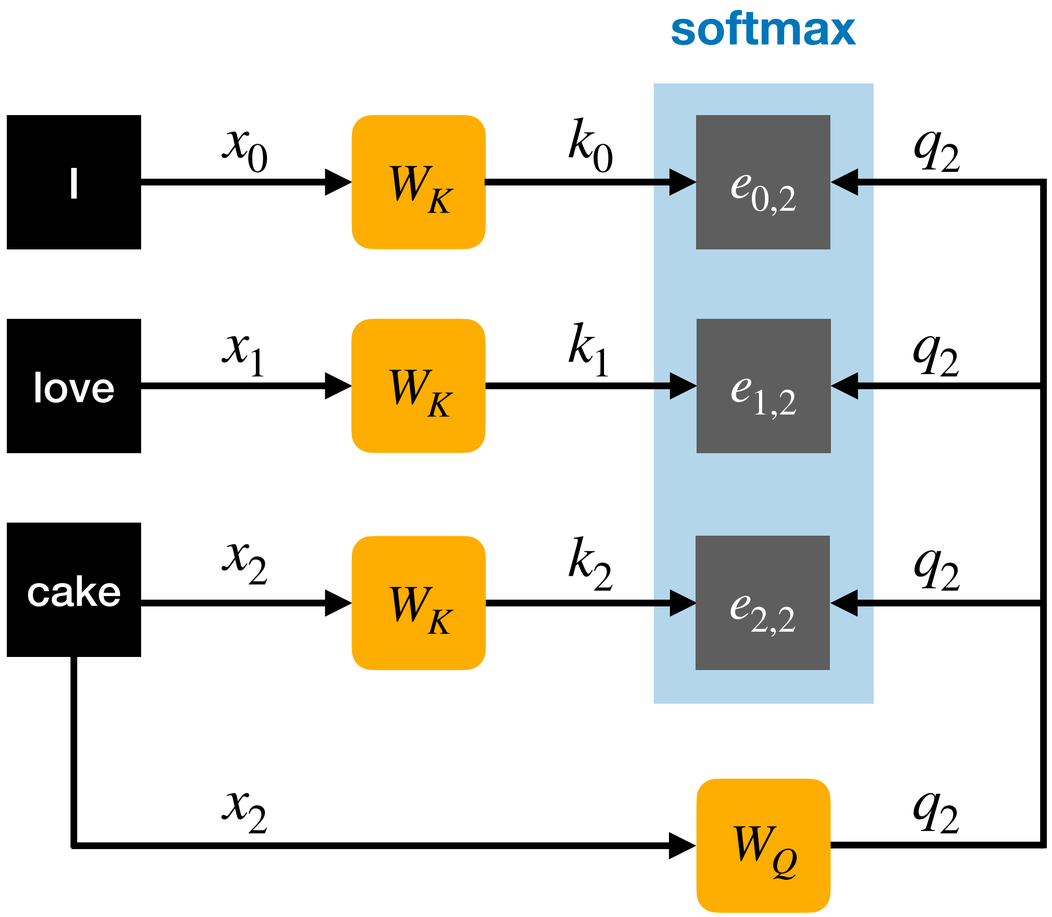
LLM Inference

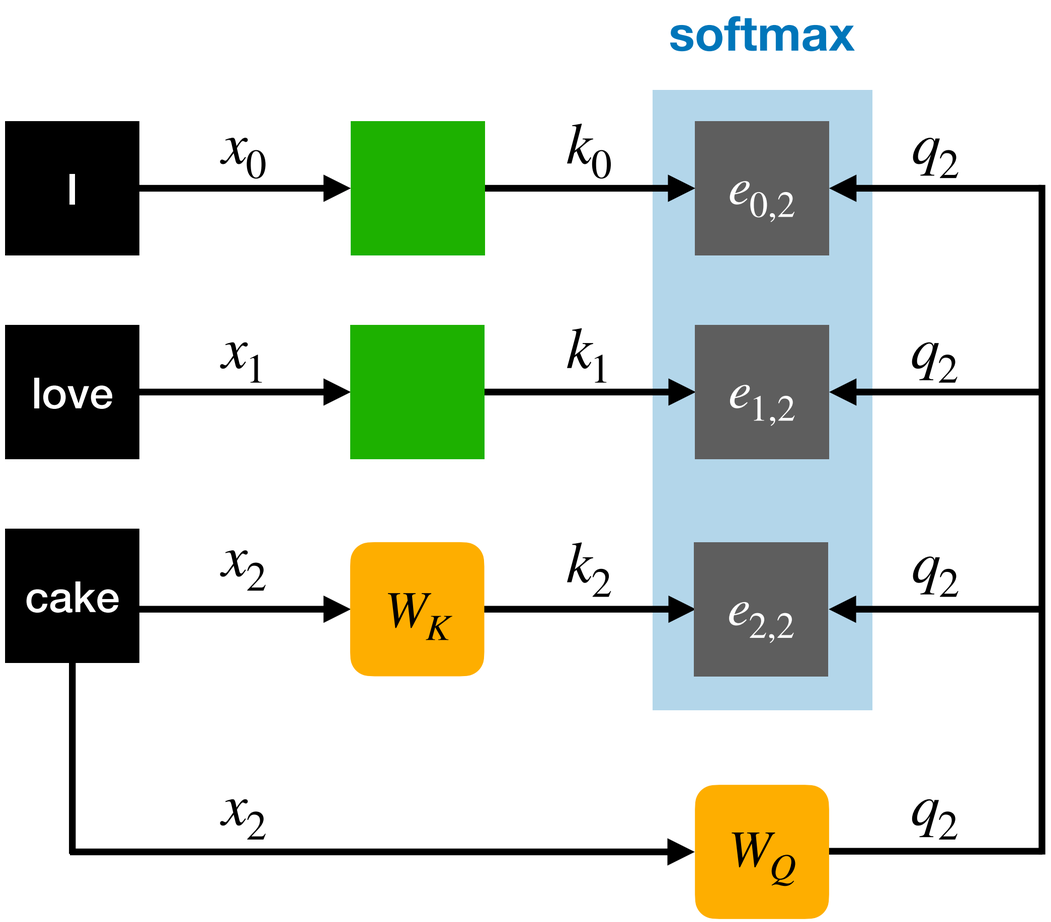
KV Caching

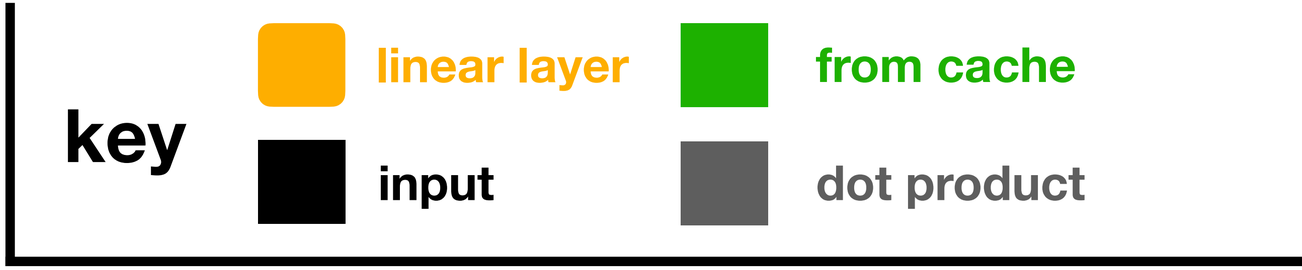
- When generating the $k+1$-th token, we must compute…
  - Key & value vectors for the k previous tokens — computed during previous generations
  - Query & key & value vectors for the $k+1$-th token — computed for the first time on this generation
  - Attention logits using the $k+1$-th query and the $[1..k, k+1]$ keys
  - Outputs using the attention scores and the $[1..k, k+1]$ value vectors

LLM Inference

KV Caching

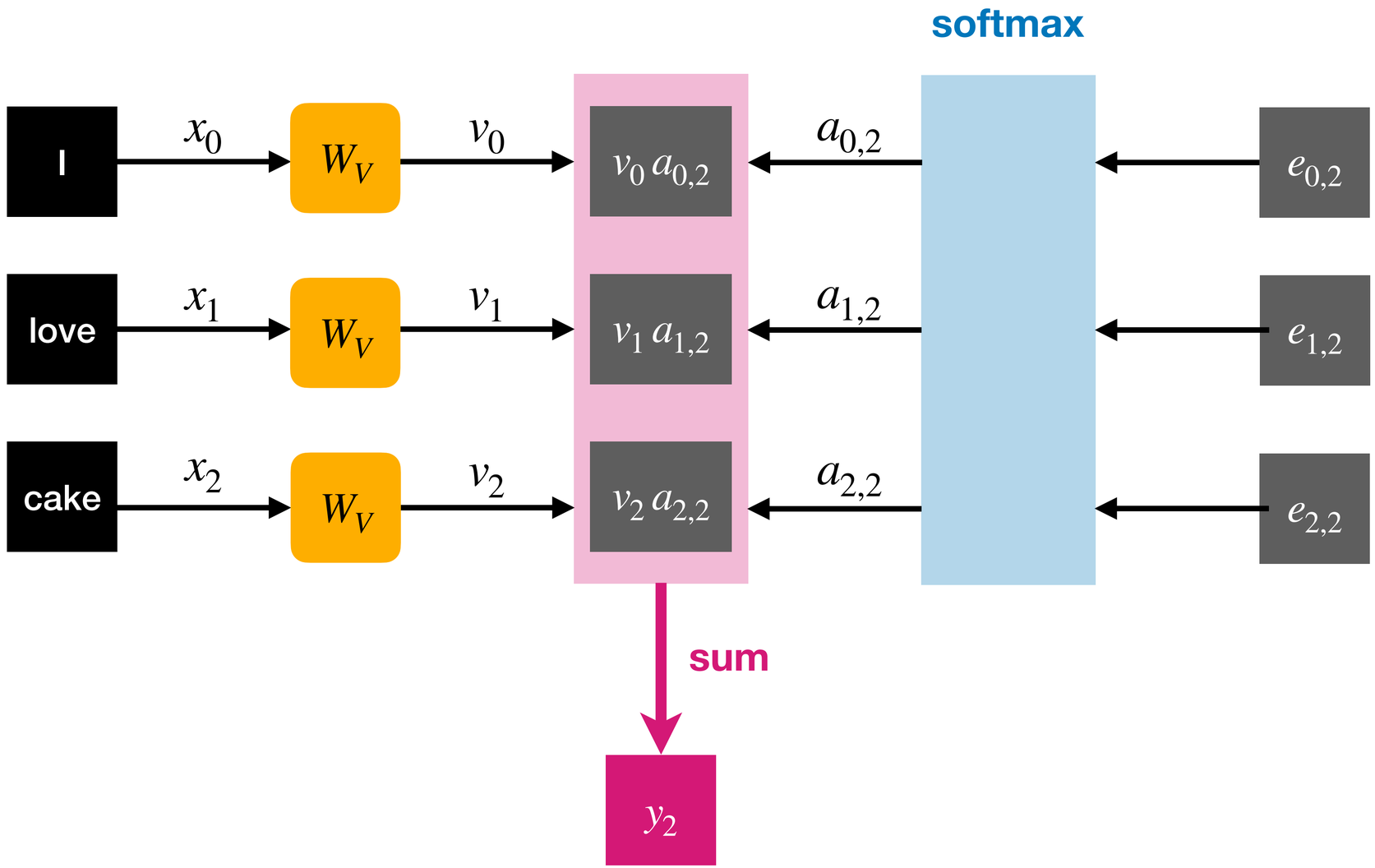






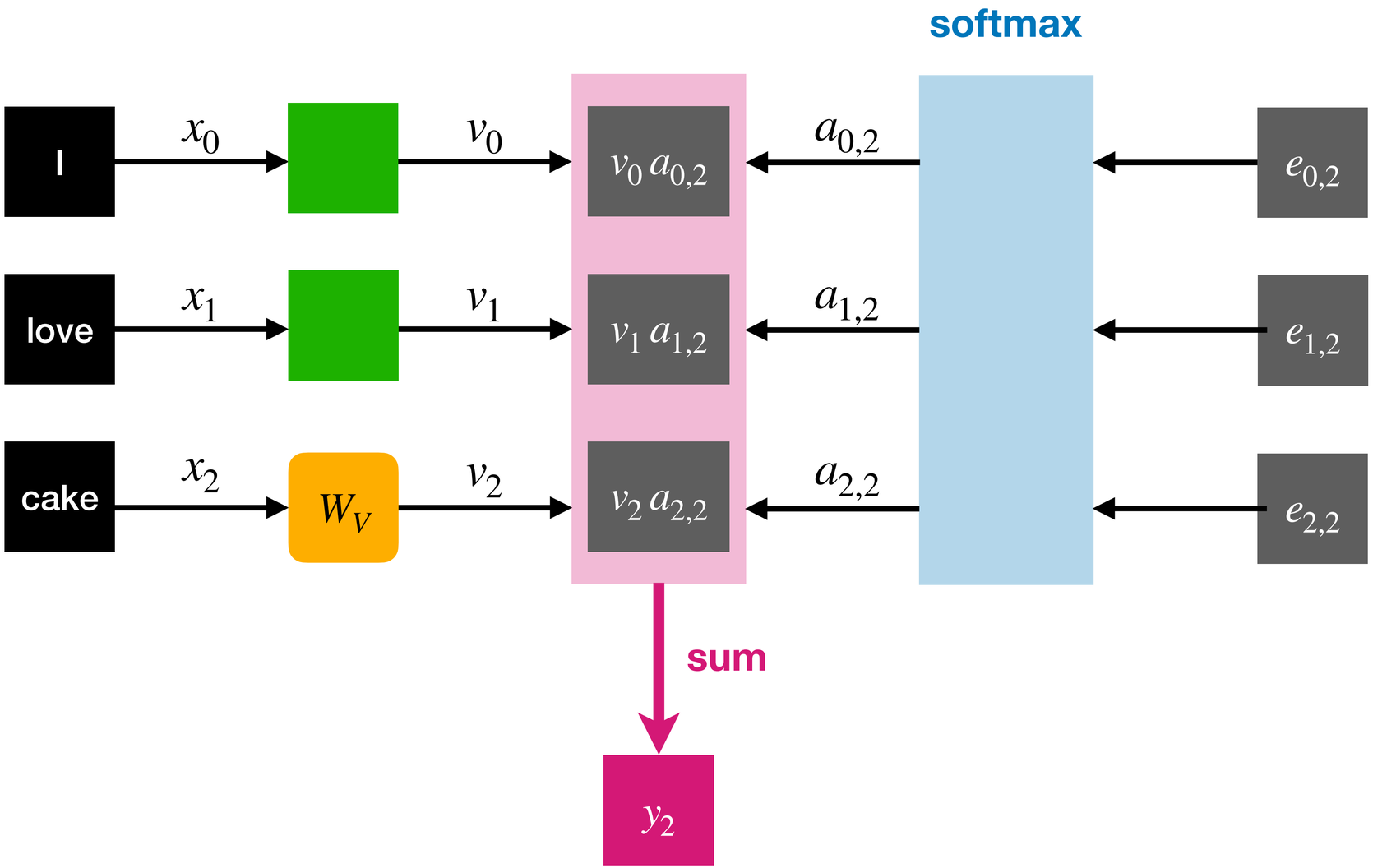
LLM Inference

KV Caching



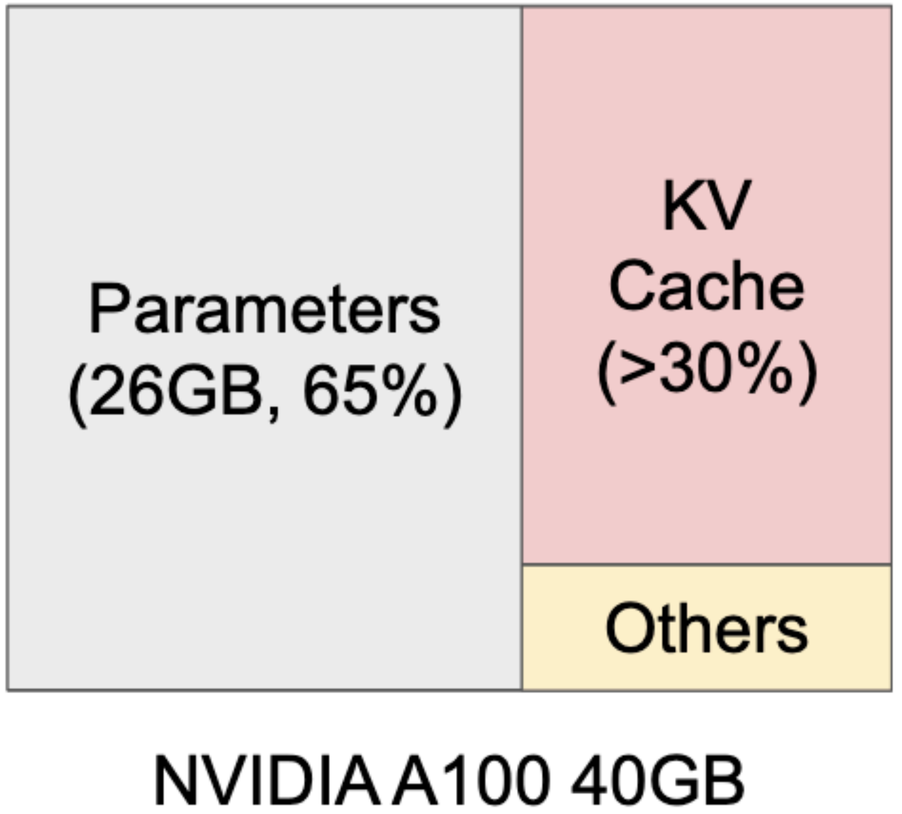
LLM Inference

KV Caching



LLM Inference

KV Caching



Kwon et al. 2023, “PagedAttention”

Systems Aside

Terminology

- **Compute-bound**: waiting on CPU calculations
  - compute the first n digits of π
  - brute-force a password hash
- **Memory-bound**: waiting on memory accesses
  - (naively) transpose a matrix
  - copy a large file between two memory buffers
- **Wall Clock Time**: how long a program takes to run (from start to finish)

Systems Aside

Terminology

- **Arithmetic Intensity**: # of math ops per byte of data fetched from memory
  - Optimal value: FLOPS-to-bandwidth ratio of device
  - High value indicates compute-bound
  - Low value indicates memory-bound

prefill and decode — compute-bound or memory-bound?

Agrawal et al. 2024, “Sarathi-Serve”

LLM Inference
Two Distinct Phases

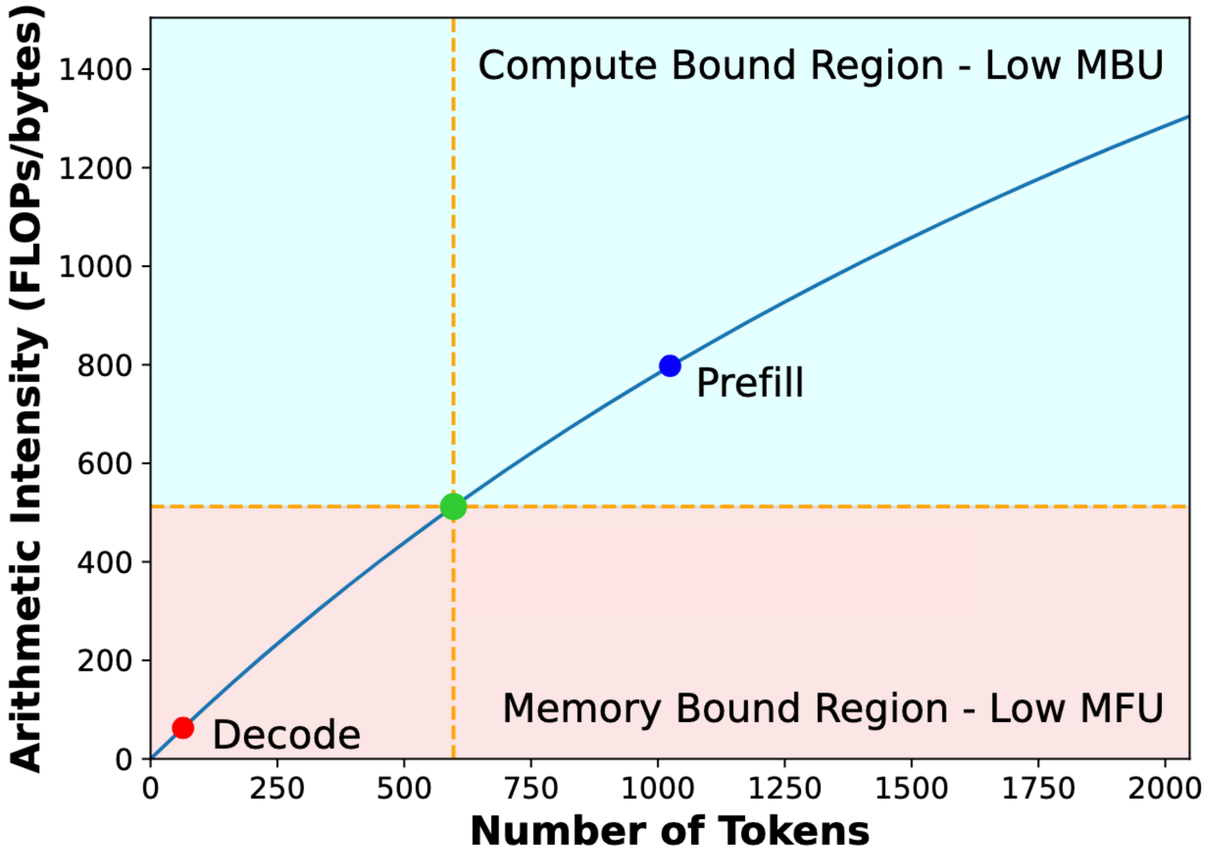
prefill and decode:
compute-bound or memory-bound?

- **Prefill**
  - processes user input prompt
  - produces first output token
- **Decode**
  - generates output tokens one-at-a-time
  - terminates when end-of-sequence token is generated

Agrawal et al. 2024, “Sarathi-Serve”

LLM Inference

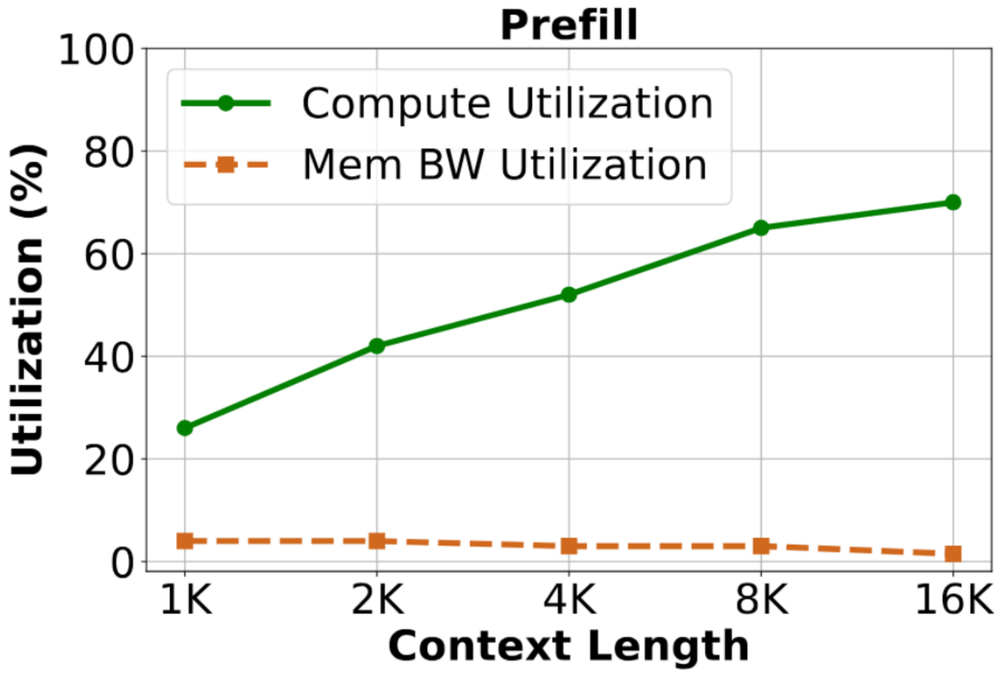
Compute vs Memory

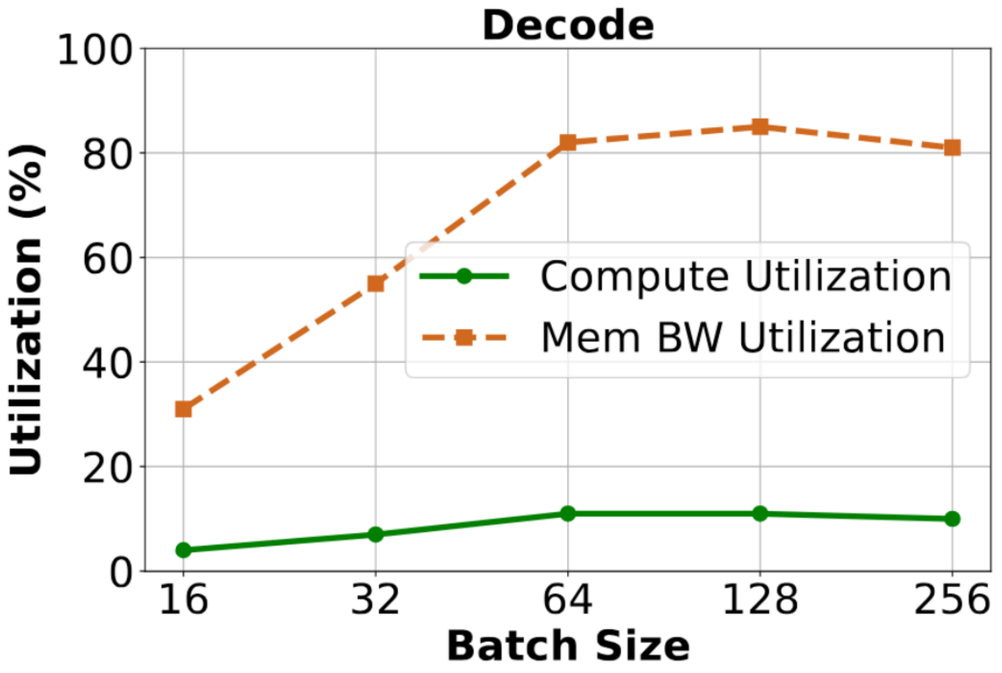


Agrawal et al. 2024, “Sarathi-Serve”

LLM Inference

Compute vs Memory





Kamath et al. 2024, “POD-Attention”

LLM Inference

Compute vs Memory

"For a 13B LLM, computing the prefill of a 512-token sequence makes an A100 compute-bound."

Kamath et al. 2024, "POD-Attention"

LLM Inference

Compute vs Memory

- We have seen that **prefill is compute bound** and **decode is memory bound**
- KV caching is the only compute optimization we will cover
- Rest of lecture will focus on memory optimizations

does this mean rest of lecture is only useful for decode?

Intuition

Attention

Compute vs Memory

“Compute-bound operations make up over 99.8% of total FLOPs but only take up 61% of total runtime”

Ivanov et al. 2021, “Data Movement”
Clint Greene 2024, “Accelerating LLMs with Flash Attention”

Attention

Compute vs Memory

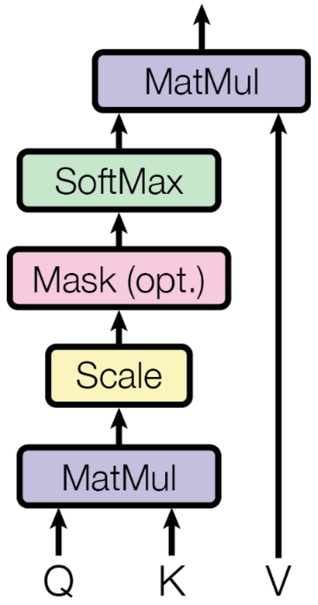
"Surprisingly, memory-bound operations consume the remaining 40% of a Transformer model's runtime, but only account for 0.02% of the total FLOPs."

Ivanov et al. 2021, "Data Movement"
Clint Greene 2024, "Accelerating LLMs with Flash Attention"

Attention

Compute vs Memory

Scaled Dot-Product Attention



linear layer

ulk of the
clock time
ent here

large result

low arithmetic intensity

linear layer

	Time (µs)	Compute Peak (%)	Memory Peak (%)
Q, K, V	306	61.2	12
QK^T	143	21.8	50
Scaled Softmax	433	1.3	32
Final Value	160	19.4	6

Image from Vaswani et al. 2017, "Attention Is All You Need"

Ivanov et al. 2021, "Data Movement"

Attention

Compute vs Memory

	Arithmetic Intensity
Ideal Ratio	138.9
Q, K, V	910
QK^T	102
Softmax	2.5

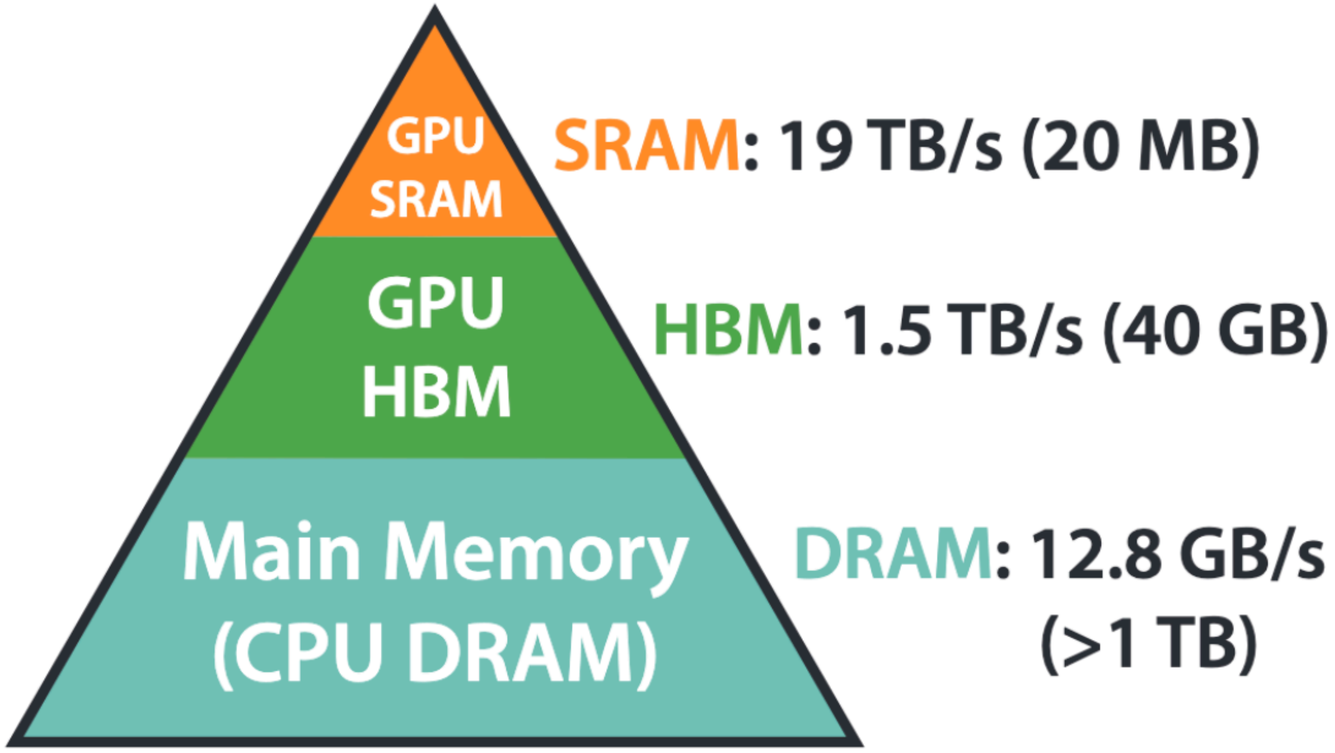
extremely memory-bound

Tested on V100s

Ivanov et al. 2021, “Data Movement”

Systems Aside

Memory Hierarchy



Dao et al. 2024, “FlashAttention”

Attention

Memory

Algorithm 1 Standard Attention

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

1: Load Q and K (by blocks) from HBM.
2: Compute $S = QK^T$ and write $S \in \mathbb{R}^{N \times N}$ to HBM.
3: Load S (by blocks) from HBM.
4: Compute $P = \text{softmax}(S)$ and write $P \in \mathbb{R}^{N \times N}$ to HBM.
5: Load P and V (by blocks) from HBM.
6: Compute $O = PV$ and write $O \in R^{N \times d}$ to HBM.
7: **Return** O.

S and P are stored and immediately read from memory

Attention
Memory

- Idea: keep intermediate results in SRAM
  - $S \in \mathbb{R}^{N \times N}$
  - GPT-2 has $N = 1024$
  - $\Rightarrow S$ contains 1024^2 elements
  - Assume each element takes 16 bits to store
  - $\Rightarrow S$ takes $1024^2 \cdot 2 = 2{,}097{,}152$ bytes ≈ 2 MB

but an A100 only has 192 KB of L1 Cache!

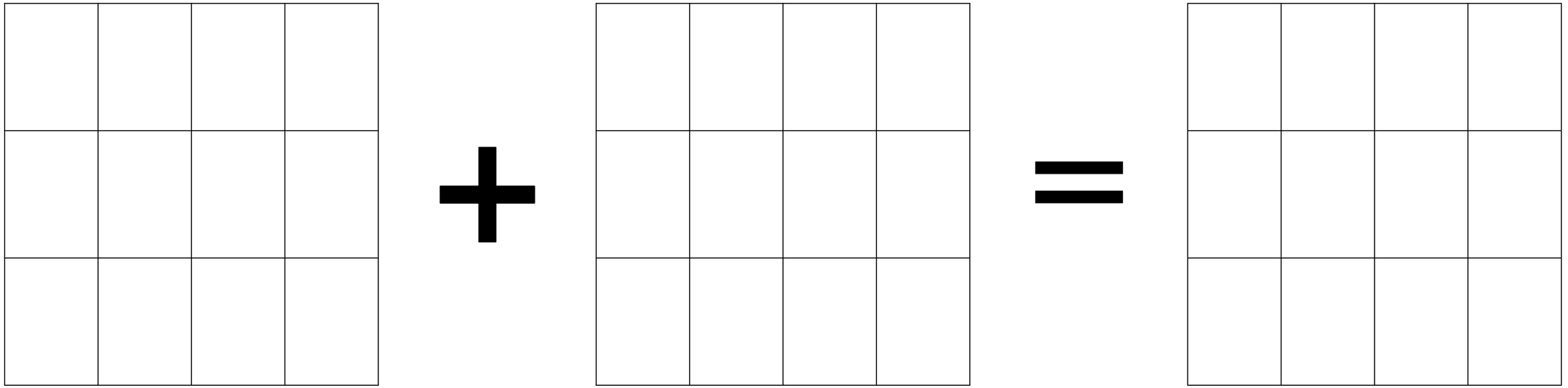
Attention

Memory

- We can’t cache intermediate values in SRAM; they’re too big
- What if we computed Softmax tile-by-tile on the inputs?

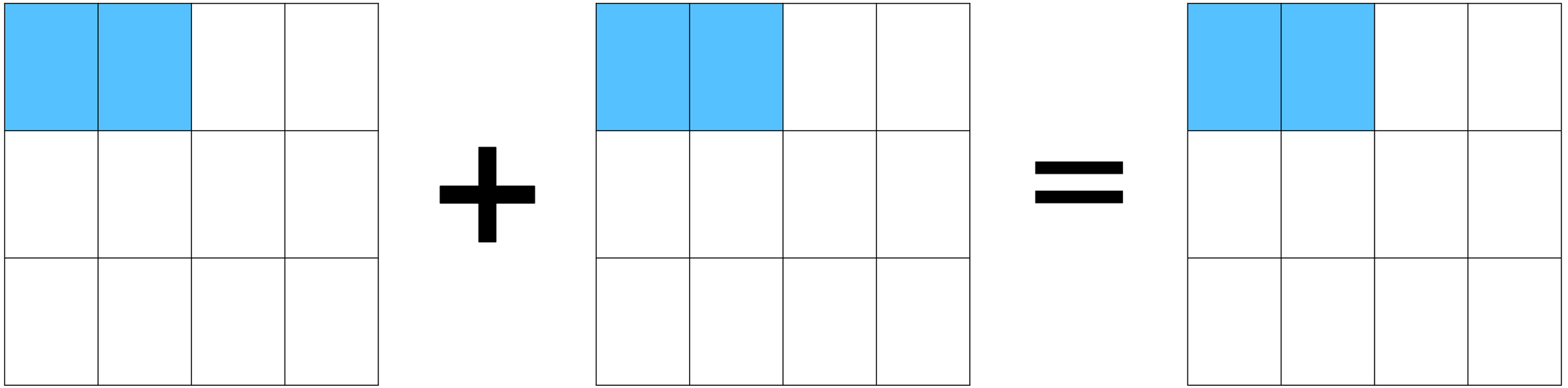
Tiling

Trivial for matrix addition



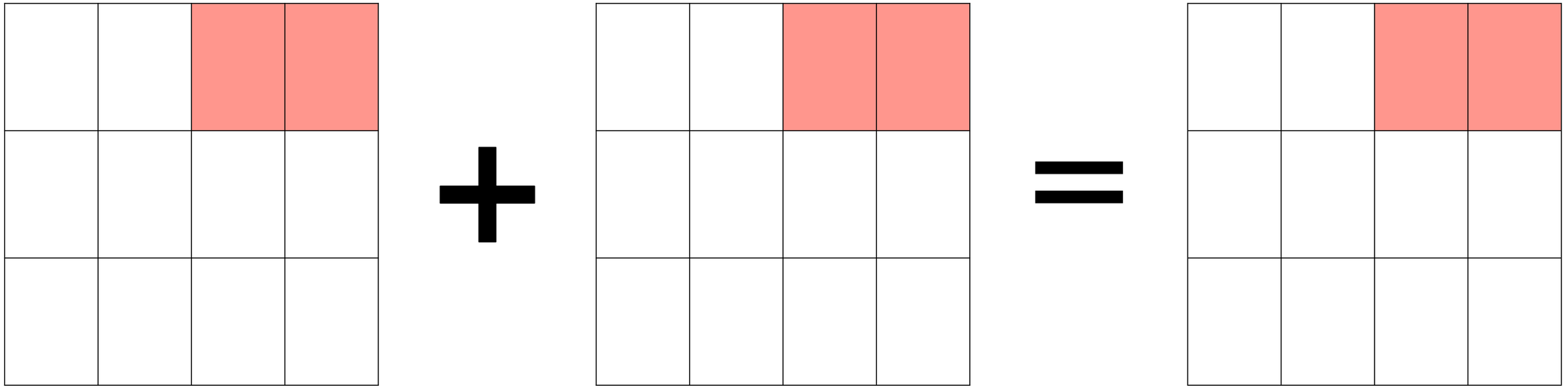
Tiling

Trivial for matrix addition



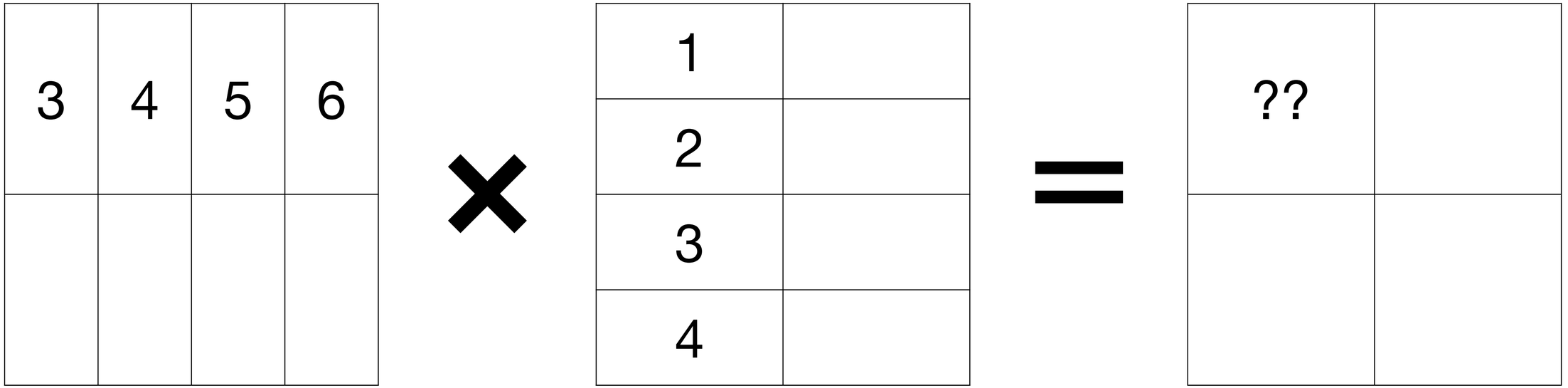
Tiling

Matrix addition



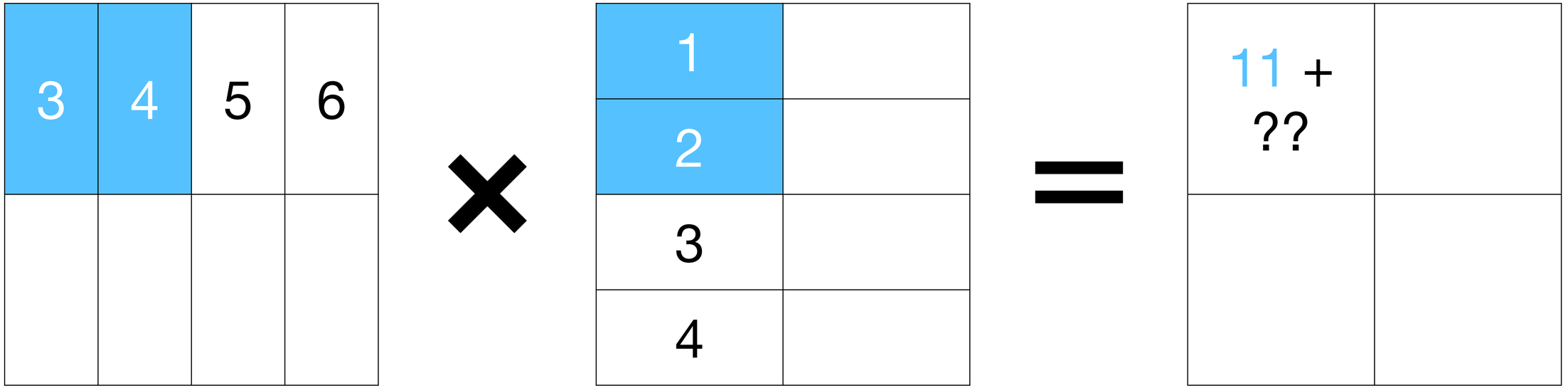
Tiling

Matrix multiplication



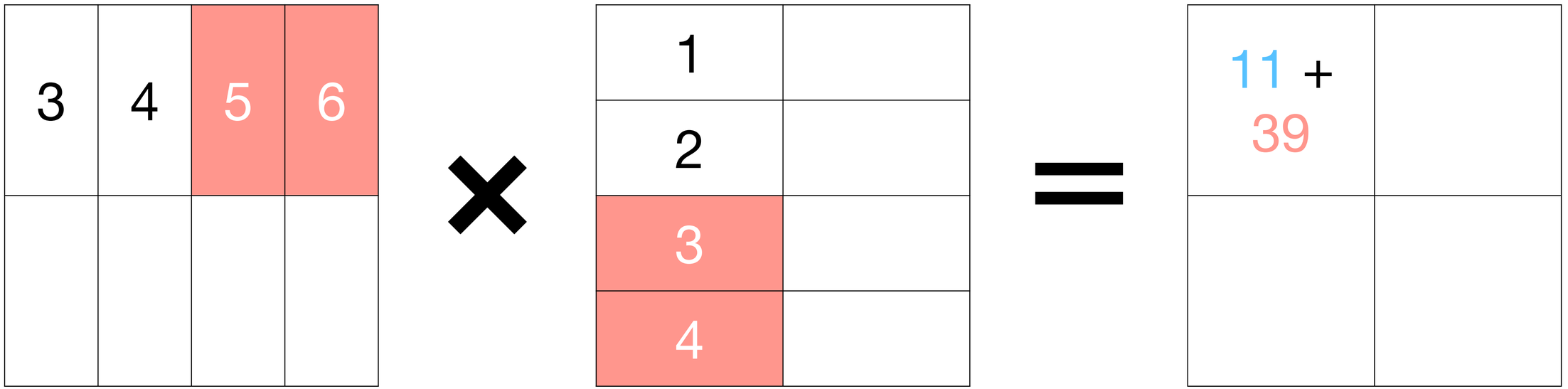
Tiling

Matrix multiplication



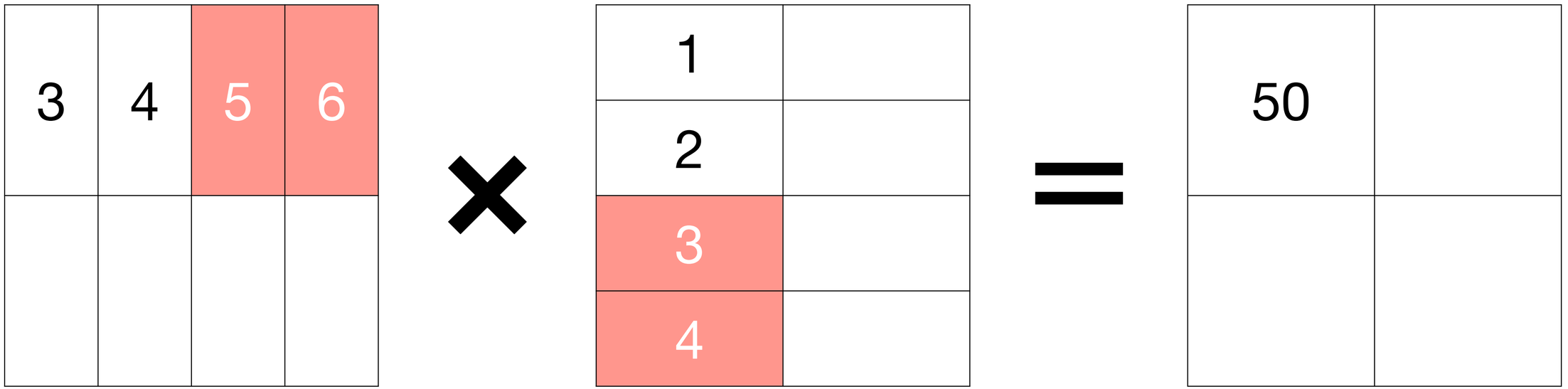
Tiling

Matrix multiplication



Tiling

Matrix multiplication



Tiling
Softmax

- Non-trivial because denominator for any one term depends on ALL terms

Online Softmax

This section follows heavily from Zihao Ye's course notes for UW CSE 599M

Softmax

$$\text{softmax}(\{x_1, \ldots, x_N\}) = \left\{ \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \right\}_{i=1}^{N}$$

Softmax

with numerical stability

$$\frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} = \frac{e^{x_i - m}}{\sum_{j=1}^{N} e^{x_j - m}}$$

Softmax

with numerical stability

- Before calculating softmax value for each input, we need:
  - Max across all elements m_i
  - Shared denominator d_i

let's naively iterate over all input elements and slowly build up these values

Softmax

Notation

$$m_0 = -\infty$$

$$m_i = \max\{x_1, x_2, \ldots, x_i\}$$

$$d_0 = 0$$

$$d_i = \sum_{j=1}^{i} e^{x_j - m_N}$$

Softmax

Notation

- Let's use a_i to refer to the softmax value for input x_i

Softmax

3-Pass Algorithm

1. $\textbf{for } i \leftarrow 1, N \textbf{ do} \qquad m_i \leftarrow \max\left(m_{i-1}, x_i\right)$
2. $\textbf{for } i \leftarrow 1, N \textbf{ do} \qquad d_i \leftarrow d_{i-1} + e^{x_i - m_N}$
3. $\textbf{for } i \leftarrow 1, N \textbf{ do} \qquad a_i \leftarrow \dfrac{e^{x_i - m_N}}{d_N}$

Softmax

3-Pass Algorithm

- In the context of attention, x_i's are pre-softmax logits given by QK^T

- Three options:
  - Cache all logits (not enough SRAM)
  - Compute logits on-the-fly 3x (not IO-efficient)
  - Store logits in memory; access them 3x (not IO-efficient)

Softmax

2-Pass Algorithm

- Let’s fuse one of the three loops
- An obvious issue arises…

Softmax

3-Pass Algorithm

1. $\textbf{for } i \leftarrow 1, N \textbf{ do} \qquad m_i \leftarrow \max\left(m_{i-1}, x_i\right)$

2. $\textbf{for } i \leftarrow 1, N \textbf{ do} \qquad d_i \leftarrow d_{i-1} + e^{x_i - m_N}$

   relies on loop #1

3. $\textbf{for } i \leftarrow 1, N \textbf{ do} \qquad a_i \leftarrow \dfrac{e^{x_i - m_N}}{d_N}$

Softmax

2-Pass Algorithm

- Let's create an additional sequence d'_i where we subtract m_i instead of m_N

$$d_0 = 0 \qquad d_i = \sum_{j=1}^{i} e^{x_j - m_N}$$

$$d'_0 = 0 \qquad d'_i = \sum_{j=1}^{i} e^{x_j - m_i}$$

Softmax

2-Pass Algorithm

- Observe that $d'_N = d_N$, so the final value for our denominator is still accurate

$$d_0 = 0 \qquad d_i = \sum_{j=1}^{i} e^{x_j - m_N}$$

$$d'_0 = 0 \qquad d'_i = \sum_{j=1}^{i} e^{x_j - m_i}$$

Softmax

2-Pass Algorithm

- Observe that d'_i is **NOT** dependent on N

$$d_0 = 0 \qquad d_i = \sum_{j=1}^{i} e^{x_j - m_N}$$

$$d'_0 = 0 \qquad d'_i = \sum_{j=1}^{i} e^{x_j - m_i}$$

Softmax

2-Pass Algorithm

1. **for** $i \leftarrow 1, N$ **do**

$$m_i \leftarrow \max\left(m_{i-1}, x_i\right)$$

$$d_i' \leftarrow \sum_{j=1}^{i} e^{x_j - m_i}$$

nested loop :(

2. **for** $i \leftarrow 1, N$ **do**

$$a_i \leftarrow \frac{e^{x_i - m_N}}{d_N'}$$

Softmax

Unrolling the Recurrence

$$d'_i = \sum_{j=1}^{i} e^{x_j - m_i}$$

Softmax

Unrolling the Recurrence

$$
\begin{aligned}
d'_i &= \sum_{j=1}^{i} e^{x_j - m_i} \\
&= \left(\sum_{j=1}^{i-1} e^{x_j - m_i} \right) + e^{x_i - m_i}
\end{aligned}
$$

Softmax

Unrolling the Recurrence

$$
\begin{aligned}
d'_i &= \sum_{j=1}^{i} e^{x_j - m_i} \\
&= \left(\sum_{j=1}^{i-1} e^{x_j - m_i} \right) + e^{x_i - m_i}
\end{aligned}
$$

Recall, $d'_{i-1} = \sum_{j=1}^{i-1} e^{x_j - m_{i-1}}$

Softmax

Unrolling the Recurrence

Goal: $\sum_{j=1}^{i-1} e^{x_j - m_i} \longrightarrow \sum_{j=1}^{i-1} e^{x_j - m_{i-1}}$

Softmax

Unrolling the Recurrence

$$e^{x_j - m_i} \longrightarrow e^{x_j - m_{i-1}}$$

$$e^{x_j - m_i} = e^{x_j - m_i - m_{i-1} + m_{i-1}}$$

Softmax

Unrolling the Recurrence

$$e^{x_j - m_i} \longrightarrow e^{x_j - m_{i-1}}$$

$$\begin{aligned} e^{x_j - m_i} &= e^{x_j - m_i - m_{i-1} + m_{i-1}} \\ &= e^{x_j - m_{i-1} + m_{i-1} - m_i} \end{aligned}$$

Softmax

Unrolling the Recurrence

$$e^{x_j - m_i} \longrightarrow e^{x_j - m_{i-1}}$$

$$\begin{aligned} e^{x_j - m_i} &= e^{x_j - m_i - m_{i-1} + m_{i-1}} \\ &= e^{x_j - m_{i-1} + m_{i-1} - m_i} \\ &= e^{x_j - m_{i-1}} \cdot e^{m_{i-1} - m_i} \end{aligned}$$

Softmax

Unrolling the Recurrence

Goal:

$$\sum_{j=1}^{i-1} e^{x_j - m_i} \longrightarrow \sum_{j=1}^{i-1} e^{x_j - m_{i-1}}$$

Solution:

$$\sum_{j=1}^{i-1} e^{x_j - m_i} = \sum_{j=1}^{i-1} \left(e^{x_j - m_{i-1}} e^{m_{i-1} - m_i} \right)$$

Softmax

Unrolling the Recurrence

Goal:

$$\sum_{j=1}^{i-1} e^{x_j - m_i} \longrightarrow \sum_{j=1}^{i-1} e^{x_j - m_{i-1}}$$

Solution:

$$\sum_{j=1}^{i-1} e^{x_j - m_i} = \sum_{j=1}^{i-1} \left(e^{x_j - m_{i-1}} e^{m_{i-1} - m_i} \right)$$

$$= e^{m_{i-1} - m_i} \sum_{j=1}^{i-1} \left(e^{x_j - m_{i-1}} \right)$$

Softmax

Unrolling the Recurrence

Goal:

$$\sum_{j=1}^{i-1} e^{x_j - m_i} \longrightarrow \sum_{j=1}^{i-1} e^{x_j - m_{i-1}}$$

Solution:

$$\begin{aligned}
\sum_{j=1}^{i-1} e^{x_j - m_i} &= \sum_{j=1}^{i-1} \left(e^{x_j - m_{i-1}} e^{m_{i-1} - m_i} \right) \\
&= e^{m_{i-1} - m_i} \sum_{j=1}^{i-1} \left(e^{x_j - m_{i-1}} \right) \\
&= e^{m_{i-1} - m_i} \cdot d'_{i-1}
\end{aligned}$$

Softmax

Unrolling the Recurrence

$$
\begin{aligned}
d'_i &= \sum_{j=1}^{i} e^{x_j - m_i} \\
&= \left(\sum_{j=1}^{i-1} e^{x_j - m_i} \right) + e^{x_i - m_i} \\
&= \left(\sum_{j=1}^{i-1} e^{x_j - m_{i-1}} \right) e^{m_{i-1} - m_i} + e^{x_i - m_i}
\end{aligned}
$$

Softmax

Unrolling the Recurrence

$$
\begin{aligned}
d'_i &= \sum_{j=1}^{i} e^{x_j - m_i} \\
&= \left(\sum_{j=1}^{i-1} e^{x_j - m_i} \right) + e^{x_i - m_i} \\
&= \left(\sum_{j=1}^{i-1} e^{x_j - m_{i-1}} \right) e^{m_{i-1} - m_i} + e^{x_i - m_i} \\
&= d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}
\end{aligned}
$$

Softmax

2-Pass Algorithm

1 **for** $i \leftarrow 1, N$ **do**

$$m_i \leftarrow \max\left(m_{i-1}, x_i\right)$$

$$d'_i \leftarrow d'_{i-1} \cdot e^{m_{i-1} - m_i} + e^{x_i - m_i}$$

2 **for** $i \leftarrow 1, N$ **do**

$$a_i \leftarrow \frac{e^{x_i - m_N}}{d'_N}$$

Softmax

1-Pass Algorithm

IMPOSSIBLE

Attention

Compute vs Memory

	Arithmetic Intensity
Ideal Ratio	138.9
Q, K, V	910
QK^T	102
Softmax	2.5

extremely memory-bound

Tested on V100s

Ivanov et al. 2021, “Data Movement”

Attention

Compute vs Memory

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 $\mathbf{O}$.

Dao et al. 2024, “FlashAttention”

FlashAttention

This section follows heavily from the original FlashAttention paper

FlashAttention

Notation

Given:

- Q is divided into blocks of size B_r
- K, V are divided into blocks of size B_c

FlashAttention

Notation

We initialize:

- $O \in \mathbb{R}^{N \times d}$ is our output matrix
- $l \in \mathbb{R}^{N}$ is a vector holding the Softmax denominator for all N rows
- $m \in \mathbb{R}^{N}$ is a vector holding the maximum value seen for all N rows

FlashAttention

Algorithm

Iterate on i and j through # of blocks for Q and K respectively

On the (i, j)-th iteration, we have:

- K_j and V_j **given**
- Q_i and O_i **must compute**
- l_i and m_i

FlashAttention

Algorithm

On the (i, j)-th iteration,

- $S_{ij} \leftarrow Q_i K_j^T \in \mathbb{R}^{B_r \times B_c}$: the pre-Softmax scores for this block
- $\tilde{m}_{ij} \leftarrow \text{rowmax}(S_{ij}) \in \mathbb{R}^{B_r}$: max element for each row *in this block*
- $\tilde{P}_{ij} \leftarrow \exp(S_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$: post-Softmax scores for this block
- $\tilde{l}_{ij} \leftarrow \text{rowsum}(\tilde{P}_{ij}) \in \mathbb{R}^{B_r}$: Softmax denominator for this block (local)

FlashAttention

Algorithm

On the (i, j)-th iteration,

- $m_i^{\text{new}} \leftarrow \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}_r^B$
- $l_i^{\text{new}} \leftarrow \exp(m_i - m_i^{\text{new}}) \cdot l_i + \exp(\tilde{m}_{ij} - m_i^{\text{new}}) \cdot \tilde{l}_{ij} \in \mathbb{R}^{B_r}$
- $O_i \leftarrow \text{diag}(l_i^{\text{new}})^{-1}(\text{diag}(l_i) \cdot e^{m_i - m_i^{\text{new}}} O_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{P}_{ij} V_j)$
- $m_i \leftarrow m_i^{\text{new}}$
- $l_i \leftarrow l_i^{\text{new}}$

must be written to HBM

FlashAttention

Notation

Let $\boldsymbol{x}^{(j)}$ be the value of some tensor $\boldsymbol{x}$ after the j-th iteration.

Let $\boldsymbol{A}_{:j}$ be the first $j \cdot B_c$ rows of some matrix $\boldsymbol{A}$

Let $\boldsymbol{A}_{:,:j}$ be the first $j \cdot B_c$ columns of some matrix $\boldsymbol{A}$

FlashAttention

Proof: Inductive Hypothesis

By induction.

After the j-th of the outer loop, HBM stores:

- $\boldsymbol{m}^{(j)} = \text{rowmax}(\boldsymbol{S}_{:,:j}) \in \mathbb{R}^N$
- $\boldsymbol{l}^{(j)} = \text{rowsum}(\exp(\boldsymbol{S}_{:,:j} - \boldsymbol{m}^{(j)})) \in \mathbb{R}^N$
- $\boldsymbol{O}^{(j)} = \boldsymbol{P}_{:,:j}\boldsymbol{V}_{:j} \in \mathbb{R}^{N \times D}$

why does this Inductive Hypothesis guarantee our algorithm's correctness?

FlashAttention

Proof: Base Case

Initialize such that IH holds for $j = 0$:

- $\boldsymbol{m}^{(j)} = -\infty$
- $\boldsymbol{l}^{(j)} = \mathbf{0} \in \mathbb{R}^{N}$
- $\boldsymbol{O}^{(j)} = \mathbf{0} \in \mathbb{R}^{N \times D}$

FlashAttention

Proof: Inductive Step (m)

goal $\boldsymbol{m}^{(j+1)} = \text{rowmax}(\boldsymbol{S}_{:,:j+1}) \in \mathbb{R}^N$

$$\boldsymbol{m}^{(j+1)} = \max\left(\boldsymbol{m}^{(j)}, \tilde{\boldsymbol{m}}\right)$$

FlashAttention

Proof: Inductive Step (m)

goal $\boldsymbol{m}^{(j+1)} = \text{rowmax}(\boldsymbol{S}_{:,:j+1}) \in \mathbb{R}^N$

$$\begin{aligned}\boldsymbol{m}^{(j+1)} &= \max\left(\boldsymbol{m}^{(j)}, \tilde{\boldsymbol{m}}\right) \\ &= \max\left(\text{rowmax}(\boldsymbol{S}_{:,:j}), \tilde{\boldsymbol{m}}\right)\end{aligned}$$

FlashAttention

Proof: Inductive Step (m)

goal $\boldsymbol{m}^{(j+1)} = \text{rowmax}(\boldsymbol{S}_{:,:j+1}) \in \mathbb{R}^N$

$$
\begin{aligned}
\boldsymbol{m}^{(j+1)} &= \max\left(\boldsymbol{m}^{(j)}, \tilde{\boldsymbol{m}}\right) \\
&= \max\left(\text{rowmax}(\boldsymbol{S}_{:,:j}), \tilde{\boldsymbol{m}}\right) \\
&= \max\left(\text{rowmax}\left(\boldsymbol{S}_{:,:j}\right), \text{rowmax}\left(\boldsymbol{S}_{:,j:j+1}\right)\right)
\end{aligned}
$$

FlashAttention

Proof: Inductive Step (m)

goal $\boldsymbol{m}^{(j+1)} = \text{rowmax}(\boldsymbol{S}_{:,:j+1}) \in \mathbb{R}^N$

$$
\begin{aligned}
\boldsymbol{m}^{(j+1)} &= \max\left(\boldsymbol{m}^{(j)}, \tilde{\boldsymbol{m}}\right) \\
&= \max\left(\text{rowmax}(\boldsymbol{S}_{:,:j}), \tilde{\boldsymbol{m}}\right) \\
&= \max\left(\text{rowmax}\left(\boldsymbol{S}_{:,:j}\right), \text{rowmax}\left(\boldsymbol{S}_{:,j:j+1}\right)\right) \\
&= \text{rowmax}\left(\boldsymbol{S}_{:,:j+1}\right)
\end{aligned}
$$

FlashAttention

Proof: Inductive Step (l)

goal $\boldsymbol{l}^{(j+1)} = \text{rowsum}(\exp(\boldsymbol{S}_{:,:j+1} - \boldsymbol{m}^{(j+1)}))$

$$\boldsymbol{l}^{(j+1)} = e^{\boldsymbol{m}^{(j)} - \boldsymbol{m}^{(j+1)}} \boldsymbol{l}^{(j)} + e^{\tilde{\boldsymbol{m}} - \boldsymbol{m}(j+1)} \tilde{\boldsymbol{l}}$$

FlashAttention

Proof: Inductive Step (l)

goal $l^{(j+1)} = \text{rowsum}(\exp(S_{:,:j+1} - m^{(j+1)}))$

$$= \text{rowsum}\left(\exp\left(\boldsymbol{S}_{:,:j} - \boldsymbol{m}^{(j)} + \boldsymbol{m}^{(j)} - \boldsymbol{m}^{(j+1)}\right)\right) + \exp\left(\tilde{\boldsymbol{m}} - \boldsymbol{m}^{(j+1)}\right) \cdot \text{rowsum}\left(\exp\left(\boldsymbol{S}_{:,j:j+1} - \tilde{m}\right)\right)$$

FlashAttention

Proof: Inductive Step (O)

goal $\boldsymbol{O}^{(j)} = \boldsymbol{P}_{:,:j}\boldsymbol{V}_{:j}$

$$\boldsymbol{O}^{(j+1)} = \left(\mathbf{1} \oslash \operatorname{diag}(\boldsymbol{l}^{(j+1)})\right)\left(\operatorname{diag}(\boldsymbol{l}^{(j)})\exp(\boldsymbol{m}^{(j)} - \boldsymbol{m}^{(j+1)})\boldsymbol{O}^{(j)} + \exp(\tilde{\boldsymbol{m}} - \boldsymbol{m}^{(j+1)})\tilde{\boldsymbol{P}}_{j+1}V_{j:j+1}\right)$$

FlashAttention

Proof: Inductive Step (O)

goal $\boldsymbol{O}^{(j)} = \boldsymbol{P}_{:,:j}\boldsymbol{V}_{:j}$

$$= \left(1 \oslash \text{diag}(\boldsymbol{l}^{(j+1)})\right) \left(\exp(\boldsymbol{m}^{(j)} - \boldsymbol{m}^{(j+1)}) \exp(\boldsymbol{S}_{:,:j} - \boldsymbol{m}^{j})\boldsymbol{V}_{:j} + \exp(\boldsymbol{S}_{:,j:j+1} - \boldsymbol{m}^{(j+1)})\boldsymbol{V}_{j:j+1}\right)$$

FlashAttention

Backward Pass

- Avoid storing $O(N^2)$ values for S, P in HBM
- Instead, recompute them on-the-fly using cached O, m, l and inputs Q, K, V
- More FLOPs, but fewer HBM accesses

FlashAttention

Results

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×)**

Dao et al. 2024, "FlashAttention"

FlashAttention

Results

Model implementations	Context length	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Megatron-LM	1k	18.2	4.7 days (1.0×)
GPT-2 small - FlashAttention	1k	18.2	**2.7 days (1.7×)**
GPT-2 small - FlashAttention	2k	17.6	3.0 days (1.6×)
GPT-2 small - FlashAttention	4k	**17.5**	3.6 days (1.3×)

Dao et al. 2024, “FlashAttention”

Conclusion

Summary

Efficient models are critical.

- KV caching removes redundant computations
- FlashAttention optimizes I/O to yield a 2-3x speedup for attention
- Lots of exciting work being done on optimizing ML serving
  - CSE 554: Systems for ML