https://hao-ai-lab.github.io/cse234-w25/

# CSE 234: Data Systems for Machine Learning
# Winter 2025

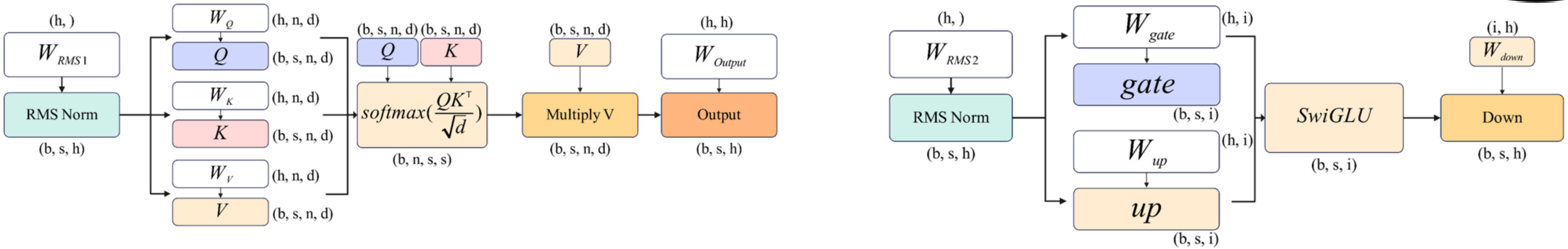
LLMSys

Optimizations and Parallelization

MLSys Basics

large b

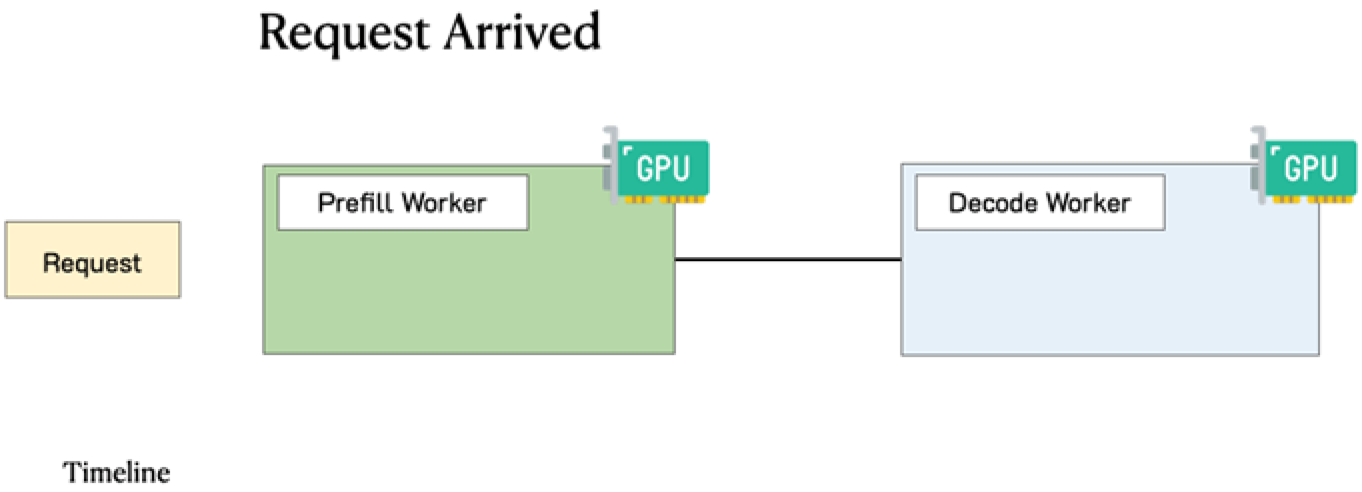
# Potential Bottleneck of LLM Inference in Serving




- How to batch
  - Prompts have different length and unknown #generated tokens
  - Sol: continuous batching
- Memory
  - KV cache memory becomes a bottleneck
  - Sol: paged attention
- In the presence of SLOs (beyond throughput)
  - Interference of prefill and decoding
  - Sol: disaggregate prefill and decode

# Disaggregating Prefill and Decode

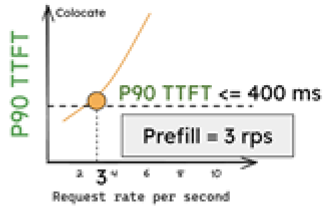
Disaggregation is a technique that

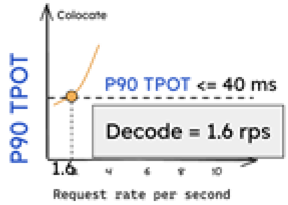


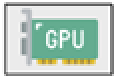
# Disaggregation achieves better goodput

## Colocate

1 GPU for both Prefill and Decode




Max System goodput

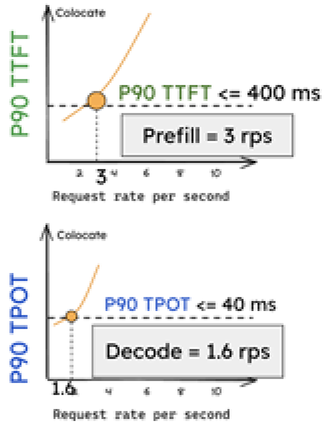
= Min(Prefill, Decode)

= 1.6 rps / GPU

# Disaggregation achieves better goodput

## Colocate

1 GPU for both Prefill and Decode



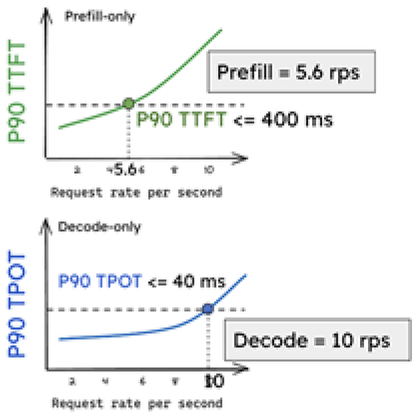

Max System goodput

= Min(Prefill, Decode)

= 1.6 rps / GPU

## Disaggregate (2P1D)

2 GPU for Prefill + 1 GPU for Decode




Disaggregate (2P1D) goodput

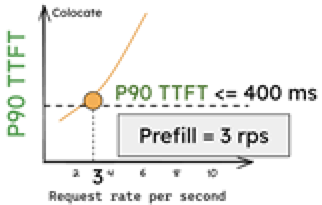
= Min (5.6 x **2**, 10) rps / 3 GPU

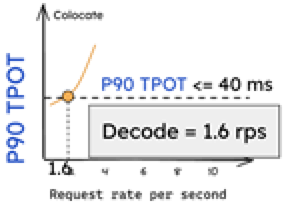
= 3.3 rps / GPU

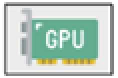
# Disaggregation achieves better goodput

## Colocate

1 GPU for both Prefill and Decode



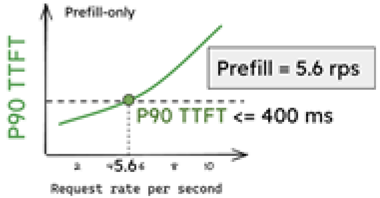

Max System goodput

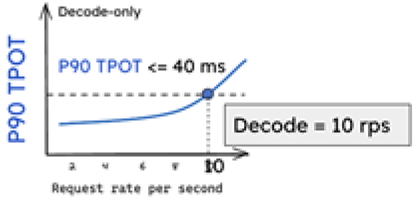
= Min(Prefill, Decode)

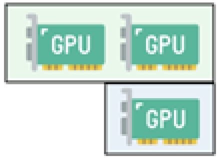
= 1.6 rps / GPU

## Disaggregate (2P1D)

2 GPU for Prefill + 1 GPU for Decode




Disaggregate (2P1D) goodput

= Min (5.6 x **2**, 10) rps / 3 GPU

= 3.3 rps / GPU

Simple Disaggregation

achieves **2x** goodput

(per GPU)

# Disaggregation

- Published in 2024 at UCSD (yes, Hao's lab)
- Soon become the chosen architecture replacing continuous batching at large scale
- Deepseek-v3 uses prefill-decode disaggregation combined with different parallelisms for prefill and decoding instances.

# Continuous batching -> disaggregation

- It seems we are going back and forth

- Actually no:
  - Continuous batching: improve GPU utlization hence throughput
  - Disaggregation: to address goodput, throughput s.t. SLOs

- Also, key insights of CB carries to disaggregation
  - Batch attentions and MLPs differently
  - Exit finished request and pick up new request asap

# LLM Inference Now is High-stake research topic

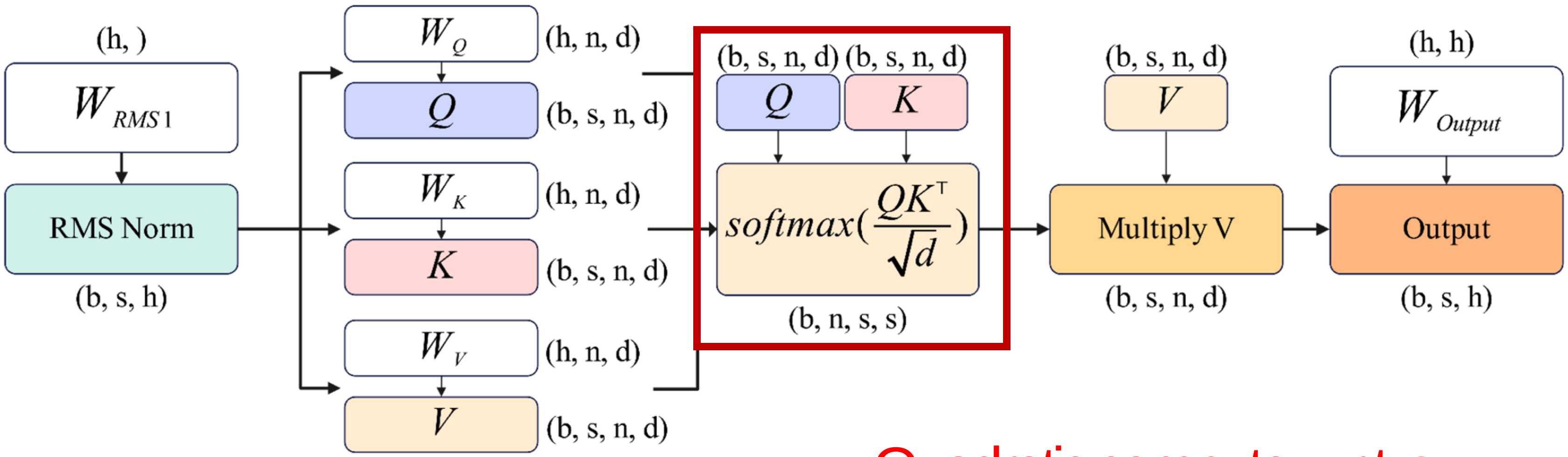
- Scheduling
  - Continuous batching
  - Chunked prefill
  - Disaggregated prefill and decoding
- Speculative Decoding
- Address the memory bottleneck of KV Cache
  - New attention mechanisms: paged, sparse, etc.
  - Sparse KV cache
- Kernel optimizations

large b

# Large Language Models

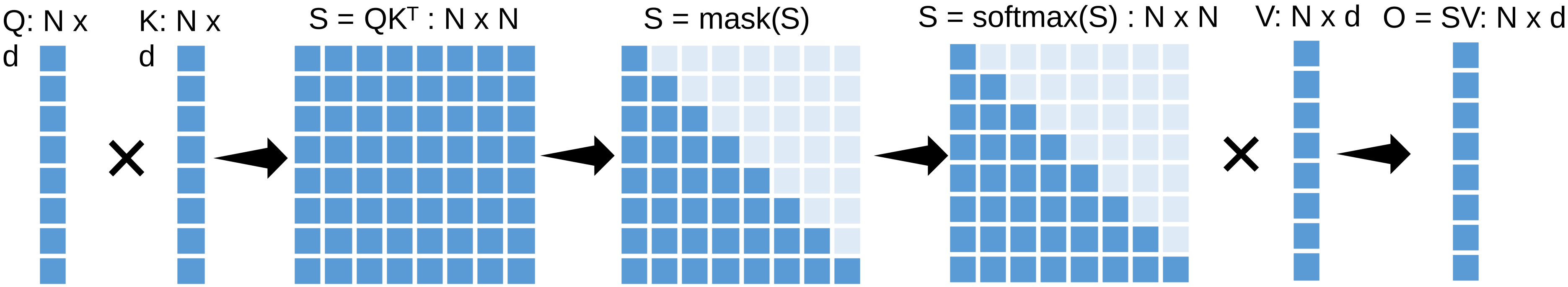
- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - **Flash attention ← come back to this later next week**
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

# The Rest of bottleneck




- Quadratic compute w.r.t. s
- Quadratic memory w.r.t. s

# Attention: $O = \mathrm{Softmax}(QK^T)\ V$



# Which would hit bottleneck first

Compute: $4bs^2h$

Memory: $bs^2n$

Assume b = 1K, n =32, h = 4k

- Assume s = 4K (sequence length)
- compute: 4 * 1K * 4K * 4K * 4K
  - = 256 Tflops
- memory: 1K * 4K * 4K * 32
  - = 512G bytes

| | H100 SXM |
|---|---|
| FP64 | 34 teraFLOPS |
| FP64 Tensor Core | 67 teraFLOPS |
| FP32 | 67 teraFLOPS |
| TF32 Tensor Core* | 989 teraFLOPS |
| BFLOAT16 Tensor Core* | 1,979 teraFLOPS |
| FP16 Tensor Core* | 1,979 teraFLOPS |
| FP8 Tensor Core* | 3,958 teraFLOPS |
| INT8 Tensor Core* | 3,958 TOPS |
| GPU Memory | 80GB |
| GPU Memory Bandwidth | 3.35TB/s |
| Decoders | 7 NVDEC<br>7 JPEG |

GPU memory is more scarce than compute at this moment

# The Large [bnss] matrix makes thing even worse

**Algorithm 0** Standard Attention Implementation

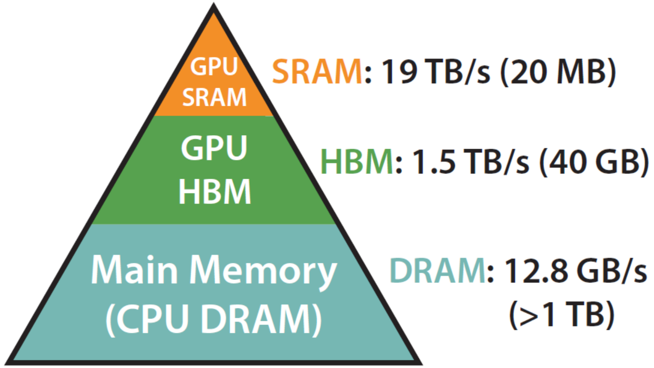
**Require:** Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

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 $\mathbf{O}$.

**Additional Challenges**:

- Repeated reads/writes from HBM -> SRAM of the large bnss matrix

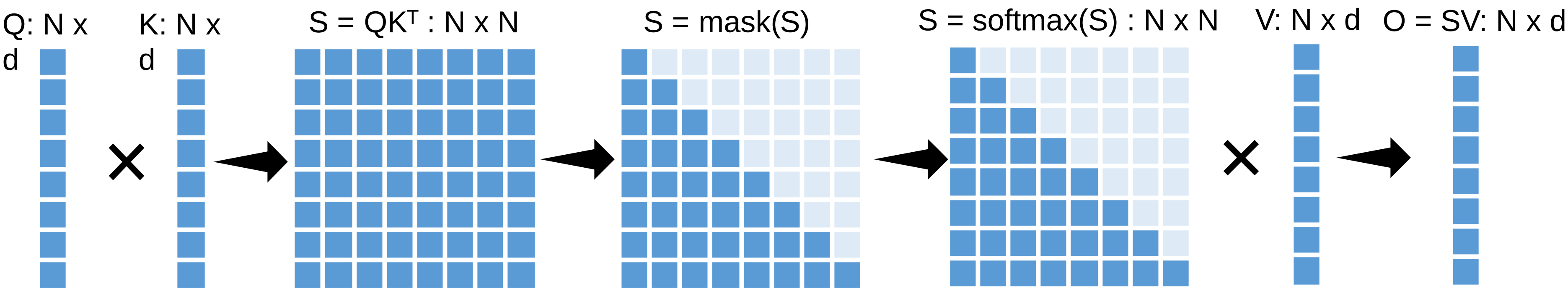
# Revisit: GPU Memory Hierarchy




Memory Hierarchy with
Bandwidth & Memory Size


now is 3

# Problem: How to tile softmax?




**Challenges: We must avoid materializing S while**

- Compute softmax reduction O w/o access to NxN at forward
- Compute backward even without saving the NxN softmax forward activations

# Recall in the Matmul Class

```
dram float A[n/v1][n/v3][v1][v3];
dram float B[n/v2][n/v3][v2][v3];
dram float C[n/v1][n/v2][v1][v2];

for (int i = 0; i < n/v1; ++i) {
   for (int j = 0; j < n/v2; ++j) {
      register float c[v1][v2] = 0;
      for (int k = 0; k < n/v3; ++k) {
        register float a[v1][v3] = A[i][k];
        register float b[v2][v3] = B[j][k];
        c += dot(a, b.T);
      }
      C[i][j] = c;
   }
}
```

- We were able to compute the final results without fully materialzing the input / output

# The Large [bnss] matrix makes thing even worse

**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 $\mathbf{O}$.

Core Q: How to tile softmax?

# How to Implement Softmax

**Algorithm 1** Naive softmax

$d_0 \leftarrow 0$
**for** $j \leftarrow 1, V$ **do**
    $d_j \leftarrow d_{j-1} + e^{x_j}$
**end for**
**for** $i \leftarrow 1, V$ **do**
    $y_i \leftarrow \frac{e^{x_i}}{d_V}$
**end for**

Problem

- Can easily go overflow because of sum (e^x)

# Safe Softmax

$$y_i = \frac{e^{x_i - \max\limits_{k=1}^{V} x_k}}{\sum\limits_{j=1}^{V} e^{x_j - \max\limits_{k=1}^{V} x_k}}$$

**Algorithm 2** Safe softmax

$m_0 \leftarrow -\infty$
**for** $k \leftarrow 1, V$ **do**
    $m_k \leftarrow \max(m_{k-1}, x_k)$
**end for**
$d_0 \leftarrow 0$
**for** $j \leftarrow 1, V$ **do**
    $d_j \leftarrow d_{j-1} + e^{x_j - m_V}$
**end for**
**for** $i \leftarrow 1, V$ **do**
    $y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$
**end for**

# Can we fuse?

Create alternative sequence

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

With:
$d_V' = d_V$

**Algorithm 2** Safe softmax

$m_0 \leftarrow -\infty$
**for** $k \leftarrow 1, V$ **do**
  $m_k \leftarrow \max(m_{k-1}, x_k)$
**end for**
$d_0 \leftarrow 0$
**for** $j \leftarrow 1, V$ **do**
  $d_j \leftarrow d_{j-1} + e^{x_j - m_V}$
**end for**
**for** $i \leftarrow 1, V$ **do**
  $y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$
**end for**

# Further Note:

Create alternative sequence

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

With:

$$d'_V = d_V$$

The sequence exhibits recurrence

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

# Online, Safe Softmax

**Algorithm 3** Safe softmax with online normalizer calculation

$m_0 \leftarrow -\infty$
$d_0 \leftarrow 0$
**for** $j \leftarrow 1, V$ **do**
    $m_j \leftarrow \max\left(m_{j-1}, x_j\right)$
    $d_j \leftarrow d_{j-1} \times e^{m_{j-1}-m_j} + e^{x_j-m_j}$
**end for**
**for** $i \leftarrow 1, V$ **do**
    $\boxed{y_i \leftarrow \frac{e^{x_i-m_V}}{d_V}}$
**end for**

Q： can we further fuse these two loops?

# Self attention v2: with Online Safe Softmax

NOTATIONS

$Q[k,:]$: the $k$-th row vector of $Q$ matrix.

$K^T[:,i]$: the $i$-th column vector of $K^T$ matrix.

$O[k,:]$: the $k$-th row of output $O$ matrix.

$V[i,:]$: the $i$-th row of $V$ matrix.

$\{\boldsymbol{o}_i\}$: $\sum_{j=1}^{i} a_j V[j,:]$, a row vector storing partial aggregation result $A[k,:i] \times V[:i,:]$

BODY

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

$$
\begin{aligned}
x_i &\leftarrow Q[k,:]\, K^T[:,i] \\
m_i &\leftarrow \max\left(m_{i-1}, x_i\right) \\
d'_i &\leftarrow d'_{i-1}\, e^{m_{i-1}-m_i} + e^{x_i - m_i}
\end{aligned}
$$

**end**

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

Q1: can we further fuse these two loops?

Q2: How to get x_i?

$$
\begin{aligned}
a_i &\leftarrow \frac{e^{\boxed{x_i} - m_N}}{d'_N} \\
\boldsymbol{o}_i &\leftarrow \boldsymbol{o}_{i-1} + a_i\, V[i,:]
\end{aligned}
$$

**end**

$$O[k,:] \leftarrow \boldsymbol{o}_N$$

# Self attention

NOTATIONS
$Q[k,:]$: the $k$-th row vector of $Q$ matrix.
$K^T[:,i]$: the $i$-th column vector of $K^T$ matrix.
$O[k,:]$: the $k$-th row of output $O$ matrix.
$V[i,:]$: the $i$-th row of $V$ matrix.
$\{\boldsymbol{o}_i\}$: $\sum_{j=1}^{i} a_j V[j,:]$, a row vector storing partial aggregation result $A[k,:i] \times V[:i,:]$

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

$$\begin{aligned} x_i &\leftarrow Q[k,:]\, K^T[:,i] \\ m_i &\leftarrow \max\left(m_{i-1}, x_i\right) \\ d_i' &\leftarrow d_{i-1}' \, e^{m_{i-1}-m_i} + e^{x_i - m_i} \end{aligned}$$

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

$$\begin{aligned} a_i &\leftarrow \frac{e^{x_i - m_N}}{d_N'} \\ \boxed{\boldsymbol{o}_i} &\leftarrow \boldsymbol{o}_{i-1} + a_i \, V[i,:] \end{aligned}$$

**end**

$$O[k,:] \leftarrow \boldsymbol{o}_N$$

in self attention, we just want o, not a

$$\boldsymbol{o}_i := \sum_{j=1}^{i} \left( \frac{e^{x_j - m_N}}{d_N'} V[j,:] \right)$$

Create alternative sequence with $o_N = o'_N$

$$\boldsymbol{o}_i' := \left( \sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d_i'} V[j,:] \right)$$

# Derive the Recurrence

$$
\begin{aligned}
\boldsymbol{o}_i' &= \sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d_i'} V[j,:] \\
&= \left( \sum_{j=1}^{i-1} \frac{e^{x_j - m_i}}{d_i'} V[j,:] \right) + \frac{e^{x_i - m_i}}{d_i'} V[i,:] \\
&= \left( \sum_{j=1}^{i-1} \frac{e^{x_j - m_{i-1}}}{d_{i-1}'} \frac{e^{x_j - m_i}}{e^{x_j - m_{i-1}}} \frac{d_{i-1}'}{d_i'} V[j,:] \right) + \frac{e^{x_i - m_i}}{d_i'} V[i,:] \\
&= \left( \sum_{j=1}^{i-1} \frac{e^{x_j - m_{i-1}}}{d_{i-1}'} V[j,:] \right) \frac{d_{i-1}'}{d_i'} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d_i'} V[i,:] \\
&= \boldsymbol{o}_{i-1}' \frac{d_{i-1}' \, e^{m_{i-1} - m_i}}{d_i'} + \frac{e^{x_i - m_i}}{d_i'} V[i,:]
\end{aligned}
$$

Insight: $o_i'$ only depends on: $o_{i-1}', d_{i-1}', d_i', m_i, m_{i-1}$

# Finally: Flash Attention

1. we only read $x_1$ (Q, K) once
2. We never materialize the full S
3. we never materialize the full a (softmax(S))

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

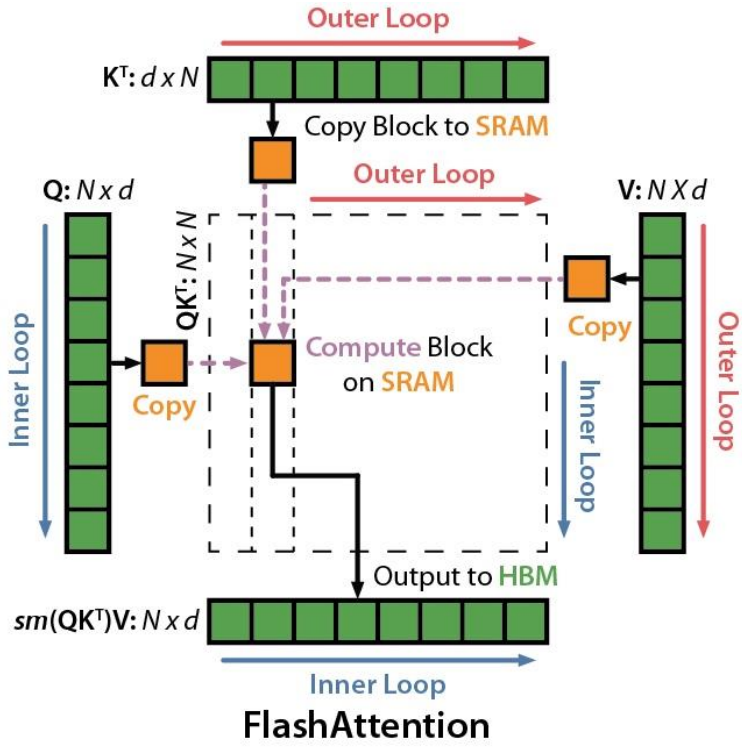
$$
\begin{aligned}
x_i &\leftarrow Q[k,:]\,K^T[:,i] \\
m_i &\leftarrow \max\left(m_{i-1}, x_i\right) \\
d_i' &\leftarrow d_{i-1}'\, e^{m_{i-1}-m_i} + e^{x_i - m_i} \\
\boldsymbol{o}_i' &\leftarrow \boldsymbol{o}_{i-1}' \frac{d_{i-1}'\, e^{m_{i-1}-m_i}}{d_i'} + \frac{e^{x_i - m_i}}{d_i'} V[i,:]
\end{aligned}
$$

**end**

$$O[k,:] \leftarrow \boldsymbol{o}_N'$$

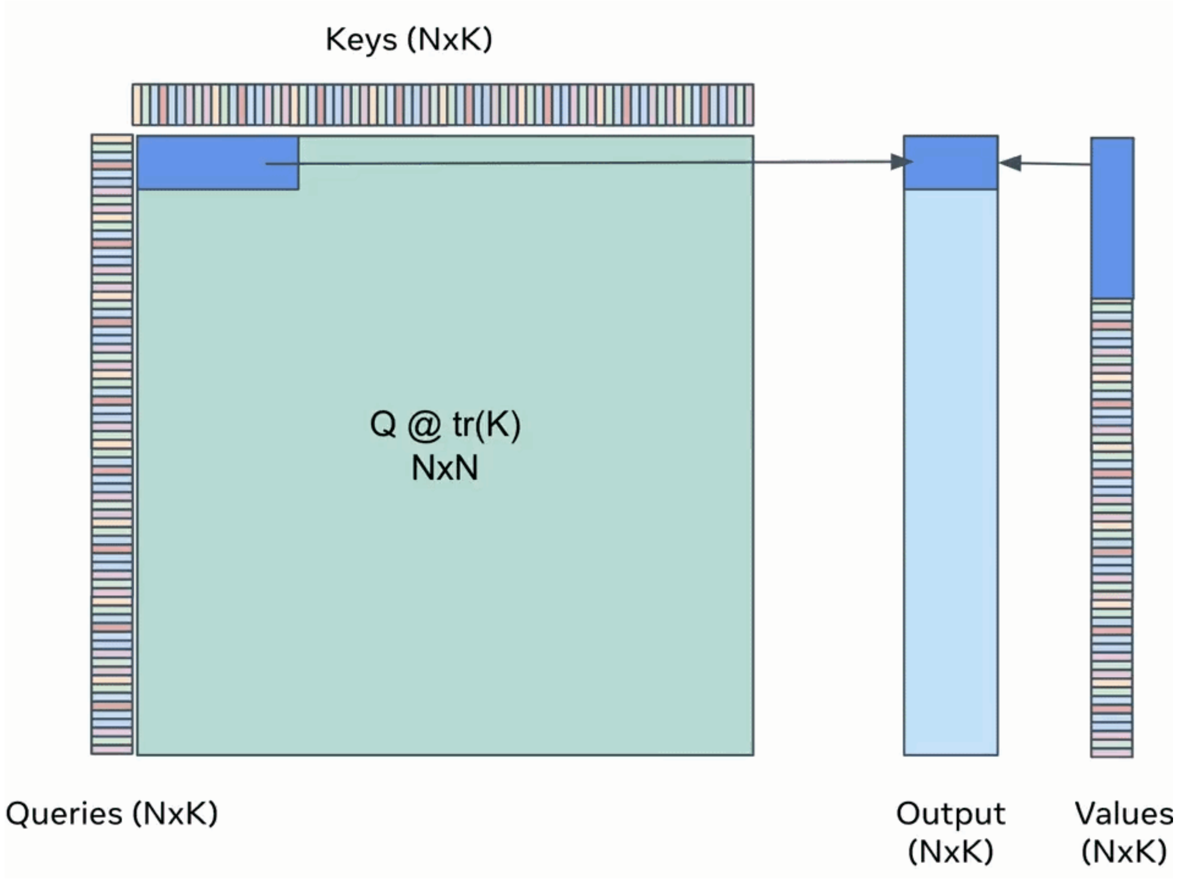
# Tiling: Decompose Large Softmax into smaller ones by Scaling

1. Load inputs by blocks from global to shared memory
2. On chip, compute attention output wrt the block
3. Update output in device memory by scaling



# + Tiling

1. Load inputs by blocks from global to shared memory
2. On chip, compute attention output wrt the block
3. Update output in device memory by scaling

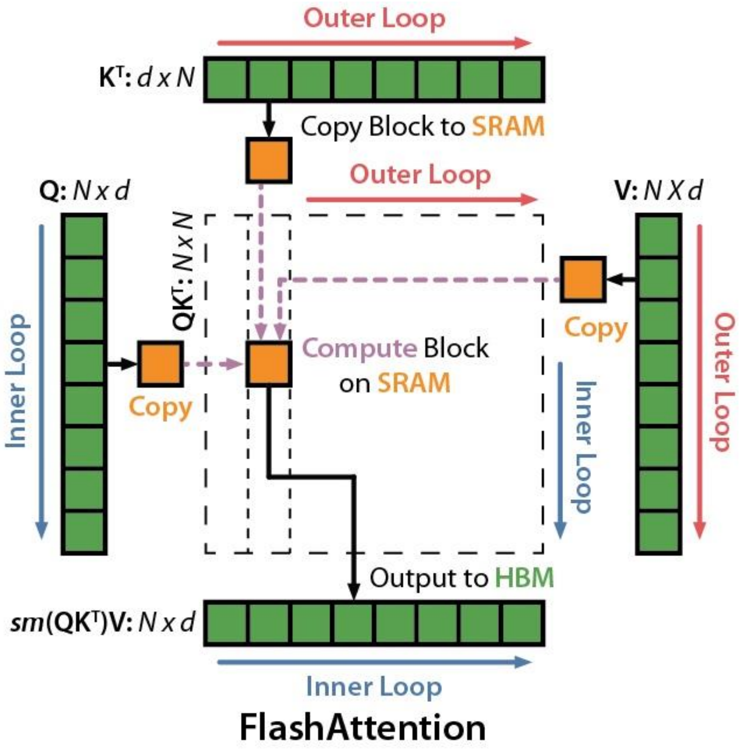



Animation credit: Francisco Massa

# Recomputation: Backward Pass

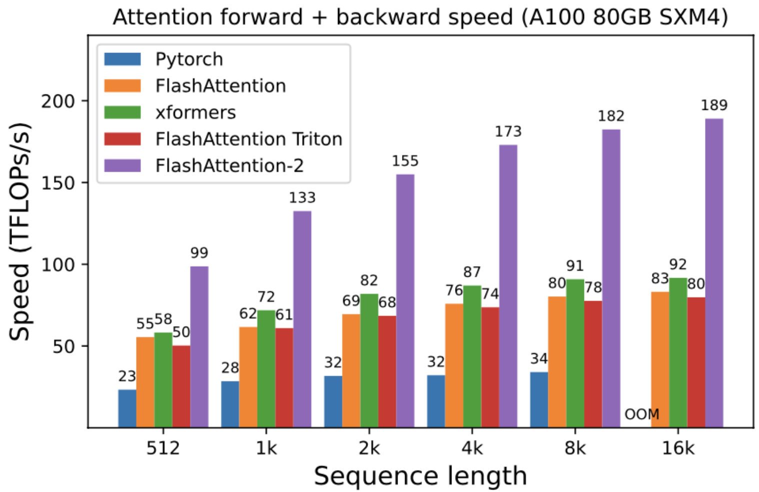
By storing softmax normalization factors from forward (size N), recompute attention in the backward from inputs in

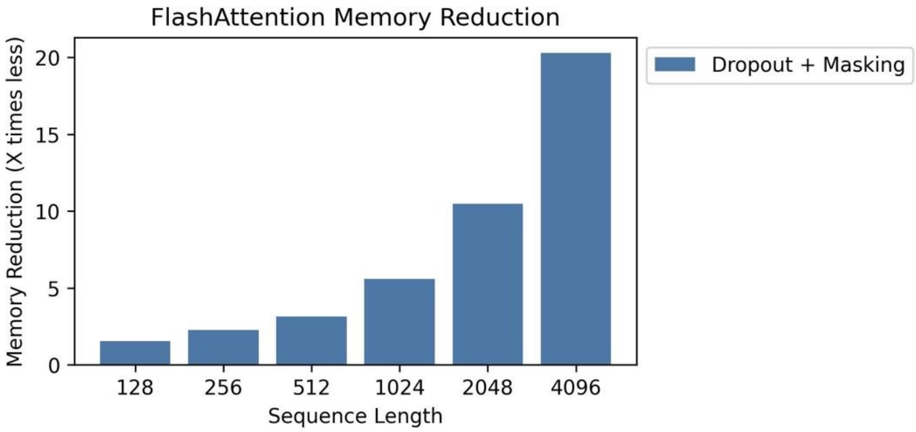
| Attention | Standard | FlashAttention |
|---|---|---|
| GFLOPs | 66.6 | 75.2 |
| Global mem access | 40.3 GB | 4.4 GB |
| Runtime | 41.7 ms | 7.3 ms |




**Speed up backward pass with with increased FLOPs!**

# FlashAttention: 2-4x speedup, 10-20x memory reduction



# High-level GISTs of FA

- It completely avoid materializing the large S of size $bs^2n$
- It will make the compute $4bs^2h$ even worse
- It will greatly reduce memory movement between memory hierarchy

Why FA is even greater win than thought -- Cascaded effect:

- Because it saves the memory of $bs^2n$, it enables two possibilities
  - Can train with a large b (in fact, pre-FA, most train with b=1)
    - -> high AI
  - Can turn off gradient checkpointing
    - -> No need to pay extra forward (25% flops)

# FA2 and FA3

- More kernel-level optimizations over FA
  - More fusion
  - Improved memory access patterns

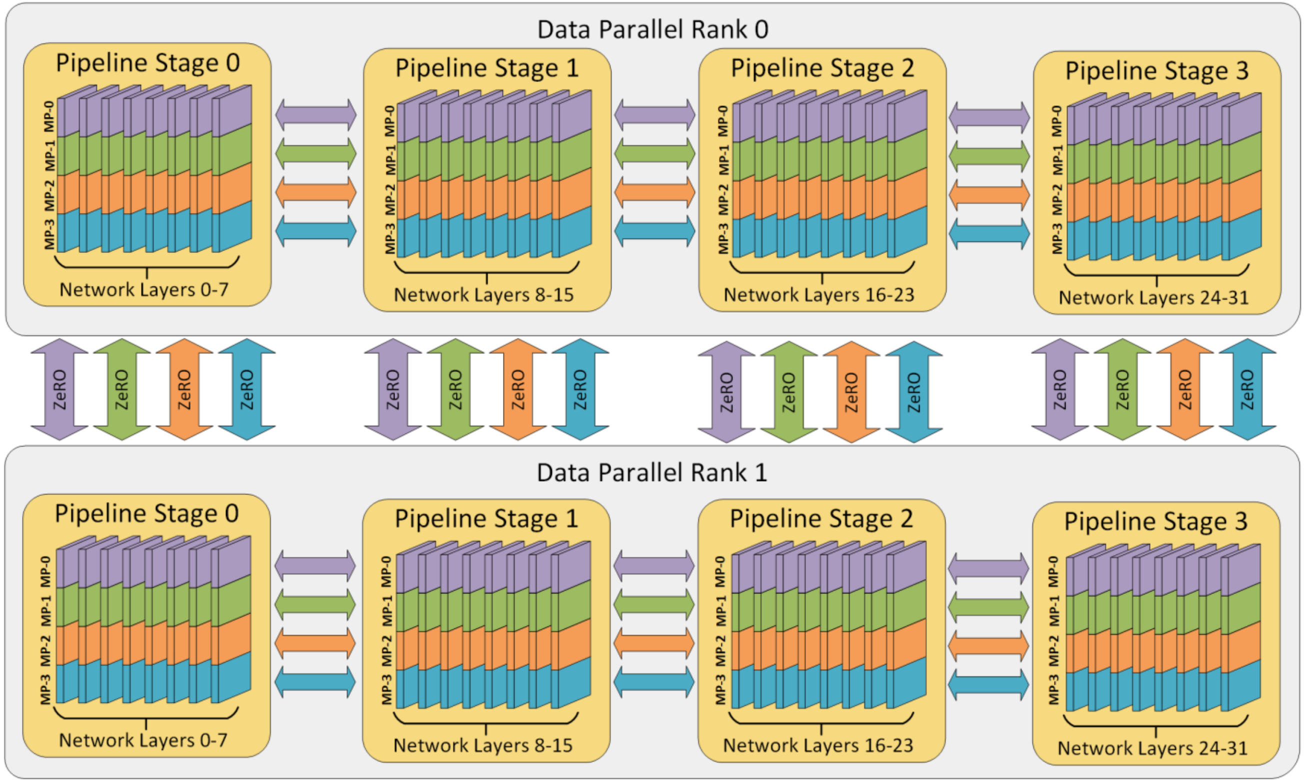
- Discussion: why FA took off?

large b

# Large Language Models

- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - **Flash attention ← come back to this later next week**
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

# In Practice: How LLMs are trained today



# Summary: How LLMs are trained today

- Outer Loop 1:
  - Inter-op parallelism + 1F1B
- Outer Loop 2: Intra-op parallelism based on model architecture
  - Zero-2 / Zero-3 + data parallelism
  - Megatron-LM tensor parallelism or Expert parallelism
- Outer Loop 3:
  - Gradient checkpointing and recomputation at backward
- Inner Loop 4:
  - Graph fusion
- Inner Loop 5:
  - Operator-level optimization: tiling, flash attention, etc.

# Deepseek-v3 overview

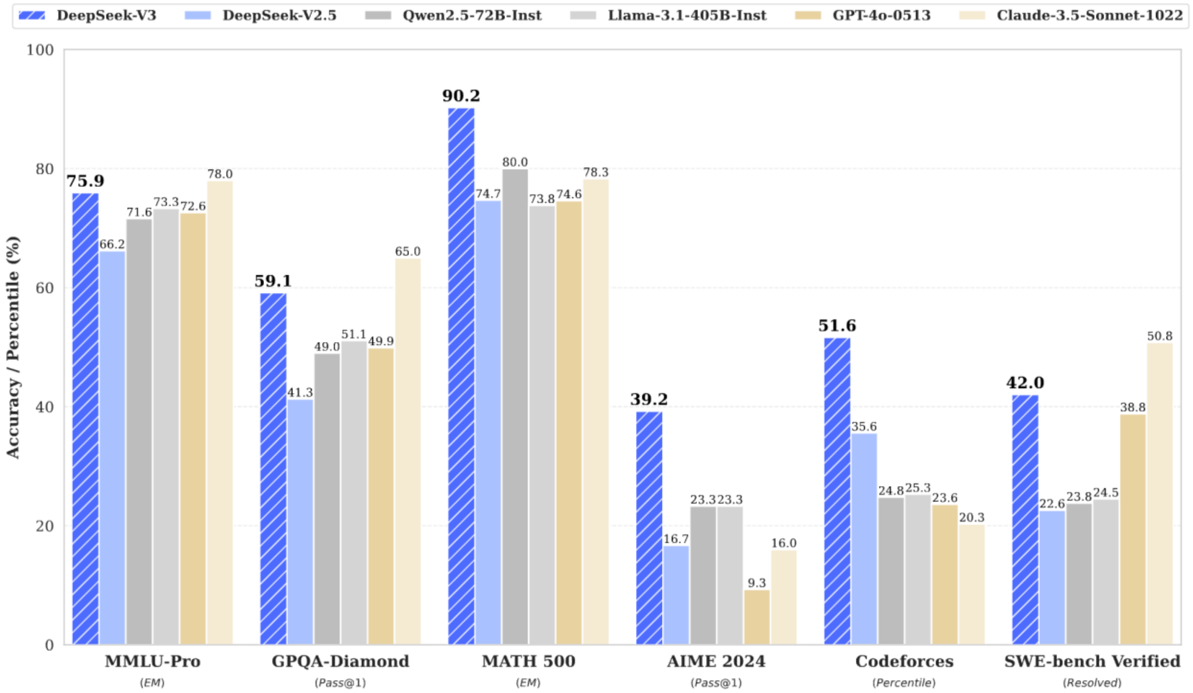



Figure 1 | Benchmark performance of DeepSeek-V3 and its counterparts.

# Deepseek-v3 overview

| **Training Costs** | **Pre-Training** | **Context Extension** | **Post-Training** | **Total** |
|---|---|---|---|---|
| in H800 GPU Hours | 2664K | 119K | 5K | 2788K |
| in USD | $5.328M | $0.238M | $0.01M | $5.576M |

Caveats: very good marketing because the majority (100 – 1000x more) of the GPU hours ($) are spent on the **planning phase**, not the actual training phase

# Outline

Deepseek-v3

- Model architecture
- System optimizations

Deepseek-r1

- RL
- Maybe next course 🙂?

# Deepseek is an MoE with 256 Experts

First 3 layers: dense
Other layers: MoE

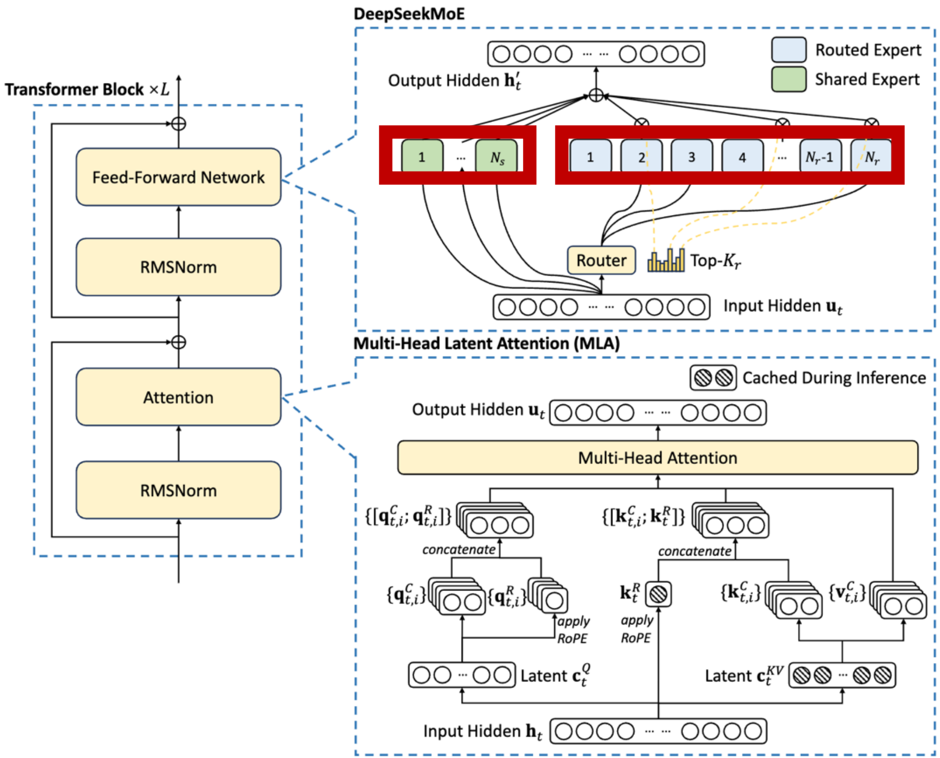



Figure 2 | Illustration of the basic architecture of DeepSeek-V3. Following DeepSeek-V2, we adopt MLA and DeepSeekMoE for efficient inference and economical training.

$N_s = 1$
$N_r = 256$
#activated = 8

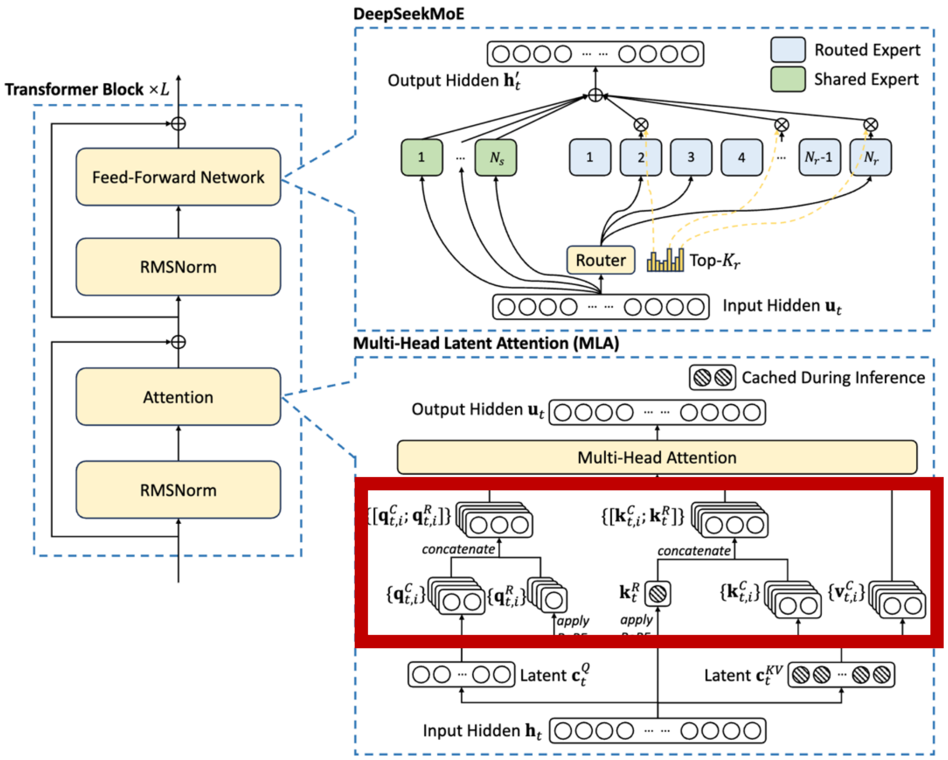
# Cons of MoE

Large number of Parameters -> Need expert parallelism

Need expert parallelism -> expert balancing

Will come back to this later

# Deepseek is an MoE with 256 Experts




MLA: multi-head latent attention

Figure 2 | Illustration of the basic architecture of DeepSeek-V3. Following DeepSeek-V2, we adopt MLA and DeepSeekMoE for efficient inference and economical training.

# MLA: multi-head latent attention

$$\mathbf{q}_t = W^Q \mathbf{h}_t, \quad (1)$$
$$\mathbf{k}_t = W^K \mathbf{h}_t, \quad (2)$$
$$\mathbf{v}_t = W^V \mathbf{h}_t, \quad (3)$$

$W^Q, W^K, W^V \in \mathbb{R}^{d_h n_h \times d}$ $\qquad$ $\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t \in \mathbb{R}^{d_h n_h}$

$$[\mathbf{q}_{t,1}; \mathbf{q}_{t,2}; \ldots; \mathbf{q}_{t,n_h}] = \mathbf{q}_t,$$
$$[\mathbf{k}_{t,1}; \mathbf{k}_{t,2}; \ldots; \mathbf{k}_{t,n_h}] = \mathbf{k}_t,$$
$$[\mathbf{v}_{t,1}; \mathbf{v}_{t,2}; \ldots; \mathbf{v}_{t,n_h}] = \mathbf{v}_t,$$
$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h}}\right) \mathbf{v}_{j,i},$$
$$\mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \ldots; \mathbf{o}_{t,n_h}],$$

$$\mathbf{c}_t^Q = W^{DQ} \mathbf{h}_t,$$
$$[\mathbf{q}_{t,1}^C; \mathbf{q}_{t,2}^C; \ldots; \mathbf{q}_{t,n_h}^C] = \mathbf{q}_t^C = W^{UQ} \mathbf{c}_t^Q,$$
$$[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; \ldots; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R = \mathrm{RoPE}(W^{QR} \mathbf{c}_t^Q),$$
$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R],$$
$$\boxed{\mathbf{c}_t^{KV}} = W^{DKV} \mathbf{h}_t,$$
$$[\mathbf{k}_{t,1}^C; \mathbf{k}_{t,2}^C; \ldots; \mathbf{k}_{t,n_h}^C] = \mathbf{k}_t^C = W^{UK} \mathbf{c}_t^{KV},$$
$$\boxed{\mathbf{k}_t^R} = \mathrm{RoPE}(W^{KR} \mathbf{h}_t),$$
$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R],$$
$$[\mathbf{v}_{t,1}^C; \mathbf{v}_{t,2}^C; \ldots; \mathbf{v}_{t,n_h}^C] = \mathbf{v}_t^C = W^{UV} \mathbf{c}_t^{KV},$$
$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}}\right) \mathbf{v}_{j,i}^C,$$
$$\mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \ldots; \mathbf{o}_{t,n_h}],$$

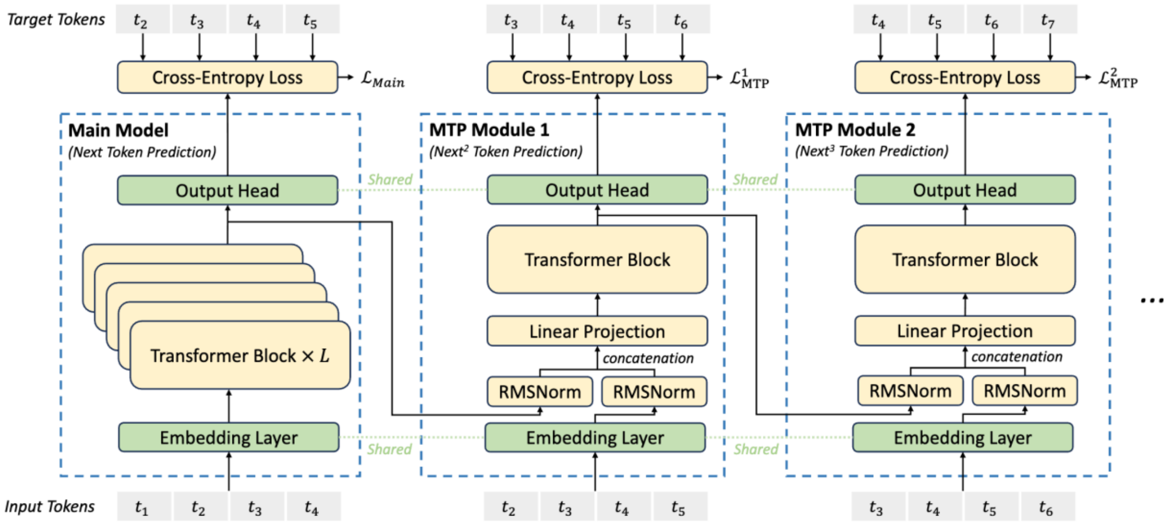
# MLA: multi-head latent attention

| **Attention Mechanism** | **KV Cache per Token (# Element)** | **Capability** |
|---|---|---|
| Multi-Head Attention (MHA) | $2n_h d_h l$ | Strong |
| Grouped-Query Attention (GQA) | $2n_g d_h l$ | Moderate |
| Multi-Query Attention (MQA) | $2d_h l$ | Weak |
| MLA (Ours) | $(d_c + d_h^R)l \approx \frac{9}{2} d_h l$ | Stronger |

# Multi-token Prediction: 2 claims

Training with MTP improves pretraining

Training with MTP gives a free speculation head

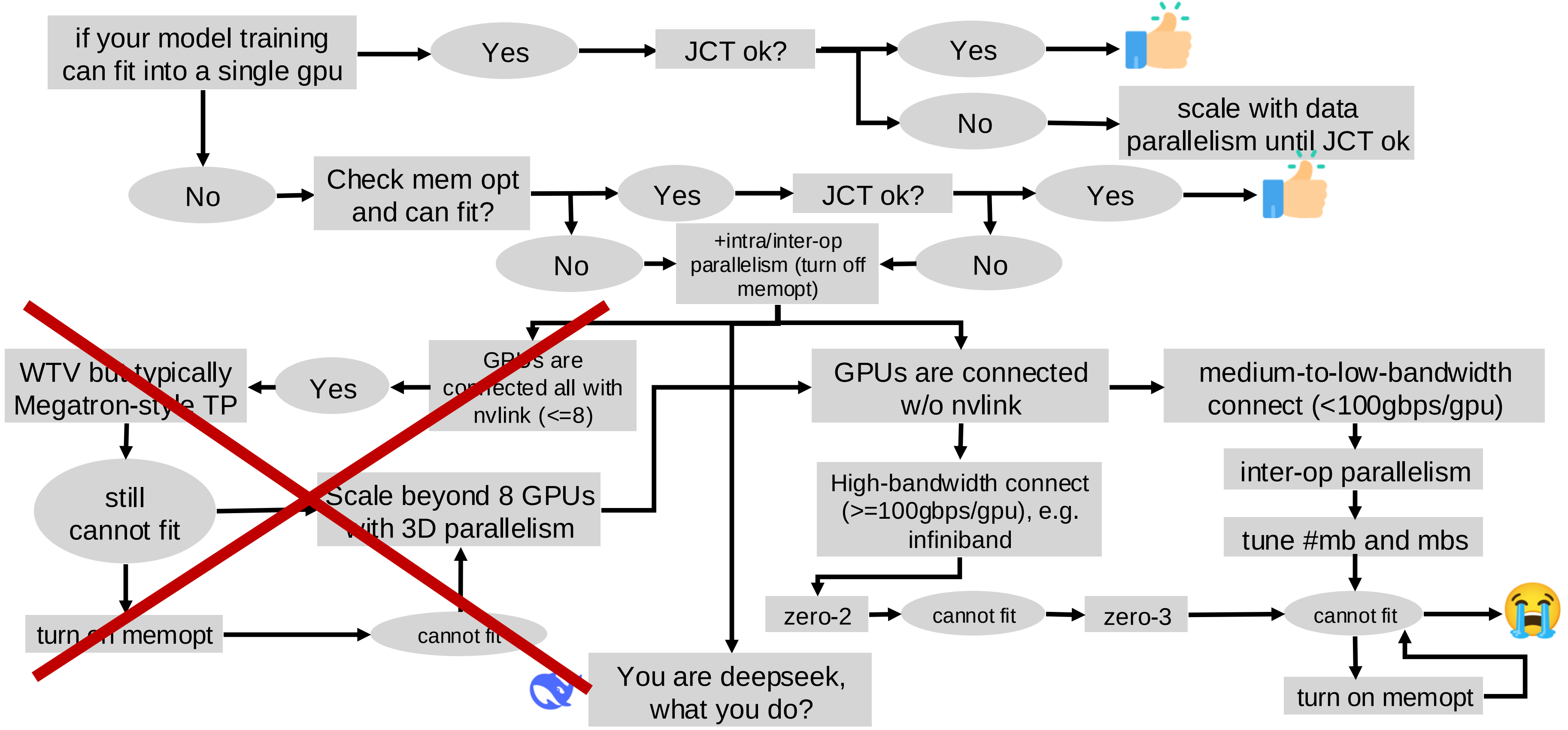


# Outline

Deepseek-v3

- Model architecture
- **System optimizations**

# Hao’s Ultimate Guide



# Cons of MoE

**Large number of Parameters -> Need expert parallelism**

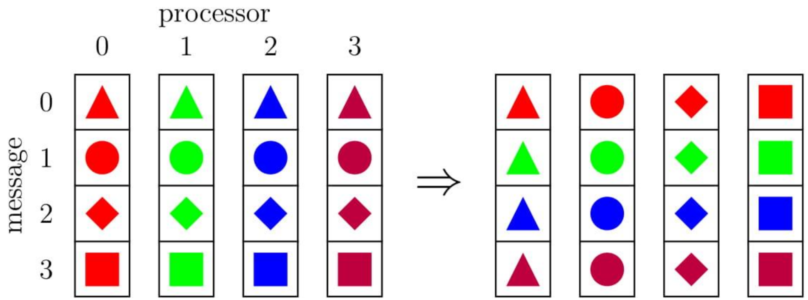
Need expert parallelism -> expert balancing

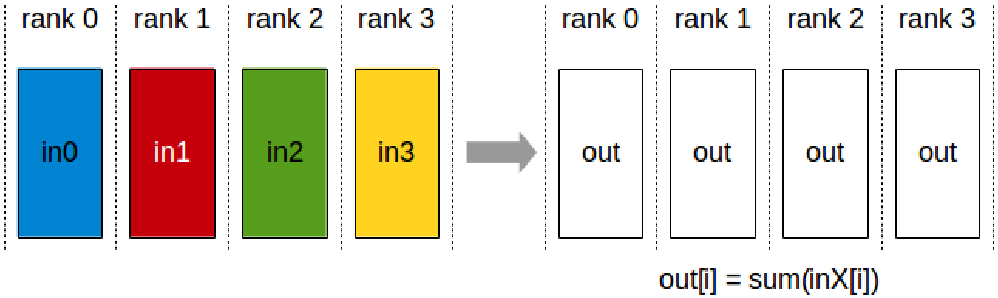
They only have H800 -> no tensor parallelism

-> pipeline parallelism + data parallelism + expert parallelism

# They still cannot use TP anyway (even without MoE)

All-reduce is Nx more complex than all-to-all



# Cons of MoE

**Large number of Parameters -> Need expert parallelism**

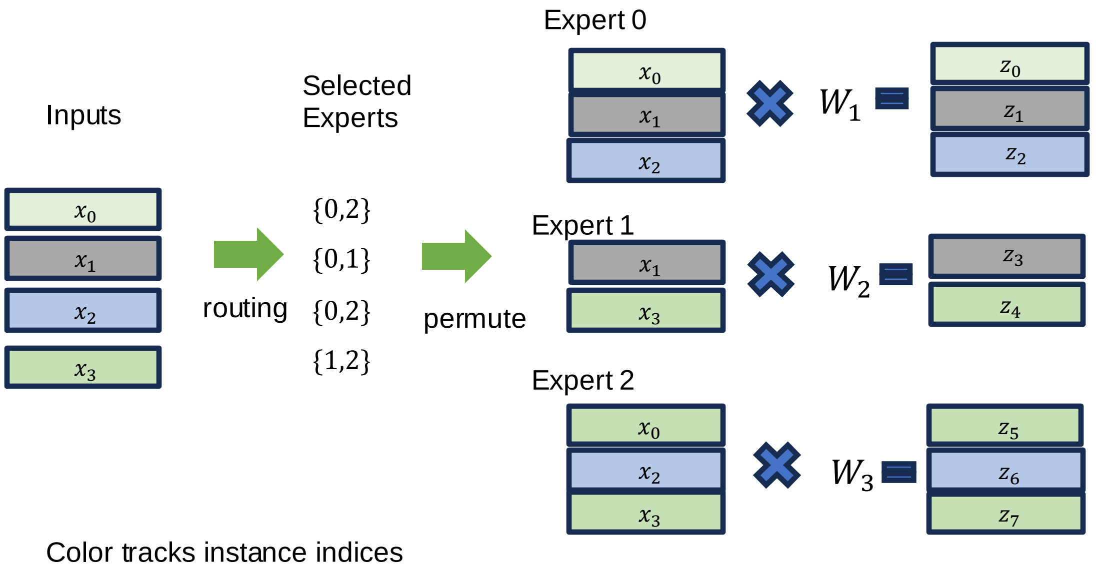
**Need expert parallelism -> expert balancing**

They only have H800 -> no tensor parallelism

-> pipeline parallelism + data parallelism + expert parallelism

We'll come back to this later

# Potential Problems of MoE?




*Gale et.al* MegaBlocks: Efficient Sparse Training with Mixture-of-Experts

# Fixing Expert Parallelism: Loss-free balancing

Typical: add a expert balancing loss

Deepseek's arguments:

- This loss hurts the pretraining
- Use a auxiliary-loss-free bias term

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} + b_i \in \operatorname{Topk}(\{s_{j,t} + b_j | 1 \leqslant j \leqslant N_r\}, K_r), \\ 0, & \text{otherwise.} \end{cases}$$

# Fixing Expert Parallelism: Loss-free balancing

Sequence level balancing loss

- encourages the expert load on each sequence to be balanced.
- Hao's comments: ???

$$\mathcal{L}_{\text{Bal}} = \alpha \sum_{i=1}^{N_r} f_i P_i,$$

$$f_i = \frac{N_r}{K_r T} \sum_{t=1}^{T} \mathbb{1}\left(s_{i,t} \in \operatorname{Topk}(\{s_{j,t} | 1 \leqslant j \leqslant N_r\}, K_r)\right),$$

$$s'_{i,t} = \frac{s_{i,t}}{\sum_{j=1}^{N_r} s_{j,t}},$$

$$P_i = \frac{1}{T} \sum_{t=1}^{T} s'_{i,t},$$

# Parallelism

“16 way pipeline parallelism

64 way expert parallelism

the rest is ZERO data parallelism”

Assuming 2048 GPUs (which they claimed):

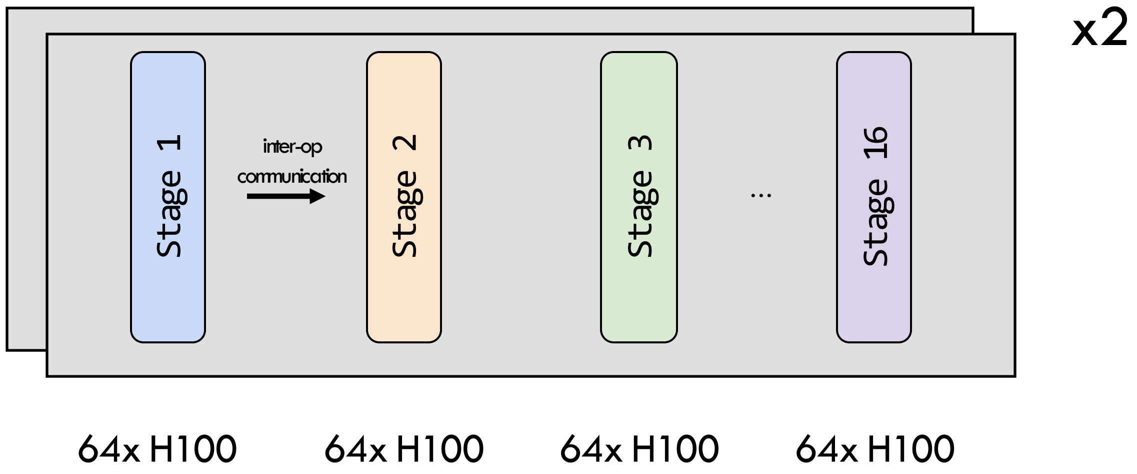
* 16 PP x 64EP x 2DP = 2048 GPUs

# Parallelism: 16-way PP

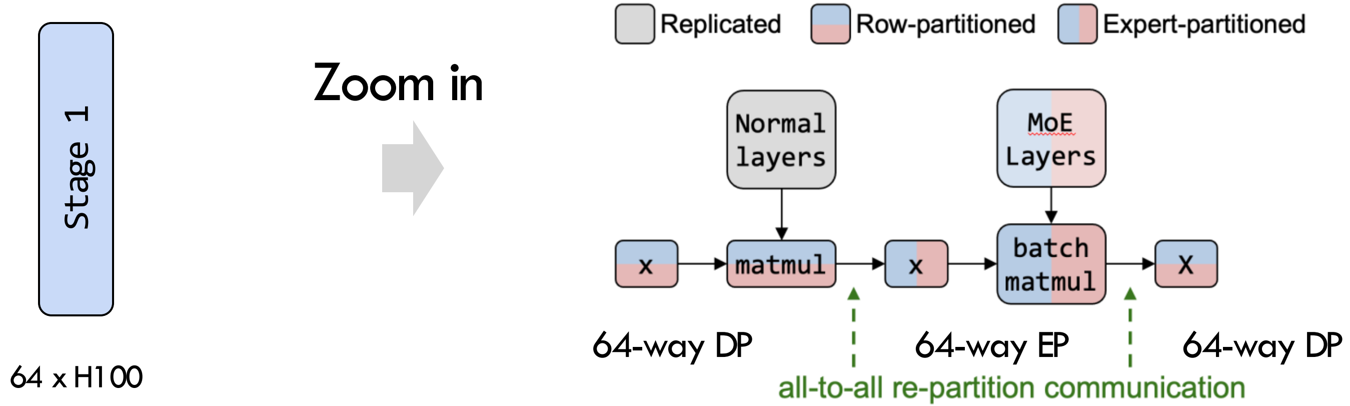
16 way pipeline parallelism

64 way expert parallelism

2-way ZERO-1 data parallelism



# Parallelism: 64-way Expert Parallelism

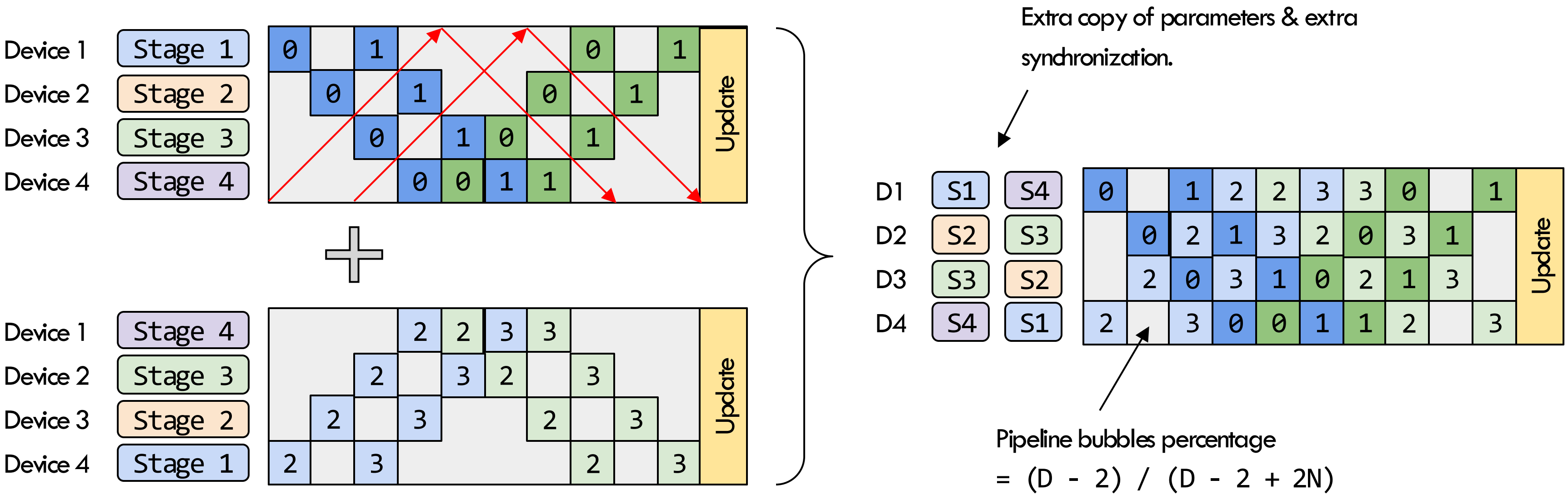


# Pipeline Parallelism Optimization

- DualPipe
- Communication and Computation Overlapping
- All-to-all kernel

# Recap: Chimera

**Idea:** Store bi-directional stages and combine bidirectional pipeline to further reduce pipeline bubbles.




Pipeline bubbles percentage

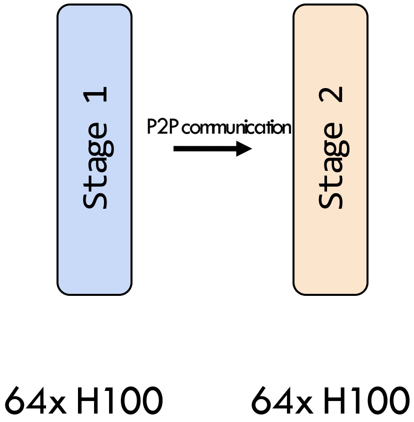
= (D - 2) / (D - 2 + 2N)

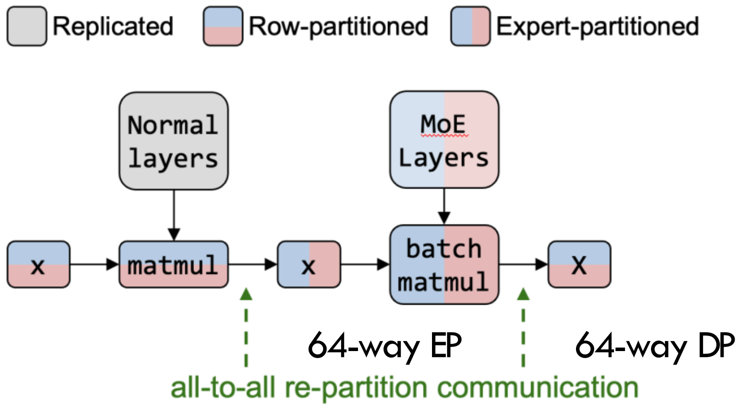
with D devices and N micro-batches.

---

Li, Shigang, and Torsten Hoefler. "Chimera: efficiently training large-scale neural networks with bidirectional pipelines." *SC 21*.

# Compute&Communication Pattern




- attention -> all-to-all dispatch (1st) -> MLP/expert -> all-to-all combine (2nd) -> P2P communication

# Overlapping Opportunities with a Reverse Pipeline

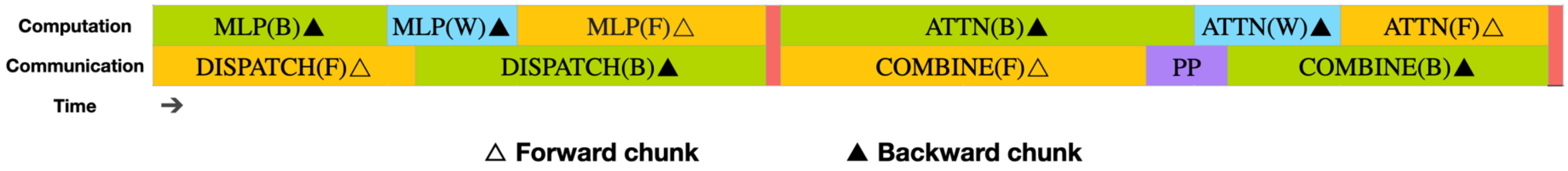


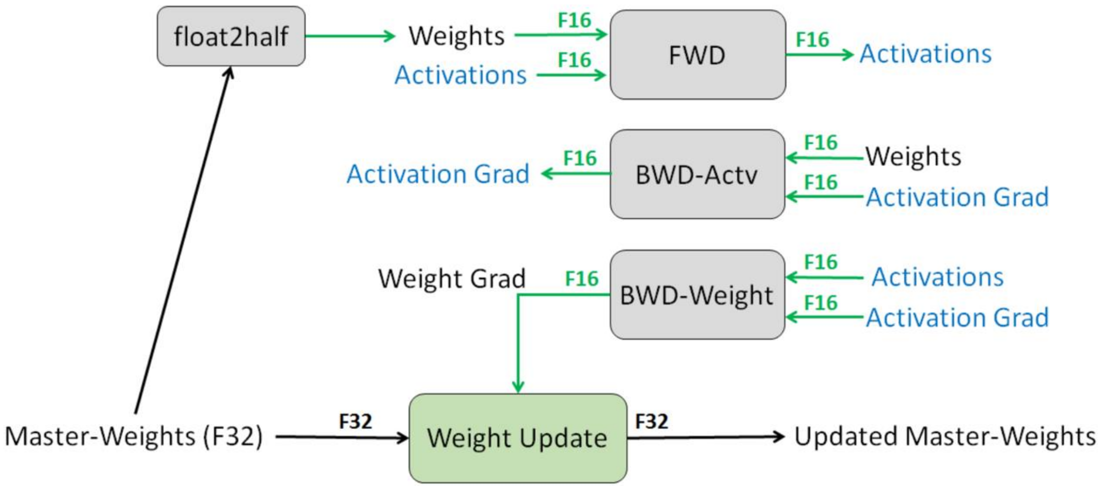

Figure 4 | Overlapping strategy for a pair of individual forward and backward chunks (the boundaries of the transformer blocks are not aligned). Orange denotes forward, green denotes "backward for input", blue denotes "backward for weights", purple denotes PP communication, and red denotes barriers. Both all-to-all and PP communication can be fully hidden.

# All-to-All Communication Kernel: Very cool

**Undefined-behavior PTX usage**

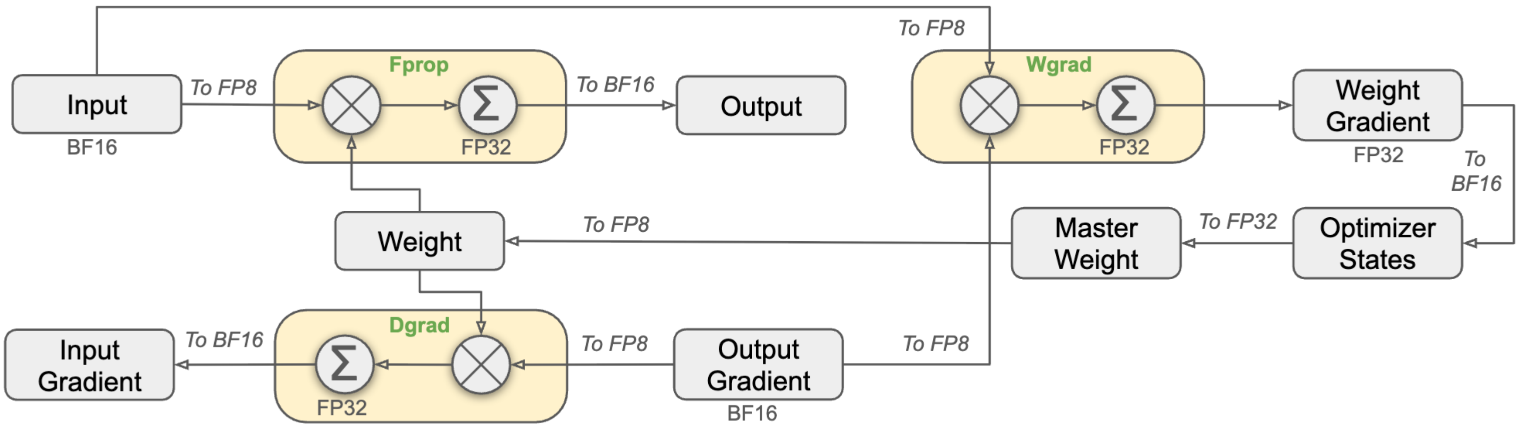
- For extreme performance, we discover and use an undefined-behavior PTX usage: using read-only PTX `ld.global.nc.L1::no_allocate.L2::256B` to **read volatile data**. The PTX modifier `.nc` indicates that a non-coherent cache is used. But the correctness is tested to be guaranteed with `.L1::no_allocate` on Hopper architectures, and performance will be much better. The reason we guess may be: the non-coherent cache is unified with L1, and the L1 modifier is not just a hint but a strong option, so that the correctness can be guaranteed by no dirty data in L1.
- Initially, because NVCC could not automatically unroll volatile read PTX, we tried using `__ldg` (i.e., `ld.nc`). Even compared to manually unrolled volatile reads, it was significantly faster (likely due to additional compiler optimizations). However, the results could be incorrect or dirty. After consulting the PTX documentation, we discovered that L1 and non-coherent cache are unified on Hopper architectures. We speculated that `.L1::no_allocate` might resolve the issue, leading to this discovery.
- If you find kernels not working on some other platforms, you may add `DISABLE_AGGRESSIVE_PTX_INSTRS=1` to `setup.py` and disable this, or file an issue.

# Traditional FP16-FP32 mixed precision Training




- Master copy (fp32) = 4 *M
- Grad (fp16) = 2 * M
- Running copy (fp16) = 2 * M
- Adam mean and variance (fp32) = 2 * 4 *M
- **Rule the thumb: (4 + 2 + 2 + 4 + 4) N = 16N memory for an LLM**

# Deepseeks' FP8-FP16-FP32 mix precision training




- Master copy (fp32) = 4 *M
- Grad (fp16) = 2 * M
- Running copy (fp16) = M
- Adam mean and variance (fp32) = 2 * 2 *M

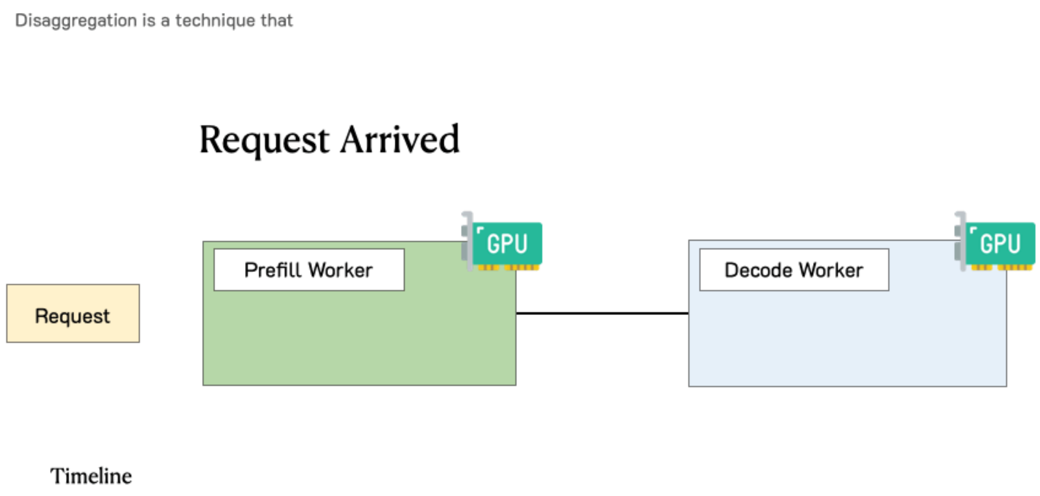
- **(4 + 2 + 1 + 2 + 2) N = 13N memory for an LLM**
- **Using fp8 tensorcore (2x peak flops of fp16 core)**

# Other Optimizations

- Fine-grained quantization
  - fp8 kernels

- Inference
  - Prefill-decode disaggregation (Hao's 2024 work😍)

# Recap of Prefill-Decode Disaggregation




4 TP/SP + 8 DP
32EP in MoE
+ redundant experts

4 TP/SP + DP80
EP320

# Large Language Models

- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - **Flash attention ← come back to this later next week**
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics: prefill-decode disaggregation

# Hope You Have Enjoyed the Content

- ML Systems
- CUDA Kernels
- ML Distributed systems
- Efficient ML algorithms
- The current technology market

LLMSys

Optimizations and Parallelization

MLSys Basics

# The End

- The world now is changing 10x faster than before
- Innovations happen 10x faster
  - What you have learned can be replaced in 1 – 2 years
  - This will become a **norm**

- What we really hope you learn from this course:
  - Ability to identify the right problems
  - Ability to understand "trends"
  - Ability to "predict the future" (I hope so)

# Once you have such skills

Identify a good problem and write an influential paper

| CITED BY | YEAR |
|---|---|
| 42086 | 2020 |

Name: You
Employer: the next OpenAI
Package: 💰💰💰💰💰

Invest the right future and become the next