DATA 8005 Advanced Natural Language Processing

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Guicheng Qi

Fall 2024

Outline

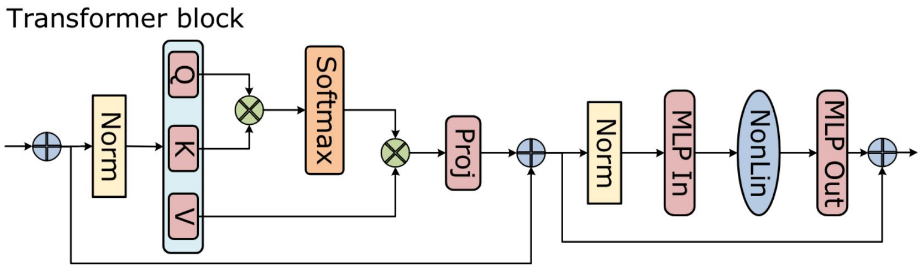
- Transformers are slow and memory-hungry
- Hardware performance
- Standard attention implementation
- FlashAttention

Transformers are slow and memory-hungry

Self-Attention:

$$O = \mathrm{softmax}(Q K^T) V$$

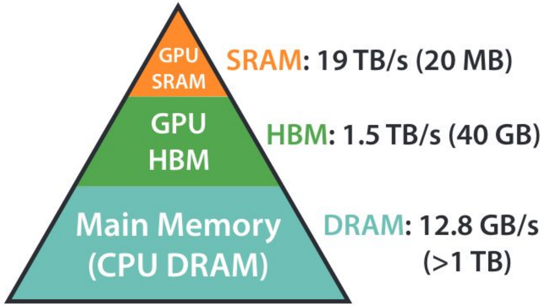
$$\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$$



- N is the sequence length, d is the head dimension
- often N >> d (for GPT2, N = 1024 and d = 64)
- time and memory complexity are quadratic in sequence length

Hardware performance

- GPU memory hierarchy, smaller memory being faster
- GPUs have a massive number of threads to execute an operation (called a kernel)
- Each kernel loads inputs from HBM to registers and SRAM, computes, then writes outputs to HBM



Memory Hierarchy with Bandwidth & Memory Size

Standard attention implementation

- Three stages, where softmax is applied row-wise

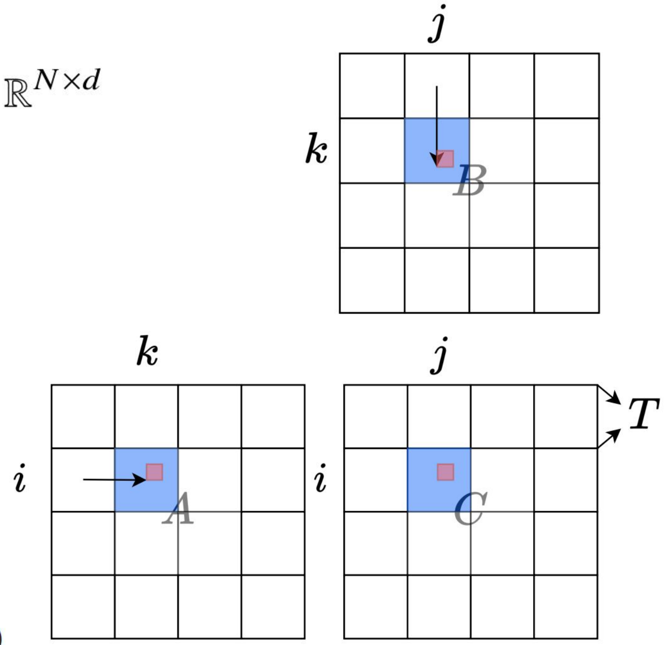
$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{N\times N}, \quad \mathbf{P} = \mathrm{softmax}(\mathbf{S}) \in \mathbb{R}^{N\times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N\times d}$$

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 is needed for MatMul due to limited SRAM
- Softmax should be applied to each row of S by 3-pass



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

FlashAttention

Online softmax

- remove the dependency on N: $d_i' := \sum_{j=1}^{i} e^{x_j - m_i}$ $\{m_i\}$: $\max_{j=1}^{i}\{x_j\}$
- find a recurrence relation:

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

- 2-pass online softmax:

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

$$
\begin{aligned}
m_i &\leftarrow \max(m_{i-1}, x_i) \\
d_i' &\leftarrow d_{i-1}' e^{m_{i-1} - m_i} + e^{x_i - m_i}
\end{aligned}
$$

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

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

end

FlashAttention

Still need 2-pass

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**

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

end

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

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

end

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

FlashAttention

- again, remove dependency on N: $\boldsymbol{o}_i := \sum_{j=1}^{i} \left(\frac{e^{x_j - m_N}}{d'_N} V[j,:] \right)$ $\quad$ $\boldsymbol{o}'_i := \left(\sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d'_i} V[j,:] \right)$
- find a recurrence relation:

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

- one-pass FlashAttention:

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

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

FlashAttention

- Implementation

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 $\mathbf{O}$.

- Effects

Attention	Standard	FlashAttention
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

FlashAttention

Takeaways:

1. FlashAttention proposes a one-pass algorithm to fuse the three stages in original self-attention into one stage.
2. By doing so, FlashAttention reduces times of accessing HBM to achieve faster self-attention computation.
3. The core idea of the algorithm is similar to online softmax.

Thank you!

DATA 8005 Advanced Natural Language Processing

Mamba:

Linear-Time Sequence Modeling with Selective State Spaces

Zijian Ye

Fall 2024

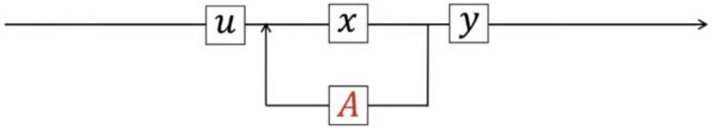
Potential alternative of transformer?

Rejected by ICLR

Figure 4: Application of State Space Models (SSMs) Across Various Domains.

Used in different regions

The origin of mamba: State Space Model(SSM)



$$x' = \boldsymbol{A}x + \boldsymbol{B}u$$
$$y = \boldsymbol{C}x + \boldsymbol{D}u$$

Function:
To describe the relation between u(input) and y(output)

$$h'(t) = Ah(t) + Bx(t)$$
$$y(t) = Ch(t) + Dx(t)$$

$$h_k = \bar{A}h_{k-1} + \bar{B}x_k,$$
$$y_k = \bar{C}h_k + \bar{D}x_k,$$
$$\bar{A} = e^{\Delta A},$$
$$\bar{B} = (e^{\Delta A} - I)A^{-1}B,$$
$$\bar{C} = C$$

discretization

Use of convolution kernel to implement SSM

$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$
$$y(t) = Ch(t) \quad (1b)$$

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \quad (2a)$$
$$y_t = Ch_t \quad (2b)$$

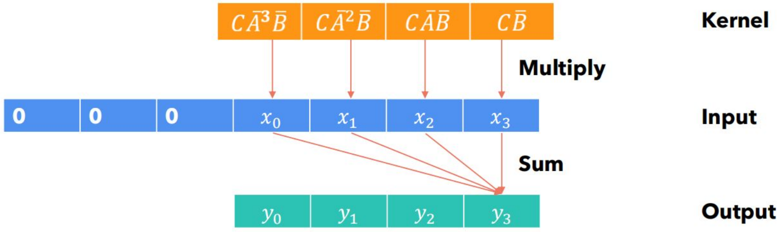
$$\overline{K} = (C\overline{B}, C\overline{AB}, \dots, C\overline{A}^k\overline{B}, \dots) \quad (3a)$$
$$y = x * \overline{K} \quad (3b)$$

$$\begin{aligned} y_2 &= Ch_2 \\ &= C\left(\bar{A}h_1 + \bar{B}x_2\right) \\ &= C\left(\bar{A}\left(\bar{A}h_0 + \bar{B}x_1\right) + \bar{B}x_2\right) \\ &= C\left(\bar{A}\left(\bar{A}\cdot\bar{B}x_0 + \bar{B}x_1\right) + \bar{B}x_2\right) \\ &= C\left(\bar{A}\cdot\bar{A}\cdot\bar{B}x_0 + \bar{A}\cdot\bar{B}x_1 + \bar{B}x_2\right) \\ &= C\cdot\bar{A}^2\cdot\bar{B}x_0 + C\cdot\bar{A}\cdot\bar{B}\cdot x_1 + C\cdot\bar{B}x_2 \end{aligned}$$

$$y_3 = \mathbf{C\overline{AAAB}}x_0 + \mathbf{C\overline{AAB}}x_1 + \mathbf{C\overline{AB}}x_2 + \mathbf{C\overline{B}}x_3$$

$$y_3 = \begin{pmatrix} \mathbf{C\overline{AAAB}} & \mathbf{C\overline{AAB}} & \mathbf{C\overline{AB}} & \mathbf{C\overline{B}} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

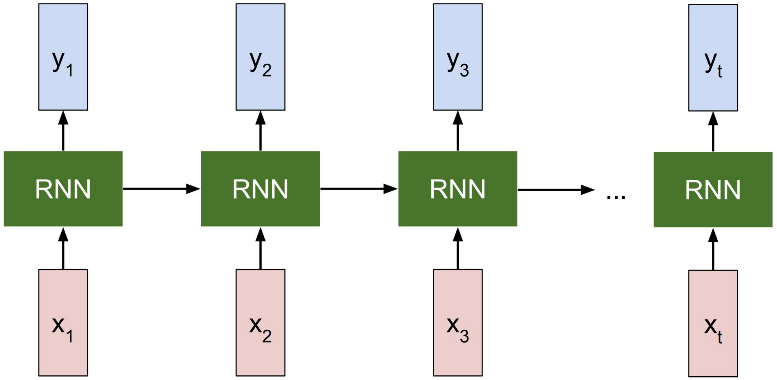
$$y_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + \cdots + C\bar{A}\bar{B}x_{k-1} + C\bar{B}x_k$$



$$y_3 = C\bar{A}^3\bar{B}x_0 + C\bar{A}^2\bar{B}x_1 + C\bar{A}\bar{B}x_2 + C\bar{B}x_3$$

Use of convolution operation to efficiently train SSM model

Linear RNN



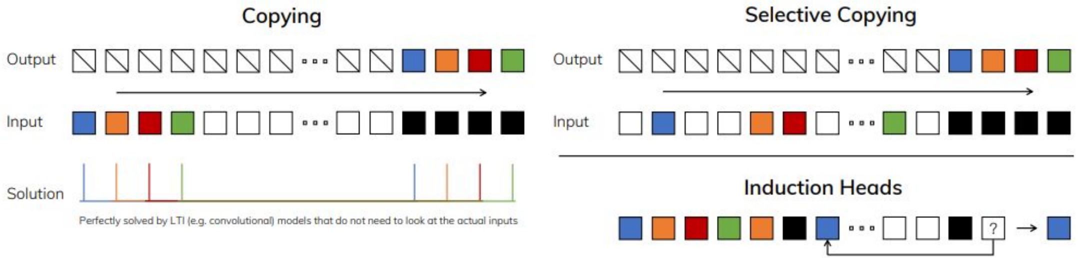
$$h_t = f_W(h_{t-1}, x_t)$$
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

$$h_t = \overline{A} h_{t-1} + \overline{B} x_t \quad (2a)$$
$$y_t = C h_t \quad (2b)$$

Activation: nonlinear operation to linear operation
RNNs are not efficient at training because of sequential computing
Similar to RNN, in mamba, A is used for long term memory,
B is to write into the RNN memory, C is to read from RNN memory

From SSM to Mamba



Motivation:
Same parameter for different inputs causes limited capacity for models

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
$\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
$\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\mathsf{Parameter})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
return y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\mathsf{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

Data dependent:
Make parameters related to data can help solve this kind of problem

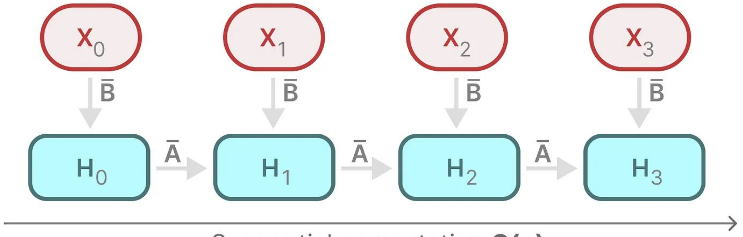
New Challenge: how to parallel?

$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$
$$y(t) = Ch(t) \quad (1b)$$

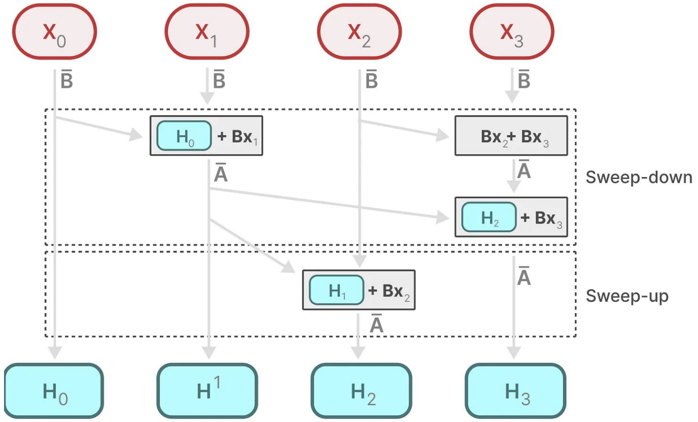
$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \quad (2a)$$
$$y_t = Ch_t \quad (2b)$$

$$\overline{K} = (C\overline{B}, C\overline{AB}, \ldots, C\overline{A}^k\overline{B}, \ldots) \quad (3a)$$
$$y = x * \overline{K} \quad (3b)$$

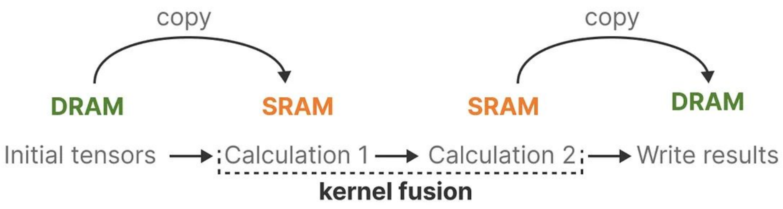
When A,B,C is dependent on input, we can't use convolution anymore.



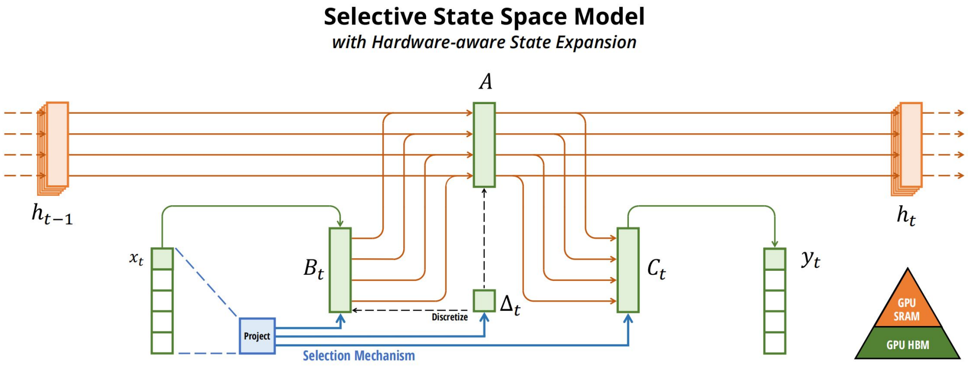
Sequential computation **O(n)**



Parallel computation **O(n/t)**

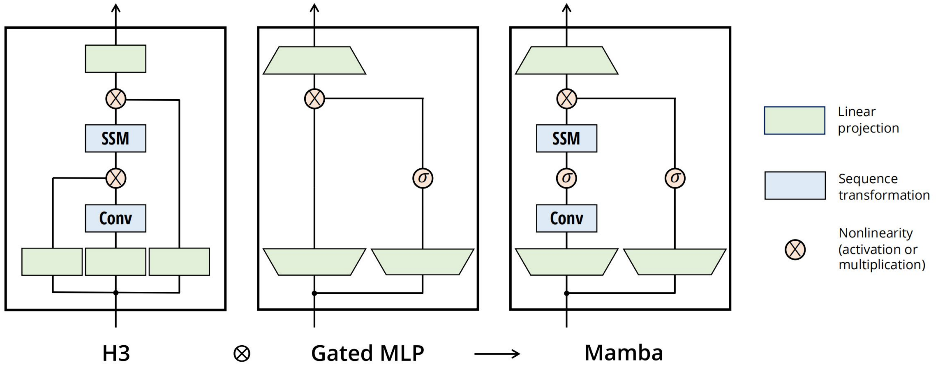


Mamba structure

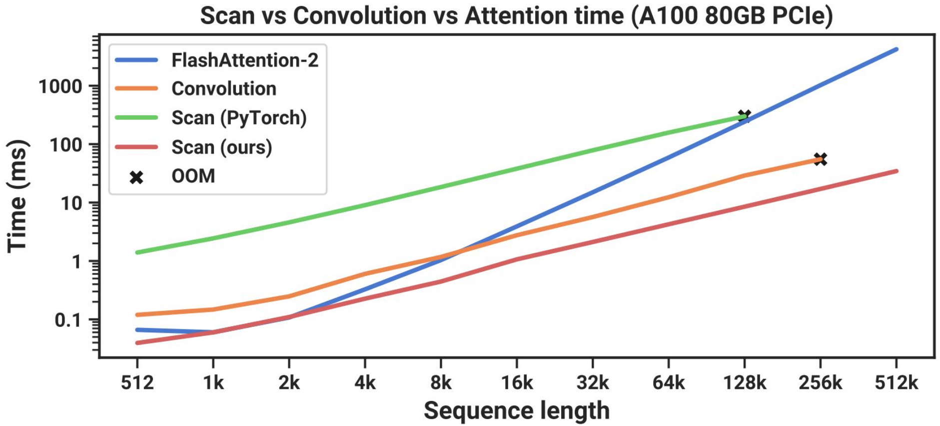


data flow

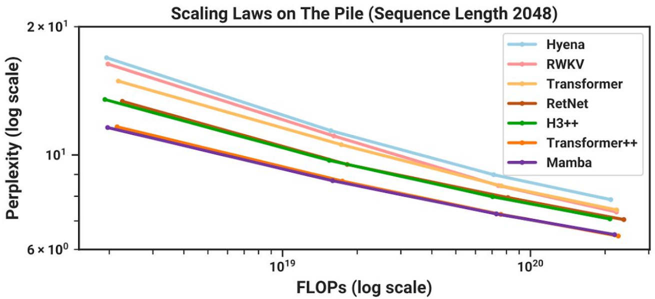
model structure

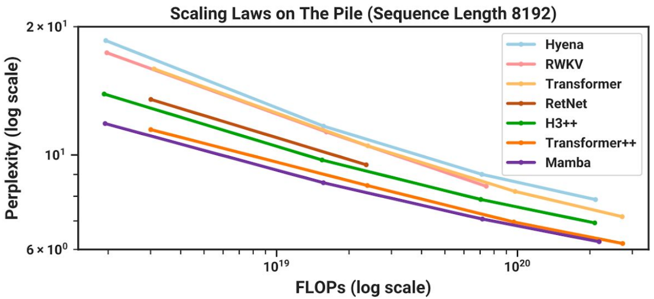


Experiment results



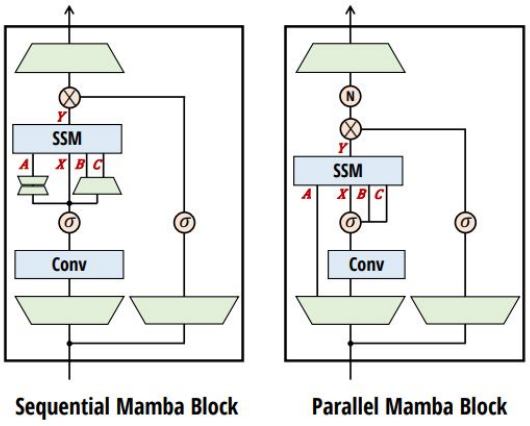
high speed
without OOM problem

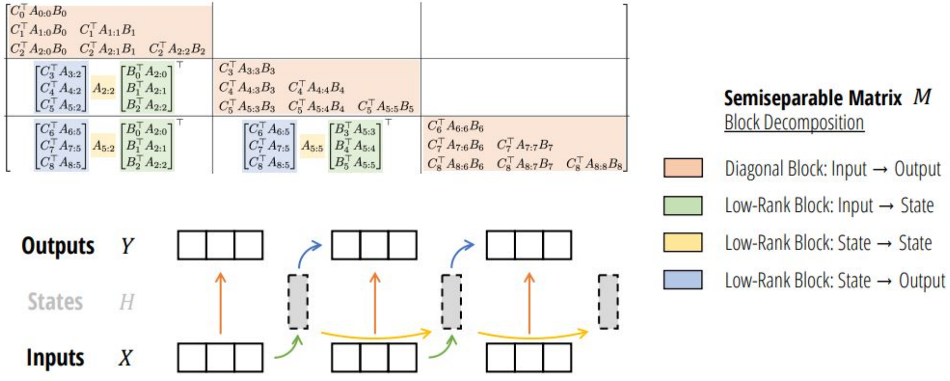




scaling law

Following





Mamba2
New model structure
Faster scan ways

tiiuae/falcon-mamba-7b-Q8_0-GGUF
Updated 20 days ago • 206 • 2

tiiuae/falcon-mamba-7b-F16-GGUF
Updated 20 days ago • 123 • 1

tiiuae/falcon-mamba-7b-BF16-GGUF
Updated 20 days ago • 129 • 1

tiiuae/falcon-mamba-7b-instruct-Q4_K_M-GGUF
Updated 20 days ago • 4.61k • 3

tiiuae/falcon-mamba-7b-instruct-Q8_0-GGUF
Updated 20 days ago • 269 • 5

tiiuae/falcon-mamba-7b-instruct-F16-GGUF
Updated 20 days ago • 140 • 1

Falcon-mamba-7b

ai21labs/Jamba-tiny-dev
Updated 29 days ago • 19.2k • 8

ai21labs/AI21-Jamba-1.5-Large
Text Generation • Updated Sep 17 • 2.81k • 195

ai21labs/AI21-Jamba-1.5-Mini
Text Generation • Updated Sep 17 • 5.76k • 236

ai21labs/Jamba-v0.1
Text Generation • Updated Sep 12 • 45.8k • 1.17k

ai21labs/Jamba-tiny-random
Text Generation • Updated Apr 24 • 8.88k • 12

Jamba

Discussion

Is mamba a potential alternative of transformer?

DATA 8005 Advanced Natural Language Processing

Mixture of Experts

SHEN Che

Fall 2024

What is a Mixture of Experts (MoE)?

- Training a larger model for fewer steps is better than training a smaller model for more steps.
- Mixture of Experts enable models to be pretrained with far less compute.

a MoE consists of two main elements:

- **Sparse MoE layers** are used instead of dense feed-forward network (FFN) layers.
- A **gate network or router**, that determines which tokens are sent to which expert.

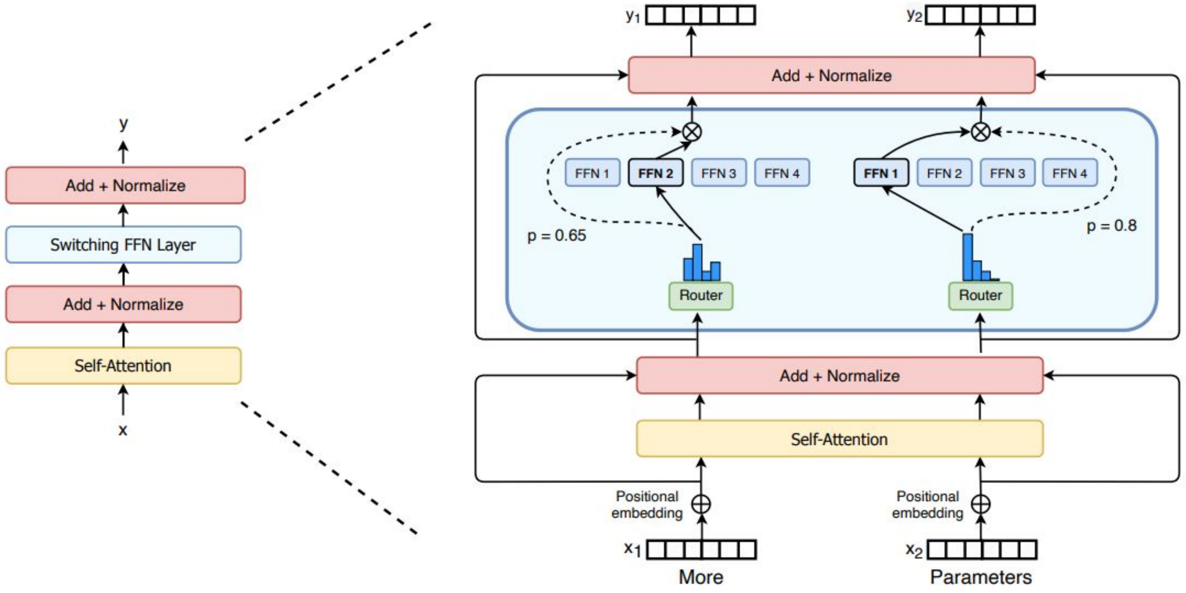


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

What is a Mixture of Experts (MoE)?

- In MoEs we replace every FFN layer of the transformer model with an MoE layer, which is composed of a gate network and a certain number of experts.

Challenges:

- Struggled to generalize during fine-tuning, leading to overfitting.
- All parameters need to be loaded in RAM, so memory requirements are high.

What is Sparsity?

- While in dense models all the parameters are used for all the inputs, sparsity allows us to only run some parts of the whole system.
- The idea of conditional computation (parts of the network are active on a per-example basis) allows one to scale the size of the model without increasing the computation.

Load balancing tokens for MoEs

Challenge: uneven batch sizes and underutilization.

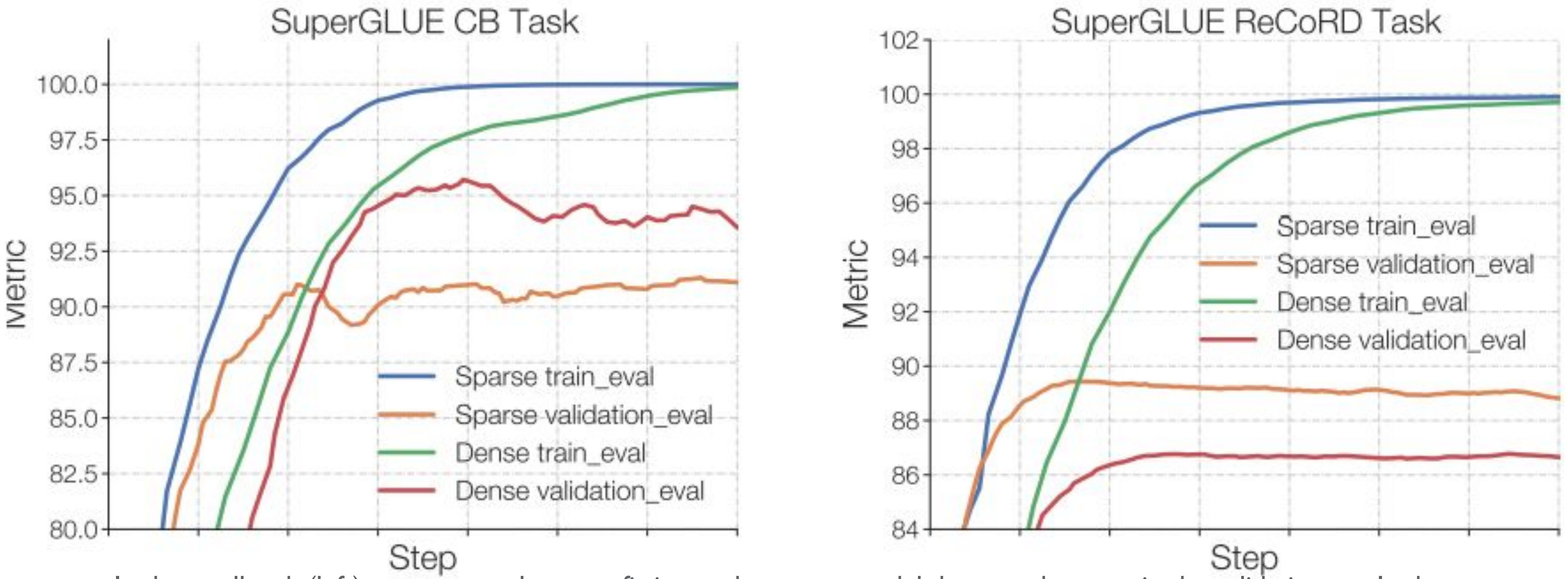
Solution:

- **Auxiliary loss:** an auxiliary loss is added to encourage giving all experts equal importance, which ensures that all experts receive a roughly equal number of training examples.
- **Random routing:** in a top-2 setup, we always pick the top expert, but the second expert is picked with probability proportional to its weight.
- **Expert capacity:** we can set a threshold of how many tokens can be processed by one expert.

Fine-tuning MoEs

- Sparse models are more prone to overfitting, so we can explore higher regularization (e.g. dropout) within the experts themselves (e.g. we can have one dropout rate for the dense layers and another, higher, dropout for the sparse layers).
- At a fixed pretrain perplexity, the sparse model does worse than the dense counterpart in downstream tasks, especially on reasoning-heavy tasks such as SuperGLUE.
- On the other hand, for knowledge-heavy tasks such as TriviaQA, the sparse model performs disproportionately well.

Fine-tuning MoEs

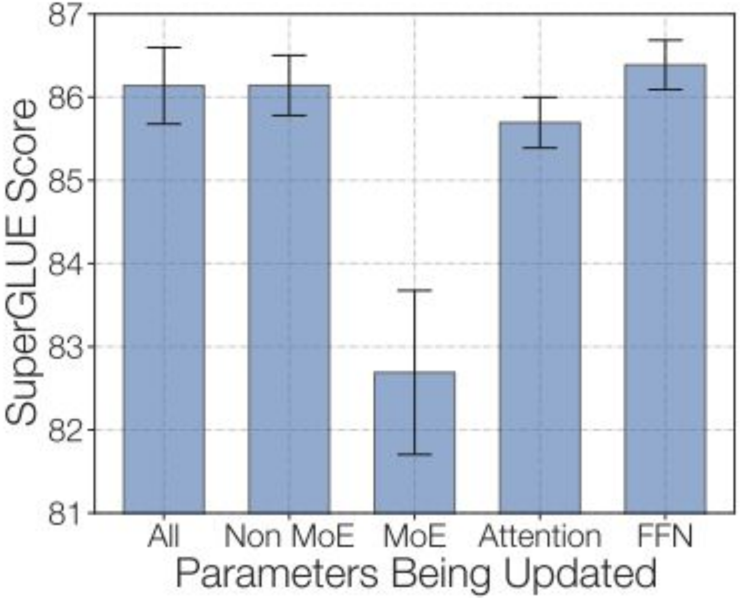


In the small task (left), we can see clear overfitting as the sparse model does much worse in the validation set. In the larger task (right), the MoE performs well.

Fine-tuning MoEs

- Freezing all non-expert weights and only updating the MoE layers leads to a huge performance drop.
- Freezing only the parameters in MoE layers worked almost as well as updating all parameters, which is somewhat counter-intuitive as 80% of the parameters are in the MoE layers.
- The hypothesis for that architecture is that, as expert layers only occur every 1/4 layers, and each token sees at most two experts per layer, updating the MoE parameters affects much fewer layers than updating other parameters.

Fine-tuning MoEs



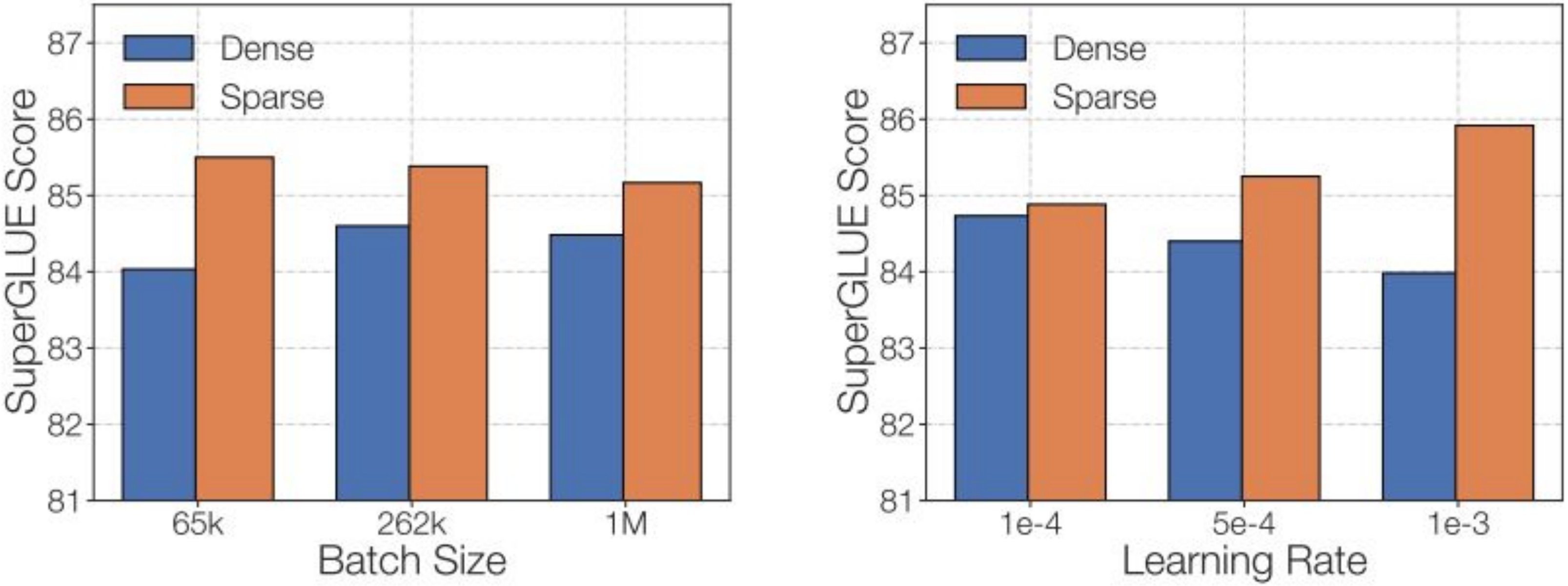
By only freezing the MoE layers, we can speed up the training while preserving the quality.

Fine-tuning MoEs

- Sparse models tend to benefit more from smaller batch sizes and higher learning rates.

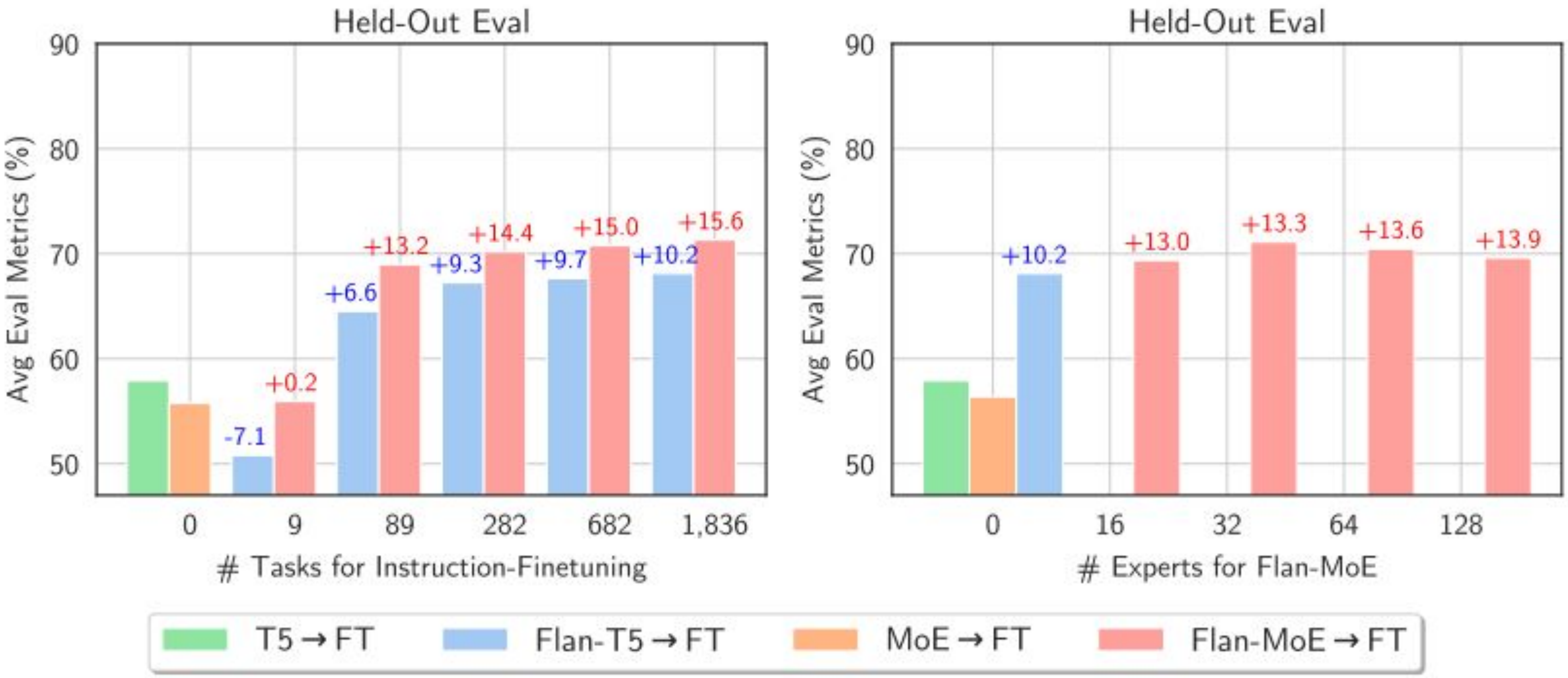
- MoEs might benefit much more from instruction tuning than dense models.

Fine-tuning MoEs



Sparse models fine-tuned quality improves with higher learning rates and smaller batch sizes.

Fine-tuning MoEs



Sparse models benefit more from instruct-tuning compared to dense models.

When to use sparse MoEs vs dense models?

- Experts are useful for high throughput scenarios with many machines. Given a fixed compute budget for pretraining, a sparse model will be more optimal. For low throughput scenarios with little VRAM, a dense model will be better.

Parallelism

- **Expert parallelism**: experts are placed on different workers. If combined with data parallelism, each core has a different expert and the data is partitioned across all cores.
- With expert parallelism, experts are placed on different workers, and each worker takes a different batch of training samples. For non-MoE layers, expert parallelism behaves the same as data parallelism. For MoE layers, tokens in the sequence are sent to workers where the desired experts reside.

TL;DR

MoEs:

- Are **pretrained much faster** vs. dense models
- Have **faster inference** compared to a model with the same number of parameters
- Require **high VRAM** as all experts are loaded in memory
- Face many **challenges in fine-tuning**, but recent work with MoE **instruction-tuning is promising**

Discussions

- Distilling Mixtral into a dense model
- Explore model merging techniques of the experts and their impact in inference time
- Perform extreme quantization techniques of Mixtral