SNaX: SPARSE NARROW ACCELERATED MIXTURE OF EXPERTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture of Experts (MoE) models have emerged as the de-facto architecture for scaling up language models without significantly increasing the computational cost. Existing MoE methods optimize system efficiency or model architecture independently. We show that as MoE models get more granular and sparser, they become more memory-bound, and jointly optimizing the algorithms and the kernel design leads to a major improvement in MoE training throughput. We first propose a memory-efficient algorithm to compute the forward and backward of MoE with minimal activation saved. We then design GPU kernels that overlap memory IO latency with compute, benefiting all MoE architectures. Finally, we propose a novel "token rounding" method that minimizes the wasted compute brought by tile quantization. As a result, our method SNaX **reduces 45% activation memory and has 1.87x compute throughput improvement** on Hopper GPUs compared to state-of-the-art BF16 MoE kernel for a fine-grained 7B MoE. Concretely, SNaX on 64 H100s achieves almost the same total throughput as ScatterMoE on 96 H100s for a 7B MoE model training with token-choice rounding while training with FSDP-2. Under high MoE sparsity settings, our tile-aware token rounding algorithm yields an additional **1.18x** speedup on kernel execution time compared to vanilla top-K routing while maintaining similar downstream performance.

1 INTRODUCTION

Mixture of Experts (MoE) (Shazeer et al., 2017) models have emerged as a key technique for scaling up transformers (Vaswani et al., 2017) without increasing the computational requirements for training. Recent models have reached over hundreds of billions of parameters (DeepSeek-AI et al., 2024) or even trillions of parameters (Kimi et al., 2025). Modern transformers often have layers comprised of a sequence mixer block (e.g., Multi-Head Attention (Vaswani et al., 2017)) followed by a channel mixer block, where MoEs are excellent substitutes for dense MLPs for FLOPs efficiency. MoE layers exploit sparse computation thus reducing FLOPs for the same number of parameters. However, reducing FLOPs does not directly translate to better hardware utilization since hardware-friendly sparse computation is not an easy task and MoEs often have more IO access than dense models.

Scaling law studies on MoEs Clark et al. (2022); Krajewski et al. (2024); Tian et al. (2025) predict better model quality per FLOPS as one increase expert granularity (ratio between the model embedding dimension and each expert's intermediate size) and sparsity. Recent MoE models like DeepSeek V3 (DeepSeek-AI et al., 2024), QWen3 MoE (QwenLM, 2025) and GPT-OSS-120B (OpenAI, 2025), have demonstrated superior performance of "fine-grained" MoEs over "coarse-grained" MoE at scale. These findings have been adopted with recent model architectures such as PEER (He, 2024), Memory layers (Berges et al., 2024), and UltraMem (Huang et al., 2025b;a). Besides granularity, the pursuit of MoEs with better model quality while keeping computational requirements constant have also led to modern MoEs become extremely sparse. For example, Kimi K2 (Kimi et al., 2025) has the same amount of activated parameters as DeepSeek V3 (DeepSeek-AI et al., 2024) but much larger total parameters. Overall, granularity and sparsity for MoEs have only increased over time as shown in Table 5.

However, granular and sparse MoEs, optimizing mainly for model quality per FLOPS, suffers from hardware inefficiency due to: (1) larger activation memory footprint for granular MoE models as activation size typically scales linearly with number of activated experts, (2) worse hardware utilization due to the lower arithmetic intensity and increased IO costs by granular experts and

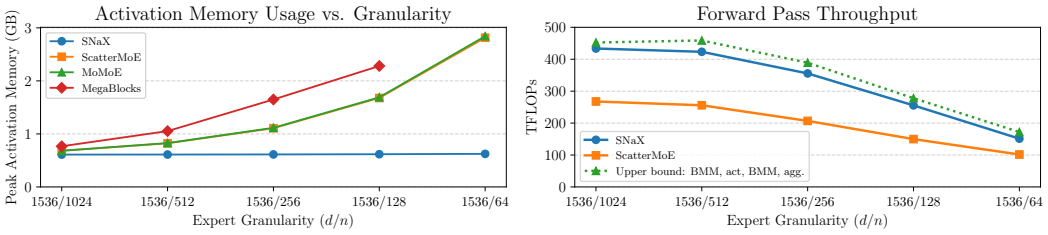

Figure 1: SNaX's per-layer activation memory footprint (left) stays constant even as expert granularity increases, and is 1.5-4x more memory-efficient than baselines. SNaX's forward pass throughput (right) (router gemm + top-k + softmax + grouped gemm + activation + grouped gemm + aggregation) reaches 93%+ the speed of the upper bound of BMM + activation + BMM + aggregation.

(3) wasted computations due to tile quantization effects of grouped GEMM for highly sparse MoEs. Fundamentally, the high granularity and sparsity both push the MoE training towards the memory-bound regime where MoE kernels should take care of the increased IO costs. Existing state-of-the-art MoE kernels such as ScatterMoE (Tan et al., 2024) and MoMoE (Costin et al., 2025) are not designed to handle these high IO costs case where they suffer significant throughput degradation.

We propose to co-design MoE architecture with a hardware-aware GPU kernel and a novel routing method. (1) We first derive an algorithm to compute the MoE backward pass in a different order than the standard, leading to much smaller activation size that does not increase as experts get more granular. (2) We then carefully design MoE GPU kernels to overlap memory IO with computation, benefiting all MoEs but most importantly fine-grained MoEs. (3) We propose a hardware-aware token rounding routing method where the routed number of tokens to an expert are always a multiple of the GEMM tile size. We show through extensive experiments that this proposed routing strategy could be 18% faster than token-choice routing with the same downstream language modeling performance. With (1) and (2), we can increase the end-to-end training throughput of 7B MoE model by 50% (with no architecture change from typical top-K token choice routing), and our new token rounding routing further improve another 10-20% TFLOPs under high MoE sparsity setting without accuracy loss.

Summary of contributions. Our work proposes a co-design solution SNaX on addressing the training efficiency problems, making the following contributions:

- **Memory Efficient MoE forward and backward:** We analyze the impact of MoE granularity to the MoE layer's forward and backward pass and identify that increasing MoE granularity with constant FLOPs leads to a linear increase in activation memory required for backward computation. We propose to carefully change the computation graph to avoid caching the activations needed for router gradient computation while being mathematically equivalent to the normal MoE. As a result, for a finegrained 7B MoE, SNaX reduces activation memory usage by 45% per layer.
- **Efficient MoE kernel with overlapped IO and compute:** We also demonstrate that increasing both granularity and sparsity leads to MoEs becoming increasingly memory bandwidth bound. We exploit the asynchrony of GEMM and IO and carefully reduce the IO latency and accesses to maximize the compute throughput for finegrained MoE. For the same finegrained 7B MoE, SNaX's MoE kernel reduces improves the kernel TFLOPs by 87% compared to state-of-the-art BF16 MoE kernel.
- **Token rounding routing:** We introduce a drop-in routing algorithm that *rounds* the per-expert token count to the tile size (e.g., 128) multiples used by grouped GEMM in MoE kernel to reduce wasted compute by padding while prioritizing the original token-choice expert selection. This routing algorithm favors token choice result such that for each expert, the maximum deviation from token choice top-K result is at most 1 tile. This routing method eliminates padding waste in grouped GEMM while maintaining the same total tokens in expectation, and has robust token-choice inference quality even under sparse MoE training regime. We validate the performance of this token rounding strategy in a highly sparse training setting at 1.4B parameter scale. We also show that token rounding's compute throughput over vanilla token-choice top-K is consistently larger once we enter highly sparse MoE training regime and can make 18% TFLOPs difference on kernel runtime.

Our implementation of SNaX, written in CuTe-DSL (Nvidia, 2025) with a PyTorch interface, will be released with a permissive license to benefit researchers and practitioners.

2 BACKGROUND

We first provide an overview of the MoE architecture and a standard MoE kernel employing grouped GEMM in Section 2.2. Based on such formulation, we examine existing MoE implementations. We then discuss key factors that influence MoE's training efficiency in Section 2.3 and illustrate why prior kernel design fails to resolve the new efficiency challenges with finegrained and sparse MoEs[1]. We also examine the influence of routing on MoE's training efficiency in Section 2.4.

2.1 GEMM AND GROUPED GEMM

A GEMM (general matrix multiply) kernel on GPUs is often structured into the prologue, mainloop and epilogue. The kernel first applies tiling (dividing large matrices into smaller blocks), and optionally pads dimensions so computation align with hardware-friendly tile sizes. The mainloop loads data from High Bandwidth Memory (HBM) to on-chip SRAM and performs the tiled matrix multiply and accumulates over the $\mathbf{K}$ dimension. The epilogue applies post-processing on the GEMM results such as activation functions and writes results back to HBM. Grouped GEMM, commonly used in MoE, is a variant of GEMM that runs a list of GEMMs with different $\{\mathbf{M}_i,\mathbf{N}_i,\mathbf{K}_i\}_{i\in[E]}$.

2.2 MOE ARCHITECTURE AND EXISTING KERNEL DESIGN

A MoE block typically has a token router and multiple smaller (often equally-sized) subnetworks, called "experts". The router is responsible for dispatching tokens to the experts which are subsequently used by the specific expert for computation. The outputs from all experts in the layer are then aggregated and passed onto the next layer. Algorithm 1 formulates such computation with grouped GEMM, which can leverage high throughput Tensor Cores on modern GPUs (NVIDIA, 2022).

Algorithm 1 MoE forward with Grouped GEMM

Input : $X \in \mathbb{R}^{T\times d}$, $W_1 = \{W_{1,e}\}_{e\in[E]} \in \mathbb{R}^{d\times 2n}$, $W_2 = \{W_{2,e}\}_{e\in[E]} \in \mathbb{R}^{n\times d}$, routing scores $S\in\mathbb{R}^{T\times E}$, $\pi\in\{0,1\}^{T\times E}$ as a binary valued matrix where $\pi_{t,e}$ represents if token t is routed to expert e.

Output : output activation $O\in\mathbb{R}^{T\times d}$

Parallel for $e\in[E]$ **do**

- $X_e \leftarrow \mathrm{Gather}(X,\pi_{:,e})$
- $Z_e \leftarrow X_e W_{1,e}$ `// Grouped GEMM`
- $Y_{1,e} \leftarrow \mathrm{SwiGLU}(Z_e)$
- $Y_{2,e} \leftarrow Y_{1,e} W_{2,e}$ `// Grouped GEMM`

Parallel for $t\in[T]$ **do**

- $O_t = \sum_{e\in[E]} \pi_{t,e} S_{t,e} Y_{2,e,t}$

return O

For MoE training, MegaBlocks (Gale et al., 2023) proposed to gather and pad the tokens and then apply block-sparse matrix multiplication to compute the expert output. ScatterMoE (Tan et al., 2024) fuses the gather with Grouped GEMM, and scatters each expert result after the down projection. DeepGEMM (Zhao et al., 2025b;a) has also emerged as a performant kernel for FP8-MoE training, however DeepGEMM doesn't do epilogue fusion and gather fusion with the grouped GEMM. MoMoE (Costin et al., 2025) does epilogue fusion and gives more control over how much activations to save during training.

Existing implementations like MoMoE, ScatterMoE, and MegaBlocks are already implemented in low-level GPU programming languages (Triton (Tillet et al., 2019) and CUDA), but they have yet to take full advantage of the asynchronous nature of modern accelerators to overlap compute and memory IO. Modern NVIDIA GPUs like H100s feature advanced features like asynchronous Tensor Cores, TMA (Tensor Memory Accelerator) for data movement etc. (NVIDIA, 2022; NVIDIA). For instance, on H100s it is possible to asynchronously fetch a tile of data while the Tensor Cores are still running the GEMM for the previous tile. It is important to use such asynchrony on modern GPUs to achieve close to peak compute throughput as demonstrated by FlashAttention-3 (Shah et al., 2024).

2.3 MOE'S TRAINING EFFICIENCY

Arithmetic intensity is defined as the ratio of FLOPs over the number of transferred bytes. It quantitatively categorizes whether a kernel is memory-bound or compute-bound. If we denote T_e as the number of routed tokens received for expert e, the up-projection of expert e uses $2T_e\cdot 2n\cdot d$ FLOPs with $2T_e d + 2\cdot 2n\cdot d + 2T_e n$ HBM memory transfer bytes, and similarly, the down projection uses $2T_e nd$ FLOPs with $2T_e n + 2nd + 2T_e d$ HBM memory transfer bytes. Assuming $\rho = \frac{K}{E}$ as the inverse of sparsity, $G = \frac{d}{n}$ as the granularity and uniform routing i.e $T_e = T\rho$, the arithmetic intensity (ignoring

[1] Here we refer "finegrained MoE" as MoE with small granularity i.e. n is smaller than d. We assume the setting of both iso-FLOPs and iso-params.

the writes for Z_e) for the forward pass of an expert is

$$\frac{2T_e \cdot 2n \cdot d + 2T_e nd}{4T_e n + 6nd + 4T_e d} = \frac{3}{\frac{2}{d} + \frac{2}{n} + \frac{3}{T_e}} = \frac{3}{\frac{2+2G}{d} + \frac{3}{T\rho}} \tag{1}$$

For a specific model size (constant d), it can be seen that increasing granularity (increasing G) or increasing sparsity (decreasing ρ) leads to a decreasing arithmetic intensity. Our speed benchmarks in Figure 2 suggest that when $n < 256$, we enter the memory-bound regime where the observed TFLOPs decrease nearly linearly with n even under the most ideal case (cuBLAS dense BMM) due to the linear scaling between IO costs and granularity $\left(G = \frac{d}{n}\right)$. This suggests the importance of hiding IO latency with compute for low expert intermediate size (n). This is achieved using a series of asynchronous load/stores overlapped with compute described in Section 4.

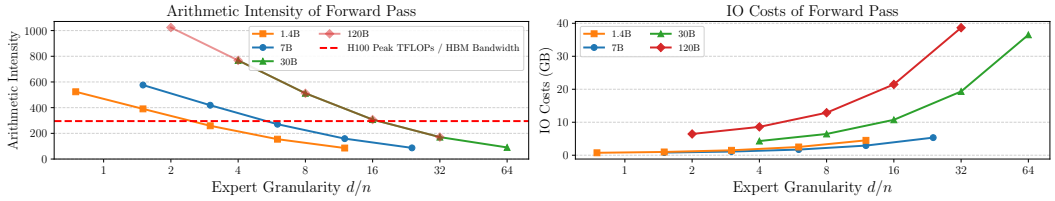

Figure 2: Arithmetic intensity and the corresponding IO costs of MoE's forward pass w.r.t. expert granularity across MoE configurations from 1.4B to 120B.

2.4 MoE routing methods

MoE models rely on conditional computation to reduce the model FLOPs requirement. For conditional computation, a routing subnetwork routes tokens to a subset of experts. The most widely used routing method is token choice routing (Shazeer et al., 2017). In token-choice routing, the tokens pick which experts to use for their computation, i.e the routing decision is $\text{TopK}_{e \in [E]}(S_{:,e}, K)$ where TopK is the top-K function and $S_{:,e}$ is the expert score for each token. Expert choice routing has also been proposed to avoid load imbalance during training (Zhou et al., 2022). In expert choice routing, the experts are instead responsible for choosing the tokens. However, expert choice routing is not easily usable for auto-regressive inference since it creates a mismatch between training and inference and breaks causality. Other works such as Zeng et al. (2024) propose AdaMoE which dynamically adjusts the number of activated experts by adding null experts. In this paper, we focus on token-choice routing for experimentation since its the most widely used method for training large MoE models.

3 Memory-efficient MoE algorithm

We explain the activation memory bottleneck in existing MoE methods, and show how to reduce activation size by performing the backward pass in a different order (with the same gradients mathematically).

The FLOPs of MoE forward & backward pass is $(6+12)TnKd$ and if we fix microbatch size (keep T constant) and keep FLOPs constant[2], nK should be a constant. Therefore, increasing granularity means scale down n and proportionally scale up K and E. In this case, any variables that has size $O(TKd)$ should not be cached otherwise we would observe a linear dependency of activation memory w.r.t. granularity, as the case for community MoE kernels such as ScatterMoE.

During the forward and backward pass, Y_2 (fwd down-proj results), X_e (fwd up-proj gathered activation) and dO_e (bwd down-proj gathered activation) all require $2TKd$ bytes. Existing MoE methods such as ScatterMoE need to store Y_2 in the forward to compute the gradient w.r.t. the router scores s, as one can see from the forward pass in Algorithm 1.

To minimize activation memory, we need to avoid avoid caching Y_2, X_e, dO_e in the forward (and ideally do not incur extra matmul FLOPS in the backward pass). We propose to:

- fuse grouping operation into the prologue and do not materialize X_e and dO_e in global memory
- identify a computation path that circumvents the need of Y_2 without incurring any additional FLOPs, by reordering the summation order in the backward pass, as derived in App. F.

[2]We always assume embedding dimension d is constant.

Algorithm 2 SNaX's MoE kernel forward pass with SwiGLU architecture. Variables stored in HBM are colored blue.

Input : X, S, π, W_1, W_2 same as Algorithm 1.
Output : Output activation O

Up-proj Y_1 kernel $(X, W_1, \pi) \to (Z, Y_1)$:
// Gather + Grouped GEMM + SwiGLU
  Parallel for $e \in [E]$ **do**
    $X_e, W_{1,e}, \pi_{:,e} \leftarrow \text{load}(X_e, W_{1,e}, \pi_{:,e})$
    $X_e \leftarrow \text{Gather}(X, \pi_{:,e})$
    $Z_e \leftarrow X_e W_{1,e}$
    $Y_{1,e} \leftarrow \text{SwiGLU}(Z_e)$
    $Z_e, Y_{1,e} \leftarrow \text{store}(Z_e, Y_{1,e})$

Down-proj Y_2 kernel $(Y_1, W_2) \to Y_2$:
// Grouped GEMM
  Parallel for $e \in [E]$ **do**
    $Y_{1,e}, W_{2,e} \leftarrow \text{load}(Y_{1,e}, W_{2,e})$
    $Y_{2,e} \leftarrow Y_{1,e} W_{2,e}$
    $Y_{2,e} \leftarrow \text{store}(Y_{2,e})$

Expert aggregation O Kernel $(Y_2, S, \pi) \to O$:
// reduce over each expert results
  Parallel for $t \in [T]$ **do**
    $Y_{2,e,t}, S_{t,e}, \pi_{t,e} \leftarrow \text{load}(Y_{2,e,t}, S_{t,e}, \pi_{t,e})$
    $O_t \leftarrow \sum_{e \in [E]} \pi_{t,e} S_{t,e} Y_{2,e,t}$
    $O_t \leftarrow \text{store}(O_t)$

Algorithm 3 SNaX's MoE kernel backward pass of down projection with SwiGLU architecture.

Input : S, π, W_2, dO.
Output : dZ, dW_2, dS.

Down-proj act dZ kernel $(dO, W_2, S, \pi) \to (dZ_e, dS, Y_4)$:
// Gather + Grouped GEMM + dSwiGLU + dS + Y_4 for dW_2
  Parallel for $e \in [E]$ **do**
    $dO_e, \pi_{:,e} \leftarrow \text{load}(dO_e, W_{2,e}, S, \pi_{:,e})$
    $dO_e \leftarrow \text{Gather}(dO, \pi_{:,e})$
    $Y_{3,e} \leftarrow dO_e W_{2,e}^\top$ // $Y_{3,e} \in \mathbb{R}^{T_e \times n}$
    $dY_{1,e} \leftarrow \text{Broadcast}(\mathbf{s}_e) Y_{3,e}$
    $Y_{1,e}, dZ_e \leftarrow \text{dSwiGLU}(dY_{1,e}, Z_e)$
    $dS_{e,t} \leftarrow \langle Y_{3,e,t}, Y_{1,e,t} \rangle$
    $\mathbf{s}_e \leftarrow \text{Gather}(S, \pi_{:,e})$
    $Y_{4,e} \leftarrow \text{Broadcast}(\mathbf{s}_e) Y_{1,e}$ // $Y_4 \in \mathbb{R}^{T_e \times n}$
    $dZ_e, dS, Y_{4,e} \leftarrow \text{store}(dZ_e, dS, Y_{4,e})$

Down-proj weight dW_2 kernel $(dO, Y_4, \pi) \to dW_2$:
// Gather + Grouped GEMM
  Parallel for $e \in [E]$ **do**
    $dO_e, Y_{4,e}, \pi_{:,e} \leftarrow \text{load}(dO_e, Y_{4,e}, \pi_{:,e})$
    $dO_e \leftarrow \text{Gather}(dO, \pi_{:,e})$;
    $dW_{2,e} \leftarrow Y_{4,e}^\top dO_e^\top$
    $dW_{2,e} \leftarrow \text{store}(dW_{2,e})$

As a result, we only cache X and Z and small routing metadata with total size $2Td + 4TKn + O(TE)$ bytes per layer **similar as the usage for a dense model with activated number of parameters**.[3]

We present the SNaX algorithm in Algorithm 2,3,6 [4]. **SNaX achieves effectively the minimum activation memory without GEMM recomputation**. In Figure 4, we profile SNaX's activation memory in a 7B MoE configuration and also demonstrate that the activation memory of SNaX is *independent of expert granularity*. More results from 1.4B to 120B are included in Figure 4.

4 IO-AWARE KERNEL DESIGN

The expressivity of fine-grained MoE comes from the diversity of every token's expert selection, which in turns leads to linearly-scaled IO costs w.r.t. expert granularity as shown in Figure 2. For memory bound GEMM kernels, we want to maximally overlap the IO latency with compute. Here we describe an IO-aware optimization that minimizes the induced IO latency across multiple kernels.

IO access reduction: We perform most computations on-chip to fully exploit registers and shared memory, thereby reducing costly IO access. To further minimize IO, we reformulate the computation of dZ (see Appendix F), and fuse the calculations of dZ, dS, and Y_4 into a single kernel, resulting in a heavy epilogue that can be further overlapped with MMA using Ping-Pong.

Ping-Pong scheduling: On NVIDIA Hopper GPUs, GEMM is usually performed asynchronously with a producer-consumer paradigm (Shah et al., 2024) where producer warpgroups are dedicated to handling IO while consumer warpgroups are responsible for GEMM computation.Suppose we have 2 consumer warpgroups, we can either let them cooperatively issue the Hopper GEMM instruction (WGMMA) with a large tile size, or overlap the IO of 1 warpgroup with GEMM of another warpgroup with a smaller tile size. Once this is finished, we switch the roles of the warpgroups (effectively interleaving IO and GEMM). This is often referred to as "Ping-Pong" scheduling (Wright & Hoque,

[3] Although we still materialize a Y_2 variable, we can recycle Y_2 after each layer. As long as the number of MoE layers (typically 32+ for 7B+ MoE) is larger than K, the transient memory usage of Y_2 will be overshadowed. We note that removing such materialization requires an atomic add which creates new issue with determinism (He & Machines, 2025) and numerical accuracy (for BF16 atomic add).

[4] Our only recomputation is Y_1 from Z, and this is cheap during epilogue of dZ as shown in Algorithm 5.

2024) in Figure 8. Pingpong scheduling[5] is particularly useful to maintain high Tensor Core (TC) utilization in GEMM kernels with long epilogues (compared to the mainloop). For example, the Y_2 kernel's epilogue has heavy HBM store IO (1.21 GB per layer for a 7B model) and in the dZ kernel's epilogue, we need to asynchronously load Z and execute multiple math and reduction operations.

Asynchronous epilogue load for dZ computation: In the dZ kernel's epilogue, we need to load Z to compute dZ from dY_1. We create a dedicated pipeline with asynchronous TMA (NVIDIA, 2022) load for Z to overlap it with other epilogue operations across epilogue stages.

5 Token Rounding for Sparse MoE

In this section, we analyze the hardware efficiency under sparse MoE training regime and identify that as MoEs become sparser, the wasted compute on padded GEMM tiles accumulate to a nontrivial amount, known as "tile quantization" effects. In response, we propose a novel routing method "token rounding" to eliminate tile quantization effects.

5.1 Training Efficiency of Sparse MoE

Besides granularity, the arithmetic intensity of MoE also depends on the inverse of sparsity ρ as shown in Equation 1. When we scale down ρ, the expected number of received tokens per expert $\mathbb{E}_{e\in[E]}T_e = T\rho$ will also linearly decrease and the GEMM computation shifts towards memory-bound regime.

Tile quantization effect: Moreover, GEMM on modern GPUs are often computed in tiles (NVIDIA, 2022) and we always need to pad to the next tile-sized multiples if any dimensions of $\mathbf{M}$,$\mathbf{N}$,$\mathbf{K}$ are not fully divisible by tile sizes. Once the size of input (e.g. token dimension per expert) is small, the wasted TFLOPs by padding be nontrivial (Fig. 3).

For the case of MoE with token-choice routing, we often have variable number of tokens per expert and during the forward and activation gradient backward, we waste

$$\frac{\lceil \frac{T_e}{\text{tile}_M} \rceil * \text{tile}_M - T_e}{T_e} \cdot E \cdot (6dn) \tag{2}$$

FLOPs per forward pass by padding. As such, we propose to use token rounding to avoid launching such extra tiles leading to more efficient training. We also show that our token rounding method does not affect model quality while achieving much higher training throughput.

Algorithm 4 Token rounding routing

Input : $X \in \mathbb{R}^{T\times d}$; number of experts E and expected activated number of experts K per token; tile size Q; router scores $S \in \mathbb{R}^{T\times E}$ and $0 \le S_{e,i} \le 1$. round_and_sparsify that determines rounding up or down.

Output : tile-rounded router scores $\lfloor S \rceil$,

(1) Top-K token choice sorting

$(S_{\text{topK}}, I_{\text{topK}}) \leftarrow \text{TopK}(S, K)$

(2) Calculate each expert's received token frequencies and its tile-rounded multiples

$f_e \leftarrow \sum_t \mathbf{1}_{\{e \in I_{\text{topK},t}\}}$

$f_e^{\wedge} \leftarrow \lceil f_e/Q \rceil \cdot Q; \quad f_e^{\vee} \leftarrow \lfloor f_e/Q \rfloor \cdot Q$

(3) Build Top-K-preferred S' for expert-wise ranking

$S'_e \leftarrow S_e - 1$ `// ensure non-top-K entries are smaller than top-K`

for $t \in [T]$ & $k \in [K]$ *in parallel* **do**

  $S'_{t, I_{\text{topK}}(t,k)} \leftarrow S_{\text{topK},t,k}$

(4) Token rounding per expert **for** $e \in [E]$ *in parallel* **do**

  $r_e, s_e \leftarrow \text{sort}(S'_e)$ `// token ordering and sorted scores`

  $r'_e, s'_e \leftarrow \text{round\_and\_sparsify}(r_e, s_e, f_e, f_e^{\wedge}, f_e^{\vee})$ `// token rounding`

  $\lfloor S \rceil_e \leftarrow \text{Gather}(S, r_e)$

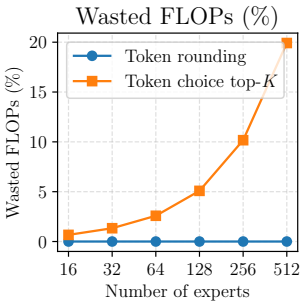

Figure 3: Wasted FLOPs by padding during forward & backward by MoE model with $T = 32k$, $d = 4k$, $K = 4$, which is the same as the bottom right row of Figure 7.

5.2 Token Rounding: Leverage the Wasted Compute

As such, we propose to use token rounding method as a 2-step sorting algorithm as shown in Algorithm 4. The token rounding algorithm will first gather the vanilla token-choice (TC) routing results and apply an expert-choice (EC) sorting step to choose to either discard some TC tokens or pad some additional EC tokens. Between these 2 steps, we process the routing weight matrix such that the TC tokens are always preferred before EC tokens so the discarding TC tokens or padding new EC tokens only affect the last tile.

[5]The selection between cooperative and pingpong is largely determined by the selection of a larger tile size or more epilogue overlap. Fine-grained MoE needs both: the dW_1, dW_2 kernels often have long mainloop and cooperative nearly always win, while Y_2, dZ kernels have heavy epilogue and Ping-Pong is usually favorable.

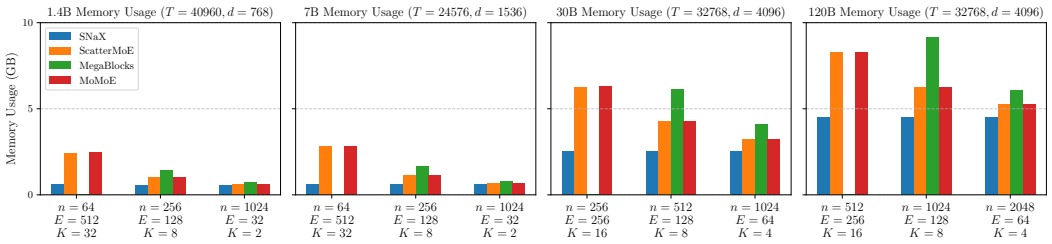

Figure 4: Peak activation memory usage (↓) per layer across different model scales (1.4B–120B). Our method consistently reduces memory consumption compared to ScatterMoE, MegaBlocks, and MoMoE. MegaBlocks does not support small n.

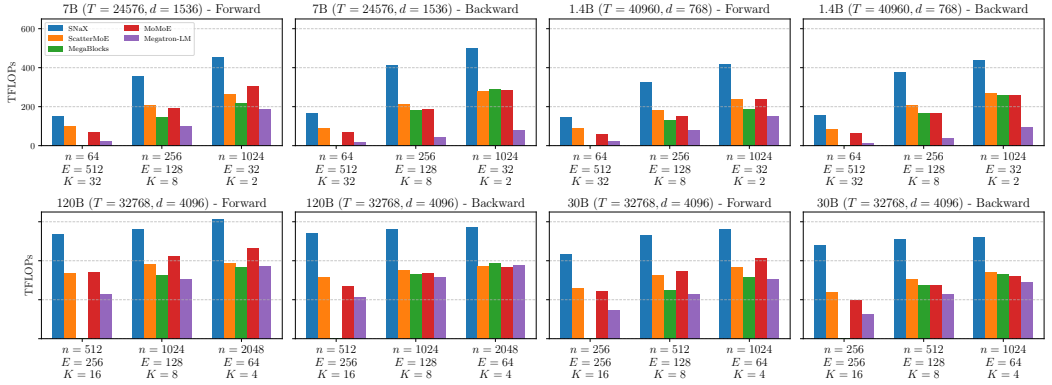

Figure 5: Forward & backward TFLOPs (TFLOP/second ↑) for different MoE kernels. H100's theoretical limit is 989 TFLOPS of BF16 GEMM (NVIDIA, 2022). Same MoE config as in Figure 4.

This simple algorithm guarantees that for each expert, the maximum deviation from token-choice routing is at most 1 tile. We find that this property has a surprisingly robust performance even under sparse MoE training regime and can serve as an in-place substitute for token-choice under sparse MoE training settings. We validate the performance of this token rounding strategy in a highly sparse training setting at 1.4B parameter scale. We also show that token rounding's compute throughput over vanilla token-choice top-K is consistently larger once we enter highly sparse MoE training regime and can make 18% TFLOPs difference on kernel runtime.

Note that token rounding algorithm needs to take a $\mathrm{round\_and\_sparsify}$ subroutine to do discard or pad decision. We choose the simplest as round to nearest on token count (as "nearest rounding"). We further conduct an ablation in Table 7 and find that our token rounding algorithm is quite robust w.r.t. the underlying round subroutine call.

6 Experiments

We evaluate SNaX's speed and activation memory requirements compared to other baseline MoE implementations. We also demonstrate the efficacy of token rounding routing strategy and show that its possible to use token choice as a drop-in replacement after training with token rounding routing.

6.1 SNaX's activation memory

We demonstrate that the peak activation memory for SNaX has the lowest activation memory footprint for a single MoE layer as in Figure 4 across all scales. For the 7B model with $n = 256$, our approach reduces memory usage by **45%** compared to ScatterMoE, and more significantly compared to MoMoE. For 30B and 120B models, the gap becomes even wider: for example, at 120B scale, our method saves more than **5GB memory per layer** compared to MoMoE. We also investigate the effect of expert

granularity on activation memory in Figure 1 and we find that SNaX's activation memory, as expected, stays constant with *independent of* expert granularity.

6.2 SNAX 'S TRAINING THROUGHPUT

Figure 5 reports the compute throughput of forward and backward pass. Across all model scales, our method consistently achieves the highest TFLOPs. In small-scale settings (1.4B and 7B), our approach already shows a clear advantage, improving TFLOPs by **40%** compared to ScatterMoE and MoMoE, while also outperforming MegaBlocks. The advantage becomes more pronounced in larger models: for 30B and 120B MoE, SNaX sustains above **500 TFLOPs** in forward and backward passes, whereas other baselines either fail to support certain n sizes (MegaBlocks) or suffer from significant performance degradation (MoMoE). **We measure the training throughput of a 7B MoE model: SNaX on 64 H100s gets roughly the same total throughput as ScatterMoE on 96 H100s.**

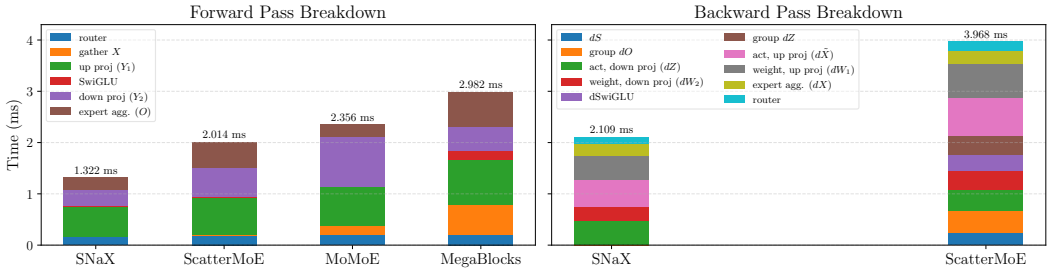

Figure 6: Profile breakdown of forward & backward pass of different MoE kernels

6.3 TOKEN ROUNDING'S TASK EVALUATION WITH SPARSE MOE

We also want to assess the quality of trained model by our token rounding ("TR") algorithm. This naturally leads to comparison with vanilla top-K token choice ("TC") routing. We use token rounding for training and during evaluation we switch to top-K token choice. This assesses the capability of in-place replacement of token rounding with token choice routing after training[6].

We conduct our experiments on 0.5B and 1.4B MoE model scale by varying sparsity in Table 1. We use the OLMoE codebase for running these experiments and construct MoE models with OLMoE base architecture and use OLMo tokenizer (Muennighoff et al., 2025; Groeneveld et al., 2024). We use a deduplicated version of FineWeb-Edu (Ben Allal et al., 2024)[7] for pretraining corpus. More details of our experiments are described in Appendix E.

Across various sparse MoE configurations in Table 1, we observe comparable quality during inference. In fact, we observe empirically that TR achieves slightly lower validation perplexity and higher average accuracy under the extreme sparse MoE (sparsity $< 1/32$) settings for the subtable (c) and (e) in Table 1.

We also benchmark the token rounding's MoE kernel runtime (without router) against top-K token choice routing. We focus on the settings of iso-FLOPs so we fix (T, n, K). We linearly increase the number of experts E to increase sparsity. The results are presented in Figure 7. As we linearly increase the number of experts (consequently sparsity), we observe a drop in TFLOPs for token-choice routing. This is due to the tile quantization effect as the wasted FLOPs spent on padding roughly linearly increases with the MoE sparsity as shown in Figure 3.

For the top right row with 128 experts in Figure 7, we have a MoE model with 1/64 MoE sparsity and intermediate size as 1k, token rounding will materialize 18.2% TFLOPs difference on forward and 4.1% on backward, and end-to-end of 8.5% difference. For the bottom right row with 512 experts in Figure 7 with 1/128 MoE sparsity, token rounding will materialize 28.6% TFLOPs difference on forward and 12.9% on backward, and end-to-end of 17.9% difference. In general, **we observe that the larger intermediate size (as more compute-bound) and the higher MoE sparsity, the gap between token rounding and vanilla token choice will become larger.**

[6]Token rounding is not a token-choice routing method which creates difficulty for autoregressive inference. Here we do not apply any adaptation and switch to vanilla token choice during inference.

[7]`https://huggingface.co/datasets/HuggingFaceTB/smollm-corpus`

Table 1: Comparison of token rounding vs token choice under various sparse MoE training regime. "PPL" refers to the validation perplexity (lower is better) at the end of training. "Avg" is the mean accuracy (higher is better) across the 11 downstream tasks.

(a) **0.5B params, 20B tokens, 8/64 sparsity** (avg # tokens/expert per microbatch = 4096)

Method	PPL	Wino	SIQA	SciQ	PIQA	OBQA	HS	COPA	CSQA	BoolQ	ArcE	ArcC	Avg
TC	16.12	51.0	42.2	80.4	66.0	31.2	37.9	63.0	31.9	59.7	59.3	29.8	50.2
TR	15.94	51.9	41.3	80.8	65.5	35.0	38.7	63.0	31.2	61.4	58.9	27.1	50.4

(b) **0.5B params, 40B tokens, 2/64 sparsity** (avg # tokens/expert per microbatch = 512)

TC	16.57	49.7	40.7	77.3	64.0	33.8	36.5	67.0	31.9	61.3	53.3	27.8	49.4
TR	15.92	51.4	41.6	78.4	65.4	31.6	38.1	65.0	31.0	61.1	57.4	29.1	50.0

(c) **1.8B params, 40B tokens, 8/256 sparsity** (avg # tokens/expert per microbatch = 512)

TC	13.40	50.3	41.2	82.3	70.3	35.4	44.4	69.0	32.4	59.7	61.9	31.1	52.5
TR	13.10	53.4	42.1	81.7	69.6	35.2	45.3	70.0	33.2	61.4	63.0	33.4	53.5

(d) **1.4B params, 50B tokens, 8/128 sparsity** (avg # tokens/expert per microbatch = 2048)

TC	13.35	52.9	42.1	83.5	69.0	34.0	45.1	69.0	34.2	57.6	62.5	30.1	52.7
TR	13.28	52.6	42.6	81.5	69.6	33.6	45.4	67.0	34.8	57.3	63.7	28.1	52.4

(e) **1.4B params, 100B tokens, 2/128 sparsity** (avg # tokens/expert per microbatch = 512)

TC	13.61	49.8	42.0	82.4	67.7	34.2	42.4	68.0	32.8	58.7	59.6	28.8	51.5
TR	13.22	52.8	41.8	80.8	68.7	33.0	43.4	67.0	33.6	60.2	60.7	29.8	52.0

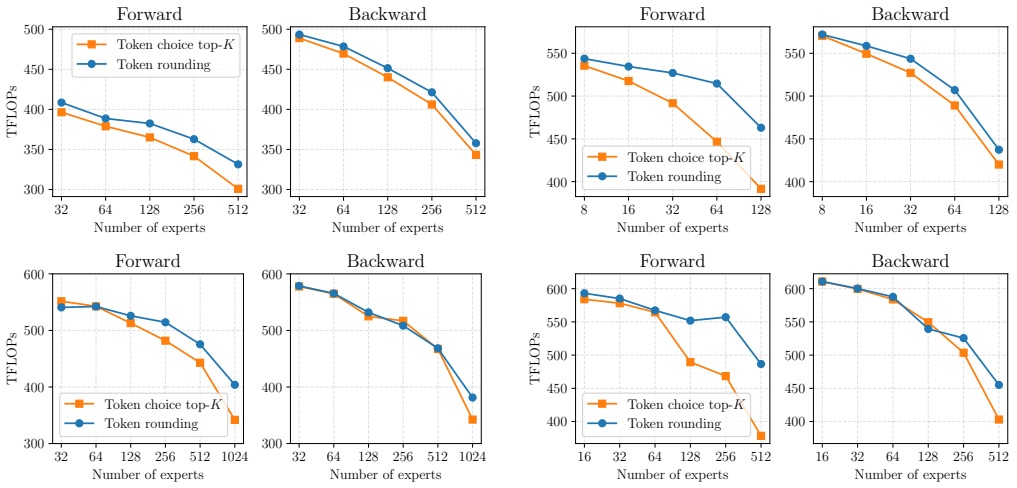

Figure 7: Forward & backward TFLOPs (↑) for different MoE kernels. We use the 7B (first row, left with $n = 256$ and right with $n = 1k$) and 30B MoE (second row, left with $n = 512$ and right with $n = 1k$) configuration and keep K constant while varying E.

7 Conclusion

We presented SNaX, a co-design solution that jointly optimizes MoE architecture and GPU kernels to address the training challenges of granular and sparse MoEs. Our contributions include: (1) a memory-efficient algorithm that minimizes activation size even as MoEs become more fine-grained, (2) GPU kernels that overlap IO with computation for throughput improvement, and (3) tile-aware token rounding that yields additional speedup without quality loss. Future directions include extending to low-precision and microscaling formats (FP8, MXFP8, MXFP4) for further memory savings, and overlapping communication with computation in distributed settings like expert parallelism. We envision future model architecture designs that optimize for quality per compute hour rather than just quality per FLOP—jointly considering algorithmic and hardware efficiency.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Smollm-corpus, 2024. URL https://huggingface.co/datasets/HuggingFaceTB/smollm-corpus.

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. *arXiv preprint arXiv:2412.09764*, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *International conference on machine learning*, pp. 4057–4086. PMLR, 2022.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Bobby Costin, Timor Averbuch, Dhruv Pai, Nathan Chen, and Ben Keigwin. Momoe: Memory optimized mixture of experts. *Tilde Research Blog*, 7 2025. URL https://www.tilderesearch.com/blog/momoe. Blog post.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, and et al. Deepseek-v3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5:288–304, 2023.

IBM Granite. Granite 3.1 language models. https://github.com/ibm-granite/granite-3.1-language-models, 2024. GitHub repository.

Dirk Groeneveld, Iz Beltagy, Evan Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. In *ACL (1)*, 2024.

Horace He and Thinking Machines. Defeating nondeterminism in llm inference, 2025. URL https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/#true-on-policy-rl.

Xu Owen He. Mixture of a million experts. *arXiv preprint arXiv:2407.04153*, 2024.

Zihao Huang, Yu Bao, Qiyang Min, Siyan Chen, Ran Guo, Hongzhi Huang, Defa Zhu, Yutao Zeng, Banggu Wu, Xun Zhou, et al. Ultramemv2: Memory networks scaling to 120b parameters with superior long-context learning. *arXiv preprint arXiv:2508.18756*, 2025a.

Zihao Huang, Qiyang Min, Hongzhi Huang, Yutao Zeng, Defa Zhu, Ran Guo, et al. Ultra-sparse memory network. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Matt Gardner Johannes Welbl, Nelson F. Liu. Crowdsourcing multiple choice science questions. 2017.

Team Kimi, Yifan Bai, Yiping Bao, Guanduo Chen, and et al. Kimi k2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024.

Microsoft. Announcing the availability of phi-3.5 moe in azure ai studio and github. https://techcommunity.microsoft.com/blog/azure-ai-foundry-blog/announcing-the-availability-of-phi-3-5-moe-in-azure-ai-studio-and-github/4256278, 2024. Microsoft Tech Community.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Mistral. Mixtral of experts: A high quality sparse mixture-of-experts. https://mistral.ai/news/mixtral-of-experts, 2024. Mistral AI News.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. Olmoe: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*, 2024.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Evan Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. Olmoe: Open mixture-of-experts language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

NVIDIA. Nvidia blackwell architecture technical brief. Whitepaper, NVIDIA. URL https://resources.nvidia.com/en-us-blackwell-architecture?ncid=no-ncid. Blackwell Architecture.

NVIDIA. Nvidia h100 tensor core gpu architecture: Exceptional performance, scalability, and security for the data center. Whitepaper V1.01, NVIDIA, March 2022. URL https://www.advancedclustering.com/wp-content/uploads/2022/03/gtc22-whitepaper-hopper.pdf. Grace Hopper "Hopper" Architecture.

Nvidia. Nvidia cutlass documentation, 2025. URL https://docs.nvidia.com/cutlass/media/docs/pythonDSL/cute_dsl_general/dsl_introduction.html.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Team Qwen. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

QwenLM. Qwen3: Think deeper, act faster. https://qwenlm.github.io/blog/qwen3/, 2025. Official Blog.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. In *Conference on Empirical Methods in Natural Language Processing*, 2019.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024. URL https://arxiv.org/abs/2407.08608.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421.

Shawn Tan, Yikang Shen, Rameswar Panda, and Aaron Courville. Scattered mixture-of-experts implementation. *arXiv preprint arXiv:2403.08245*, 2024.

Databricks The Mosaic Research Team. Introducing dbrx: A new state-of-the-art open llm. https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm, 2024. Databricks Blog, March 27, 2024.

Changxin Tian, Kunlong Chen, Jia Liu, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. Towards greater leverage: Scaling laws for efficient mixture-of-experts language models. *arXiv preprint arXiv:2507.17702*, 2025.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

Less Wright and Adnan Hoque. Deep dive on cutlass ping-pong gemm kernel, November 2024. URL https://docs.pytorch.org/blog/cutlass-ping-pong-gemm-kernel/. Accessed: 2025-09-21.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. AdaMoE: Token-adaptive routing with null experts for mixture-of-experts language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6223–6235, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.361. URL https://aclanthology.org/2024.findings-emnlp.361/.

Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Huazuo Gao, Jiashi Li, Liyue Zhang, Panpan Huang, Shangyan Zhou, Shirong Ma, Wenfeng Liang, Ying He, Yuqing Wang, Yuxuan Liu, and Y.X. Wei. Insights into deepseek-v3: Scaling challenges and reflections on hardware for ai architectures. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ISCA '25, pp. 1731–1745, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400712616. doi: 10.1145/3695053.3731412. URL https://doi.org/10.1145/3695053.3731412.

Chenggang Zhao, Liang Zhao, Jiashi Li, and Zhean Xu. Deepgemm: clean and efficient fp8 gemm kernels with fine-grained scaling, 2025b. URL https://github.com/deepseek-ai/DeepGEMM.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

A Appendix

A Notations

We present the notations used in this work as follows.

Table 2: Notations used in this paper

Term/Symbol	Explanation
T	number of tokens
d	hidden size
n	expert intermediate size
d	number of model parameters
E	number of experts
K	number of activated experts
W_1	$\mathbb{R}^{E\times d\times 2n}$, weight of up projection
W_2	$\mathbb{R}^{E\times n\times d}$, weight of down projection
π	$T\times E$, a binary valued matrix where $\pi_{t,e}$ represents if token t is routed to expert e
S	$T\times K$, router scores
Z	$TK\times 2n$, output of up projection
Y_1	$TK\times n$, output of SwiGLU
Y_2	$TK\times d$, output of down projection

B Benchmark Configurations

We present a detailed configuration for the benchmark in Fig.4 in Tab.3, and configuration for the benchmark in Fig.7 in Tab.4.

Table 3: Benchmark configurations for different model sizes.

Model Size	T	d	n	E	K
1.4B	40960	768	64	512	32
	40960	768	256	128	8
	40960	768	1024	32	2
7B	24576	1536	64	512	32
	24576	1536	256	128	8
	24576	1536	1024	32	2
30B	32768	4096	256	256	16
	32768	4096	512	128	8
	32768	4096	1024	64	4
120B	32768	4096	512	256	16
	32768	4096	1024	128	8
	32768	4096	2048	64	4

Table 4: Benchmark configurations for token rounding.

Model Size	d	E	K	minibatch size	lr	weight decay	lr scheduler	% of warmup steps
0.5B	768	64	8	0.5M	6e-4	0.01	cosine w/. warmup	10
	768	64	2	1M	6e-4	0.01	cosine w/. warmup	10
	768	256	8	1M	6e-4	0.01	cosine w/. warmup	10
1.5B	768	128	8	1M	4e-4	0.01	cosine w/. warmup	10
	768	128	2	2M	4e-4	0.01	cosine w/. warmup	10

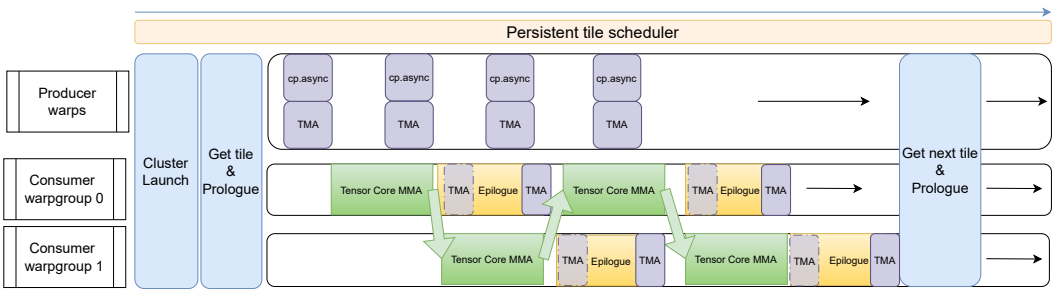

Figure 8: Ping-Pong warpgroup scheduling in Hopper GPU. The green arrows in the figure mean a consumer warpgroup signals the start of epilogue and the other consumer warpgroup can proceed with MMA. Once this step is complete, the roles will be switched. **We mainly use Ping-Pong for Y_2 & dZ kernel as they both have heavy epilogue.**

C More Kernel Details

Algorithm 5 presents the computation of SwiGLU and its derivative in the backward pass, while Algorithm 6 describes the backward pass of the up-projection.

Algorithm 5 SwiGLU and its Derivative

Input : $z = [a, b] \in \mathbb{R}^{*\times n}$, with $a, b, \partial\ell/\partial y_1 \in \mathbb{R}^{*\times(n/2)}$
Output : da, db, y_1

$y_1 \leftarrow a \odot \big(b \odot \sigma(b)\big)$.

$g = \partial\ell/\partial y_1,\ da \leftarrow g \odot \big(b \odot \sigma(b)\big)$,

$db \leftarrow g \odot a \odot \Big(\sigma(b) + b \odot \sigma(b)\big(1 - \sigma(b)\big)\Big)$.

Algorithm 6 SNaX's MoE kernel backward pass of up projection with SwiGLU architecture.

Input : X, π, W_1, dZ
Output : dX, dW_1.

Up-proj act $d\tilde{X}$ kernel $(dZ_e, W_1) \to d\tilde{X}$:
// Grouped GEMM
  Parallel for $e \in [E]$ **do**
    $d\tilde{X}_e \leftarrow dZ_e W_{1,e}^\top$

Up-proj weight dW_1 kernel $(X, dZ, \pi) \to dW_1$:
// Gather + Grouped GEMM
  Parallel for $e \in [E]$ **do**
    $X_e \leftarrow \mathrm{Gather}(X, \pi_{:,e})$
    $dW_{1,e} \leftarrow X_e^\top dZ_e$

Expert aggregation dX kernel $(d\tilde{X}, \pi) \to dX$:
// reduce over each expert results
  Parallel for $t \in [T]$ **do**
    $dX_t \leftarrow \sum_{e\in[E]} \pi_{t,e} d\tilde{X}_{e,t}$

We further describe the Ping-Pong scheduling and index prefetching strategies incorporated in our group GEMM kernel, as illustrated in Fig. 8 and Fig. 9, respectively.

D More Ablations

We conduct ablation studies to assess the effectiveness of MoE granularity and token rounding. In particular, we examine the impact of granularity and find that increasing it consistently improves accuracy, as shown in Tab. 6, which is also consistent with the MoE Scaling Trends mentioned in Tab.5. We further compare token rounding with subroutine as nearest rounding ("NR") with per-expert token count with other rounding methods. Specifically, we compare against stochastic rounding with

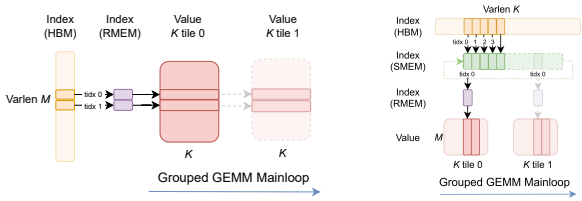

Figure 9: Index fetching strategies for gathering $\mathcal{M}$ (left, used by Y_1 and dZ kernels) and gathering $\mathcal{K}$ (right, used by dW_1 and dW_2 kernels)

Table 5: **MoE Scaling Trends:** *granular & sparser.* Activation ratio is shown as experts activated per token / total experts. Expert granularity is shown as model embedding dimension / expert intermediate size. Here we do not include the shared experts to the MoE sparsity calculation.

Model	Release date	Parameters	MoE sparsity	MoE granularity
Mixtral 8x22B (Mistral, 2024)	11/23	131B	25% (2/8)	$6144/16384 = 0.375$
DBRX (The Mosaic Research Team, 2024)	03/24	132B	25% (4/16)	$6144/10752 = 0.57$
Phi-3.5-MoE (Microsoft, 2024)	09/24	42B	12.5% (2/16)	$4096/6400 = 0.64$
OLMoE (Muennighoff et al., 2024)	09/24	7B	12.5% (8/64)	$2048/1024 = 2$
Granite 3.1-MoE (Granite, 2024)	12/24	3B	20% (8/40)	$1536/512 = 3.00$
DeepSeek-V3 (DeepSeek-AI et al., 2024)	**12/24**	**671B**	**3.125% (8/256)**	$7168/2048 = 3.5$
Qwen3 MoE (QwenLM, 2025)	**04/25**	**235B**	**6.25% (8/128)**	$2048/1536 = 1.33$
QWen3-30B-A3B (Qwen, 2025)	**05/25**	**30.5B**	**6.25% (8/128)**	$2048/768 = 2.67$
Kimi K2 (Kimi et al., 2025)	**07/25**	**1.04T**	**2.08% (8/384)**	$7168/2048 = 3.5$
GPT-OSS-120B (OpenAI, 2025)	08/25	120B	3.125% (4/128)	$2880/2880 = 1$
GLM-4.5-Air (Zeng et al., 2025)	**08/25**	**106B**	**6.25% (8/128)**	$4096/1408 \approx 2.91$

per-expert token count ("SR"), always round up ("UP"). The results are shown in Table 7 and we find that our token rounding algorithm in general is robust to the specific rounding subroutines.

Table 6: Evaluation of MoE w.r.t. granularity under iso-FLOPs regime. "Train" and "Val" refers to the evaluated perplexity (↓) on training and validation set at the end of training. "Avg" is the mean accuracy (↑) across the 11 downstream tasks.

(E, K, n)	Train	Val	Wino	SIQA	SciQ	PIQA	OBQA	HS	COPA	CSQA	BoolQ	ArcE	ArcC	Avg
16, 2, 1024	16.31	16.42	51.6	41.0	78.0	65.5	33.4	37.2	63.0	30.5	61.8	56.3	28.1	49.7
64, 8, 256	16.05	16.03	51.1	41.6	78.9	66.6	31.8	38.6	64.0	32.5	59.7	59.5	32.1	**50.6**
256, 32, 64	16.04	16.12	51.0	42.2	80.4	66.0	31.2	37.9	63.0	31.9	59.7	59.3	29.8	50.2
Dense, iso-FLOPs	19.71	19.90	48.9	41.4	74.9	62.2	30.2	32.6	62.0	31.6	61.7	53.2	27.1	47.8
Dense, iso-param	15.38	15.46	52.1	41.5	78.9	65.3	34.0	39.2	69.0	32.2	58.5	59.3	28.8	50.8

Table 7: Token rounding ablation: comparison of different rounding methods

Method	Train	Val	Wino	SIQA	SciQ	PIQA	OBQA	HS	COPA	CSQA	BoolQ	ArcE	ArcC	Avg
TC	16.85	16.57	49.7	40.7	77.3	64.0	33.8	36.5	67.0	31.9	61.3	53.3	27.8	49.4
NR	16.22	15.92	51.4	41.6	78.4	65.4	31.6	38.1	65.0	31.0	61.1	57.4	29.1	50.0
SR	16.05	15.93	50.8	40.9	77.4	66.9	33.0	38.4	64.0	31.1	60.7	55.8	28.1	49.7
UP	16.07	15.89	50.5	40.9	78.6	64.5	32.2	38.2	68.0	29.9	55.2	54.2	30.1	49.3

E HYPERPARAMETER DETAILS FOR PRETRAINING

We use the OLMoE codebase (Muennighoff et al., 2025) and its downstream tasks in the official configuration: WinoGrande ("wino")(Sakaguchi et al., 2020), Social IQA ("SIQA") (Sap et al., 2019), SciQ(Johannes Welbl, 2017), PIQA(Bisk et al., 2020), OpenBookQA ("OBQA")(Mihaylov et al., 2018), HellaSwag (Zellers et al., 2019), COPA(Roemmele et al., 2011), CommonsenseQA ("CSQA")(Talmor et al., 2019), BoolQ(Clark et al., 2019), Arc-Easy and Arc-Challenge ('ArcE" and "ArcC")(Clark et al., 2018) datasets. We use deduplicated version of FineWebEdu for pretraining corpus and all of model use sequence length 4k. More details of our experiments (e.g. hyperparameters) are described in Appendix E. and all of models use sequence length 4k.

F Deriving the Gradient dZ

For an expert e, let

$$X_e \in \mathbb{R}^{T_e \times d}, \quad W_{1,e} \in \mathbb{R}^{d \times 2n}, \quad W_{2,e} \in \mathbb{R}^{n \times d}.$$

Forward definitions:

$$Z_e = X_e W_{1,e} \in \mathbb{R}^{T_e \times 2n}, \qquad Y_{1,e} = \mathrm{SwiGLU}(Z_e) \in \mathbb{R}^{T_e \times n}, \qquad Y_{2,e} = Y_{1,e} W_{2,e} \in \mathbb{R}^{T_e \times d}.$$

Token aggregation with scores $S = \{s_{t,e}\}$ is

$$O_t = \sum_e s_{t,e} Y_{2,e,t}, \qquad dO_t = \frac{\partial L}{\partial O_t}.$$

(A) Gradients through aggregation. By linearity,

$$\frac{\partial L}{\partial Y_{2,e,t}} = s_{t,e} dO_t \quad \Longrightarrow \quad dY_{2,e} = \mathrm{Broadcast}(s_e) dO_e. \tag{3}$$

Define the grouped-GEMM output

$$Y_{3,e} := dO_e W_{2,e}^\top \in \mathbb{R}^{T_e \times n},$$

then from equation 3

$$dY_{1,e} = dY_{2,e} W_{2,e}^\top = \mathrm{Broadcast}(s_e) Y_{3,e}.$$

The score gradient is

$$\boxed{dS_{t,e} = \langle dO_t, Y_{2,e,t} \rangle = \left\langle dO_t W_{2,e}^\top, Y_{1,e,t} \right\rangle = \langle Y_{3,e,t}, Y_{1,e,t} \rangle}.$$

(B) Gradient through SwiGLU**.** Since $Y_{1,e} = \mathrm{SwiGLU}(Z_e)$ (elementwise),

$$\boxed{dZ_e = \mathrm{dSwiGLU}(dY_{1,e}, Z_e)}.$$

(If $Z = [A, B]$ with $\mathrm{SwiGLU}(Z) = \mathrm{SiLU}(A) \odot B$, then $dA = (dY_1 \odot B) \odot \mathrm{SiLU}'(A)$ and $dB = dY_1 \odot \mathrm{SiLU}(A)$, which the epilogue dSwiGLU computes.)

(C) Down-projection weight gradient. Using equation 3,

$$dW_{2,e} = Y_{1,e}^\top dY_{2,e} = Y_{1,e}^\top \big(\mathrm{Broadcast}(s_e) dO_e\big) = \big(\underbrace{\mathrm{Broadcast}(s_e) Y_{1,e}}_{Y_{4,e}}\big)^\top dO_e.$$

Hence it is sufficient to cache

$$\boxed{Y_{4,e} := \mathrm{Broadcast}(s_e) Y_{1,e}} \quad \text{and then} \quad \boxed{dW_{2,e} = Y_{4,e}^\top dO_e}.$$

Conclusion. The three quantities produced in the algorithm block,

$$\boxed{dS_{t,e} = \langle Y_{3,e,t}, Y_{1,e,t} \rangle}, \qquad \boxed{dZ_e = \mathrm{dSwiGLU}(dY_{1,e}, Z_e)}, \qquad \boxed{Y_{4,e} = \mathrm{Broadcast}(s_e) Y_{1,e}},$$

are exactly the analytical gradients and the sufficient cached tensor required to compute $dW_{2,e}$ via a grouped GEMM, thus matching the paper.

G Use of Large Language Models

In accordance with ICLR 2026's policies on Large Language Model (LLM) usage, we disclose that LLMs were employed in the preparation of this submission. Specifically, LLMs assisted in refining the language and generating initial versions of plotting code. Subsequent to their generation, all outputs were thoroughly reviewed and validated by human authors to ensure accuracy and compliance with ethical standards. This approach aligns with ICLR's Code of Ethics, which mandates transparency and accountability in research practices.