FARSKIP-COLLECTIVE: UNHOBBLING BLOCKING COMMUNICATION IN MIXTURE OF EXPERTS MODELS

Yonatan Dukler [1] **Guihong Li** [1] **Deval Shah** [1] **Jiang Liu** [1] **Vikram Appia** [1] **Emad Barsoum** [1]

ABSTRACT

Blocking communication presents a major hurdle in running MoEs efficiently in distributed settings. To address this, we present *FarSkip-Collective* which modifies the architecture of modern models to enable overlapping of their computation with communication. Our approach modifies the architecture to skip connections in the model and it is unclear a priori whether the modified model architecture can remain as capable, especially for large state-of-the-art models and while modifying all of the model layers. We answer this question in the affirmative and fully convert a series of state-of-the-art models varying from 16B to 109B parameters to enable overlapping of their communication while achieving accuracy that is comparable with their original open-source releases. For example, we convert Llama 4 Scout (109B) via self-distillation and achieve average accuracy within 1% of its instruction tuned release averaged over a wide range of downstream evaluations. In addition to demonstrating retained accuracy of the large modified models, we realize the benefits of FarSkip-Collective through optimized implementations that explicitly overlap communication with computation, accelerating both training and inference in existing frameworks. For inference, we demonstrate 32.6% speedup in Time To First Token when serving a converted DeepSeek-V3 architecture with expert parallelism in SGLang and achieve 97.3% communication-computation overlap during the prefill stage. During training, our approach enables 88.9% communication overlap of the all-to-all communication collectives when pre-training DeepSeek-V3 MoE layers with expert parallelism.

1 INTRODUCTION

Mixture of Experts (MoE) models have emerged as the de-facto model architecture for leading large language models (LLMs) in recent years (Shazeer et al., 2017; Fedus et al., 2022; Dai et al., 2024). MoEs activate a sparse subset of their total parameters, usually via a mixture of experts layer to replace the dense MLP sub-block. The sparsity and reduced computational cost of MoEs makes them amenable to even larger parameter scaling with new open-source MoE models routinely exceeding 500B total parameters (DeepSeek-AI, 2025; Kimi et al., 2025; Meta AI, 2025). Because of the fixed memory capacity of compute accelerators, MoEs require even more distribution for training and inference setups as compared to their dense LLM counterparts (DeepSeek-AI, 2024).

However, distributed training and inference is not without a cost as activations and model weights need to be communicated between devices quickly; especially when the communication operations are *blocking*, where the communication can only commence at a particular stage of processing and the next operation in the compute graph relies on it. Blocking communication patterns result in *exposed* idle time when the accelerator is not running computations; this commonly appears in popular parallelism techniques such as Expert, Sequence, and Tensor Parallelism (Narayanan et al., 2021; Shazeer et al., 2017) and is especially tricky to overcome during inference. The new age of large and sparsely activated MoE architectures along with improved hardware computation speeds exacerbates these issues as communication becomes a relatively larger portion of the end-to-end workload (Zhao et al., 2025).

In this work we present a method to modify models' architectures to use available activations for the next computation at the onset of the communication call, which may be outdated or partially materialized, in order to avoid the blocking communication and start the next computation during the communication operation. We name our approach FarSkip-Collective as we start the next computation immediately using available activations and run the communication collective in parallel, far-skipping the communicated result to the residual of the next layer and making that activation available for future layers. By running communication overlapped with computation, as long as the duration of the computation leading up to the next residual is longer than the communication we avoid idle compute time.

[1]Advanced Micro Devices Inc. (AMD) . Correspondence to: Yonatan Dukler <yonatan.dukler@amd.com>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026. Copyright 2026 by the author(s).

Mathematically, FarSkip-Collective is "dropping" connections of the network as the input to the next computation is now a residual which does not include the latest communicated output block. Characteristically, this can damage the capabilities of the model architecture. We therefore focus on evaluating whether the modified architecture connectivity can perform comparably with regular MoE connectivity and study the capabilities of models exhaustively over a wide array of evaluations and model scales.

By developing FarSkip-Collective Self-Distill (FCSD) we answer in the affirmative, demonstrating that state-of-the-art open-source MoE models of scales ranging from 16B to 109B can be fully converted into FarSkip-Collective models with minimal loss of capabilities of the model demonstrated at varying scales (Qwen, 2025; Meta AI, 2025; DeepSeek-AI et al., 2024). FCSD is a simple yet effective knowledge distillation recipe we identified via a systematic study which can be applied to any model with the absence of a powerful teacher. In addition, when ablating the FarSkip-Collective architecture by pretraining from scratch, we observe on-par performance at the 16B model scale, further corroborating the viability of the architecture.

Independently of this work, we became aware of recent works exploring similar approaches, modifying the model architecture and running computations with either "outdated" (Zhang et al., 2025) or "partial" (Prabhakar et al., 2024; Lamprecht et al., 2025) activations to overlap communication. In contrast with our work, existing approaches focus has been limited to tensor-parallelism in dense models which are not designed for the communication patterns of MoEs. Such works have also only studied the problem of model capabilities at order of magnitude smaller-scale models or achieved only partial modification of the model layers. It therefore remained unclear whether FarSkip's modified connectivity can be applied to overlap communication in all layers of an MoE and continue to perform at the scale of today's state-of-the-art MoE LLMs. It is not uncommon for model architecture modification to show promise at a smaller size but scale poorly when studied at frontier-LLM scale with more challenging tasks. We find the results with FCSD encouraging in that even at the 100B+ model scale over a wide range of generation and likelihood-based tasks, FarSkip-Collective can achieve within 1% on average from the original model's accuracy.

Just because the new model architecture obviates dependencies in the model that regularly lead to blocking communication, it does not imply the architecture will automatically overlap computation and communication in practice if not implemented carefully. To this end, we realize the overlapping opportunities of the models by developing performant and overlapped implementations for training and inference which we have tested extensively. For training, we develop on top of Megatron-LM (Narayanan et al., 2021) and Primus (AMD TAS, 2025). We achieve 88.4% computation-communication overlapping of the Expert Parallelism communication (87.6% in forward, 89.0% in backward) using asynchronous execution of communication collectives and novel scheduling techniques at the PyTorch API layer. On the inference side, we implement our method on top of vLLM & SGLang and integrate our approach with HIP/CUDA-graphs achieving up to 97.6% communication overlap. Overall we implement our approach for wide use across different hardware and avoid low-level kernel optimizations in favor of more general implementation at the PyTorch layer. We open-source our overlapped implementations and provide easy integration with the upstream frameworks at `https://github.com/AMD-AGI/FarSkip-Collective`.

We summarize our contributions as follows:

- We present FarSkip-Collectives, a method to convert the execution dependency of model layers that eliminates blocking communication patterns in MoEs, thereby allowing inference and training speedups.
- We demonstrate at the 100B+ parameter scale that models using the FarSkip-Collective architecture modifications retain the capabilities of modern transformer blocks while speeding up their execution in distributed settings. In particular we fully convert the Llama 4 Scout MoE (109B) while observing only small drop in model performance of 1% on average compared with the instruction-tuned open-sourced model release. (In the case of pre-training from scratch we demonstrate an average drop in performance of 0.3% when ablating the DeepSeek-V2-Lite architecture (16B)).
- We develop FCSD, an efficient and general knowledge self-distillation recipe to convert existing LLMs into FarSkip models and demonstrate it by converting DeepSeek-V2 Lite (16B), Qwen-3-30B-MoE (30B), and Llama 4 Scout (109B) with $<$ 10B training tokens.
- For large-scale training, we integrate our method into Megatron-LM and achieve 88.9% communication overlap for the previously blocking all-to-all communication responsible for MoE expert parallelism.
- For model serving, we develop an optimized implementation of FarSkip in SGLang and vLLM that overlaps the communication for distributed inference. For example, for the modified Llama-4 Scout model, we achieve 18.5% speedup in Time To First Token.

The rest of the paper is organized as follows, in Section 2 we present background followed by our approach in Section 3

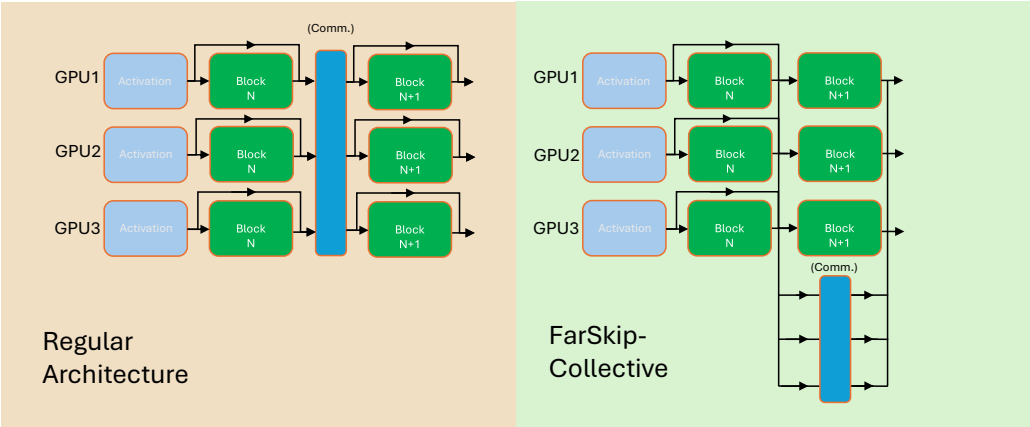


Figure 1: FarSkip-Collective modifies the connectivity between sub-blocks to avoid waiting for communication collectives. Computation continues with available activations, partial (e.g., Block N output) or outdated (e.g., Activation).

and explicit optimized implementation of the method in Section 4. In Section 5, we present our experimental results and review related works in Section 6 followed by conclusion (Section 7).

2 Background

2.1 MoE parallelism at training and inference

Two of the key parallelism techniques for MoE training and inference are Tensor and Expert Parallelism.

Tensor Parallelism In Tensor Parallelism (TP), an MLP or a Self-Attention sub-block will be split by slicing the sub-block's weight matrices evenly across their columns or rows. Let $A \in \mathbb{R}^{B\times d}$ be a model input activation for a modern MLP layer of the form

$$\text{MLP}(A) = \sigma(AW_1^\top \cdot g(AW_2^\top))W_3^\top,$$

with g, σ being entrywise non-linearities, $W_1, W_2 \in \mathbb{R}^{c\times d}, W_3 \in \mathbb{R}^{d\times c}$. TP of size k will split the matrices into

$$W^i_{\{1,2\}} = \big(W_{\{1,2\}}\big)_{[i*c/k:(i+1)*c/k,:]} \in \mathbb{R}^{(c/k)\times d},$$
$$W^i_3 = \big(W_3\big)_{[:,i*c/k:(i+1)*c/k]} \in \mathbb{R}^{d\times(c/k)},$$

for $0 \le i \le k-1$. Then computation of each TP rank can run independently until the end of the sub-block where an all-reduce communication-collective is applied to construct the final output,

$$\text{MLP}(A)_i = \sigma(AW_1^{i\top} \cdot g(AW_2^{i\top}))W_3^{i\top}, \quad (1)$$
$$\text{MLP}(A) = \text{all-reduce}\big(\text{MLP}(A)_i\big). \quad (2)$$

The selection of Column-Parallel $W^i_{\{1,2\}}$ split followed by Row-Parallel W^i_3 split makes it possible to only apply communication once at the end of the layer.

For multi-head self-attention (or efficient variants) implemented with TP, the computation is decomposed into independent computations partitioned across the different attention heads; this allows for a similar implementation employing Column-Parallelism (Q, K, V) followed by Row-Parallelism (O) and a single all-reduce.

Expert Parallelism The key parallelism component of MoE layers is Expert Parallelism (EP). An MoE layer with E experts generalizes the MLP as

$$\text{MoE}(A) = \sum_{j=1}^{E} G(A)_j \cdot \text{MLP}^j(A),$$

where $G(A) = s(AW_R^\top)$ is a linear classification layer followed by a sparse router selection function $s()$ activating only a subset of the MLP^{j}s. Here MLP$^j(A)$ refers to a distinct "expert" for $1 \le j \le E$. With EP of rank k, subsets of $\sim E/k$ experts will be distributed across the k parallel ranks. Unlike TP, during training different input activations will be mapped to the different experts based on the router selection $G(A)$. Mechanically, specific token vectors $A_{[l,:]} \in \mathbb{R}^d$ will be grouped and mapped to a subset of experts $P_l \subset \{1, \dots k\}$ requiring permutation of A followed by an all-to-all collective that sends and receives data between the ranks according to the router-defined data-partition map.

$$A_i = R_i \times A \text{ placed on rank } i \quad \text{(Dispatch)},$$

with $R_i \in \{0,1\}^{B_i\times d}$ being the indicator of $G(A)$; this is referred to as "Dispatch" and will have different bandwidth

requirements depending on factors such as the number of experts E and the sparsity of s (e.g., the assigned "TopK" value). After each expert receives its dedicated tokens and computes $\text{MLP}^j(A)$ of the relevant experts on its rank, a dual all-to-all collective, referred to as "Combine" is applied to aggregate and sum the routed-experts' activations back into the output activation of the MoE layer. Modern MoE designs will also typically include "shared-experts" MLP layers that will run on all tokens in addition to the routed MoE experts.

Putting these together, typical training execution of an MoE transformer layer would follow 1) the attention sub-block (including layer-norm) 2) potential post-attention communication collective if TP or CP is enabled 3) compute the gating and router scores (duplicated on each rank) 4) initiate routed expert Dispatch communication 5) routed expert computation and potentially shared-experts computation, and finally 6) Combine collective operation to aggregate the routed-experts from the different ranks. This execution leads to three potential blocking communication bubbles: (a) post-attention blocking communication if TP/CP is used, (b) during Dispatch and (c) during Combine, although (c) may be partially overlapped if shared experts are present.

For inference, there are different approaches for MoE execution, with only some involving all-to-all "Dispatch + Combine" (DeepSeek-AI, 2024). In this work, we focus on the approach implemented by vLLM and SGLang where an all-reduce is used instead of the all-to-all collective and activations are replicated and indexed across the expert parallel ranks.

2.2 Model Distillation

FarSkip-Collective modifies the model architecture followed by self-distillation to recover the original model's capabilities. As a basic approach, one can simply fine-tune the model with high-quality data via Supervised Fine-tuning (SFT) according to

$$\mathcal{L}_{\text{SFT}}(\theta) = -\,\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\sum_{t=1}^{|y|}\log p_\theta(y_t \mid x, y_{<t})\right], \quad (3)$$

where θ denotes the model parameters, and $(x, y) \sim \mathcal{D}$ are input–output supervision pairs with $|y|$ output tokens in the pair.

When converting the model with a target model in mind (e.g., in our case aiming to recover the original model), one may train with knowledge distillation which is defined against a fixed teacher model q. Logit-based knowledge distillation optimizes p_θ to match the teacher's predictive distribution according to

$$\mathcal{L}_{\text{KD}}(\theta) = \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[\sum_{t=1}^{T}\text{KL}\big(q(\cdot \mid x, y_{<t}) \,\|\, p_\theta(\cdot \mid x, y_{<t})\big)\right]. \quad (4)$$

In addition to an objective on the model outputs, one may also aim to increase alignment of a model to a teacher model q by aligning the model with the intermediate representations of the teacher (Yang et al., 2025) as

$$\mathcal{L}_{L2}(\theta) = \sum_{i=1}^{L} \| o_i(\theta) - t_i \|_2^2, \quad (5)$$

where o_i and t_i denote the matching hidden activations of the student and teacher models, respectively, over L layers.

3 FarSkip-Collective Framework

Modern networks use residual connections, meaning that self-attention, MLP, and MoE sub-block outputs are incrementally added to a residual activation stream. Denote the output activation of a network after k layers as o_k, and the ith sub-block (layer) of the network as f_i. The output o_k is computed as

$$o_k = f_1(o_0) + f_2(o_1) + \cdots + f_k(o_{k-1}). \quad (6)$$

For large models, producing f_k can involve blocking communication which stops o_k from being used as an input to f_{k+1} until o_k is communicated – leading to idle computation resources. We propose to use an available activation instead, denoted as o_k^*, to be used as input to f_{k+1} and compute the next layer while the communication collective is producing o_k. o_{k+1} will be updated with o_k once it is ready; however, now the communication of o_k can be "far-skipped" and overlapped over the duration of the computation of $f_{k+1}(o_k^*)$ by using o_k^* as the input to the block.

$$\begin{aligned} o_{k+1} &= o_k + f_{k+1}(o_k^*) \\ &= f_1(o_0) + f_2(o_1^*) + \cdots + f_{k+1}(o_k^*) \end{aligned} \quad (7)$$

We consider two options for o_k^*,

$$\begin{cases} o_k^* = o_{k-1} \quad \text{(outdated)} & \text{(8a)} \\ \text{or} \\ o_k^* = o_{k-1} + f_k^*(o_{k-1}^*) \quad \text{(partial)} & \text{(8b)} \end{cases}$$

f_k^* denotes part of the activation of block f_k which is ready before the collective, e.g., $\text{MLP}_i(A)$ in Eq. 2. In both cases, the output activation o_k will consist of the same number of blocks as before, but the difference will be in terms of the input activation into each f_i. Crucially the input to f_{k+1}, o_k^*, has access to all of the previous block outputs except for the full representation of block f_k; that is all future layers f_j for $j \geq k+2$ will have access to the full f_k outputs.

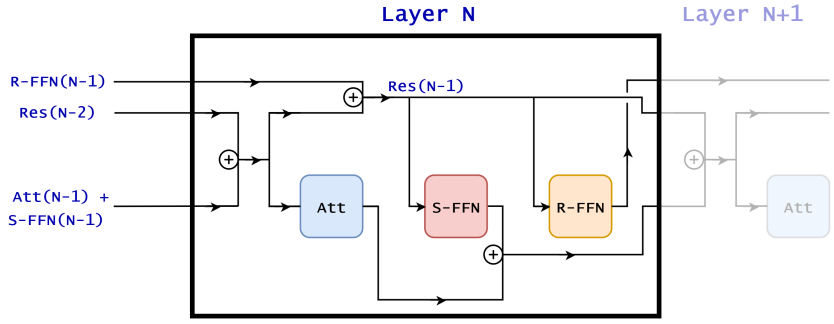


Figure 2: FarSkip-Collective MoE block diagram. The attention (Att) block's input corresponds to partial output from the previous layer. Both the shared and routed experts (S-FFN, R-FFN) parts of the MLP block receive outdated input (residual of layer N-1). Each input is at most a single layer behind while communication can be fully overlapped. Since the output of attention and shared-experts are used together, their collectives may be combined to reduce bandwidth.

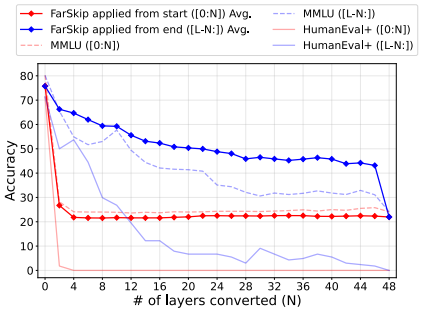


Figure 3: Accuracy of Qwen-3-30B-MoE modified with N FarSkip-Collective layers without training. Replacing the last N layers (blue) and first N layers (red).

When developing FarSkip-Collective for MoEs we balance two competing goals: minimize the number of activations dropped from each o_k^* while maximizing possible overlap. Specifically for the MoE settings, we consider the possible communication patterns of the attention block, and shared and routed experts parts of the MLP block. The routed-experts are parallelized with EP which for training will require Dispatch and Combine, whereas the shared-experts and attention computations are not router conditional and tend to have compatible parallelisms (e.g. replication / TP / CP). When developing the FarSkip-connectivity we aim to combine the latter two's communications to reduce bandwidth, while at the same time overlapping the routed-experts communication.

To this end we optimize the execution order and connectivity by considering 1) the routed-experts communication and 2) attention and shared-experts communication (if not replicated). The most minimal activation dropping scheme that allows overlapping both involves using the partial formulation for attention since we can prepare the non-routed activation of the MLP without blocking the attention computation. Then for the MLP, use the outdated formulation which can not depend on the previous attention block while remaining overlapped. We present this connectivity diagram in Fig. 2 demonstrating the input and output to each computation.

Mathematically let $o_k^*(\text{attn})$ be the input to the kth attention sub-block, then the partial activation

$$\text{attn-in}_k := o_k^*(\text{attn}) = o_{k-2} + \text{attn-out}_{k-1} + \text{shared-exp-out}_{k-1}$$

serves as the input. For the MLP block input (shared and routed experts), $o_k^*(\text{mlp})$ is the outdated activation

$$\text{mlp-in}_k := o_k^*(\text{mlp}) = o_{k-1}.$$

Compared with attn-in_k, mlp-in_k can be decomposed as $o_{k-1} = \text{attn-in}_k + \text{routed-exp-out}_{k-1}$ with the routed-expert activations added after Combine finishes.

The FarSkip-Collective connectivity has the property that attention computation is not dependent on the preceding routed-experts which means the Combine operation of the previous routed-experts can be overlapped with the first part of the attention computation (and shared-experts computation depending on the parallelism). In addition the MLP input is independent of the attention output which means that the Dispatch can follow the Combine immediately after computing the new router scores and then run overlapped with the second part of the attention computation. Hence FarSkip-Collectives combines the outdated and partial formulations for o_k^* to achieve overlap with provable minimal activation dropping for the MoE layer. We provide the details of the execution and overlapping of FarSkip-Collective in the next section.

We also leave more aggressive multi-block variants of "far-skipping" as future work, which may be useful if the communication significantly exceeds the duration of the full sub-block's computation time, for example in the case of extremely sparse and large-scale MoEs or different hardware paradigms.

3.1 Distilling existing models with the FarSkip-Collective Framework

The FarSkip-Collective method modifies the architecture connectivity without changing the model's parameter layout or dense compute operations, making it possible to execute an existing checkpoint with FarSkip-Collective connectivity using the same kernels with relatively few modifications to the model definition. In Fig. 3, we give a simple demonstra-

tion of this by loading the original Qwen-3-30B MoE model checkpoint into models with various numbers of FarSkip-Collective layers activated and evaluate its performance on different benchmarks. Without re-training, we observe that as we increase the number of converted layers, the model performance degrades considerably, and the model achieves random baseline accuracy on MMLU and 0% on HumanEval+ when fully converted. This is unsurprising, as we pass different input activations than the ones the model was trained with, leading to out-of-distribution outputs.

We, however, show that by continuing training the original checkpoint via Knowledge Distillation (4) using typical instruction tuning data, we are able to recover the original model's performance in a small fraction of the compute needed to retrain it from scratch with the FarSkip-Collective architecture ($\sim 100 - 1000\times$ cheaper). We systematically study different approaches for the distillation training which we present in Tab. 2 and find that using KL-based knowledge distillation with the original model as the teacher (self-distillation) performs best or on par as compared to the different approaches we tested. We also study the effect of different aspects such as the batch-size and learning rate and find that they also contribute to the final model performance and training stability, which culminates in our simple and robust "FarSkip-Collective Self-Distillation" (FCSD) recipe to convert any MoE model into the FarSkip-Collective connectivity. Self-distillation to the original model's logits provides granular signal that better aligns with the existing representations. Training using the logits as the training signal, also reduces the dependency on meticulous and high quality instruction tuning data as the FarSkip-Collective model is instead trained to align with the original model's predictions rather than potentially low quality data itself. On the flip side this approach will also essentially cap the final performance of the student model to that of the teacher model especially since they are of the same size. We however find FCSD attractive in its generality since unlike other distillation approaches we do not rely on strong external teacher models and training with FCSD gives strong performance under 10B tokens.

Since the modified architecture only modifies the connectivity of the original model, (i.e., all the model parameters have the same shapes etc.) FCSD is effective in reducing the distance from the original model, nonetheless later in training KL may lead to training instabilities as small discrepancies between the teacher and student model lead to occasional large gradients and training instabilities. We tested different approaches to overcome this but find that training FCSD with early stopping enables us to avoid the issue. For the early stopping validation we use the MBPP+(Liu et al., 2023a) dataset as a fast proxy for detecting instabilities and evaluate every 1000 training steps with a patience of 20 evaluations and performance delta of 2%. MBPP+ provides for quick evaluation and being a code-generation dataset it is sensitive to damaging distribution shifts caused by training instabilities. When evaluating conversion using a direct SFT objective as a baseline we apply the same early stopping procedure for fair comparison.

4 Explicit overlapping of FarSkip models

Modern GPUs, equipped with hundreds of independent Compute Units (Streaming Multiprocessors), can process multiple Queues (Streams) of kernels independently by scheduling work on different sets of processing units at the same time (Zhao et al., 2023; PyTorch Team, 2024; DeepSeek-AI, 2024). Both computation and communication operations utilize compute units to run operations, with communication operations only utilizing a fraction of the total available units, allowing for overlap with computation. Computation-communication overlap, however, requires dedicated implementation, and aside from standard patterns, modern frameworks such as PyTorch and JAX will not accomplish this automatically. Therefore, even though the modified FarSkip-Collective model will logically facilitate parallel and non-blocking flow through the computational graph, without explicit implementation, the models will not automatically overlap communication with computation. Below, we describe our optimized and extensively tested explicitly-overlapping implementation of the FarSkip-Collective framework. As a design choice, we aim to make our implementation generalizable and as hardware-independent as possible by sticking to the framework level as opposed to lower-level kernels or Triton. In particular to enable scheduling of non-blocking communication calls, we rely on `torch.dist`'s `async_op=True` parameter or using the `torch.cuda.Stream()` context that enables more granular control of scheduling of kernels on the non-main queue. Note that by overlapping operations, one diverts some of the processing units, which can lead to unavoidable slowdown in computations as compared to when they are solely executed on the hardware.

4.1 Training

For training, we consider MegatronLM GPT training with MoE layers. The basic settings of training include shared-experts, Multi-head Latent Attention (MLA) and running with EP while replicating attention (no TP), following the DeepSeek's V3 model training recipe. Note however that FarSkip also allows for full overlapping of mixed EP parallelisms during training. Regularly in this setup, the all-to-all collective will lead to two communication bubbles as part of Dispatch and Combine, appearing both during the forward and backward pass of each layer. With the blocking dependency removed by our modified architecture, we modify the

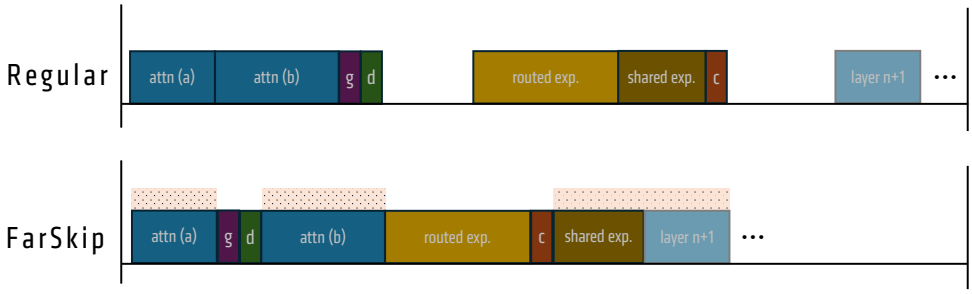


Figure 4: FarSkip-Collective MoE layer main operator execution. *g,d* and *c* refer to gating+routing, Dispatch start, and Combine start respectively. For communication operations, we only denote the starting point of the operation. The overlapping window enabled by FarSkip-Collective is illustrated with the shaded area above the operators.

execution order discussed in Section 2. In particular, we split the attention sub-block calculation into two parts: a) MLA preparation of (q, k, v) and b) core-attention + output projection. This enables us to easily launch a communication kernel asynchronously between the two parts and then immediately continue the attention calculation. For this DeepSeek model setup, we execute the FarSkip-Collective MoE layer forward pass as 1) attention part (a) computation, 2) synchronize Combine if last layer was a FarSkip-Collective MoE layer, 3) compute MoE gating and router scores, 4) initiate Dispatch (async mode will return immediately), 5) compute attention part (b). At this point, attention computation finished and the routed tokens should be dispatched; we 6) synchronize the Dispatch collective followed by computing the routed-experts and then 7) initiate Combine (async mode), lastly followed by 8) running shared-experts computation. We provide a visual demonstration of this layer execution as compared with the regular operation flow in Fig. 4. With FarSkip we maximize the window of all-to-all communication overlap during the forward call as

$$\begin{aligned} T_{\text{Dispatch}} + T_{\text{Combine}} &\leq T_{\text{overlapped computation}} \\ &= T_{\text{layer}} - (T_{\text{Routed-Experts}} + T_{\text{Gate}}). \end{aligned} \tag{9}$$

The only computations of the layers that cannot be overlapped with Dispatch and Combine communication are the routed-experts and gating operator. This is because with the modified architecture the routed-experts consume input from the previous Dispatch and produce outputs that will serve as the input for the next Combine call. Similarly, the gating operator will consume input from the previous Combine call and produce outputs that will serve as an input to the next Dispatch call.

For the backward pass, we would like to overlap the Combine and Dispatch gradient calls which will also trigger blocking all-to-all communication collectives. If run naively, one will need to run the backward communication outside of async mode as its outputs will need to be synchronized for the next gradient in the graph, making the communication blocking again. The standard approach to avoid this is to explicitly control the operator ordering in backward by using a custom `torch.autograd.Function` for the *entire* MoE transformer block layer computation. Implementing just Dispatch and Combine with a custom backward or sub-parts does not enable one to define the synchronization points outside of it, which is needed for overlap. Implementing such a large layer's backward computation graph manually, however, is tedious and error-prone as each of the operations and weights needs to be wired correctly to their next input.

Instead, we present two innovative techniques that together cleanly achieve overlap while continuing to rely on the automatic autograd for backward propagation. First, we achieve async-safe all-to-all backward communication functions that are both async and yet have synchronization points right before accessing their gradient results. Second, we "hijack" the priority ordering of `torch.autograd` via the Sequence Number PyTorch autograd's internal implementation[1] to control the gradient node execution ordering and ensure the async-safe backward calls are seperated far apart to have sufficient overlap before their synchronization points are triggered. We provide an expanded explanation and details of our novel overlapped backward technique in Appendix C. Using our optimized implementation, we achieve an overall overlap of 88.4% of the all-to-all communication time when training a DeepSeek-V2 Lite with EP8 parallelism as observed in Tab. 4. Note that the first all-to-all in backward and last all-to-all in forward cannot be overlapped as there are no additional computation candidates.

4.2 Inference

For inference, we implemented FarSkip in vLLM and later extended it to SGLang. SGLang and vLLM serve as modern LLM inference engines with TP, EP, and PP support for MoEs such as DeepSeek. Unlike other MoE EP implementations that use a pair of all-to-all collectives for Dispatch

[1] See "forward-backward correlation" discussion of Sequence Number autograd internals in paragraph below the anchor https://docs.pytorch.org/docs/stable/autograd.html#torch.autograd.profiler.emit_nvtx

and Combine, in vLLM and SGLang model activations are replicated across the ranks but model weights including expert weights are still distributed via EP and TP. This approach eliminates the need for Dispatch and Combine and is implemented with all-reduce operations applied to the activations after the MLP layers finish. For the attention sub-block, vLLM and SGLang adopt a regular TP approach with an all-reduce collective and both of the all-reduce calls are regularly blocking as the activations are needed in the next layer.

To implement FarSkip-Collective for the routed-experts MoE layer, we run the `all-reduce` in async-op mode and synchronize it only before the next MoE computations, as those activations are no longer needed for Dispatch and can be overlapped with attention computation. To reduce communication bandwidth, we present an optimized delayed approach for the communication of the attention outputs where usually communication appears in `RowParallelLinear` of the output projection. Instead we defer their communication and combine the attention and shared-experts activations into a single all-reduce call of the same message size by reducing the summed activation. This all-reduce call can then be overlapped with the routed-expert's computation. In inference with this parallelism settings, the overlappable window corresponds to the complement of the shared-experts computation ($T_{\text{Layer}} - T_{\text{shared Experts}}$).

For specialized attention such as MLA in DeepSeek models, prefill and generation will run different fused kernels, and we treat each case separately but defer and combine the all-reduce async-op call in each scenario. To integrate FarSkip with HIP/CUDA-graphs we use graph-compatible communication API calls and use direct Python binding of NCCL (PyNCCL). We test our inference pipeline using the self-distilled models fine-tuned with FCSD and observe that our distillation recovers the model performance in chat-based generation.

5 Experiments

In this section we describe our experiments evaluating the model capabilities of FarSkip-Collective models, followed by evaluation of the FarSkip-enabled optimized overlapped implementation.

5.1 Model Capabilities

We present the main results of our distillation experiments in Tab. 1, where we consider three open-source state-of-the-art MoEs at different scales: DeepSeek-V2-Lite (16B-A3B), Qwen-3-30B MoE (30B-A3B), and Llama-4 Scout (109B-A17B). Each model's checkpoint corresponds to the instruction-tuned / chat version of the open-source model release. We apply FarSkip-Collective to all of the model's layers and train each model for up to 10B tokens of instruction tuning data. We train with standard settings using AdamW, cosine-annealing learning rate scheduler, and 1000-step warm-up period. We use relativity large batch-size and learning rate with FCSD and run short sweeps to identify the best batch-size and learning rate for each model. In particular we conduct two sweeps for 2000 training steps each, first for batch-size selection among $\{2^{16}, 2^{17}, 2^{18}\}$ with a learning rate of 2e-5 followed by a learning rate sweep among {2e-5,4e-5,8e-5} where we use the training loss for selection. We observe rapid initial improvement on all benchmarks using the KL objective and further gradual improvement as training continues. We also observe occasional training instabilities where the distilled FarSkip-Collective model exhibits mode-collapse later in training. We tested different approaches to overcome this in Tab. 2, and resort to using early stopping with MBPP+. As a baseline conversion method, we test standard SFT training with the same training schedule and sweep selection for the batch-size and learning rate which is similar to the approach in (Zhang et al., 2025). In addition we apply the same early stopping as FCSD. Overall, SFT significantly underperforms the FCSD recipe and the resulting model exhibits catastrophic forgetting, particularly in generation tasks. With our knowledge distillation training, even for code generation task such as HumanEval+ which are more easily affected by distribution shifts, FarSkip-Collective models are able to achieve performance close to the original instruction-tuned checkpoint, demonstrating the inherent capacity of the modified architecture. Nonetheless we note that for some datasets there remains a gap from the original instructed tuned checkpoint, for example for Llama-4-Scout FSCD still results in 4.1 gap in MMLU performance vs. 10.3 gap with SFT. Overall, we believe additional scaling of FCSD, improved data mixtures and other techniques such as model merging can aid in closing any remaining performance gaps in the model conversion which we leave for future work.

In this vein, we report pre-training from scratch results in Tab. 3 and observe on par performance with 0.3% gap on average, when conducting an architecture ablation with FarSkip-Collective while fixing all non-architectural aspects. In the pre-training settings we use a DeepSeek-V2-Lite model architecture (16B) and train with a 200B token budget, evaluating the final checkpoint. We further discuss the pre-training results in Appendix A.

In Tab. 2, we study the effect of different distillation techniques and the effect of partial conversion of the model into FarSkip-Collective layers. We start from DeepSeek-V2-Lite MoE and use a short training schedule of 300M tokens using a batch size of 2^{16} tokens, 2e-5 learning rate annealed to 1e-5 and test 1) SFT training (Eq. 3) 2) KL + INTER. L2 Combining KL with intermediate activation L2 loss (Eq. 4 + Eq. 5) for which we sweep over different L2 loss coefficients. 3)

Table 1: Original and distilled FarSkip-Collective model performance on downstream evaluation tasks.

Model	Params	PIQA	ARC-E	ARC-C	HS	CSQA	WG	HEval+	MMLU	OpenBook	GSM-8K	MBPP+	Avg
DeepSeek-V2-Lite (Original)	16B	80.1	80.2	53.8	80.8	69.1	72.1	40.2	56.8	45.2	70.1	60.8	64.5
DeepSeek-V2-Lite (FCSD)	16B	79.9	78.9	50.0	76.9	70.1	68.4	41.5	50.5	41.8	64.2	59.8	62.0
DeepSeek-V2-Lite (SFT)	16B	78.2	74.3	43.8	74.1	65.5	69.0	11.0	48.0	41.2	54.3	45.8	55.0
Qwen-3-30B MoE (Original)	30B	80.5	84.8	61.9	79.7	84.8	72.9	73.8	80.2	45.0	86.9	84.4	75.9
Qwen-3-30B MoE (FCSD)	30B	80.4	83.3	58.5	77.2	84.9	74.0	73.2	74.0	42.8	87.6	74.4	73.7
Qwen-3-30B MoE (SFT)	30B	77.8	69.4	44.9	75.6	68.9	65.6	0.6	63.1	41.4	76.0	71.7	59.5
Llama-4-Scout (Original)	109B	81.1	87.3	64.6	82.9	84.4	76.6	62.2	80.0	45.2	88.6	83.6	76.0
Llama-4-Scout (FCSD)	109B	80.8	87.0	62.4	82.0	82.4	75.8	63.4	75.9	44.4	89.8	81.7	75.1
Llama-4-Scout (SFT)	109B	80.7	80.3	52.4	80.0	72.0	76.2	14.0	69.7	43.8	78.6	73.5	65.6

Table 2: Downstream performance of different training settings of FarSkip-Collective distillation. We evaluate different training settings and conversion settings training for 300M tokens.

Model	ARC-C	HEval+	MBPP+	MMLU	Avg-11
Original	53.8	40.2	60.8	56.8	64.5
KL (Far 50%)	51.0	42.7	60.6	57.3	63.9
KL (Far 75%)	46.6	22.0	47.4	48.3	56.3
KL (Far 90%)	46.0	20.1	38.4	43.0	52.5
KL (Far 100%)	42.0	16.5	29.9	38.7	48.6
KL	42.0	16.5	29.9	38.7	48.6
KL + inter. L2	41.6	14.0	26.2	35.8	46.0
SFT	37.4	8.5	28.3	34.0	44.7
KL ❄ embed.	42.3	17.7	30.7	38.4	48.6
KL 4×bs (1.2B tokens)	42.6	18.9	33.1	39.9	50.3

Table 3: Pretraining model performance of Regular and FarSkip-Collective MoE using DS-V2-Lite architecture (16B) trained on 200B tokens.

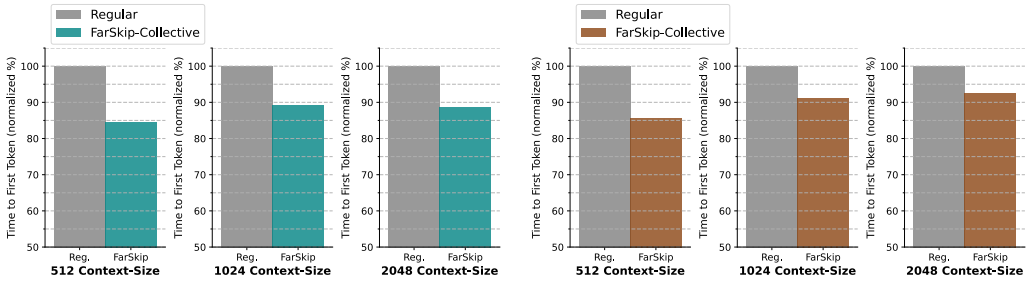
Benchmark	DS-V2-Lite-arch Reg.	DS-V2-Lite-arch Far.
PIQA	78.2	79.2
ARC-E	70.3	70.4
ARC-C	43.9	44.5
HS	69.2	69.3
WG	62.4	62.6
MMLU	43.3	41.7
OpenBook	41.0	40.0
GSM-8K	30.9	31.0
HEval+	26.8	23.8
MBPP+	48.7	49.2
Avg	51.5	51.2

KL ❄ embed. freezing the embedding and LM-head layers to reduce training instabilities 4) varying batch-sizes but maintaining the same number of training steps. Overall we observe that using the KL objective is the most effective and that freezing the embedding layers does not lead to a significant effect in the model's performance. In addition we study the effect of applying FarSkip-Collective to only a subset of the layers, with the layers applied to from the end, i.e., 75% corresponds to the last 75% layers of the model converted into FarSkip-Collective layers (cf. Fig. 3). In this settings we still optimize all of the model's parameters and observe that converting fewer layers makes the conversion task significantly easier especially for generation datasets.

We continue to study the effect of the number of modified layers in Fig. 3 where we use the original checkpoint of Qwen-3-30B MoE and evaluate it under different number of modified layers without training. We observe modifying the initial layers is more detrimental for performance, which we suspect is the result of two factors. 1) corrupting the early layers will cascade down as corrupted input to later layers and 2) for layer at depth k, f_k will have full access to $\frac{k-1}{k}$ of the previous layers via the residual connection, making it less likely to lose critical dependencies for larger k.

5.2 Explicit Overlapping

We measure single-node performance of our overlapped training implementation in Megatron-LM in Tab. 4, specifically focusing on the all-to-all collectives appearing in the MoE layers. We benchmark training on a node with 8xMI325X and consider two models, DeepSeek-V2 Lite (DS-V2 Lite) (16B) training with a micro-batch size of 4 and global batch-size of 128, and a short DeepSeek-V3 (DS-V3) model with 6 MoE layers (71B) with a micro batch-size of 1. Both models are trained with EP8 and sequence length of 4096. We use the short DS-V3 (L=6) model as it has the same layer dimensions and allows us to study the computation-communication trade-off of the full model while isolating orthogonal factors such as Pipeline-Parallelism (PP). We observe using FarSkip-Collective leads to high degree overlap in both the forward (87.6%, 92.9%) and backward pass (89.0%, 84.1%) leading to 4% and 13% end-to-end speedups in single-node settings for DS-V2 Lite



(a) Llama 4 Scout (b) DeepSeek V2

Figure 5: Time To First Token (prefill stage) with vLLM inference engine under varying prompt length. Each model is served with EP=8 for the MLP sub-block and TP=8 for attention serving 16 concurrent requests.

Table 4: Computation-communication overlap of all-to-all collectives in overlapped FarSkip-Collective MegatronLM training with EP=8. We evaluate the training of DeepSeek-V2 Lite and a shortened DeepSeek-V3 model with 6 layers.

METHOD	ALL-TO-ALL % OVERLAP			END-TO-END SPEEDUP
	FWD	BWD	TOTAL	
DS-V2 LITE REGULAR	0.0	0.0	0.0	1.0X
DS-V2 LITE FARSKIP	87.6	89.0	88.4	1.12X
DS-V3 (L=6) REGULAR	0.0	0.0	0.0	1.0X
DS-V3 (L=6) FARSKIP	92.9	84.1	88.9	1.04X

and DS-V3 respectively. This benchmark does not incorporate optimizations such as fused MLA attention that will enable additional acceleration and will make the exposed communication in the model even more critical.

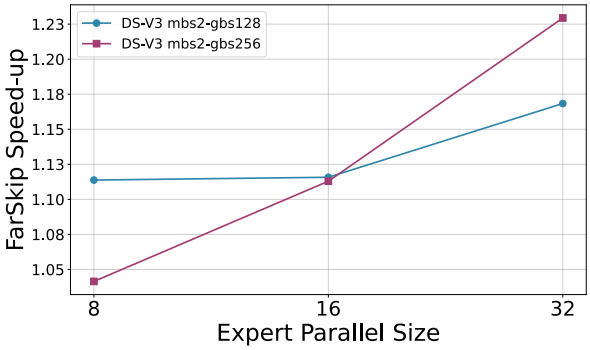


Figure 6: DeepSeek-V3 ($L = 6$) FarSkip-Collective training speedup under increasing Expert-Parallelism sizes and different batch size configurations.

We extend the training benchmarking of FarSkip-Collective to multi-node training scenarios on a 4 node system with each node equipped with 8xMI325X GPUs and inter-node communication bandwidth of 400Gbps between GPUs. We study the end-to-end speedup of the DS-V3 $L = 6$ model with FarSkip as compared with the regular model training when increasing the number of nodes from 1 to 4 in proportion with the EP size. As we increase the number of nodes and EP we keep the micro (mbs) and global (gbs) batch-sizes fixed (strong-scaling). In Fig. 6 we observe that FarSkip-Collective improvement scales up as we increase the EP size, with EP=32 leading to 1.22x end-to-end training speedup.

For inference with vLLM, we benchmark the prefill phase which has a considerable communication component where we consider the DeepSeek-V2 (235B) and Llama 4 Scout (109B) models. We test both models using a single node with 8xMI300X. In the benchmarking we adopt standard practices and use FP8 quantization and fused-MoE forward kernel (for routed-experts). With this setup we evaluate the Time-To-First Token (TTFT) with different input context lengths (L=512, 1024, 2048), per-device batch size of (BS=2) and (EP=8). For the attention layer the vLLM implementation will mirror the EP size with TP=8. We observe speedups of 8.2% - 16.8% and 12.2% - 18.5% in both DeepSeek-V2 and Llama-4 in using FarSkip-collective. The smaller number of experts in LLama-4 as compared with DeepSeek-V2 leads to faster computation and makes exposed communication more critical. In addition, we achieve communication overlap of the all-reduce of 95.3% and 97.6% for Llama-4 and DeepSeek-V2 (compared with 0% overlap in regular execution). In the appendix we share layer execution traces for both training and inference that demonstrate the computation-communication overlap enabled by our implementation.

For SGLang inference, we evaluate the optimized DeepSeek-V3 (671B) model architecture equipped with FarSkip-Collective for large-scale MoE inference for both prefill and decoding. In the prefill phase, e.g. benchmarking Time-to-First-Token (TTFT) FarSkip enables up to 1.34x speedup with TP=8, EP=8. The prefill stage is compute-bound and Fig. 7 (left) demonstrates linear scaling of the duration of TTFT with the numbers of tokens processed.

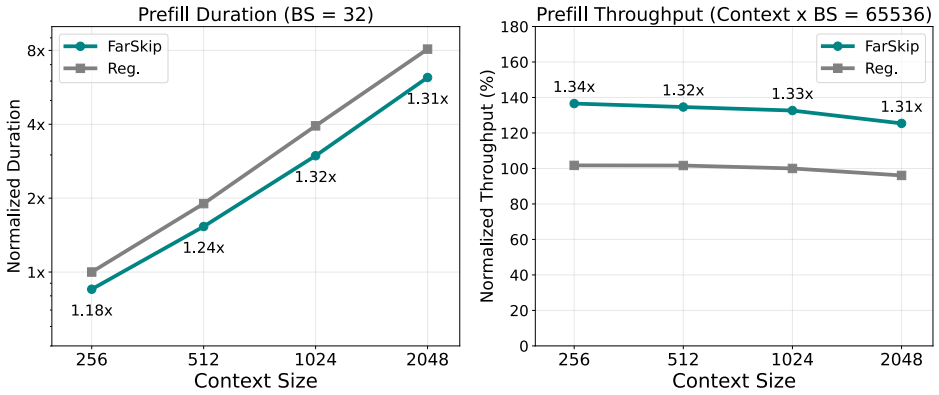


Figure 7: DeepSeek-V3 Time To First Token (prefill stage) with SGLang under varying batch-size and prompt length. Each model is served with EP=8 for the MLP sub-block and TP=8 for attention.

In both Fig. 7 (left) and (right) we also observe a fairly consistent speedup provided by FarSkip. The duration and the consistent speedup behavior can be attributed to the fact that both the compute-bound computation portion and the bandwidth-bound blocking communication portion scale directly with the number of tokens processed.

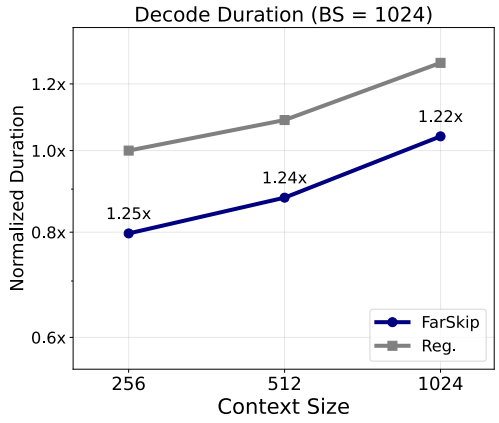


Figure 8: Time Between Tokens (decode stage) with SGLang DeepSeek-V3 for 2-node inference under varying prompt length. Each model is served with EP=16 for the MLP sub-block and TP=16 for attention.

We also create an analytic performance model to experiment with different configurations of models modified with FarSkip-Collective which we present in Fig. 9. Using the theoretical model helps isolating factors such as the presence of optimized kernels and fused operations. In particular we estimate the speedup provided by FarSkip-Collective under different MoE sparsity levels based on the DeepSeek-V3 architecture (32x sparsity point) and observe that the projected speedup of FarSkip-Collective continues even in sparser settings than DeepSeek-V3 such as 64x and 128x sparsity-level models. We provide additional results and details about the analytic performance model in Appendix B.

Unlike the prefill phase, LLM decoding is memory-bandwidth-bound; especially in large MoEs such as DeepSeek-V3. In single-node settings, the large parameter count that needs to be loaded per-GPU, translates to slower decoding and also leads to reduced maximum batch-size due to the limited memory capacity left for the KV cache. On the communication side, the all-reduce calls are applied to just the newly predicted tokens which will have smaller message-sizes as compared to prefill, especially with the smaller batches dictated by single-node serving. Together, this makes computation time (that is dictated by memory bandwidth) dominate the single-node large MoE workload as compared with communication.

Nonetheless, in a multi-node set-up FarSkip leads to a significant benefit. Distributing the MoE experts over a larger number of GPUs both directly decreases the computation time and increases the communication time. With wide-EP, the number of experts and parameters per GPU directly decrease and allow for significantly larger batch-sizes. This

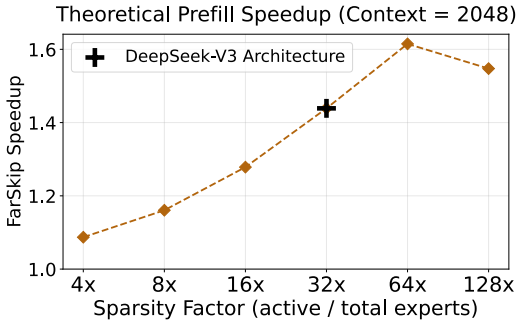


Figure 9: Performance model of FarSkip-Collective prefill speedup under different sparsity levels applied to a Deepseek-V3 architecture.

increases the throughput significantly making large MoE decoding suitable for large-scale distributed setups in general. At the same time, by switching to multi-node serving one relies on scale-out interconnects and the bigger batch-sizes also lead to larger message sizes. Together these changes shift the computation-communication balance making FarSkip-Collective significant in this setting. To this end, we evaluate DeepSeek-V3 decoding with TP=16, EP=16 in the large-batch setting (BS=1024) on a 2-node 8xMI325X system connected with 8 400Gbps NICs for inter-node communication. In Fig. 8 we observe consistent speedup of up to 1.25x with FarSkip under varying prompt lengths as multi-node settings allow for larger batch-sizes.

6 Related Work

Computation-communication overlap in distributed deep learning traditionally focuses on "bit-exact" approaches that retain the mathematical formulation of the model and instead focus on improved execution of the algorithm on hardware. Most existing parallelism techniques aim to achieve minimal exposed communication (Zhao et al., 2023; Rasley et al., 2020). A common theme to achieve overlap is decomposing operators into smaller pieces and scheduling computation and communications in tandem, this includes operator decomposition such as AsyncTP (Wang et al., 2023; PyTorch Team, 2024), and multi-layer pipelines (Zhu et al., 2025; Huang et al., 2019; Li et al., 2023).

Closer to our approach are model architectural changes aimed at reducing exposed communication at runtime. In the partial formulation front the works of (Prabhakar et al., 2024) and (Lamprecht et al., 2025) use un-communicated TP activation shards as input to a sub-block. Nonetheless neither technique can be extended to the MoE settings as the un-communicated shards with EP lead to the full activation in case the selected expert is on the device and a 0 activation otherwise which destabilizing training and inference. Further both works study the modeling aspect on order of magnitude smaller scales with a restricted set of likelihood tasks compared to our settings. On the implementation side FarSkip-Collective training is the first to enable communication-overlapping in the backward pass and on the inference side coalesces communication to reduce bandwidth by 33%. In the outdated formulation the recent work of (Zhang et al., 2025) uses outdated activations for dense transformers with tensor-parallelism. In contrast our work is the first to develop a communication-overlapped architecture for MoEs tackling expert parallelism which entails Dispatch and Combine collectives within the MoE block as opposed to outside the blocks. Our work also demonstrates successful full model conversion (vs. 50%) and proves out FarSkip-Collective at order-of-magnitude larger scale. Other communication-overlapped architecture works include (Naumov et al., 2019) that designs the model architecture for computation-communication pipelining of the network and computation heavy layers. The works of (Black et al., 2022; Wang & Komatsuzaki, 2021) reduce required communication in transformer blocks by designing parallel MLP and attention sub-blocks however neither allows for overlap of the remaining communication. The work of (Gunter et al., 2024) further reduces required model communication for large-scale models via "track-parallelism" but also maintains non-overlapped synchronization points.

7 Conclusion

In this work we present a modified connectivity architecture for MoEs that removes model dependencies between network blocks and therefore has the potential to unhobble blocking communication in their training and execution. We demonstrate the modified architecture has the capabilities to perform well at scale and by developing FCSD we are able to convert a series of state-of-the-art MoE models with increasing scale of up to 109B model parameters efficiently with minor performance degradation. After demonstrating the modified architecture is viable as a replacement of the regular MoE, we move to realize the benefits of FarSkip-Collective by developing optimized training and inference implementations that enable 88.9% overlapping of all-to-all communication during MoE training and on the inference side enable up to 1.34x speedup in time-to-first-token of large MoEs. Moving forward, by making model execution amenable to overlapping we hope one can rethink existing model architectural configurations and parallelism techniques and explore a larger architecture search space.

References

AMD TAS. Primus: A lightweight, unified training framework for large models on amd gpus, 2025. URL https://github.com/AMD-AGI/Primus.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., Pieler, M., Prashanth, U. S., Purohit, S., Reynolds, L., Tow, J., Wang, B., and Weinbach, S. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode #5 – Workshop on Challenges*

& Perspectives in Creating Large Language Models, pp. 95–136, virtual+Dublin, 2022. Association for Computational Linguistics.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021.

Dai, D., Deng, C., Zhao, C., Xu, R., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

DeepSeek-AI, Shao, Z., Dai, D., et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Gunter, T., Wang, Z., Wang, C., Pang, R., Narayanan, A., Zhang, A., Zhang, B., Chen, C., Chiu, C.-C., Qiu, D., Gopinath, D., Yap, D. A., Yin, D., Nan, F., Weers, F., Yin, G., Huang, H., Wang, J., Lu, J., Peebles, J., Ye, K., Lee, M., Du, N., Chen, Q., Keunebroek, Q., Wiseman, S., Evans, S., Lei, T., Rathod, V., Kong, X., Du, X., Li, Y., Wang, Y., Gao, Y., Ahmed, Z., Xu, Z., Lu, Z., et al. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M. X., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, volume 32, pp. 103–112, 2019.

Kimi, T., Bai, Y., Bao, Y., Chen, G., Chen, J., Chen, N., Chen, R., Chen, Y., Chen, Y., Chen, Y., et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Lamprecht, I., Karnieli, A., Hanani, Y., Giladi, N., and Soudry, D. Tensor-parallelism with partially synchronized activations. *arXiv preprint arXiv:2506.19645*, 2025.

Li, F., Zhao, S., Qing, Y., Chen, X., Guan, X., Wang, S., Zhang, G., and Cui, H. Fold3d: Rethinking and parallelizing computational and communicational tasks in the training of large dnn models. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1432–1449, 2023.

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=1qvx610Cu7.

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b.

Meta AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. Meta AI Blog, April 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., and Zaharia, M. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.

Naumov, M., Mudigere, D., Shi, H. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C., Azzolini, A. G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X., Chen, W., Rao, V., Jia, B., Xiong, L., and Smelyanskiy, M. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019. URL https://arxiv.org/abs/1906.00091.

Prabhakar, R. B., Zhang, H., and Wentzlaff, D. Kraken: Inherently parallel transformers for efficient multi-device inference. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

PyTorch Team. Distributed w/ torchtitan: Introducing async tensor parallelism in pytorch. PyTorch Discussion Forum, 2024. URL https://discuss.pytorch.org/t/distributed-w-torchtitan-introducing\-async-tensor-parallelism-in-pytorch/209487.

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

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. doi: 10.1145/3394486.3406703.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Talmor, A., Herzig, J., Lourie, N., and Berant, J. 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.

Wang, B. and Komatsuzaki, A. Gpt-j-6b: A 6 billion parameter autoregressive language model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Wang, S., Wei, J., Sabne, A., et al. Overlap communication with dependent computation via decomposition in large deep learning models. In *ASPLOS*, pp. 93–106, 2023. doi: 10.1145/3567955.3567959.

Yang, M., Rezagholizadeh, M., Li, G., Appia, V., and Barsoum, E. Zebra-llama: Towards extremely efficient hybrid models. In *Advances in Neural Information Processing Systems*, volume 39, 2025. NeurIPS 2025.

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

Zhang, M., Mishra, M., Zhou, Z., Brandon, W., Wang, J., Kim, Y., Ragan-Kelley, J., Song, S. L., Athiwaratkun, B., and Dao, T. Ladder-residual: parallelism-aware architecture for accelerating large model inference with communication overlapping. *arXiv preprint arXiv:2501.06589*, 2025.

Zhao, C., Deng, C., Ruan, C., Dai, D., Gao, H., Li, J., Zhang, L., Huang, P., Zhou, S., Ma, S., Liang, W., He, Y., Wang, Y., Liu, Y., and Wei, Y. X. Insights into deepseek-v3: Scaling challenges and reflections on hardware for ai architectures. *arXiv preprint arXiv:2505.09343*, 2025. URL https://arxiv.org/abs/2505.09343.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.-C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., Desmaison, A., Balioglu, C., Nguyen, P., Chauhan, B., Hao, Y., Mathews, A., and Li, S. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12):3848–3860, 2023. doi: 10.14778/3611540.3611569.

Zhu, K., Zhao, Y., Zhao, L., Zuo, G., Gu, Y., Xie, D., Gao, Y., Xu, Q., Tang, T., Ye, Z., Kamahori, K., Lin, C.-Y., Wang, S., Krishnamurthy, A., and Kasikci, B. Nanoflow: Towards optimal large language model serving throughput. In *Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '25, Santa Clara, CA, USA, 2025. USENIX Association.