GATED ALL-REDUCE: FASTER INFERENCE WITH POST-TRAINING COMMUNICATION PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

While tensor-parallelism reduces overall latency during large language models' inference, it introduces two all-reduce operations in every transformer block, which account for a large share of inference latency. We propose Gated All-Reduce, a post-training method that removes a predefined fraction of these all-reduce operations. When skipping an all-reduce operation, each device accumulates its local contribution and merges it at the next retained all-reduce, so every layer's output eventually reaches the shared residual stream. Our formulation is differentiable, except for the top-k gate selection, whose gradients we estimate with a rectangular straight-through estimator. This lets us learn which all-reduce operations to skip via backpropagation. After a brief joint fine-tuning of LoRA adapters and gates, our method nearly recovers the base model's performance. Skipping 50% of all-reduce operations, our method preserves 98.7% of the average downstream accuracy of Llama-3.1-70B (dense) and 98.2% for Qwen-2-57B (MoE).

1 INTRODUCTION

Large language models (LLMs) have advanced rapidly in recent years, driven in large part by improvements in compute and communication infrastructure. Their substantial computational and memory requirements often necessitate partitioning them across multiple devices to reduce latency, increase throughput, and lower per-device memory consumption. Among the available parallelization techniques, tensor-parallelism (TP) (Shoeybi et al., 2020) plays a central role in both LLM training and inference. TP partitions individual tensor operations, such as matrix multiplications, across multiple devices, thereby distributing their computational and memory requirements. Other prominent techniques, such as data parallelism (DP) (Rajbhandari et al., 2020) and pipeline parallelism (PP) (Harlap et al., 2018), can increase throughput but do not generally reduce single-request latency. Under these approaches, each matrix multiplication executes on a single device, so an individual operation is not parallelized as it is under TP. However, the latency benefits of TP come at the cost of additional communication, as the partial results computed by the participating devices must be aggregated with all-reduce operation.

In dense transformer models, tensor-parallel communication typically occurs twice per transformer layer, once in the attention block and once in the feed-forward block. In mixture-of-experts (MoE) (Fedus et al., 2022; Blakeman et al., 2026; Du et al., 2022) models that employ expert-parallelism, the MoE block instead uses expert-parallel communication (all-to-all), and tensor-parallel communication (all-reduce) typically occurs only once per transformer layer. During inference, TP communication can account for a substantial fraction of forward-pass latency. Moreover, tensor-parallelism is commonly limited to a degree of eight because its communication-intensive nature generally confines it to devices within a single node connected by a high-bandwidth interconnect such as NVLink. Although the balance between computation and communication depends on the hardware, workload, and interconnect, communication overhead can become a major performance bottleneck (Pati et al., 2023). Consequently, considerable algorithmic and engineering effort has been devoted to reducing this overhead.

We propose Gated All-Reduce, a novel post-training method in the line of work that redesigns the model architecture to be communication-aware (Zhang et al., 2025; Lamprecht et al., 2026; Kim et al., 2025). Unlike previous work, our method learns which all-reduce operations to remove, ad-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

dressing both bandwidth-bound and latency-bound scenarios. In addition, our method is not limited to skipping only the post-attention all-reduce, and is applicable to MoE models served with combined tensor-parallelism and expert-parallelism. We achieve this with a differentiable formulation, except for the top-k gate selection, whose gradients we estimate with a rectangular straight-through estimator. This approach lets us learn which all-reduce operations to skip via backpropagation, and jointly fine-tune LoRA adapters and all-reduce gates.

We demonstrate our method with two recent mid-sized transformer models — Llama-3.1-70B (dense) (Grattafiori et al., 2024) and Qwen-2-57B (MoE) (Yang et al., 2024). We nearly recover the base model's accuracy across several high skip ratios, with only $\sim$5.2M tokens consumed throughout the training pipeline. We show inference speedup with our method in various skip ratios and settings.

In summary, our main contributions are:

- We propose Gated All-Reduce, a novel architecture that skips all-reduce operations and accumulates their local contributions to the residual stream at the next retained all-reduce.
- We show that our method is trainable with a rectangular straight-through estimator. Unlike prior communication-aware methods that fix which collectives to drop using hand-designed sensitivity heuristics, our gate vector is discovered by backpropagation and, crucially, co-trained with the recovery adapters so that gate vector and weights adapt to each other.
- We show that our method accelerates inference across various settings, while nearly preserving the base model's accuracy.

2 Related Work

The challenge of communication bottleneck while applying tensor-parallelism has attracted much attention, both in inference and training settings. Some works focused on system optimization, others have tried to compress the communicated tensor, and a few works have attempted to redesign the model architecture to be communication-aware.

System optimization. Several approaches mitigate communication overhead by overlapping communication with computation through automated tensor slicing (Wang et al., 2024; Li et al., 2025). CoCoNet (Jangda et al., 2022) expresses computation and communication in a single language, letting a compiler fuse, reorder and overlap collectives with the surrounding GEMMs. FLUX (Chang et al., 2024) decomposes both into fine-grained tiles fused into a single GPU kernel. However, these system-level methods leave the transferred volume unchanged and can only hide communication as long as there is enough computation to overlap it behind, a condition that breaks down at high tensor-parallel degrees or low batch sizes. They also demand substantial ongoing engineering effort to support constantly evolving hardware and software architectures.

Compressed communication. Another line of research addresses the communication bottleneck by compressing the exchanged tensors (Li et al., 2024; Hansen-Palmus et al., 2026). However, compression introduces additional computation overhead, and only reduces bandwidth demands, failing to mitigate the communication latency that dominates in low-latency settings. These techniques are orthogonal to our work and can be applied on top of it.

Communication-aware architecture. A third line of work modifies the model itself so that it requires less synchronization to begin with. CAAT-Net (Lamprecht et al., 2026) shows that a model with partially synchronized activations can be trained and converge, reducing communication bandwidth demands; however, it requires pre-training the model from scratch, and since it only reduces bandwidth demands it does little for latency-dominated regimes. Ladder-Residual (Zhang et al., 2025) reroutes the residual stream so that the all-reduce of one block is overlapped with the computation of the next. However, on NVIDIA GPUs the Streaming Multiprocessors (SM), which perform the computation, are also needed for communication. When overlapping communication and computation, some SM cycles are spent on communication-related tasks instead of computation. Furthermore, methods that reduce total communication can potentially lower power consumption.

SPD (Kim et al., 2025) removes sync-points by skipping the all-reduce on selected attention outputs, with a modified block design and sensitivity-aware treatment of the layers; yet it only targets the post-attention all-reduce, degrades at high skip ratios, and does not extend to MoE models served with combined tensor-parallelism and expert-parallelism.

Straight-through estimators. Bengio et al. (2013) proposed propagating gradients through discrete operations by substituting a surrogate in the backward pass, and Hubara et al. (2016) introduced the rectangular-window STE for training binarized networks, the surrogate we adopt here. STEs have also been used to train models with top-k selection. Sparsely gated MoE models train through top-k expert selection by propagating gradients only through the selected experts (Shazeer et al., 2017; Fedus et al., 2022). More recently, Zhang et al. (2026) learned hard top-k token selection in diffusion transformers, using a surrogate based on soft ranks rather than a rectangular window around the k-th largest score. Rajamanoharan et al. (2024) train discontinuous feature thresholds in sparse autoencoders. Like us, they use a rectangular-window surrogate, but their operation does not enforce the selection of exactly k elements. To our knowledge, no prior work has either learned communication-dropping decisions with straight-through estimators or used Rect-STE as the surrogate for hard top-k selection.

3 METHOD

Our method applies to any transformer model served with tensor-parallelism. We first define the notation for a standard transformer model with tensor-parallelism (Shoeybi et al., 2020), the notation for MoE models is defined in Appendix A. Rather than indexing transformer blocks, we index sync sites — the points at which tensor-parallelism would ordinarily place an all-reduce operation: namely after the attention output projection and after the feed-forward output projection of each block. A dense model with L blocks therefore has $N = 2L$ sites. Let D be the number of tensor-parallel devices, R_i the residual stream entering site i, and $F_{m,i}(\cdot)$ the composition of all layers on the m device that end at site i: normalization, attention or feed-forward, and the output projection. The all-reduce followed by skip connection yields

$$R_{i+1} = R_i + \sum_{n}^{D} F_{n,i}(R_i)\,. \tag{1}$$

Consequently, in standard tensor-parallelism, the residual stream is identical on all devices.

3.1 GATED ALL-REDUCE

Gating mechanism. To reduce communication overhead, we skip a predefined ratio ρ of all-reduce operations. We never skip the final all-reduce operation because it merges all local contributions into the shared residual stream. For the remaining $N-1$ sites, we define a gate vector $g \in \{0,1\}^{N-1}$, where $g_i = 0$ indicates that site i is skipped. To learn which sites are most important to retain, we introduce a learnable selection vector $M \in \mathbb{R}^{N-1}$ and normalize it with a Softmax to obtain the selection scores

$$S = \mathrm{Softmax}\,(M)\ . \tag{2}$$

We retain the $k = \lceil (1-\rho)\cdot N \rceil - 1$ sites with the highest selection scores, defining

$$g_i = \begin{cases} 1, & S_i \in \mathrm{TopK}(S,k), \\ 0, & \text{otherwise}, \end{cases} \tag{3}$$

where $\mathrm{TopK}(S,k)$ denotes the set of the k largest values in S. The Softmax operation is order-preserving and therefore does not change which sites are selected. We describe how the selection vector is learned in Section 3.2.

Residual correction. When an all-reduce is skipped, the local contribution is aggregated and added to the residual stream with the inherent skip connections and the next retained all-reduce, as shown in Figure 1. To write it we define

- $C_i \in \mathbb{R}^{t\times e}$ — the consistent stream, identical on all devices.

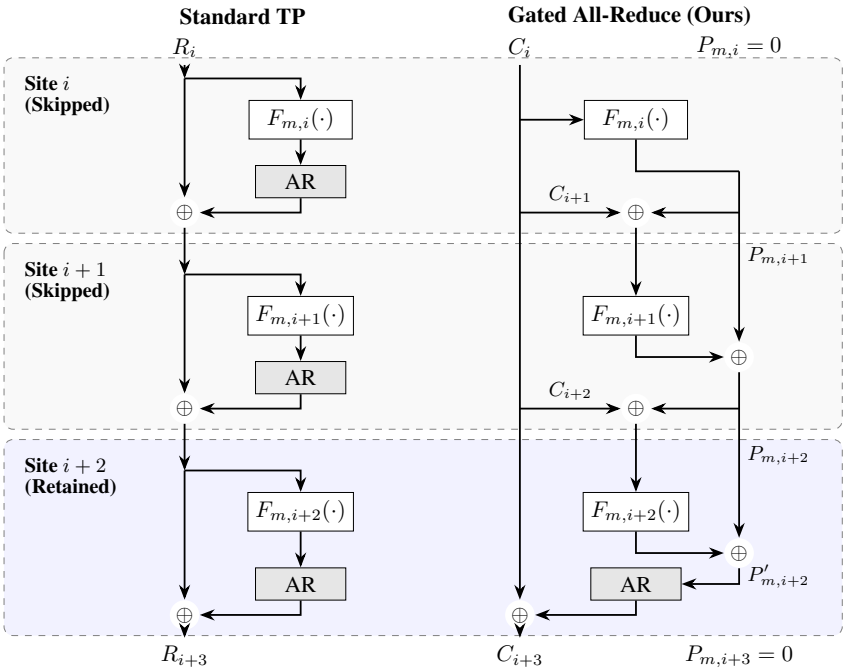


Figure 1: Unrolled execution comparison. In standard tensor-parallelism (left), every site enforces an all-reduce operation. In Gated All-Reduce (right), skipped sites accumulate their local output into a pending stream P without communication overhead. Upon reaching a retained site, the accumulated pending stream is synchronized via All-Reduce (AR block) and resolved back into the main consistent stream, effectively resetting the pending stream to zero.

- $P_{m,i} \in \mathbb{R}^{t \times e}$ — the pending stream, per-device local stream awaiting the next all-reduce.

where t is number of input tokens and e is embedding dimension. Initialized as $C_1 = X$, where $X \in \mathbb{R}^{t \times e}$ is the model's input, and $P_{m,1} = 0 \quad \forall m$, the forward pass is

$$P'_{m,i} = P_{m,i} + F_{m,i}\left(C_i + P_{m,i}\right) \tag{4}$$

$$C_{i+1} = C_i + g_i \cdot \sum_{n}^{D} P'_{n,i} \tag{5}$$

$$P_{m,i+1} = (1 - g_i) \cdot P'_{m,i} \tag{6}$$

Three properties of this formulation are worth noting:

1. Right after a retained all-reduce, the contributions of all previously skipped sites are summed into the residual stream. The only approximation is therefore that the layers between two retained all-reduce operations read a residual in which the intervening skipped sites contribute their device-local part rather than the cross-device sum. In Section 4.1 we show that this perturbation is small and recoverable.
2. Once the gates are static, the only difference from the traditional tensor-parallel transformer is that for any skipped site, an all-reduce is replaced by a local element-wise addition, as shown in Figure 1. In Section 4.2 we show that, despite this added overhead, our method achieves most of the available speedup during inference.
3. Apart from the $\mathrm{TopK}(S, k)$ in Equation 3, the formulation is fully differentiable.

3.2 Learning the Gates

Indicator backward pass relaxation. As mentioned in Section 3.1, our formulation is differentiable except for the gate calculation in equation 3. Let $\kappa(S)$ denote the k-th largest entry of S. We

can equivalently rewrite the gate using an indicator function as

$$g_i = \mathbb{1}_{\{S_i - \kappa(S) \geq 0\}} \,. \tag{7}$$

This is exactly the same function as equation 3. Its true derivative is zero almost everywhere and undefined at the selection threshold $\kappa(S)$. Writing the gate in this form allows us to use Rect-STE, which replaces the true derivative during the backward pass with the surrogate

$$\frac{\partial g_i}{\partial S_i} = \frac{1}{h} \cdot \mathrm{Rect}\left(\frac{S_i - \kappa(S)}{h}\right), \quad \mathrm{Rect}(x) = \mathbb{1}_{\{|x| < 0.5\}} \,. \tag{8}$$

Rect-STE assigns nonzero gradients to selection scores within a window of width h around the selection threshold. It therefore introduces a tradeoff between estimation error and gradient sparsity: a small h yields a lower estimation error, but only a few gates close to the selection threshold $\kappa(S)$ receive gradients, while a large h has the opposite effect. Therefore, h must be tuned together with the other training hyperparameters.

Other surrogate derivatives can also be used. Two examples are Identity STE, which passes the upstream gradient unchanged to every selection score,

$$\frac{\partial g_i}{\partial S_i} = 1 \,, \tag{9}$$

and Top-K gradients, which propagate gradients only through the selected gates,

$$\frac{\partial g_i}{\partial S_i} = \begin{cases} 1, & g_i = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

We compare these alternatives with Rect-STE in the ablation experiments in Section 4.3.

Training objective. Following previous works (e.g., Bercovich et al. (2025)) we use the Kullback-Leibler divergence (KLD) between the output distribution of the gated and base models as the training objective, since it is a common proxy for preserved downstream performance. We use KLD loss for all phases throughout this work: gate vector initialization, gates training, and recovery training.

$$\mathcal{L}_{\mathrm{KLD}} = \sum_{i}^{V} p_i \log\left(\frac{p_i}{q_i}\right) \,, \tag{11}$$

where p_i and q_i represent the probability distributions derived from the logits of the base and gated models, respectively, and V is the vocabulary size.

Logit vector initialization. We initialize M, the logit vector, by measuring the KLD loss when removing one site at a time, and applying $\tanh$ to the results (to cap outliers).

$$M_i^{\mathrm{init}} = \tanh\left(\mathcal{L}_{\mathrm{KLD}}\left(g_i = 1, \ldots, g_{i+1} = 1, g_i = 0, g_{i+1} = 1, \ldots, g_N = 1\right)\right) \,. \tag{12}$$

3.3 Recovery Training

After training the gates with the original weights frozen, we add a recovery training phase. In the recovery phase, we train LoRA (Hu et al., 2021) adapters to adjust the model weights and recover the model's performance. Prior work on sparsification has shown that the optimal mask before recovery training is not necessarily the optimal one afterward (Zhou et al., 2021). Therefore, we let the logit vector M keep training in the recovery phase as well. In this stage only, we add a regularization term inspired by SR-STE (Zhou et al., 2021) that encourages fewer changes in the selected gates by adding a positive value to the gradients of closed gates and a negative value to the gradients of open gates. With γ_s being the learning-rate at time s, and λ the regularization magnitude term, the update rule for M becomes

$$M_{s+1} = M_s - \gamma_s \left(\nabla_M \mathcal{L}_{\mathrm{KLD}} + \lambda\left(1 - 2g\right)\right) \tag{13}$$

In Section 4.3 we report an ablation result for this method, these results highlight the value of jointly training LoRA adapters and all-reduce gates, enabled by our differentiable formulation.

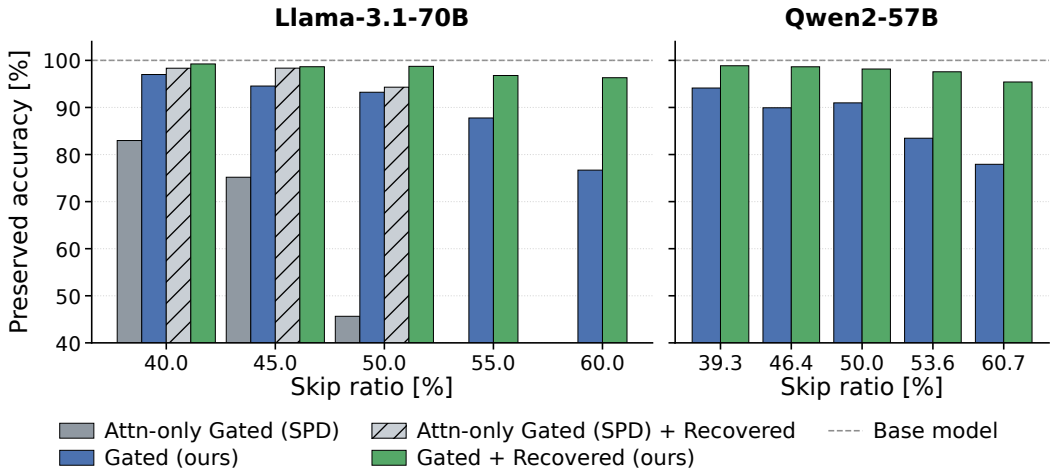


Figure 2: Preserved average accuracy on downstream evaluation tasks relative to the base model, before and after recovery, across different skip ratios. On the left side we report on Llama-3.1-70B and on the right side on Qwen-2-57B. We report our implementation of SPD only up to its maximum skip ratio (50%), and do not report it on Qwen-2-57B since SPD is not applicable with combined TP and EP.

4 Experiments

We use two mid-size transformer models to evaluate our method: Llama-3.1-70B (Grattafiori et al., 2024), a dense model with 80 transformer blocks, and Qwen-2-57B (Yang et al., 2024), a MoE model with 14B active parameters and 28 transformer blocks. We train on a node with 8 NVIDIA A100 GPUs, with 80GB of HBM each and with NVLink interconnect. For training, we use The Pile (Gao et al., 2020) dataset, an open-source collection of English text created for LLM pre-training. The gates training phase consumes $\sim$2.6M training tokens, and the joint recovery phase consumes another $\sim$2.6M. We train the Llama-3.1-70B model for tensor-parallelism on 8 GPUs, and the Qwen-2-57B model for combined tensor-parallelism and expert-parallelism on 8 GPUs.

4.1 Accuracy

To evaluate the ability of our method to preserve the base model's accuracy, we run our full pipeline at skip ratios ranging from 40% to 60% on both models. We evaluate the accuracy performance of our method at two different points: after gates training phase, and after the joint recovery phase. We measure the following metrics: KLD loss and cross-entropy (CE) loss on the evaluation set, WikiText (Merity et al., 2016) perplexity and average score on tasks from the Language Model Evaluation Harness (Gao et al., 2021) framework: ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2019), SciQ (Welbl et al., 2017), WinoGrande (Sakaguchi et al., 2019) and MMLU (Hendrycks et al., 2021). We compare our method before recovery with our implementation of SPD approach for zero-shot dropping. In addition, we compare our method after recovery to SPD with the same recovery process (except that its gates are frozen) . We compare only up to a 50% skip ratio, since this is the maximum ratio achievable with SPD, and do not compare against SPD on Qwen-2-57B, since SPD is not applicable with combined tensor-parallelism and expert-parallelism serving.

The results for preserved accuracy on downstream evaluation tasks are available in Figure 2 and the other evaluation metrics are available in Table 1. We find that before the recovery training SPD degrades substantially at high skip ratios: at 50% it preserves only 45.6% of Llama-3.1-70B base model's accuracy. In comparison, our method degrades mildly, and before the recovery training it preserves 93.2% of Llama-3.1-70B base model's accuracy. At 40% and 45% skip ratio the recovery training closes most of the gap for both our method and SPD, but at the 50% skip ratio our method outperforms with 98.7% preserved accuracy against only 94.3% with SPD. We also observe that with our method the joint recovery phase closes most of the accuracy gap in higher skip ratios, for

Table 1: **Accuracy evaluation** before and after the recovery phase. Val KLD and Val CE are measured on the evaluation set, Wiki PPL is WikiText perplexity, and LM-Eval is the average downstream accuracy, with the preserved accuracy relative to the base model (%) in parentheses. SPD is reported only up to its maximum skip ratio (50%) and only for Llama-3.1-70B, as it does not support combined TP and EP. Per-task results are in Appendix B.1.

		Before recovery				After recovery			
Skip	**Method**	**KLD**	**CE**	**PPL**	**LM-Eval**	**KLD**	**CE**	**PPL**	**LM-Eval**
Llama-3.1-70B (dense, TP=8)									
0%	Base	0.0000	1.8038	3.43	80.59 (100.0)	–	–	–	–
40%	SPD	0.4657	2.2582	11.26	66.87 (83.0)	0.0694	1.8557	4.24	79.25 (98.3)
	Ours	0.1658	1.9644	6.59	78.17 (97.0)	0.0733	1.8668	4.95	79.98 (99.2)
45%	SPD	0.7656	2.5482	15.88	60.59 (75.2)	0.1033	1.8801	4.38	79.26 (98.4)
	Ours	0.2441	2.0397	7.62	76.20 (94.5)	0.1188	1.9087	5.85	79.50 (98.7)
50%	SPD	3.8216	5.5814	1285.25	36.78 (45.6)	0.2587	2.0245	6.10	76.00 (94.3)
	Ours	0.3354	2.1329	9.10	75.14 (93.2)	0.1128	1.9002	5.69	79.57 (98.7)
55%	Ours	0.4830	2.2717	10.68	70.73 (87.8)	0.1899	1.9718	7.03	78.00 (96.8)
60%	Ours	0.8229	2.6101	17.51	61.80 (76.7)	0.2134	1.9920	7.36	77.62 (96.3)
Qwen-2-57B (MoE, TP=8, EP=8)									
0%	Base	0.0000	1.8462	6.37	74.50 (100.0)	–	–	–	–
39.3%	Ours	0.1438	1.9826	7.26	70.13 (94.1)	0.0693	1.8939	6.65	73.65 (98.9)
46.4%	Ours	0.3257	2.1530	8.61	67.00 (89.9)	0.0897	1.9152	6.79	73.49 (98.6)
50%	Ours	0.2936	2.1257	8.38	67.78 (91.0)	0.0808	1.9065	6.73	73.14 (98.2)
53.6%	Ours	0.4084	2.2368	9.36	62.18 (83.5)	0.1146	1.9393	6.95	72.70 (97.6)
60.7%	Ours	0.6403	2.4644	11.76	58.06 (77.9)	0.1283	1.9502	7.03	71.08 (95.4)

both Llama-3.1-70B and Qwen-2-57B. Some evaluation tasks are more affected than the average, the results for each task are available in the Appendix B.1.

4.2 SPEEDUP

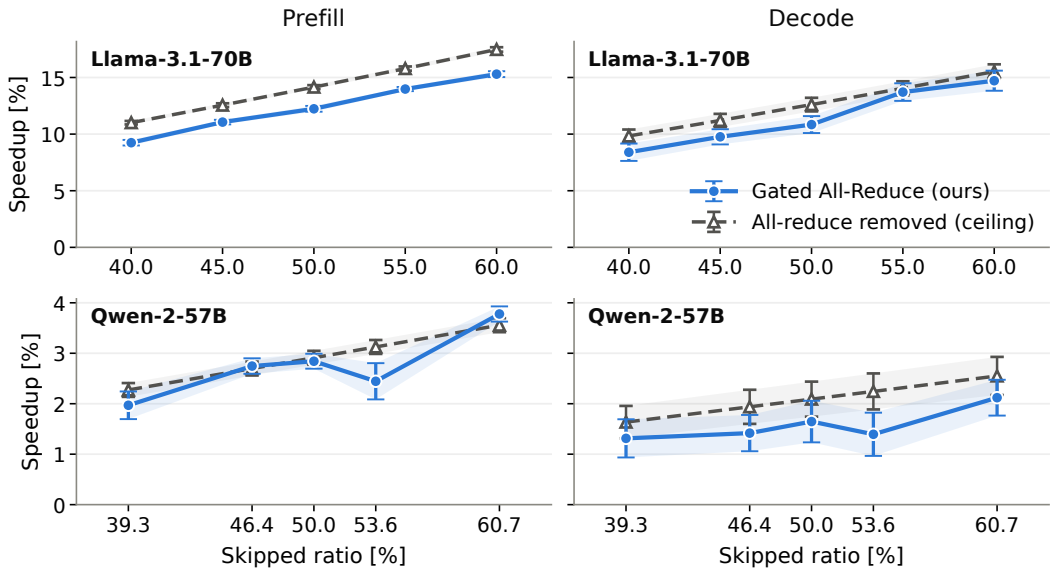
To measure end-to-end speedup, we implement Gated All-Reduce in vLLM (Kwon et al., 2023), a production-grade open-source inference engine, and run it on a node with 8 NVIDIA A100 GPUs, with 80GB of HBM each and NVLink interconnect. We measure the latency of both the prefill and decode phases. All measurements are taken after warm-up, and we report the mean and standard deviation over 15 repetitions. To quantify the overhead of our method, we also report a ceiling: the speedup from removing the same all-reduce operations outright, without adding our method overhead.

Figure 3a shows the speedup across skip ratios. At a 50% skip ratio, our method accelerates Llama-3.1-70B by $12.23 \pm 0.26\%$ in prefill and $10.84 \pm 0.75\%$ in decode, and Qwen-2-57B by $2.84 \pm 0.15\%$ and $1.65 \pm 0.41\%$ respectively. The gains are smaller for Qwen-2-57B because, with combined tensor-parallelism and expert-parallelism, only one all-reduce per block is eligible for skipping and the all-to-all operation of the MoE layers is unaffected. In addition, Qwen-2-57B is less communication-bound than Llama-3.1-70B. Averaged over skip ratios and phases, our method achieves 88.5% of the ceiling speedup for Llama-3.1-70B and 84.8% of the ceiling speedup for Qwen-2-57B.

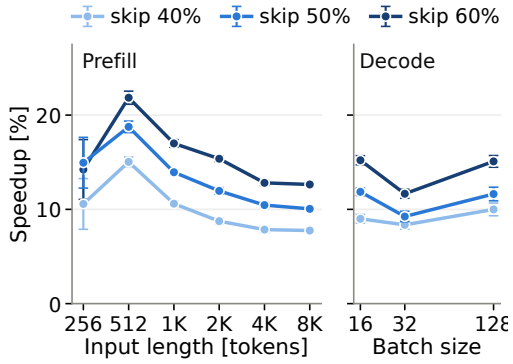
Figure 3b shows that the speedup holds across batch sizes for Llama-3.1-70B during decode, and slightly decreases for long prefills, where computation accounts for a larger share of latency. Figure 3c extends the measurement to two additional models, Llama-3.1-8B and Qwen3-235B, for which we report speedup only. Absolute latencies for all configurations are listed in the Appendix B.2.

4.3 ABLATIONS

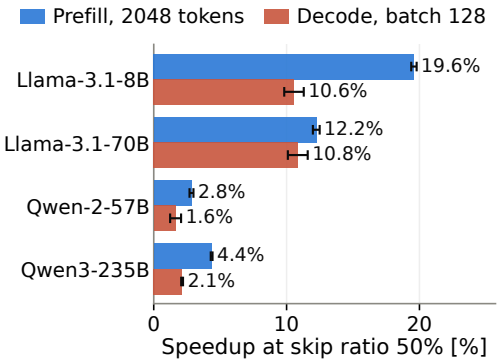
Gates learning ablation. To isolate the contribution of learning the gates, we compare three gate vectors at a 50% skip ratio, all evaluated before recovery training. *Random* vector retains k sites sampled uniformly from eligible sites. The *Init* vector retains the top-k sites according to the ini-



(a) Skip-ratio sweep. Prefill: 2048 tokens, batch 1; decode: batch 128.



(b) Llama-3.1-70B, varying sequence length and batch size.



(c) Models at 50% skip.

Figure 3: Speedups on 8 A100 GPUs with a vLLM-based implementation.

tialization M^{init}, as described in Section 3.2. The *Trained* vector is the output of our gate-training phase with Rect-STE. We report the KLD on the evaluation set for each method in Table 2. The *Init* vector already substantially outperforms random selection, confirming that site importance is highly non-uniform. Training the gates further reduces the KLD by 88.1% for Llama-3.1-70B and 70.2% for Qwen-2-57B relative to the Init vector. This indicates that single-site sensitivity is a useful initialization, but does not capture the interactions between skipped sites, which our differentiable formulation accounts for.

Rect-STE ablation. To evaluate our choice of Rect-STE for the surrogate derivative $\partial g_i / \partial S_i$, we compare it with the two alternatives defined in Section 3.2: Identity STE and Top-K gradients. For each method, we report the final KLD loss on the evaluation set after the gate-training phase, where lower values indicate better agreement with the base model. Table 3 shows that Rect-STE substantially outperforms Top-K gradients on both models. It also achieves lower mean KLD than Identity STE for both models, although the differences are not statistically significant. We therefore use Rect-STE in all remaining experiments.

Recovery training ablation. To evaluate the benefit of our recovery method described in Section 3.3, we report the KLD loss on evaluation set throughout the recovery phase with different λ values,

Table 2: **Gates learning ablation.** Evaluation-set KLD before recovery with different gate vectors, at a 50% skip ratio. Mean ± std over 5 seeds (3 seeds for the trained Llama-3.1-70B vector).

Gate vector	Llama-3.1-70B	Qwen-2-57B
Random	5.869 ± 0.585	7.531 ± 1.358
Init	2.473 ± 0.040	0.990 ± 0.222
Trained	0.296 ± 0.023	0.295 ± 0.035

Table 3: **Top-K backward relaxation ablation.** Evaluation-set KLD after gate training with different surrogate gradients, at a 50% skip ratio. Mean ± std over 5 seeds (Qwen-2-57B) and 3 seeds (Llama-3.1-70B).

Surrogate	Llama-3.1-70B	Qwen-2-57B
Top-K grads	0.836 ± 0.084	2.013 ± 0.955
Identity STE	0.306 ± 0.021	0.336 ± 0.054
Rect-STE	0.296 ± 0.023	0.295 ± 0.035

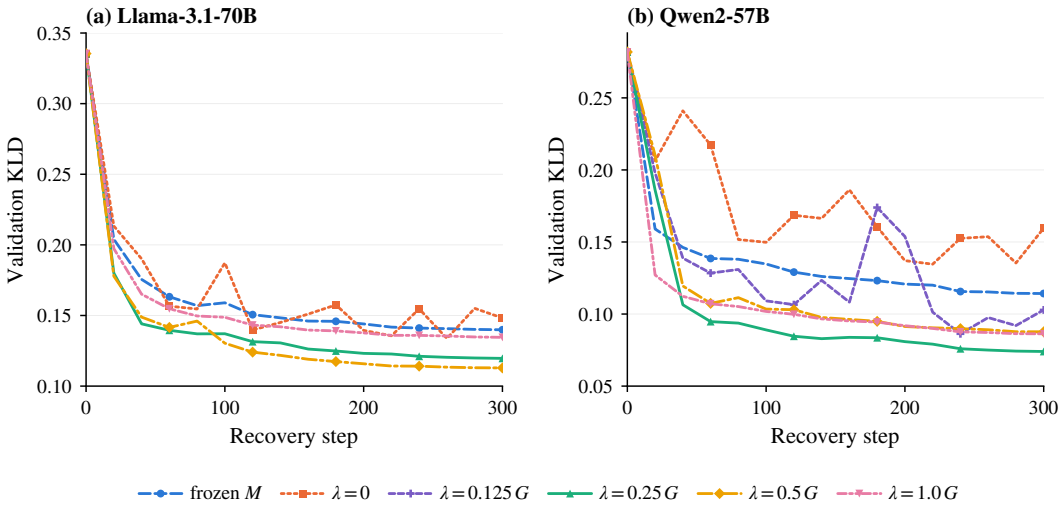


Figure 4: **Gates training during recovery ablation.** Comparison of recovery training with frozen gates versus trained gates with SR-STE regularization. On the left, Llama-3.1-70B, and on the right, Qwen-2-57B, both with 50% skip ratio. G is the median value taken over 5 calibration training steps, on the RMS of non-zero M gradients.

in addition to recovery with frozen M. Figure 4 supports our claim that continuing to train M during recovery lowers the final KLD, but only when the regularization term is applied; without it, the gates vector oscillates and performs worse than keeping M frozen.

5 Conclusion

In this paper, we propose Gated All-Reduce, which allows skipping all-reduce operations. We show that our method is trainable with a rectangular straight-through estimator and that jointly fine-tuning the model and gates achieves better accuracy. We experiment with two transformer models and show that our method achieves significant speedup on dense model and smaller speedup on MoE model with only a minor degradation in both models' accuracy compared to the base model.

AI use statement

In this work, we used generative AI tools to provide feedback on research methodology or experiments and to interpret results. We have not used generative AI tools to formulate mathematical claims, clean and reformat dataset, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for create or modify scientific figures or images, edit software code and identify relevant literature. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013.

Akhiad Bercovich et al. Puzzle: Distillation-based nas for inference-optimized llms, 2025. URL https://arxiv.org/abs/2411.19146.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Aaron Blakeman, Aaron Thomas, Aastha Jhunjhunwala, Abhibha Gupta, Abhinav Khattar, Adam Rajfer, Adi Renduchintala, Adil Asif, Aditya Vavre, Adriana Flores Miranda, et al. Nemotron 3 ultra: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2606.15007*, 2026.

Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Chengji Yao, Ziheng Jiang, Haibin Lin, Xin Jin, and Xin Liu. Flux: Fast software-based communication overlap on gpus through kernel fusion, 2024. URL https://arxiv.org/abs/2406.06858.

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, 2018. URL https://arxiv.org/abs/1803.05457.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pp. 5547–5569. PMLR, 2022.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021. URL https://doi.org/10.5281/zenodo.5371629.

Aaron Grattafiori et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Jan Hansen-Palmus, Michael Truong Le, Oliver Hausdörfer, and Alok Verma. Communication compression for tensor parallel llm inference, 2026. URL https://arxiv.org/abs/2411.09510.

Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training, 2018. URL https://arxiv.org/abs/1806.03377.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Sarikivi. Breaking the computation and communication abstraction barrier in distributed machine learning workloads, 2022. URL https://arxiv.org/abs/2105.05720.

Han-Byul Kim, Duc Hoang, Arnav Kundu, Mohammad Samragh, and Minsik Cho. Spd: Sync-point drop for efficient tensor parallelism of large language models, 2025. URL https://arxiv.org/abs/2502.20727.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Itay Lamprecht, Asaf Karnieli, Yair Hanani, Niv Giladi, and Daniel Soudry. Tensor-parallelism with partially synchronized activations, 2026.

Qingyuan Li, Bo Zhang, Liang Ye, Yifan Zhang, Wei Wu, Yerui Sun, Lin Ma, and Yuchen Xie. Flash communication: Reducing tensor parallelization bottleneck for fast large language model inference, 2024. URL https://arxiv.org/abs/2412.04964.

Shengwei Li, Zhiquan Lai, Dongsheng Li, Yanqi Hao, Weijie Liu, Keshi Ge, Xiaoge Deng, and Kai Lu. Oases: Efficient large-scale model training on commodity servers via overlapped and automated tensor model parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 36 (9):1828–1840, 2025. ISSN 2161-9883. doi: 10.1109/tpds.2025.3583165. URL http://dx.doi.org/10.1109/TPDS.2025.3583165.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016. URL https://arxiv.org/abs/1606.06031.

Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew D. Sinclair. Computation vs. communication scaling for future transformers on future hardware, 2023. URL https://arxiv.org/abs/2302.02825.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with JumpReLU sparse autoencoders. *arXiv preprint arXiv:2407.14435*, 2024.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2020. URL https://arxiv.org/abs/1910.02054.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

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.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

Guanhua Wang, Chengming Zhang, Zheyu Shen, Ang Li, and Olatunji Ruwase. Domino: Eliminating communication in llm training via generic tensor slicing and overlapping, 2024. URL https://arxiv.org/abs/2409.15241.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions, 2017. URL https://arxiv.org/abs/1707.06209.

An Yang et al. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Jiaji Zhang, Hailiang Zhao, Guoxuan Zhu, Ruichao Sun, Jiaju Wu, Xinkui Zhao, Hanlin Tang, Weiyi Lu, Kan Liu, Tao Lan, et al. Shiva-DiT: Residual-based differentiable top-k selection for efficient diffusion transformers. *arXiv preprint arXiv:2602.05605*, 2026.

Muru Zhang, Mayank Mishra, Zhongzhu Zhou, William Brandon, Jue Wang, Yoon Kim, Jonathan Ragan-Kelley, Shuaiwen Leon Song, Ben Athiwaratkun, and Tri Dao. Ladder-residual: parallelism-aware architecture for accelerating large model inference with communication overlapping, 2025. URL https://arxiv.org/abs/2501.06589.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch, 2021. URL https://arxiv.org/abs/2102.04010.

A NOTATION FOR MOE MODELS

For MoE models served with combined tensor-parallelism and expert-parallelism, the MoE output goes through an all-to-all operation, not an all-reduce operation. The only sync sites are therefore after the attention output projection of each block, so $N = L$. Here $F_{m,i}(\cdot)$ denotes the normalization, attention, and output projection of block i. $\mathrm{MoE}_i(x, r)$ denotes the MoE layers of block i with x being the experts' input and r being the router's input. With $C_1 = X$ and $P_{m,1} = 0 \quad \forall m$, the forward pass is

$$P'_{m,i} = P_{m,i} + F_{m,i}\left(C_i + P_{m,i}\right) \tag{14}$$

$$\hat{C}_i = C_i + g_i \cdot \sum_{n}^{D} P'_{n,i} \tag{15}$$

$$P_{m,i+1} = (1 - g_i) \cdot P'_{m,i} \tag{16}$$

$$C_{i+1} = \hat{C}_i + \mathrm{MoE}_i\left(\hat{C}_i + P_{m,i+1}\,;\, \hat{C}_i\right) \tag{17}$$

Feeding the router a consistent stream $\hat{C}_i$ guarantees identical routing decisions across devices, avoiding router divergence. The experts read the full local residual, so the pending contributions are still used.

B ADDITIONAL RESULTS

B.1 DOWNSTREAM EVALUATION FULL RESULTS

Table 4: Downstream Task Performance: SPD

Skipped	MMLU	ARC	HellaSwag	LAMBADA	PIQA	SciQ	Winogrande	AVG
– (Base)	76.42 (100.0%)	61.18 (100.0%)	85.75 (100.0%)	79.25 (100.0%)	84.28 (100.0%)	95.80 (100.0%)	81.45 (100.0%)	80.59 (100.0%)
40%	46.54 (60.9%)	46.50 (76.0%)	68.93 (80.4%)	65.28 (82.4%)	79.76 (94.6%)	89.10 (93.0%)	71.98 (88.4%)	66.87 (83.0%)
45%	34.15 (44.7%)	40.36 (66.0%)	64.87 (75.6%)	53.23 (67.2%)	76.77 (91.1%)	86.70 (90.5%)	68.03 (83.5%)	60.59 (75.2%)
50%	23.25 (30.4%)	27.30 (44.6%)	31.92 (37.2%)	3.96 (5.0%)	59.85 (71.0%)	59.90 (62.5%)	51.30 (63.0%)	36.78 (45.6%)

Table 5: Downstream Task Performance: SPD + Recovered

Skipped	MMLU	ARC	HellaSwag	LAMBADA	PIQA	SciQ	Winogrande	AVG
– (Base)	76.42 (100.0%)	61.18 (100.0%)	85.75 (100.0%)	79.25 (100.0%)	84.28 (100.0%)	95.80 (100.0%)	81.45 (100.0%)	80.59 (100.0%)
40%	72.29 (94.6%)	59.73 (97.6%)	85.24 (99.4%)	77.82 (98.2%)	84.11 (99.8%)	95.60 (99.8%)	79.95 (98.2%)	79.25 (98.3%)
45%	72.01 (94.2%)	60.75 (99.3%)	85.10 (99.2%)	77.99 (98.4%)	83.68 (99.3%)	95.50 (99.7%)	79.79 (98.0%)	79.26 (98.4%)
50%	66.38 (86.9%)	57.59 (94.1%)	82.38 (96.1%)	69.03 (87.1%)	83.13 (98.6%)	95.20 (99.4%)	78.30 (96.1%)	76.00 (94.3%)

Table 6: Llama-3.1-70B full downstream Task Performance: Gated

Skipped	MMLU	ARC	HellaSwag	LAMBADA	PIQA	SciQ	Winogrande	AVG
– (Base)	76.42 (100.0%)	61.18 (100.0%)	85.75 (100.0%)	79.25 (100.0%)	84.28 (100.0%)	95.80 (100.0%)	81.45 (100.0%)	80.59 (100.0%)
40%	75.90 (99.3%)	59.04 (96.5%)	84.13 (98.1%)	70.27 (88.7%)	83.30 (98.8%)	95.00 (99.2%)	79.56 (97.7%)	78.17 (97.0%)
45%	74.11 (97.0%)	57.17 (93.4%)	82.76 (96.5%)	63.89 (80.6%)	82.48 (97.9%)	95.00 (99.2%)	77.98 (95.7%)	76.20 (94.5%)
50%	69.76 (91.3%)	55.63 (90.9%)	81.09 (94.6%)	65.13 (82.2%)	82.10 (97.4%)	94.50 (98.6%)	77.74 (95.4%)	75.14 (93.2%)
55%	64.97 (85.0%)	46.25 (75.6%)	76.18 (88.8%)	61.19 (77.2%)	79.82 (94.7%)	91.10 (95.1%)	75.61 (92.8%)	70.73 (87.8%)
60%	37.37 (48.9%)	40.10 (65.6%)	63.83 (74.4%)	55.85 (70.5%)	76.55 (90.8%)	88.60 (92.5%)	70.32 (86.3%)	61.80 (76.7%)

Table 7: Llama-3.1-70B full downstream Task Performance: Gated + Recovered

Skipped	MMLU	ARC	HellaSwag	LAMBADA	PIQA	SciQ	Winogrande	AVG
– (Base)	76.42 (100.0%)	61.18 (100.0%)	85.75 (100.0%)	79.25 (100.0%)	84.28 (100.0%)	95.80 (100.0%)	81.45 (100.0%)	80.59 (100.0%)
40%	76.26 (99.8%)	60.84 (99.4%)	85.23 (99.4%)	78.19 (98.7%)	83.62 (99.2%)	95.60 (99.8%)	80.11 (98.4%)	79.98 (99.2%)
45%	75.65 (99.0%)	59.81 (97.8%)	84.49 (98.5%)	78.13 (98.6%)	83.13 (98.6%)	95.60 (99.8%)	79.72 (97.9%)	79.50 (98.7%)
50%	73.84 (96.6%)	61.35 (100.3%)	84.77 (98.9%)	77.12 (97.3%)	83.79 (99.4%)	95.70 (99.9%)	80.43 (98.7%)	79.57 (98.7%)
55%	72.30 (94.6%)	58.11 (95.0%)	83.50 (97.4%)	75.51 (95.3%)	82.37 (97.7%)	94.90 (99.1%)	79.32 (97.4%)	78.00 (96.8%)
60%	69.72 (91.2%)	59.30 (96.9%)	83.11 (96.9%)	75.30 (95.0%)	82.59 (98.0%)	95.60 (99.8%)	77.74 (95.4%)	77.62 (96.3%)

Table 8: Qwen-2-57B full downstream Task Performance: Gated

Skipped	MMLU	ARC	HellaSwag	LAMBADA	PIQA	SciQ	Winogrande	AVG
Base	74.50 (100.0%)	49.15 (100.0%)	72.42 (100.0%)	76.58 (100.0%)	81.08 (100.0%)	94.20 (100.0%)	73.58 (100.0%)	74.50 (100.0%)
39.3%	66.52 (89.3%)	45.39 (92.4%)	70.75 (97.7%)	69.25 (90.4%)	80.83 (99.7%)	90.90 (96.5%)	67.25 (91.4%)	70.13 (94.1%)
46.4%	57.03 (76.5%)	46.33 (94.3%)	67.25 (92.9%)	59.92 (78.2%)	79.92 (98.6%)	89.90 (95.4%)	68.67 (93.3%)	67.00 (89.9%)
50%	59.26 (79.5%)	43.00 (87.5%)	67.25 (92.9%)	68.67 (89.7%)	76.75 (94.7%)	91.10 (96.7%)	68.42 (93.0%)	67.78 (91.0%)
53.6%	48.26 (64.8%)	41.98 (85.4%)	65.08 (89.9%)	54.67 (71.4%)	78.42 (96.7%)	80.30 (85.2%)	66.58 (90.5%)	62.18 (83.5%)
60.7%	39.69 (53.3%)	36.43 (74.1%)	62.08 (85.7%)	46.50 (60.7%)	75.08 (92.6%)	85.60 (90.9%)	61.00 (82.9%)	58.06 (77.9%)

Table 9: Qwen-2-57B full downstream Task Performance: Gated + Recovered

Skipped	MMLU	ARC	HellaSwag	LAMBADA	PIQA	SciQ	Winogrande	AVG
Base	74.50 (100.0%)	49.15 (100.0%)	72.42 (100.0%)	76.58 (100.0%)	81.08 (100.0%)	94.20 (100.0%)	73.58 (100.0%)	74.50 (100.0%)
39.3%	69.45 (93.2%)	50.60 (103.0%)	71.00 (98.0%)	75.08 (98.0%)	80.75 (99.6%)	95.10 (101.0%)	73.58 (100.0%)	73.65 (98.9%)
46.4%	68.97 (92.6%)	51.45 (104.7%)	70.92 (97.9%)	73.92 (96.5%)	81.00 (99.9%)	94.50 (100.3%)	73.67 (100.1%)	73.49 (98.6%)
50%	69.16 (92.8%)	50.00 (101.7%)	70.17 (96.9%)	74.92 (97.8%)	80.92 (99.8%)	94.80 (100.6%)	72.00 (97.8%)	73.14 (98.2%)
53.6%	66.04 (88.7%)	50.68 (103.1%)	70.83 (97.8%)	73.58 (96.1%)	81.08 (100.0%)	94.40 (100.2%)	72.25 (98.2%)	72.70 (97.6%)
60.7%	61.52 (82.6%)	48.89 (99.5%)	70.17 (96.9%)	72.25 (94.3%)	80.50 (99.3%)	94.00 (99.8%)	70.25 (95.5%)	71.08 (95.4%)

B.2 SPEEDUP FULL RESULTS

Table 10: Numbers behind Figure 3a: skip-ratio sweep, prefill of 2048 tokens at batch 1 and decode at batch 128, latency time measured in milliseconds and speedup with one standard deviation over 15 repetitions. We report Ceiling for the speedup of simply removing all-reduce operations.

Model	Phase	Skip ratio	Baseline average latency [ms]	Gated All-Reduce average latency [ms]	Speedup	Ceiling
Llama-3.1-70B	prefill	40%	240.30	219.97	+9.24 ± 0.25%	+11.01 ± 0.16%
Llama-3.1-70B	prefill	45%	240.30	216.37	+11.06 ± 0.20%	+12.56 ± 0.16%
Llama-3.1-70B	prefill	50%	240.30	214.10	+12.23 ± 0.26%	+14.15 ± 0.17%
Llama-3.1-70B	prefill	55%	240.30	210.83	+13.98 ± 0.18%	+15.79 ± 0.18%
Llama-3.1-70B	prefill	60%	240.30	208.42	+15.29 ± 0.26%	+17.47 ± 0.20%
Llama-3.1-70B	decode	40%	31.50	29.06	+8.40 ± 0.77%	+9.84 ± 0.57%
Llama-3.1-70B	decode	45%	31.50	28.70	+9.76 ± 0.67%	+11.20 ± 0.58%
Llama-3.1-70B	decode	50%	31.50	28.42	+10.84 ± 0.75%	+12.61 ± 0.60%
Llama-3.1-70B	decode	55%	31.50	27.70	+13.71 ± 0.78%	+14.04 ± 0.62%
Llama-3.1-70B	decode	60%	31.50	27.46	+14.72 ± 0.89%	+15.52 ± 0.64%
Qwen-2-57B	prefill	39.3%	91.67	89.90	+1.97 ± 0.27%	+2.27 ± 0.14%
Qwen-2-57B	prefill	46.4%	91.67	89.22	+2.75 ± 0.15%	+2.70 ± 0.14%
Qwen-2-57B	prefill	50%	91.67	89.14	+2.84 ± 0.15%	+2.91 ± 0.14%
Qwen-2-57B	prefill	53.6%	91.67	89.48	+2.44 ± 0.36%	+3.12 ± 0.14%
Qwen-2-57B	prefill	60.7%	91.67	88.33	+3.78 ± 0.15%	+3.56 ± 0.14%
Qwen-2-57B	decode	39.3%	22.12	21.83	+1.31 ± 0.38%	+1.64 ± 0.32%
Qwen-2-57B	decode	46.4%	22.12	21.81	+1.42 ± 0.36%	+1.94 ± 0.34%
Qwen-2-57B	decode	50%	22.12	21.76	+1.65 ± 0.41%	+2.09 ± 0.35%
Qwen-2-57B	decode	53.6%	22.12	21.81	+1.39 ± 0.43%	+2.24 ± 0.36%
Qwen-2-57B	decode	60.7%	22.12	21.66	+2.12 ± 0.35%	+2.55 ± 0.38%

Table 11: Numbers behind Figure 3b: Llama-3.1-70B sweep over input sequence length (prefill, batch 1) and batch size (decode). Latency time measured in milliseconds and speedup with one standard deviation over 15 repetitions.

Phase	Input sequence length / batch size	Skip ratio	Baseline average latency [ms]	Gated All-Reduce average latency [ms]	Speedup
prefill	L=256	40%	42.89	38.78	+10.58 ± 2.69%
prefill	L=512	40%	84.38	73.35	+15.04 ± 0.51%
prefill	L=1024	40%	132.74	120.02	+10.60 ± 0.23%
prefill	L=2048	40%	240.83	221.46	+8.75 ± 0.30%
prefill	L=4096	40%	446.64	414.13	+7.85 ± 0.33%
prefill	L=8192	40%	878.78	815.54	+7.75 ± 0.07%
prefill	L=256	50%	42.89	37.31	+14.95 ± 2.69%
prefill	L=512	50%	84.38	71.05	+18.75 ± 0.61%
prefill	L=1024	50%	132.74	116.50	+13.94 ± 0.24%
prefill	L=2048	50%	240.83	215.11	+11.96 ± 0.35%
prefill	L=4096	50%	446.64	404.37	+10.45 ± 0.28%
prefill	L=8192	50%	878.78	798.50	+10.05 ± 0.07%
prefill	L=256	60%	42.89	37.54	+14.23 ± 3.17%
prefill	L=512	60%	84.38	69.25	+21.84 ± 0.69%
prefill	L=1024	60%	132.74	113.46	+16.99 ± 0.41%
prefill	L=2048	60%	240.83	208.75	+15.37 ± 0.31%
prefill	L=4096	60%	446.64	395.92	+12.81 ± 0.28%
prefill	L=8192	60%	878.78	780.17	+12.64 ± 0.06%
decode	B=16	40%	21.96	20.15	+9.00 ± 0.46%
decode	B=32	40%	23.43	21.62	+8.36 ± 0.46%
decode	B=128	40%	31.69	28.81	+9.99 ± 0.67%
decode	B=16	50%	21.96	19.63	+11.86 ± 0.39%
decode	B=32	50%	23.43	21.45	+9.24 ± 0.56%
decode	B=128	50%	31.69	28.39	+11.63 ± 0.72%
decode	B=16	60%	21.96	19.06	+15.21 ± 0.48%
decode	B=32	60%	23.43	20.98	+11.65 ± 0.48%
decode	B=128	60%	31.69	27.53	+15.09 ± 0.62%

Table 12: Numbers behind Figure 3c: all four models at a 50% skip ratio, prefill of 2048 tokens at batch 1 and decode at batch 128. Latency time measured in milliseconds and speedup with one standard deviation over 15 repetitions (except Llama-3.1-8B which we measure with 40 repetitions).

Model	Prefill baseline average latency [ms]	Prefill Gated All-Reduce average latency [ms]	Prefill speedup	Decode baseline average latency [ms]	Decode Gated All-Reduce average latency [ms]	Decode speedup
Llama-3.1-8B TP=8	40.90	34.20	+19.59 ± 0.20%	9.29	8.40	+10.56 ± 0.73%
Llama-3.1-70B TP=8	240.30	214.10	+12.23 ± 0.26%	31.50	28.42	+10.84 ± 0.75%
Qwen2-57B TP=8, EP=8	91.67	89.14	+2.84 ± 0.15%	22.12	21.76	+1.65 ± 0.41%
Qwen3-235B TP=8, EP=8	252.15	241.62	+4.36 ± 0.07%	68.56	67.12	+2.14 ± 0.07%