SCAPE: ACCURATE AND EFFICIENT LLM TRAINING WITH EXTREME SPARSE COMMUNICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Communication increasingly dominates the cost of Large Language Model (LLM) pre-training, especially under data-parallel and sharded training schemes, where gradient synchronization and parameter reconstruction overhead increase with model size and system scale. Existing communication-reduction methods either sparsify raw gradients, which can be unstable for modern Adam-style optimizers at high sparsity, or quantize communication, whose savings are fundamentally bounded by bit width. We present SCAPE, a communication-efficient distributed optimizer for LLM training that exploits the stability of AdamS's first-moment to enable aggressive sparsification while preserving LLM quality. Instead of constructing masks from gradients, SCAPE derives them from first-moment-based statistics, partitions mask generation across workers to align with optimizer sharding, and delays mask usage by one step so that mask synchronization can overlap with computation. SCAPE also reconstructs the quantities required for second-moment updates from a single synchronized sparse buffer, avoiding an additional collective communication. We implement SCAPE in Megatron-LM and evaluate its convergence by pre-training GPT-345M on OpenWebText and Llama-500M on SlimPajama-6B using 32 NVIDIA GH200 GPUs on TACC Vista supercomputer. In both models, SCAPE preserves training stability, validation loss, and downstream task accuracy under 90% and 99% sparsity. For Llama-500M, SCAPE reduces end-to-end pre-training wall-clock time by up to 43.3% while maintaining model quality comparable to dense AdamW and AdamS. For Llama-1.8B, SCAPE achieves up to 3.20× speedup per step compared to dense distributed AdamS.

1 INTRODUCTION

Large Language Models (LLMs) with ever-increasing sizes have achieved unparalleled performance across many fields, including math reasoning (Ahn et al., 2024), code generation (Chen et al., 2021), and autonomous laboratory (Szymanski et al., 2023). State-of-the-art LLM training requires a massive amount of accelerators, such as GPUs, to achieve faster training and accommodate enormous model parameters and optimizer states. The legacy data parallel (DP) strategy replicates the model and optimizer states (e.g., first and second moments in AdamW (Loshchilov & Hutter, 2019)) across GPUs and distributes a mini-batch of data among them. Modern sharded data parallel strategies, such as ZeRO (Rajbhandari et al., 2020), FSDP (Zhao et al., 2023), and Megatron-LM (NVIDIA, 2026), reduce spatial redundancy by partitioning the model and optimizer states across GPUs. In either case, communication (i.e., *all-reduce* and *all-gather*) is the scaling bottleneck given the dependency on model sizes and the scale (see Figure 15).

Lowering communication volume is an effective way to reduce communication overhead. Existing top-k methods are limited in applicability to modern optimizers or require low sparsity due to constraints on model performance. Many momentum-based sparsification methods, such as DGC (Lin et al., 2018) and DeMo (Peng et al., 2026), lacks empirical evidence to be effective in LLM training with Adam-like optimizers. Radius (Zheng & Zhang, 2025) proposes an error-correction mechanism for AdamW (Loshchilov & Hutter, 2019) optimizer and can achieve 40% sparsity without degrading downstream task performance. LEGACY (Essoullami et al., 2026) requires carefully manually desgined heuristic compression rates for layers of different sizes.

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.

In this work, we ask the following research question: Can we reduce communication volume to the extreme without harming the model performance? To this end, we propose **SCAPE**, a communication-efficient distributed optimizer that sparsifies the optimizer states rather than gradients. SCAPE is built upon AdamS (Zhang et al., 2025) and is inspired with two important insights: 1) Compared to AdamW, AdamS exhibits substantially smaller residual growth under high sparsity in LLM pretraining, indicating improved robustness to stale error-feedback dynamics; 2) The top-k structure of the first-moment is temporally stable across two adjacent steps, suggesting that delayed mask reuse is practical. Based on these observations, SCAPE introduces a partitioned mask-refresh mechanism aligned with optimizer sharding. SCAPE computes masks from momentum-based statistics rather than gradients and delays one step so that mask synchronization can overlap with computation. When sharded data parallel is enabled, with the model and optimizer states distributed across all GPUs, SCAPE compresses the volume using sparsity for both *reduce-scatter* for gradient synchronization and *all-gather* for reconstructing the model.

We implement SCAPE using Megatron-LM (NVIDIA, 2026) and empirically verify its convergence through pre-training GPT-345M on the OpenWebText (Gokaslan et al., 2019) dataset and Llama-500M on the SlimPajama-6B (Soboleva et al., 2023) dataset. We run experiments on the TACC Vista with evaluation on an extensive suite of downstream tasks. Our experiment results show that SCAPE with 90% and 99% sparsity can reduce the wall-clock time for Llama-500M pretraining on 32 GH200 GPUs by 35.6% and 43.3%, respectively. For Llama-1.8B training, SCAPE achieves up to 3.20× speedup per step compared to dense distributed AdamS. More importantly, under such high sparsities, SCAPE maintains for both GPT-345M's and Llama-500M's final train and validation losses close to those of dense AdamS and AdamW, without affecting their performance on downstream benchmarks, such as LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2019), and ARC (Clark et al., 2018).

2 Background and Related Work

Due to the limited space, related work is moved to Section B.

2.1 Training with Distributed Optimizer

Distributed optimizer has optimizer states (e.g. first- and second-moment of Adam optimizer) distributed onto all workers (e.g., GPUs) in the same data parallel (DP) group. Workers perform *all-gather* to collect the sharded model parameters to perform forward and backward pass. Then, they perform *reduce-scatter* to compute the average of gradients and shard the result evenly onto workers. An illustration is provided in Figure 9, and more details could be found in Appendix A. The total communication cost is the same as an *all-reduce*:

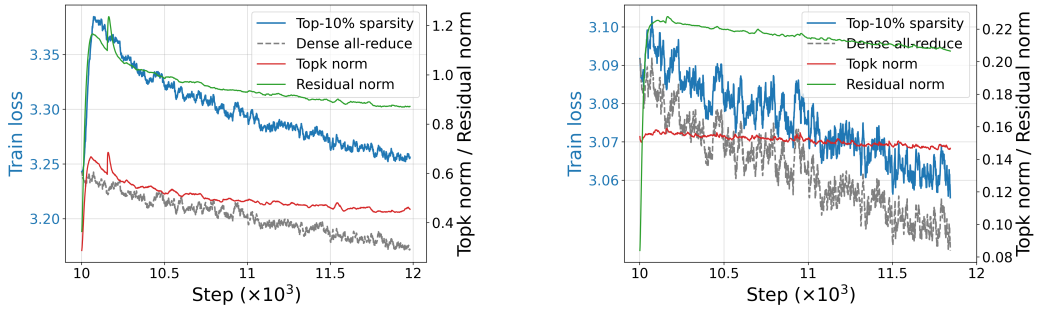
$$T_{\text{ring-}all\text{-}reduce} = T_{reduce\text{-}scatter} + T_{all\text{-}gather} = 2(N-1)\left[\alpha + \beta\frac{W}{N}\right], \tag{1}$$

2.2 Top-k Gradient Sparsification

Top-k sparsification is a technique for reducing $T_{\text{ring-}all\text{-}reduce}$ by reducing W in Equation 1, given a fixed network bandwidth $1/\beta$. Before synchronizing the gradient g, we perform a top-k operation to select gradient values with top-k largest magnitudes, and then synchronize them using a compressed buffer. This effectively reduces the communication volume from W to k.

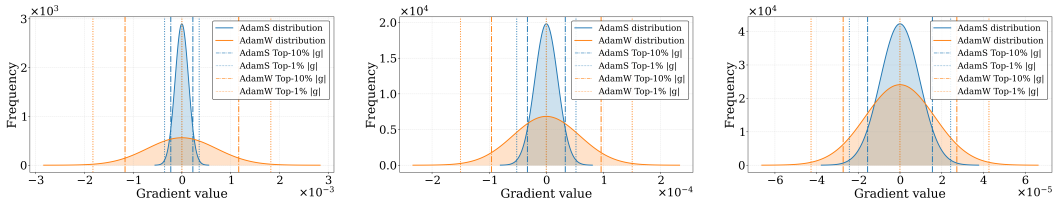
Error feedback (EF) (Stich et al., 2018) is an effective method for stabilizing training dynamics with sparsified gradient. Let e_t denote the error buffer at step t , then the compressed gradient is $\tilde{g}_t = \text{TopK}(g_t + e_t)$, and the residual is updated as $e_{t+1} = g_t + e_t - \tilde{g}_t$. In this way, the non-top-k in one iteration is not lost, but is gradually re-injected into later steps, improving optimization stability while retaining the communication savings.

Since each worker's data samples for computing the gradient g_t at step t are distinct, their top-k gradient indices are distinct. To guarantee the correspondence between top-k indices and the selected gradient values, *all-gather* is first called to form a synchronized top-k mask, and *all-reduce* then synchronizes the gradient entries selected by the globally synchronized mask. However, as described in (Li & Hoefler, 2022), when the number of workers N is large enough, the globally synchronized top-k mask is dense rather than sparse (i.e., the fill-in effect).



(a) AdamW (b) AdamS

Figure 1: Train loss curves for pretraining GPT-345M with AdamW and AdamS after switching to top-10% sparsified gradient at step 10,000.



(a) GPT-345M: D0 - Attn. dense (b) Llama-500M: D8 - Attn. dense (c) Llama-500M: D0 - MLP gate/up

Figure 2: Gradient distribution of different layers in GPT-345M and Llama-500M at step 1,000.

2.3 AdamS Optimizer

AdamS (Zhang et al., 2025) is an Adam-like optimizer which, instead of keeping a second-moment v_t as an exponential moving average (EMA) of $g_t^{\odot 2}$, uses first-moment m_t as the preconditioner:

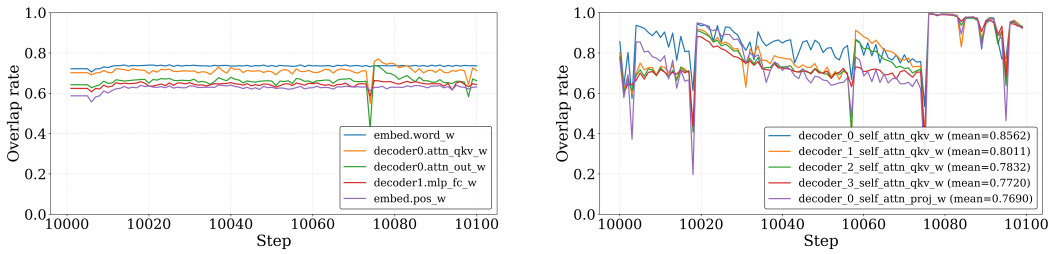
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 m_{t-1}^{\odot 2} + (1-\beta_2) g_t^{\odot 2},$$
$$\theta_{t+1} = \theta_t - \eta \left(\frac{m_t}{\sqrt{v_t} + \epsilon} + \lambda \theta_t \right), \tag{2}$$

where β_1 and β_2 are the coefficients for computing the running average, θ_t is the model parameter at step t, η is the learning rate, λ is the weight decay hyper-parameter, and ϵ maintains the numerical stability. Note that the update metric in Equation 2 is simplified for demonstration purposes, and in practice, m_t and v_t should be multiplied by their bias correction terms $1/(1-\beta_1^t)$ and $1/(1-\beta_2^t)$.

3 Observations

3.1 Gradient Distribution

We pretrain GPT-345M using AdamW and AdamS on OpenWebText (Gokaslan et al., 2019). We use dense *all-reduce* for the first 10,000 steps and then switch to using top-10% sparsity on gradient with EF. The train loss curves are shown in Figure 1, where we see that both optimizers have loss curves first spikes and then gradually decreases toward the dense *all-reduce* curve. Nevertheless, AdamS has much lower spikes compared to AdamW, as its norm of the sparsified gradient and norm of the EF buffer are much smaller than those of AdamW, suggesting that AdamS is less influenced by the staleness effect of EF. With largest stable gradient added back to AdamW's optimizer states, it causes the optimizer states to drift from the dense direction. We thus plot the distribution of AdamW's and AdamS's gradient for different layers for GPT-345M and Llama-500M in Figure 2, from which we notice that AdamS's gradient is more centralized around 0, while AdamW's spreads more evenly and is flatter. This suggests that AdamS is more suitable for adapting to top-k gradient sparsification.



(a) GPT-345M from step 10,000 (b) Llama-500M from step 10,000

Figure 3: The overlap rate of AdamS's first-moment-derived top-10% mask between two adjacent steps for different stages of pre-training GPT-345M and Llama-500M.

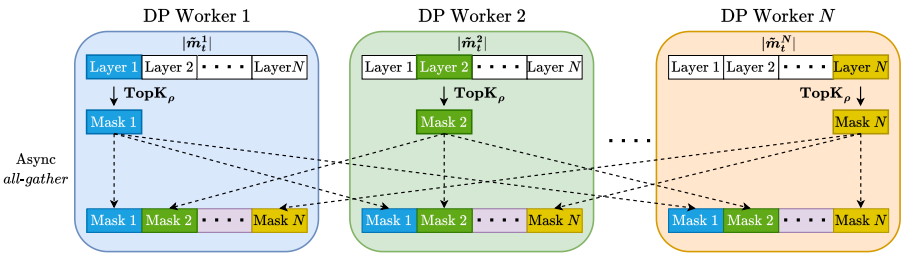


Figure 4: Communication in refreshing top-k mask. Each worker computes a sharded top-k mask and then uses *all-gather* to construct the full mask.

3.2 Temporal Stability of First-Moment-Derived Top-k Masks

We explore and observe the temporal stability of top-k masks constructed from AdamS's first-moment between two adjacent steps. As shown in Figure 3a, many layers in GPT-345M exhibit high mask overlap in the pretraining. Although Llama-500M's overlap curves in Figure 3b is noisier , the average overlap rate for each layer remains high across training stages. More overlap rate plots for other layers at different training steps could be found in Figure 12.

4 Algorithm Design

We design SCAPE for DDP in Algorithm 1, which is mainly composed of three important components: 1) distributed mask generation, 2) top-k mask computation, and 3) optimizer state update.

Mask Synchronization. We propose a new mask refresh strategy illustrated in Figure 4. Each worker in the DP group computes its own top-k mask using the disjoint, local information and then forms the globally synchronized mask M through *all-gather*: $M = M^{(1)} \cup M^{(2)} \cup \cdots \cup M^{(N)}$, where $M^{(i)}$ is the local mask from worker i. Since $M^{(i)}$ is computed from the sharded statistics and is disjoint from each other, as N increases, the density of M is guaranteed to remain the same.

Top-k Mask Computation. We use the top-k indices of $|h_t^n| = |\beta_1 m_{t-1} + (1-\beta_1) g_t^n + e_{t-1}^n|$ to generate the top-k masks. Based on the temporal stability of top-k mask observed in Section 3.2, the top-k mask at step t, M_t, could be delayed and used at step $t+1$, so that the communication cost for synchronizing masks can be overlapped by the forward-backward computation.

Optimizer State Update. In Algorithm 1, SCAPE transmits a sparsified buffer b_t^n, which has the values in the temporary, semi-updated first-moment buffer h_t^n on positions selected by the top-k mask M_{t-1}. Given that m_t is guaranteed to be synchronized, we can compute the globally averaged gradient and residual as $M_{t-1}(\bar{g}_t + \bar{e}_{t-1}/(1-\beta_1)) = M_{t-1}(\bar{b}_t - \beta_1 m_{t-1})/(1-\beta_1)$. Then, we use $\bar{g}_t + \bar{e}_{t-1}/(1-\beta_1)$ to update the first-moment m_t, second-moment v_t, and parameter θ_{t+1}.

Algorithm 1 SCAPE (Distributed Data Parallel (DDP))

Require: initial model parameters θ_1, dataset $\mathcal{D}$, loss function f, worker rank n, number of workers N, density d, weight decay λ, learning rate η, maximum gradient norm C, coefficients β_1, β_2, numerical stabilizer ϵ.

$m_0 \leftarrow \mathbf{0}$ ▷ Initialize first moment
$e_0^n \leftarrow \mathbf{0}$ ▷ Initialize EF buffer
$M_0 \leftarrow \mathbf{1}$ ▷ Initialize top-k mask
for $t = 1, 2, \ldots$ **do**
  $g_t^n \leftarrow \nabla f(\mathcal{D}_t^n, \theta_t)$
  $h_t^n \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t^n + e_{t-1}^n$
  $b_t^n \leftarrow M_{t-1} h_t^n, \quad e_t^n \leftarrow h_t^n - b_t^n$
  $\bar{b}_t \leftarrow \text{MeanAllReduce}(b_t^n)$ ▷ Compress sparsified b_t^n by dropping 0's
  $M_t \leftarrow \text{RefreshMask}(d, |h_t^n|)$ ▷ Check Figure 4
  $r_t \leftarrow \text{Clip}\left(M_{t-1}(\bar{b}_t - \beta_1 m_{t-1})/(1-\beta_1)\right)$ ▷ Global L_2-norm
  $m_t \leftarrow M_{t-1}[\beta_1 m_{t-1} + (1-\beta_1) r_t]$
  $v_t \leftarrow \beta_2 m_{t-1}^{\odot 2} + (1-\beta_2) r_t^{\odot 2}$
  $\theta_{t+1} \leftarrow (1-\eta\lambda)\theta_t - \eta \frac{m_t}{\sqrt{v_t}+\epsilon}$
end for

Optimizations for Distributed Optimizers. We can see from Algorithm 1 that since $\bar{b}_t$ only has nonzero values for indices selected by top-k mask M_{t-1}, we have $m_t[\neg M_{t-1}] = \bar{b}_t[\neg M_{t-1}] = 0$. Therefore, when computing the update metrics, we have $u_t[\neg M_{t-1}] = m_t[\neg M_{t-1}]/(\sqrt{v_t[\neg M_{t-1}]} + \epsilon) = 0$. Hence,

$$\theta_{t+1}[\neg M_{t-1}] = \theta_t[\neg M_{t-1}] - \eta(u_t[\neg M_{t-1}] + \lambda\theta_t[\neg M_{t-1}]) = (1-\eta\lambda)\theta_t[\neg M_{t-1}] \tag{3}$$

The update only depends on the current parameter θ_t, the learning rate η, and weight-decay coefficients λ. Consequently, with sharded optimizer, the unselected coordinates of parameters can be computed from its local FP32 replica maintained by Megatron-LM, eliminating the need for *all-gather*. The complete pseudocode for SCAPE using distributed optimizer (DO) is in Algorithm 2. The resulting memory overhead and optional CPU offloading are profiled in Appendix L.

5 Convergence of SCAPE

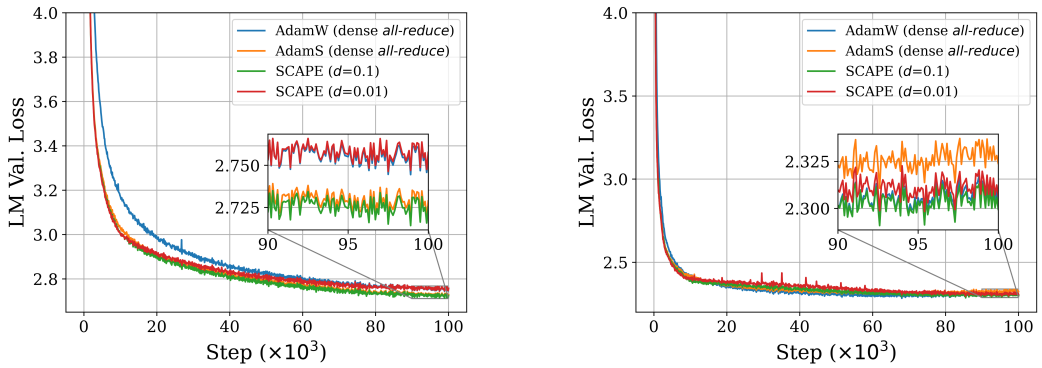
We establish a conditional stationarity rate for an *idealized fresh-mask variant* of Algorithm 1, which replaces M_{t-1} by M_t in compression, reconstruction, and the first-moment update. The analysis assumes deterministic initialization, zero initial moments and residuals, fixed block densities, inactive clipping, zero weight decay, and a constant learning rate within each run. It uses uncorrected AdamS and exact arithmetic and communication; Appendix E specifies the recursion and full assumptions.

Assumptions. The objective $F = N^{-1}\sum_n F_n$ is L-smooth and bounded below. Conditioned on the past, local stochastic gradients are independent across workers and unbiased, with $\|g_t^n\| \le G$ almost surely and bounded variances. Write bars for worker averages, let $P_t = \mathrm{Diag}(M_t)$ be the coordinate projection onto the fresh mask M_t, and express the update as $\theta_{t+1} = \theta_t - \eta H_t m_t$. We assume aggregate contraction $\|(I-P_t)\bar{h}_t\| \le q\|\bar{h}_t\|$ for $q < 1$ and bounded adaptive scaling $0 \prec H_t \preceq h_+ I$ ($h_+ = \epsilon^{-1}$ for AdamS). With $\mathbb{E}_t$ conditioning on the past, we additionally require positive adaptive alignment: $\mathbb{E}_t\langle \nabla F(\theta_t), H_t \bar{g}_t\rangle \ge \mu\|\nabla F(\theta_t)\|^2$ where $\mu > 0$. Separately, set $s_t = \beta_1 m_t + \bar{e}_t$, $\beta_1 \in [0,1)$, and $H_0 = H_1$, and define the preconditioner drift budget

$$V_{1,T} = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\big\|(H_t - H_{t-1}) s_{t-1}\big\|. \tag{4}$$

Theorem 5.1 (Conditional convergence of fresh-mask SCAPE)**.** *Under Assumptions E.1–E.4 in Appendix E, keep N, the initial objective gap, and all optimizer and assumption constants other than η independent of T. If $\eta = c/\sqrt{T}$ with $0 < c \le \mu/(2Lh_+^2)$ and $V_{1,T} = O(T^{-1/2})$, then*

$$\mathcal{G}_T := \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\|\nabla F(\theta_t)\|^2 = O(T^{-1/2}). \tag{5}$$



(a) GPT-345M val. loss

(b) Llama-500M val. loss

Figure 5: Validation loss curves for pretraining GPT-345M and Llama-500M.

Table 1: Final training and validation losses for GPT-345M and Llama-500M, with end-to-end (E2E) wall-clock time for Llama-500M. Parentheses indicate speedup relative to dense AdamW.

	GPT-345M		Llama-500M		
Method	Train loss ↓	Val. loss ↓	Train loss ↓	Val. loss ↓	E2E time (days) ↓
AdamW (dense)	2.80	2.76	2.10	2.30	2.47 (1×)
AdamS (dense)	2.77	2.73	2.11	2.32	2.47 (1×)
SCAPE (d=0.1)	2.77	2.73	2.13	2.30	1.59 (1.55×)
SCAPE (d=0.01)	2.81	2.76	2.17	2.31	1.40 (1.76×)

Proof sketch. The common projection gives $m_t = P_t\bar{h}_t$ and $\bar{e}_t = (I - P_t)\bar{h}_t$ on disjoint supports, with $\bar{h}_t = s_{t-1} + (1-\beta_1)\bar{g}_t$. Orthogonality and contraction imply, for $\tau = \sqrt{\beta_1^2 + (1-\beta_1^2)q^2} < 1$,

$$\|s_t\| \le \tau\|s_{t-1}\| + \tau(1-\beta_1)G, \qquad \|s_t\| \le B_q := \frac{\tau(1-\beta_1)G}{1-\tau},$$

where the second bound follows by iteration from $s_0 = 0$. Error feedback gives $m_t = \bar{g}_t + (s_{t-1} - s_t)/(1-\beta_1)$, so the virtual iterate satisfies

$$\begin{aligned} y_t &= \theta_t - \frac{\eta}{1-\beta_1} H_{t-1} s_{t-1}, \\ y_{t+1} &= y_t - \eta H_t \bar{g}_t + \frac{\eta}{1-\beta_1}(H_{t-1} - H_t) s_{t-1}. \end{aligned} \tag{6}$$

The state bound gives $\|y_t - \theta_t\| \le \eta h_+ B_q/(1-\beta_1) = O(\eta)$. Error feedback cancels except for preconditioner drift. Smoothness and adaptive alignment give descent; telescoping from $y_1 = \theta_1$ and using the lower bound on F yields $\mathcal{G}_T = O((\eta T)^{-1} + \eta + V_{1,T})$, proving the rate. Appendix E provides the finite-time bound and complete proofs. Aggregate contraction is an additional condition on the averaged proposal. Unbiased gradients and a decreasing choice of η do not establish adaptive alignment or drift decay. A fixed deterministic H removes those two requirements while retaining the objective, oracle, and contraction assumptions (Proposition E.10).

Delayed masks in practice. The temporal overlap in Section 3.2 and Figure 3 and comparable final losses in the GPT-345M ablation (Section 6.3 and Figure 8b) support one-step mask reuse to overlap communication with computation. Common delayed masks preserve orthogonality, but contraction on the current proposal still requires justification. The theorem's conditional stationarity guarantee is limited to the idealized fresh-mask variant and does not establish convergence of the delayed, clipped, weight-decayed implementation.

6 Experiment Results

We evaluate SCAPE with distributed optimizer by pretraining GPT-345M and Llama-500M using 32 NVIDIA GH200 GPUs on TACC Vista, where GPUs are connected by 400 Gbps InfiniBand network

Table 2: Zero-shot accuracy on downstream tasks for GPT-345M and Llama-500M. Bold values indicate the best score for each task within each model, including ties.

	GPT-345M					Llama-500M				
Method	ARC-E↑	HellaSwag↑	LAMBADA↑	PIQA↑	BoolQ↑	ARC-E↑	HellaSwag↑	LAMBADA↑	PIQA↑	BoolQ↑
AdamW (dense)	**48.82**	30.63	39.57	64.91	53.12	48.86	**34.70**	**38.60**	**67.30**	57.65
AdamS (dense)	47.77	**31.45**	39.51	**65.18**	55.72	46.97	34.57	36.66	65.94	57.55
SCAPE ($d = 0.1$)	**48.82**	31.28	**39.71**	64.80	**59.36**	**48.95**	34.38	38.39	66.76	51.90
SCAPE ($d = 0.01$)	46.93	31.06	39.06	64.42	56.42	47.47	33.93	36.64	66.32	**57.92**

and equipped with 96 GB of HBM3 memory and NVLink-C2C between the host and the device. We profile the per-step time of Llama-500M and Llama-1.8B ranging from 4 to 64 GPUs. The detailed model architecture and hyperparameters are in Appendix F and Appendix J. To ensure a consistent and reproducible software environment, all experiments used NGC PyTorch container (v26.01).

6.1 Pretraining Results

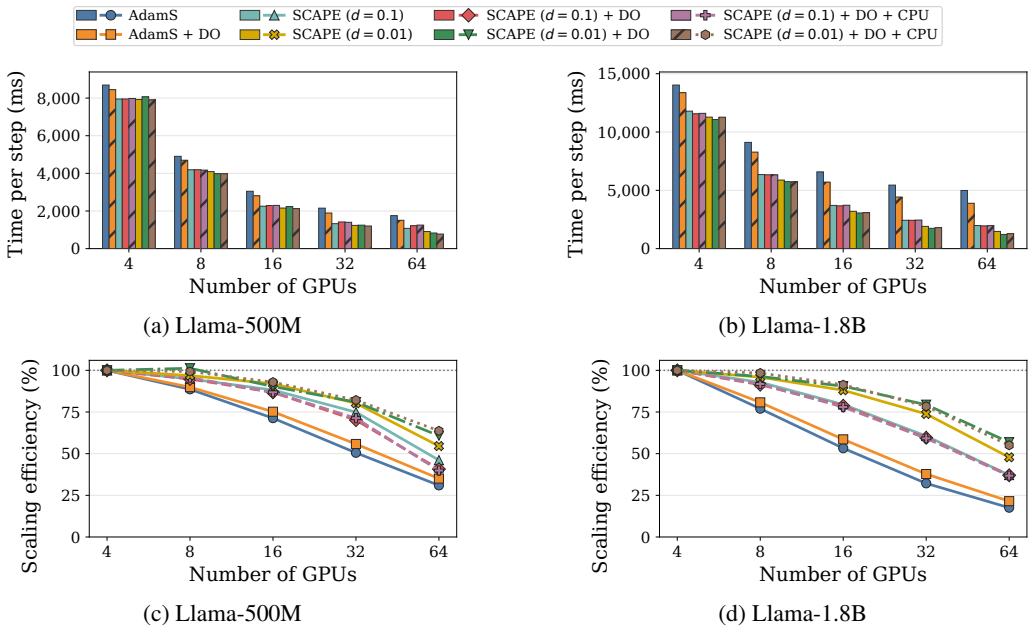
Convergence. We train both GPT-345M and Llama-500M using various methods, including AdamW, AdamS, SCAPE at $d = 0.1$ (i.e., 90% sparsity), and SCAPE at d =0.01 (i.e., 99% sparsity). Their validation loss curves are plotted in Figure 5, and the final train and validation losses are listed in Table 1. Due to the limited space, we move the train loss curves to Figure 13 in Appendix G. For both models, SCAPE at $d = 0.1$ and d =0.01 achieves train and validation losses comparable to those of AdamS and AdamW.

Downstream Task Performance. We evaluate the pretrained models on extensive zero-shot downstream benchmarks using lm-evaluation-harness (Gao et al., 2024). In Table 2, we present the scores for ARC-E (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2019), and BoolQ (Clark et al., 2019). Full results are provided in Table 3 and Table 4. Overall, SCAPE retains competitive downstream accuracy for both models under 90% and 99% sparsity, with minor influence on some tasks. For GPT-345M, SCAPE ($d = 0.1$) matches or exceeds dense AdamW on four of the five selected tasks. For Llama-500M, SCAPE achieves the highest ARC-E score at $d = 0.1$ and the highest BoolQ score at d =0.01.

Per-Step Time and Strong Scaling Efficiency. We profile Llama-500M and Llama-1.8B from 4 to 64 GPUs. The per-step training times and strong-scaling efficiencies are shown in Figure 6. We can see that compared to AdamS + DO, by using SCAPE ($d = 0.1$) and SCAPE ($d = 0.01$), the per-step time for training Llama-1.8B is reduced by $1.99\times$ and $3.20\times$, and the strong-scaling efficiency is improved from 21.48% to 37.06% and 56.88%, respectively. This shows that SCAPE is an effective method for improving the efficiency of training LLMs on large-scale systems. We have also implemented master-weight and EF-buffer offloading, with details provided in Appendix J and Appendix K.

6.2 Comparison to SOTA

We compare SCAPE to state-of-the-art (SOTA) methods, including DGC (Lin et al., 2018), PowerSGD (Vogels et al., 2019), SDP4Bit (Jia et al., 2024), LEGACY (Essoullami et al., 2026), and DeMo (Peng et al., 2026), via pretraining GPT-345M with an equivalent compression rate for all methods. We follow the Chinchilla rule (Hoffmann et al., 2022) and train all methods for 7.86B tokens. We use rank=16 and warmup steps of 1,000 for PowerSGD with EF. The configurations for LEGACY, DGC, and DeMo can be found in Appendix F. In addition to train and validation loss curves, we include the density ratio compared to dense communication using 32 GPUs to highlight the fill-in effect in large-scale training. We can see from Figure 7 that SCAPE ($d = 0.1$) achieves the lowest train and validation loss among all methods and maintains the desired compression rate throughout the training process. Since LEGACY, DGC, DeMo, and PowerSGD are implemented using purely PyTorch while SCAPE is optimized by CUDA kernels, their computation overhead cannot be fairly compared, and the per-step time comparison is omitted.



(a) Llama-500M (b) Llama-1.8B

(c) Llama-500M (d) Llama-1.8B

Figure 6: Per-step training time (a,b) and strong-scaling efficiency (c,d) for Llama-500M (sequence length 4K) and Llama-1.8B (sequence length 2K). Entries without DO use DDP. DO denotes the distributed optimizer. CPU denotes full-parameter CPU offloading.

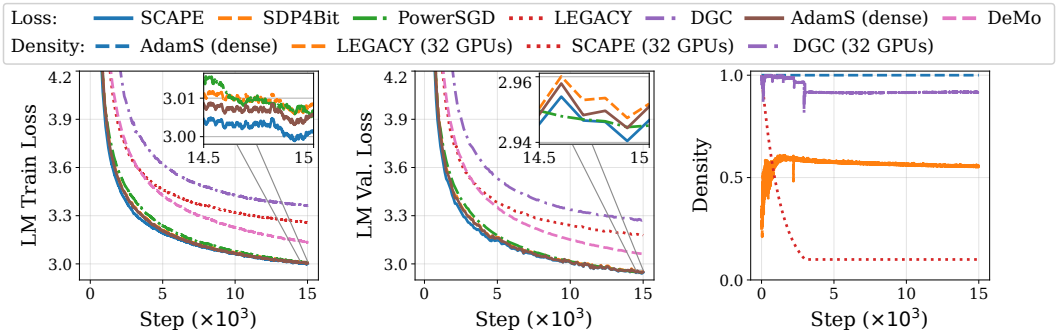


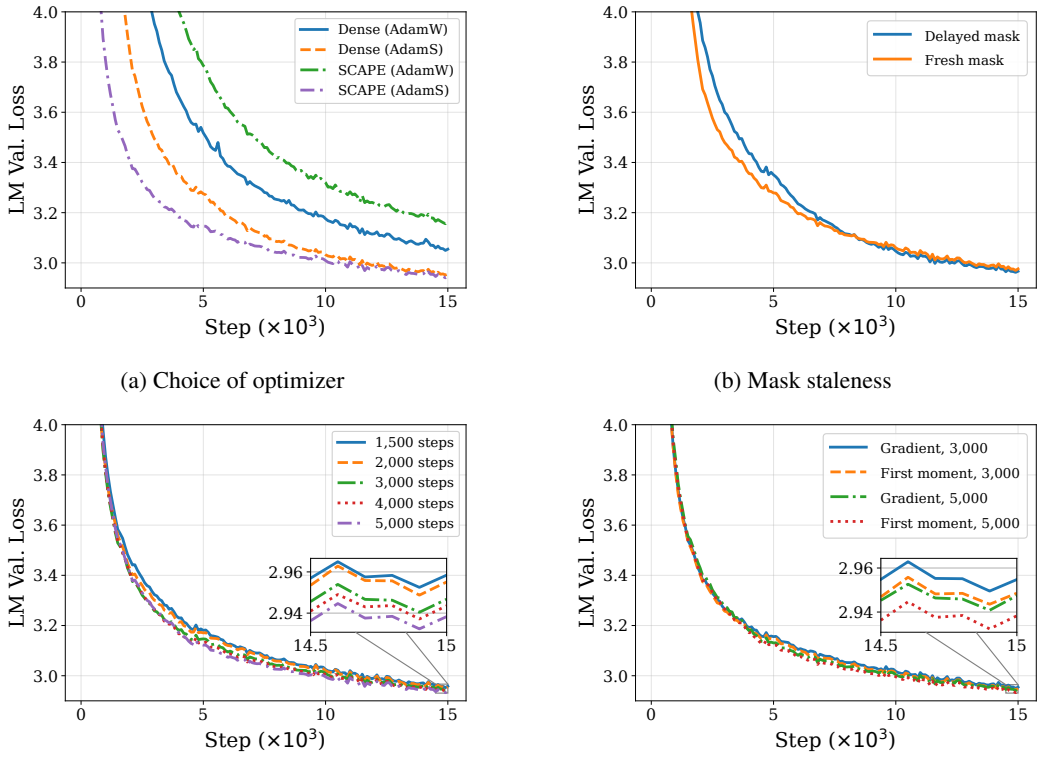
Figure 7: Comparison with dense AdamS, DGC, LEGACY, PowerSGD, and SDP4Bit.

6.3 Ablation Study

We isolate optimizer choice, mask staleness, compression warmup length, and mask score source. All models use GPT-345M and train on 7.86B tokens. Unless otherwise stated, variants share hyperparameters and SCAPE uses 3,000 compression warmup steps and $d = 0.1$. Train and validation loss curves are in Figure 14 and Figure 8, respectively.

Choice of Optimizer. We apply SCAPE to AdamW to highlight the importance of the choice of optimizer. In Figure 8a, we compare SCAPE with AdamW against dense AdamW, dense AdamS, and SCAPE with AdamS. We can see that naively applying SCAPE with AdamW results in a much higher validation loss than dense AdamW.

Mask Staleness. We investigate the impact of mask staleness in Figure 8b, where we compare the train and validation curves for fresh mask (i.e., mask generated from current step's first-moment scores; exposing the mask *all-gather* time) and mask delayed by one step. The fresh-mask run initially reaches lower loss, while the delayed-mask run ends at a similar, slightly lower loss. These empirical results support the practical use of delayed masks.



(a) Choice of optimizer

(b) Mask staleness

(c) Sparsity warmup steps

(d) Mask score source

Figure 8: Validation loss curves for the SCAPE ablation studies on GPT-345M: optimizer choice, mask staleness, density warmup steps, and mask score source.

Compression Warmup Steps. SCAPE uses exponential decay on density to gradually increase the compression rate to the target value. We investigate the impact of the number of steps for the density decay by sweeping through {1,500, 2,000, 3,000, 4,000, 5,000}, with the validation loss curves shown in Figure 8c. We notice that as the compression warmup step increases, the final validation loss decreases as well, at the cost of a lower compression rate at the beginning. Therefore, it represents a trade-off between speed and quality. However, as LLMs nowadays are over-trained with tokens exceeding Chinchilla rule, such an increase in warmup steps is insignificant.

Mask Score Generation Source. We justify our choice of using first-moment with residual $|h_t^n|$ for generating top-k mask through a comparison to using gradient with residual $|g_t^n + e_{t-1}^n|$. We see from the zoom-in plots in Figure 8d that experiments using $|h_t^n|$ for the top-k mask achieve lower validation loss than those using $|g_t^n + e_{t-1}^n|$ under different compression warmup steps.

7 Conclusion

We present SCAPE, a communication-efficient distributed optimizer for LLM pretraining achieving aggressive sparsification. SCAPE leverages the stability of AdamS's first-moment to construct top-k masks for sparse communication, aligns the top-k mask generation with optimizer sharding, and applies the top-k masks with a one-step delay to overlap the mask synchronization with computation. Through extensive evaluations, we show that SCAPE preserves the train and validation loss under both 90% and 99% sparsity, with preserved performance on downstream tasks. For Llama-500M, SCAPE reduces end-to-end pretraining wall-clock time by up to 43.3% relative to dense baselines while maintaining comparable model quality. For Llama-1.8B, SCAPE achieves 3.20× speedup on 64 GPUs compared to distributed AdamS using dense communication.

8 AI USE STATEMENT

In this work, we used generative AI tools for reviewing and formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, and designing or providing feedback on research methodology or experiments. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, implementing methods, or interpreting results, and generating synthetic data sets, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work. We have taken some AI generated suggestions to polish our paper and improve readability. We have reviewed all AI-assisted work, including proofreading the AI generated convergence proof. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

9 ETHICS STATEMENT

In this work, we propose a communication-efficient distributed optimizer for LLM pretraining, and it does not involve human subjects, personally identifiable information, or the collection of new data. All experiments use publicly available pretraining corpora and public downstream benchmarks, used in accordance with their respective terms. We do not release new datasets or pretrained model weights. By reducing communication and end-to-end training time, SCAPE may lower the compute and energy cost of LLM pretraining; it does not introduce new model capabilities beyond those of standard dense training.

10 REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our experiment, we use openly accessible framework and dataset. Concretely, we use the NGC PyTorch container (v26.01) as the experiment environment, and we use accessible pretraining dataset on Huggingface: OpenWebText and SlimPajama-6B. Moreover, the detail configurations in the experiments, including model architectures, datasets, tokenizers, optimizer hyperparameters, density-warmup schedules, and the LEGACY, DGC, and DeMo baseline configurations, are listed in Appendix F, and the PowerSGD settings are given in Section 6.2. The downstream tasks are accessed through lm-eval-harness framework. The source code of SCAPE, including the Megatron-LM integration, CUDA kernels, and training scripts, will be released publicly upon publication.

REFERENCES

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges, 2024. URL https://arxiv.org/abs/2402.00157.

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms, 2019. URL https://arxiv.org/abs/1905.13319.

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions, 2019. URL https://arxiv.org/abs/1905.10044.

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.

Mostapha Essoullami, El Houcine Bergou, and Aritra Dutta. LEGACY: A Lightweight Dynamic Gradient Compression Strategy for Distributed Deep Learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 91682–91714, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/93fce71def4e3cf418918805455d436f-Paper-Conference.pdf.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The Language Model Evaluation Harness, 07 2024. URL https://doi.org/10.5281/zenodo.12608602.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. OpenWebText Corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019. URL https://doi.org/10.5281/zenodo.3834942.

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.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Jinda Jia, Cong Xie, Hanlin Lu, Daoce Wang, Hao Feng, Chengming Zhang, Baixi Sun, Haibin Lin, Zhi Zhang, Xin Liu, and Dingwen Tao. SDP4Bit: Toward 4-bit Communication Quantization in Sharded Data Parallelism for LLM Training. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 8734–8759. Curran Associates, Inc., 2024. doi: 10.52202/079017-0279. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/10826a1a80f816ea98d559d7c7a97973-Paper-Conference.pdf.

Shigang Li and Torsten Hoefler. Near-optimal sparse allreduce for distributed deep learning. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, mar 2022. doi: 10.1145/3503221.3508399. URL https://doi.org/10.1145/3503221.3508399.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. Deep Gradient Compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1712.01887.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Ilia Markov, Adrian Vladu, Qi Guo, and Dan Alistarh. Quantized Distributed Training of Large Models with Convergence Guarantees, 2023. URL https://arxiv.org/abs/2302.02390.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1260. URL https://doi.org/10.18653/v1/D18-1260.

NVIDIA. Megatron-lm, 2026. URL https://github.com/NVIDIA/Megatron-LM.

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.

Bowen Peng, Lizhang Chen, Baiyu Su, Jeffrey Quesnelle, Diederik P. Kingma, and Qiang Liu. DeMo: Decoupled Momentum Optimization, 2026. URL https://arxiv.org/abs/2411.19870.

Pascal Pfeiffer, Philipp Singer, Yauhen Babakhin, Gabor Fodor, Nischay Dhankhar, and Sri Satish Ambati. H2O-Danube3 Technical Report, 2024. URL https://arxiv.org/abs/2407.09276.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399. URL https://doi.org/10.1609/aaai.v34i05.6399.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Jaeyong Song, Jinkyu Yim, Jaewon Jung, Hongsun Jang, Hyung-Jin Kim, Youngsok Kim, and Jinho Lee. Optimus-CC: Efficient Large NLP Model Training with 3D Parallelism Aware Communication Compression. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, pp. 560–573, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399166. doi: 10.1145/3575693.3575712. URL https://doi.org/10.1145/3575693.3575712.

Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 4452–4463, Red Hook, NY, USA, 2018. Curran Associates Inc.

Nathan J. Szymanski, Bernardus Rendy, Yuxing Fei, Rishi E. Kumar, Tanjin He, David Milsted, Matthew J. McDermott, Max Gallant, Ekin Dogus Cubuk, Amil Merchant, Haegyeom Kim, Anubhav Jain, Christopher J. Bartel, Kristin Persson, Yan Zeng, and Gerbrand Ceder. An autonomous laboratory for the accelerated synthesis of inorganic materials. *Nature*, 624(7990): 86–91, 2023. doi: 10.1038/s41586-023-06734-w. URL https://doi.org/10.1038/s41586-023-06734-w.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023. URL https://arxiv.org/abs/2307.09288.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. *PowerSGD: practical low-rank gradient compression for distributed optimization*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/4496bf24afe7fab6f046bf4923da8de6-Paper.pdf.

Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. ZeRO++: Extremely Efficient Collective Communication for Giant Model Training, 2023. URL https://arxiv.org/abs/2306.10209.

Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright. ATOMO: Communication-efficient Learning via Atomic Sparsification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/33b3214d792caf311e1f00fd22b392c5-Paper.pdf.

Qingao Yi, Jiaang Duan, Hanwen Hu, Qin Hua, Haiyan Zhao, Shiyou Qian, Dingyu Yang, Jian Cao, Jinghua Tang, Yinghao Yu, Chenzhi Liao, Kangjin Wang, and Liping Zhang. EDGC: Entropy-driven Dynamic Gradient Compression for Efficient LLM Training, 2025. URL https://arxiv.org/abs/2511.10333.

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.

Huishuai Zhang, Bohan Wang, and Luoxin Chen. AdamS: Momentum Itself Can Be A Normalizer for LLM Pretraining and Post-training. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10719–10738, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.543. URL https://doi.org/10.18653/v1/2025.emnlp-main.543.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel, 2023. URL https://arxiv.org/abs/2304.11277.

Mingkai Zheng and Zhao Zhang. Radius: Range-based Gradient Sparsity for Large Foundation Model Pre-training. In M. Zaharia, G. Joshi, and Y. Lin (eds.), *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/file/54dd9e0cff6d9214e20d97eb2a3bae49-Paper-Conference.pdf.

APPENDIX CONTENTS

A Distributed Optimizer Training **15**

B Related Work **15**

C Observations **16**

C.1 Gradient Distribution 16

C.2 Temporal Stability of first-moment-Derived Top-k Masks 16

D SCAPE with Distributed Optimizer **16**

E Complete proofs for fresh-mask SCAPE **17**

E.1 Distributed construction and filtration 18

E.2 Oracle bounds and adaptive assumptions 19

E.3 A sufficient condition for fresh-mask contraction 20

E.4 Error feedback and the virtual recursion 21

E.5 Adaptive finite-time stationarity 22

E.6 A sharper result with fixed preconditioning 23

E.7 Scope and the delayed-mask implementation 24

F Model Architecture and Hyperparameters **24**

F.1 GPT-345M 24

F.2 Llama-500M 24

F.3 LEGACY 25

F.4 DGC 25

F.5 DeMo 25

G Additional Pretraining Results **25**

H Additional Ablation Results **25**

I Downstream Task Performance **25**

J Per-Step Time **27**

K Strong Scaling Efficiency **28**

L Memory Consumption **28**

M Communication Volume **29**

A Distributed Optimizer Training

In distributed data parallel (DDP) training, each worker (e.g., a GPU) in a data parallel (DP) group duplicates all the model parameters, gradient, and optimizer states, and has a split of a micro-batch of training samples. After all workers finish computing their gradients using the local samples, they synchronize the gradient with a ring *all-reduce* operation before updating the optimizer states and model parameters. Assume there are N workers and the gradient size is W (same as the model size). This communication cost can be decomposed into a *reduce-scatter* following an *all-gather* operation, which is modeled as Equation 1, where α is the fixed latency to initialize a message, and β is the inverse of bandwidth.

Duplicating the model and optimizer states for all workers in a DP group guarantees that the forward and backward computation can finish without communication. However, optimizer states usually require full precision (FP32) to guarantee training stability, which becomes a main memory bottleneck in training LLMs with increasingly more parameters. Many frameworks have been proposed to reduce this bottleneck. For instance, Megatron-LM with sharded DP distributed optimizers partitions the optimizer states and the model parameters evenly among all DP workers. Thus, before the forward pass, workers need to perform *all-gather* to collect the model shards. Note that the model parameters can be prefetched to overlap the communication of *all-gather* with computation. After the backward pass finishes, each DP worker has its local gradient. Since the optimizer states have been partitioned evenly, gradients' *all-reduce* synchronization can be replaced by *reduce-scatter* and does not require gathering. This process is illustrated in Figure 9.

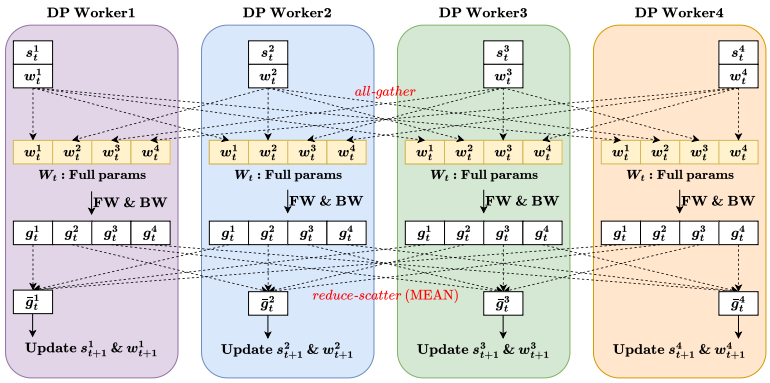


Figure 9: Megatron-LM with sharded data parallel distributed optimizer.

B Related Work

Sparsification. DGC (Lin et al., 2018) proposes a momentum-based sparsification method to reduce the communication volume of synchronizing gradient in DDP. DeMo (Peng et al., 2026) proposes a decoupled momentum optimization method which first performs DCT to orthonormalize the momentum and then uses top-k sparsification with error feedback to reduce communication volume. Ok-Top$k$ (Li & Hoefler, 2022) proposes a novel sparse *all-reduce* algorithm to achieve near asymptotically optimal communication volume. EDGC (Yi et al., 2025) proposes an entropy-driven adaptive gradient sparsification framework to dynamically specify the sparsity rate for each model layer. Radius (Zheng & Zhang, 2025) exploits the temporal stability of the top-k gradient values' indices to amortize the computation cost of top-k operations, and thus achieves lower computation overhead in gradient sparsification and higher throughput. LEGACY (Essoullami et al., 2026) applied different sparsity for layers with different sizes to achieve the desired compression rate.

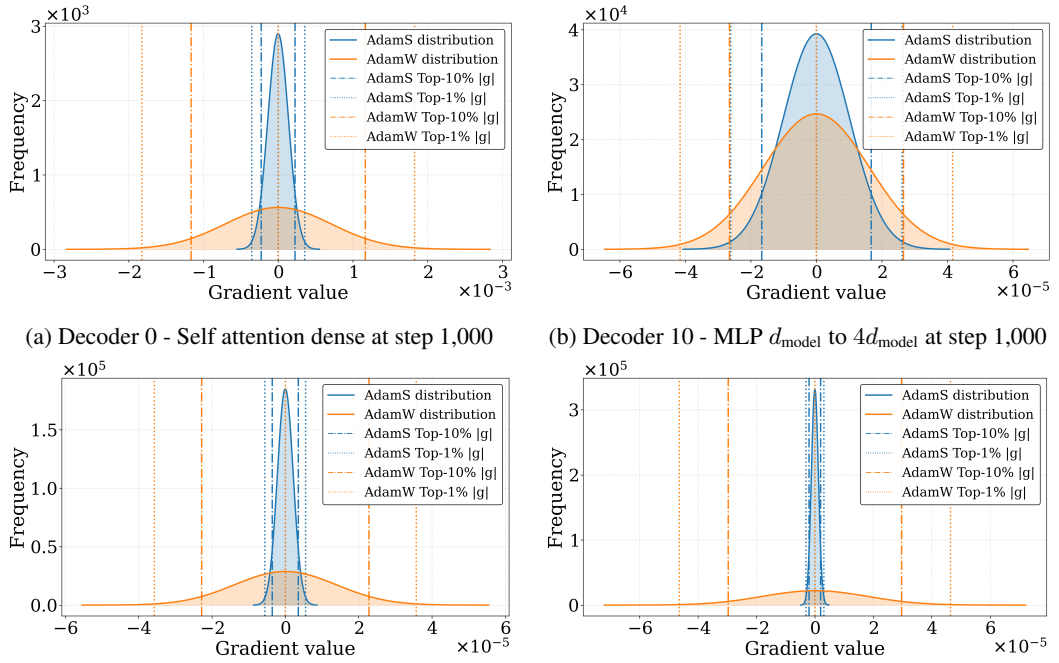
Low-Rank Compression. ATOMO (Wang et al., 2018) proposes to use singular value decomposition (SVD) to express gradients as atomic components and transmits only a subset of them to reduce communication overhead. PowerSGD (Vogels et al., 2019) proposes a low-rank approximation of the gradient by representing it with two much smaller factor matrices. Optimus-CC (Song et al., 2023) proposes a novel framework, combining 3D parallel training with PowerSGD.

Quantization. QSDP (Markov et al., 2023) proposes to extend FSDP (Zhao et al., 2023) with both model weight and gradient quantization to reduce the communication cost with convergence guaranteed. ZeRO++ (Wang et al., 2023) improves the communication efficiency of ZeRO (Rajbhandari et al., 2020) through the combination of block-wise quantized *all-gather*, communication-aware data remapping, and quantized gradient averaging built on *all-to-all* communication SDP4Bit (Jia et al., 2024) proposes to use 4-bit quantization to reduce both weight and gradient communication via quantizing the weight differences and a two-level smooth quantization scheme for gradients, while also introducing runtime optimizations to mitigate quantization overhead.

C Observations

C.1 Gradient Distribution

We show more gradient distribution plots for different layers in GPT-345M and Llama-500M in Figure 10 and Figure 11.



(a) Decoder 0 - Self attention dense at step 1,000

(b) Decoder 10 - MLP d_{model} to $4d_{\text{model}}$ at step 1,000

(c) Decoder 0 - Self attention dense at step 50,000

(d) Decoder 2 - QKV projection weight at step 50,000

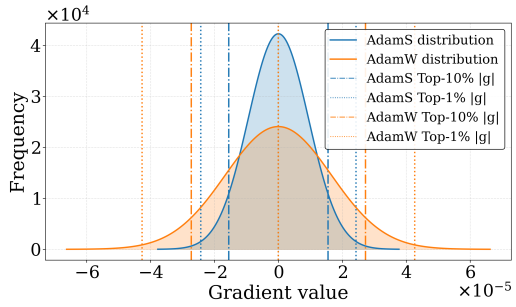
Figure 10: Gradient distribution of different layers in GPT-345M at different steps.

C.2 Temporal Stability of First-Moment-Derived Top-k Masks

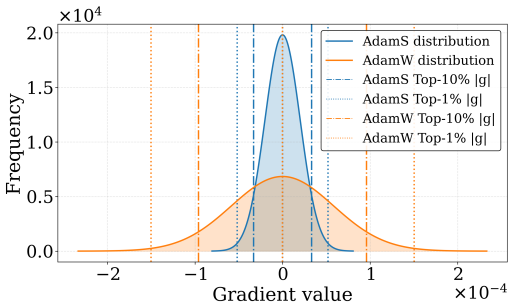
In addition to the overlap rate shown in Figure 3, more can be found in Figure 12.

D SCAPE with Distributed Optimizer

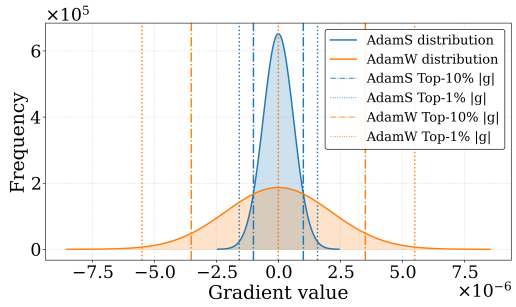
The pseudocode for SCAPE with distributed optimizer is in Algorithm 2. The EF buffer in Algorithm 1 is decomposed into two parts: the common residual $e_t^{m,n}$ retaining the unselected contribution from the shared momentum history and the worker-specific residual $e_t^{g,n}$ containing the unselected contribution from worker n's gradients. Algorithm 2 has three collective communication: *all-gather* for gathering the union of each worker's local top-k mask, *all-reduce-mean* for synchronizing the compressed gradient, and *all-reduce-sum* for synchronizing the updated model parameters (sparsified by the common mask M_{t-1}).



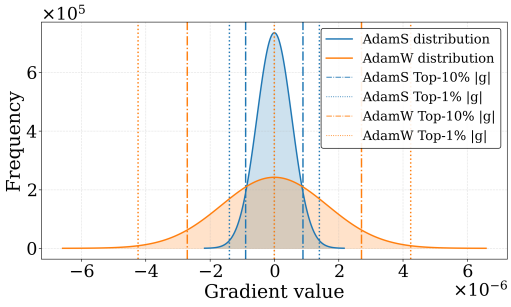
(a) Decoder 0 - gate and up project in SwiGLU at step 1,000



(b) Decoder 8 - Self attention dense at step 1,000

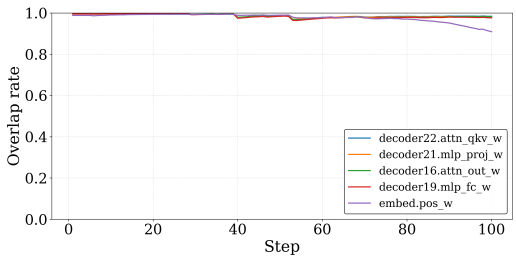


(c) Decoder 0 - gate and up project in SwiGLU at step 50,000

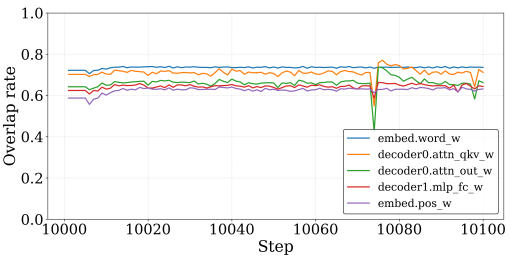


(d) Word embedding at step 50,000

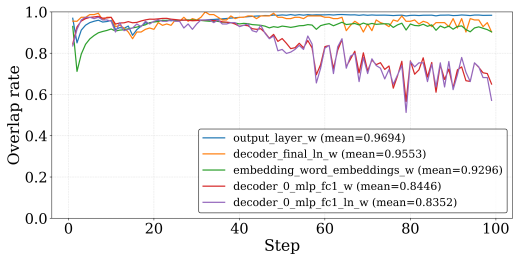
Figure 11: Gradient distribution of different layers in Llama-500M at different steps.



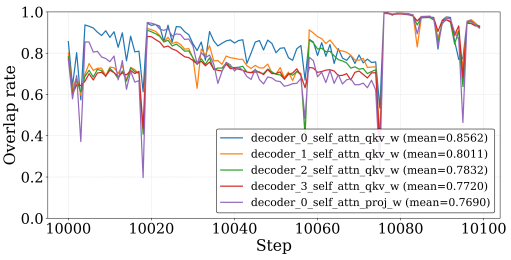
(a) GPT-345M from step 1



(b) GPT-345M from step 10,000



(c) Llama-500M from step 1



(d) Llama-500M from step 10,000

Figure 12: The overlap rate of AdamS's first-moment-derived top-10% mask between two adjacent steps for different stages of pre-training GPT-345M and Llama-500M.

E Complete proofs for fresh-mask SCAPE

We prove the conditional statements in Section 5 for an idealized fresh-mask variant: the selectors form a mask from the current proposals, synchronize it, and apply it to those same proposals.

Algorithm 2 SCAPE (Distributed Optimizer)

Require: initial model parameters θ_1, dataset $\mathcal{D}$, loss function f, worker rank n, parameter shard S_n owned by worker n, number of workers N, density d, weight decay λ, learning rate η, maximum gradient norm C, coefficients β_1, β_2, numerical stabilizer ϵ.

$e_0^{g,n}, m_0^n, e_0^{m,n} \leftarrow 0, \quad M_0 \leftarrow 1$
for $t = 1, 2, \ldots$ **do**
  $g_t^n \leftarrow \nabla f(\mathcal{D}_t^n, \theta_t)$
  $z_t^n \leftarrow (1-\beta_1) g_t^n + e_{t-1}^{g,n}$
  $w_t^n \leftarrow \beta_1 m_{t-1}^n + e_{t-1}^{m,n}$
  $h_t^n[S_n] \leftarrow z_t^n[S_n] + w_t^n$
  $M_t \leftarrow \textsc{RefreshMask}(d_{t+1}, |h_t^n[S_n]|)$
  $\chi_t \leftarrow M_{t-1}[S_n]$
  $e_t^{g,n} \leftarrow (1 - M_{t-1}) \odot z_t^n$
  $\bar{c}_t \leftarrow \textsc{MeanAllReduce}(M_{t-1} \odot z_t^n)$
  $\bar{b}_t[S_n] \leftarrow \bar{c}_t[S_n] + \chi_t \odot w_t^n$
  $e_t^{m,n} \leftarrow (1 - \chi_t) \odot w_t^n$
  $\widetilde{r}_t^n \leftarrow \chi_t \odot (\bar{b}_t[S_n] - \beta_1 m_{t-1}^n)/(1-\beta_1)$
  $r_t^n \leftarrow \mathrm{GlobalClip}_C(\widetilde{r}_t^n)$
  $m_t^n \leftarrow \chi_t \odot [\beta_1 m_{t-1}^n + (1-\beta_1) r_t^n]$
  $v_t^n \leftarrow \beta_2 (m_{t-1}^n)^{\odot 2} + (1-\beta_2)(r_t^n)^{\odot 2}$
  $\theta_{t+1}[\neg M_{t-1}] \leftarrow (1 - \eta\lambda)\theta_t[\neg M_{t-1}]$ ▷ Local decay of unselected parameters
  $p_{t,i}^n \leftarrow 0 \quad$ **if** $i \notin S_n$ **else** $(1-\eta\lambda)\theta_{t,i} - \eta (m_t^n)_i / (\sqrt{(v_t^n)_i} + \epsilon)$
  $\theta_{t+1}[M_{t-1}] \leftarrow \textsc{AllReduceSum}(p_t^n)$ ▷ Synchronize owners' updated values
end for

Throughout this appendix, gradient clipping is inactive, weight decay is zero, the learning rate is constant within a run, and block densities are fixed. The update is uncorrected and uses exact arithmetic and exact collective communication.

E.1 Distributed construction and filtration

Partition the coordinates into disjoint blocks $\ell = 1, \ldots, K$. Block ℓ has dimension W_ℓ, density $d_\ell \in (0, 1]$, and a designated selector $j(\ell) \in \{1, \ldots, N\}$. Start from a deterministic θ_1, fix $\beta_1, \beta_2 \in [0, 1)$ and $\epsilon, \eta > 0$, and initialize $m_0 = 0$ and $e_0^n = 0$ for all workers. At step t, worker n forms

$$g_t^n = \nabla f_n(\theta_t; \xi_t^n), \qquad h_t^n = \beta_1 m_{t-1} + (1-\beta_1) g_t^n + e_{t-1}^n. \tag{7}$$

The selector for block ℓ constructs

$$k_\ell = \lceil d_\ell W_\ell \rceil, \qquad P_t^{(\ell)} = \mathrm{Diag}\Big(\mathrm{TopK}_{k_\ell}(|h_t^{j(\ell),(\ell)}|)\Big),$$

where $\mathrm{TopK}_k(|x|)$ is the binary indicator of the k largest coordinate magnitudes, with deterministic tie-breaking. The selectors all-gather their disjoint mask shards before compression, giving every worker the same orthogonal coordinate projection

$$P_t = \mathrm{Diag}\big(P_t^{(1)}, \ldots, P_t^{(K)}\big), \qquad P_t = P_t^\top = P_t^2.$$

Equivalently, $P_t = \mathrm{Diag}(M_t)$, where the binary vector M_t is the fresh common mask; Algorithm 1 applies the delayed mask M_{t-1} instead. The transmitted values, local residuals, and averaged transmitted vector are

$$b_t^n = P_t h_t^n, \qquad e_t^n = (I - P_t) h_t^n, \qquad \bar{b}_t = \frac{1}{N} \sum_{n=1}^N b_t^n.$$

With clipping inactive, the supportwise reconstruction and moment update are

$$r_t = P_t \frac{\bar{b}_t - \beta_1 m_{t-1}}{1-\beta_1}, \qquad m_t = P_t \big[\beta_1 m_{t-1} + (1-\beta_1) r_t\big] = \bar{b}_t. \tag{8}$$

The last equality follows from $P_t \bar{b}_t = \bar{b}_t$ and $P_t^2 = P_t$. Writing bars for equal worker averages, we therefore have

$$\bar{h}_t = \beta_1 m_{t-1} + (1-\beta_1)\bar{g}_t + \bar{e}_{t-1}, \qquad m_t = P_t \bar{h}_t, \qquad \bar{e}_t = (I - P_t)\bar{h}_t. \tag{9}$$

Applying P_t to the first identity gives the exact reconstruction

$$r_t = P_t \frac{m_t - \beta_1 m_{t-1}}{1-\beta_1} = P_t\left(\bar{g}_t + \frac{\bar{e}_{t-1}}{1-\beta_1}\right). \tag{10}$$

Thus reconstruction is an identity on the selected support. Both the projection and the residual factor $1/(1-\beta_1)$ are retained. The adaptive scaling and parameter update are

$$\begin{aligned} v_t &= \beta_2 m_{t-1}^{\odot 2} + (1-\beta_2) r_t^{\odot 2}, \qquad H_t = \mathrm{Diag}\big((\sqrt{v_t}+\epsilon)^{-1}\big), \\ \theta_{t+1} &= \theta_t - \eta H_t m_t, \end{aligned} \tag{11}$$

where the operations defining v_t and H_t are coordinatewise.

Let $\mathcal{F}_t$ contain the initialization and all samples, masks, communicated values, residuals, and optimizer states strictly before step t, and write $\mathbb{E}_t[\cdot] = \mathbb{E}[\cdot \mid \mathcal{F}_t]$. Then θ_t, m_{t-1}, $\bar{e}_{t-1}$, and $s_{t-1} = \beta_1 m_{t-1} + \bar{e}_{t-1}$ are $\mathcal{F}_t$-measurable. The current samples, fresh projection P_t, reconstruction r_t, and scaling H_t need not be. We set $H_0 = H_1$ only as an analytical convention. Although H_1 may depend on the first samples, $H_0 s_0 = 0$, so the virtual iterate

$$y_t = \theta_t - \frac{\eta}{1-\beta_1} H_{t-1} s_{t-1}$$

satisfies $y_1 = \theta_1$ and is $\mathcal{F}_t$-measurable for every t.

E.2 Oracle bounds and adaptive assumptions

Assumption E.1 (Objective regularity)**.** *The objective $F(\theta) = N^{-1}\sum_{n=1}^N F_n(\theta)$ is bounded below by F_* and has an L-Lipschitz gradient for some $L > 0$.*

Assumption E.2 (Distributed stochastic oracle)**.** *Conditionally on $\mathcal{F}_t$, local gradients are independent across workers and satisfy $\mathbb{E}_t[g_t^n] = \nabla F_n(\theta_t)$, $\mathbb{E}_t\|g_t^n - \nabla F_n(\theta_t)\|^2 \le \sigma_n^2$, and $\|g_t^n\| \le G$ almost surely. Define $\bar{\sigma}^2 = N^{-1}\sum_n \sigma_n^2$. Consequently, $\mathbb{E}_t[\bar{g}_t] = \nabla F(\theta_t)$ and $\mathbb{E}_t\|\bar{g}_t - \nabla F(\theta_t)\|^2 \le \bar{\sigma}^2/N$.*

Assumption E.3 (Common-mask contraction)**.** *For a fixed $q \in [0,1)$ and every t, $\|(I-P_t)\bar{h}_t\| \le q\|\bar{h}_t\|$ almost surely. The condition concerns the averaged proposal under the assembled fresh mask. Distributed local Top-k selection alone does not imply it; Lemma E.5 gives a sufficient agreement condition.*

Assumption E.4 (Adaptive alignment)**.** *There are constants $h_+ > 0$ and $\mu > 0$ such that, for every t, $0 \prec H_t \preceq h_+ I$ almost surely and $\mathbb{E}_t\langle \nabla F(\theta_t), H_t \bar{g}_t\rangle \ge \mu\|\nabla F(\theta_t)\|^2$ almost surely.*

Assumption E.1 supplies an L-smooth objective $F = N^{-1}\sum_n F_n$ bounded below by F_*. Write $a_t = \nabla F(\theta_t)$ and $\bar{\sigma}^2 = N^{-1}\sum_n \sigma_n^2$. Under Assumption E.2, the centered local errors $u_t^n = g_t^n - \nabla F_n(\theta_t)$ have zero conditional means and are conditionally independent. Their conditional cross terms vanish, so

$$\begin{aligned} \mathbb{E}_t \bar{g}_t &= a_t, \\ \mathbb{E}_t\|\bar{g}_t - a_t\|^2 &= \frac{1}{N^2}\sum_{n=1}^N \mathbb{E}_t\|u_t^n\|^2 \le \frac{\bar{\sigma}^2}{N}, \\ \mathbb{E}_t\|\bar{g}_t\|^2 &= \|a_t\|^2 + \mathbb{E}_t\|\bar{g}_t - a_t\|^2 \le \|a_t\|^2 + \frac{\bar{\sigma}^2}{N}. \end{aligned} \tag{12}$$

The almost-sure local bound also gives $\|\bar{g}_t\| \le G$ and, by conditional Jensen's inequality, $\|a_t\| \le G$.

The spectral bound $0 \prec H_t \preceq h_+ I$ in Assumption E.4 implies, pathwise, $\|H_t\bar{g}_t\|^2 \le h_+^2\|\bar{g}_t\|^2$. Consequently,

$$\mathbb{E}_t\|H_t\bar{g}_t\|^2 \le h_+^2\|a_t\|^2 + \frac{h_+^2\bar{\sigma}^2}{N}. \tag{13}$$

This bound requires no independence between H_t and the current samples. For the AdamS formula in Equation 11, one may take $h_+ = \epsilon^{-1}$. This worst-case bound makes the theorem's stepsize restriction conservative; it is not a calibrated guide to the experimental learning rates. The additional alignment condition

$$\mathbb{E}_t\langle a_t, H_t\bar{g}_t\rangle \ge \mu\|a_t\|^2, \qquad \mu > 0,$$

is assumed in the adaptive result. Unbiasedness of $\bar{g}_t$ does not establish it for a matrix H_t depending on the same samples. Fresh mask selection does not establish this condition either.

Assumption E.3 requires contraction of the averaged proposal, $\|(I - P_t)\bar{h}_t\| \le q\|\bar{h}_t\|$ with $q < 1$. Since each selector ranks a local proposal, fresh blockwise Top-k selection alone does not imply this aggregate condition. The next lemma provides one sufficient condition.

E.3 A SUFFICIENT CONDITION FOR FRESH-MASK CONTRACTION

Lemma E.5 (Local-to-average agreement). *Let $\kappa_\ell = \sqrt{1 - d_\ell}$. Suppose there are constants $\delta_\ell \ge 0$ such that, for every t and every block with $k_\ell < W_\ell$,*

$$\|h_t^{j(\ell),(\ell)} - \bar{h}_t^{(\ell)}\| \le \delta_\ell \|\bar{h}_t^{(\ell)}\| \quad \textit{almost surely.}$$

Define

$$q_\ell = \begin{cases} 0, & k_\ell = W_\ell, \\ \kappa_\ell + (1 + \kappa_\ell)\delta_\ell, & k_\ell < W_\ell, \end{cases} \qquad q = \max_\ell q_\ell.$$

Then $\|(I - P_t)\bar{h}_t\| \le q\|\bar{h}_t\|$. In particular, Assumption E.3 holds if every non-full-density block obeys

$$\delta_\ell < \frac{1 - \kappa_\ell}{1 + \kappa_\ell}.$$

Full-density blocks have exact contraction factor zero and need no agreement condition.

Proof. Fix a block and abbreviate $x = h_t^{j(\ell),(\ell)}$ and $z = \bar{h}_t^{(\ell)}$. If $k_\ell = W_\ell$, then $P_t^{(\ell)} = I$ and the claim is immediate. Otherwise, the discarded squared coordinates are the $W_\ell - k_\ell$ smallest ones, so

$$\|(I - P_t^{(\ell)})x\|^2 \le \left(1 - \frac{k_\ell}{W_\ell}\right)\|x\|^2 \le (1 - d_\ell)\|x\|^2.$$

Using the triangle inequality and $\|I - P_t^{(\ell)}\|_{\mathrm{op}} \le 1$ gives

$$\begin{aligned} \|(I - P_t^{(\ell)})z\| &\le \|(I - P_t^{(\ell)})x\| + \|(I - P_t^{(\ell)})(z - x)\| \\ &\le \kappa_\ell\|x\| + \|z - x\| \\ &\le \kappa_\ell\|z\| + (1 + \kappa_\ell)\|x - z\| \\ &\le \big[\kappa_\ell + (1 + \kappa_\ell)\delta_\ell\big]\|z\|. \end{aligned}$$

The squared block norms add over the disjoint partition, hence

$$\|(I - P_t)\bar{h}_t\|^2 \le \sum_\ell q_\ell^2 \|\bar{h}_t^{(\ell)}\|^2 \le q^2\|\bar{h}_t\|^2.$$

For a non-full-density block, the stated strict threshold is equivalent to $q_\ell < 1$. □

The agreement condition is sufficient, not necessary. It holds with $\delta_\ell = 0$ for one worker, or whenever the selector's current proposal equals the current average in each block. At density $d_\ell = 0.01$, its strict threshold is

$$\delta_\ell < \frac{1 - \sqrt{0.99}}{1 + \sqrt{0.99}} \approx 0.00251.$$

This requires a relative selector-to-average discrepancy below about 0.25% in every non-full-density block at every step, almost surely. We have not established this condition empirically for our distributed training runs. Mask overlap and final loss do not verify it. In contrast, local Top-k selection without agreement can discard all of the average's energy: with two workers, one block, proposals $(2, 1)$ from worker 1 and $(-2, 1)$ from worker 2, and worker 1 as selector, the fresh Top-1 mask retains the first coordinate while $\bar{h}_t = (0, 1)$. Thus fresh selection still requires an aggregate contraction argument in the distributed setting.

E.4 Error feedback and the virtual recursion

Definition E.6 (Preconditioner variation)**.** Let $s_t = \beta_1 m_t + \bar{e}_t$, set $H_0 = H_1$, and define $\zeta_t = \|(H_t - H_{t-1})s_{t-1}\|$, $V_{1,T} = T^{-1}\sum_{t=1}^{T}\mathbb{E}\zeta_t$, and $V_{2,T} = T^{-1}\sum_{t=1}^{T}\mathbb{E}\zeta_t^2$.

This definition imposes no decay. Adaptive alignment controls the expected descent direction within a step; $V_{1,T}$ measures changes of the preconditioner acting on the accumulated state across steps.

Lemma E.7 (Bounded momentum–error-feedback state)**.** *Under Assumptions E.2 and E.3, $m_t + \bar{e}_t = \beta_1 m_{t-1} + (1-\beta_1)\bar{g}_t + \bar{e}_{t-1}$. Let $\tau^2 = \beta_1^2 + (1-\beta_1^2)q^2$ and $B_q = \tau(1-\beta_1)G/(1-\tau)$. Then $\tau < 1$ and, pathwise, $\|s_t\|^2 = \beta_1^2\|m_t\|^2 + \|\bar{e}_t\|^2 \le \tau^2\|\bar{h}_t\|^2$ and $\|s_t\| \le B_q$ for every t.*

Proof of Lemma E.7. By Equation 9,

$$m_t + \bar{e}_t = \bar{h}_t = \beta_1 m_{t-1} + (1-\beta_1)\bar{g}_t + \bar{e}_{t-1} = s_{t-1} + (1-\beta_1)\bar{g}_t. \tag{14}$$

This identity is exact before any contraction estimate is applied. Since P_t is an orthogonal coordinate projection, $m_t = P_t\bar{h}_t$ and $\bar{e}_t = (I - P_t)\bar{h}_t$ have disjoint supports. Therefore

$$\begin{aligned}\|s_t\|^2 &= \beta_1^2\|m_t\|^2 + \|\bar{e}_t\|^2 \\ &= \beta_1^2\|\bar{h}_t\|^2 + (1-\beta_1^2)\|(I-P_t)\bar{h}_t\|^2 \\ &\le \left[\beta_1^2 + (1-\beta_1^2)q^2\right]\|\bar{h}_t\|^2 = \tau^2\|\bar{h}_t\|^2.\end{aligned}$$

Because $1 - \tau^2 = (1-\beta_1^2)(1-q^2) > 0$, we have $\tau < 1$. Combining this estimate with Equation 14 and $\|\bar{g}_t\| \le G$ yields

$$\|s_t\| \le \tau\|s_{t-1}\| + \tau(1-\beta_1)G.$$

Starting from $s_0 = 0$ and summing the geometric series gives, pathwise,

$$\|s_t\| \le \frac{\tau(1-\beta_1)(1-\tau^t)}{1-\tau}G \le \frac{\tau(1-\beta_1)}{1-\tau}G = B_q.$$

In particular, $\mathbb{E}\|s_t\|^2 \le B_q^2$. □

Lemma E.8 (Exact virtual-iterate recursion)**.** *With $y_t = \theta_t - \eta H_{t-1}s_{t-1}/(1-\beta_1)$,*

$$m_t = \bar{g}_t + \frac{s_{t-1} - s_t}{1-\beta_1}, \qquad y_{t+1} = y_t - \eta H_t\bar{g}_t + \frac{\eta}{1-\beta_1}(H_{t-1} - H_t)s_{t-1}. \tag{15}$$

Proof of Lemma E.8. The relation $m_t + \bar{e}_t = s_t + (1-\beta_1)m_t$ and Equation 14 imply

$$m_t = \bar{g}_t + \frac{s_{t-1} - s_t}{1-\beta_1}.$$

Substituting this identity into the parameter update gives

$$\begin{aligned}y_{t+1} &= \theta_{t+1} - \frac{\eta}{1-\beta_1}H_t s_t \\ &= \theta_t - \eta H_t\bar{g}_t - \frac{\eta}{1-\beta_1}H_t s_{t-1} \\ &= y_t - \eta H_t\bar{g}_t + \frac{\eta}{1-\beta_1}(H_{t-1} - H_t)s_{t-1}.\end{aligned}$$

Thus the virtual recursion is exact. When H_t is fixed, its last term vanishes. □

For the variation budgets in Definition E.6, the spectral bound and Lemma E.7 imply

$$\zeta_t = \|(H_t - H_{t-1})s_{t-1}\| \le 2h_+B_q, \qquad V_{2,T} \le 4h_+^2B_q^2. \tag{16}$$

Hence a uniform bound on the second variation budget follows from the existing assumptions. Decay of the first budget remains an additional condition for the rate theorem; it follows neither from oracle unbiasedness nor from choosing a smaller learning rate as T increases.

E.5 ADAPTIVE FINITE-TIME STATIONARITY

Theorem E.9 (Finite-time stationarity of fresh-mask SCAPE)**.** *Suppose Assumptions E.1–E.4 hold. Let* $\Delta = F(\theta_1) - F_*$, $D_\eta = \eta h_+ B_q/(1-\beta_1)$, *and* $\mathcal{G}_T = T^{-1}\sum_{t=1}^{T}\mathbb{E}\|\nabla F(\theta_t)\|^2$. *If* $\eta \le \mu/(2Lh_+^2)$, *then*

$$\begin{aligned}\mathcal{G}_T \le \frac{2\Delta}{\mu\eta T} + \frac{2L\eta h_+^2\bar{\sigma}^2}{\mu N} + \frac{2L\eta h_+^2 B_q G}{\mu(1-\beta_1)} \\ + \frac{2(G+LD_\eta)V_{1,T}}{\mu(1-\beta_1)} + \frac{2L\eta V_{2,T}}{\mu(1-\beta_1)^2}.\end{aligned} \tag{17}$$

Proof of Theorem E.9. Set

$$a_t = \nabla F(\theta_t), \qquad \gamma_t = \nabla F(y_t) - a_t, \qquad \omega_t = (H_{t-1} - H_t)s_{t-1}.$$

The state bound and smoothness give

$$\|y_t - \theta_t\| \le \frac{\eta h_+ B_q}{1-\beta_1} = D_\eta, \qquad \|\gamma_t\| \le LD_\eta. \tag{18}$$

Also $\|a_t\| \le G$, $\|H_t\bar{g}_t\| \le h_+ G$, and $\|\omega_t\| = \zeta_t$. By Lemma E.8,

$$y_{t+1} - y_t = -\eta H_t\bar{g}_t + \frac{\eta}{1-\beta_1}\omega_t.$$

Smoothness and $\|x+z\|^2 \le 2\|x\|^2 + 2\|z\|^2$ imply

$$\begin{aligned}F(y_{t+1}) \le F(y_t) - \eta\langle a_t, H_t\bar{g}_t\rangle - \eta\langle\gamma_t, H_t\bar{g}_t\rangle \\ + \frac{\eta}{1-\beta_1}\langle\nabla F(y_t), \omega_t\rangle + L\eta^2\|H_t\bar{g}_t\|^2 + \frac{L\eta^2}{(1-\beta_1)^2}\zeta_t^2.\end{aligned}$$

The remaining inner products admit the pathwise estimates

$$-\langle\gamma_t, H_t\bar{g}_t\rangle \le LD_\eta h_+ G, \qquad \langle\nabla F(y_t), \omega_t\rangle \le (G+LD_\eta)\zeta_t.$$

Using the conditional alignment assumption and Equation 13, then taking total expectations, gives

$$\begin{aligned}\mathbb{E}F(y_{t+1}) \le \mathbb{E}F(y_t) - (\mu\eta - L\eta^2 h_+^2)\mathbb{E}\|a_t\|^2 + \frac{L\eta^2 h_+^2\bar{\sigma}^2}{N} \\ + \eta LD_\eta h_+ G + \frac{\eta(G+LD_\eta)}{1-\beta_1}\mathbb{E}\zeta_t + \frac{L\eta^2}{(1-\beta_1)^2}\mathbb{E}\zeta_t^2.\end{aligned} \tag{19}$$

The stepsize condition $\eta \le \mu/(2Lh_+^2)$ gives $\mu\eta - L\eta^2 h_+^2 \ge \mu\eta/2$. Since $y_1 = \theta_1$ and $F(y_{T+1}) \ge F_*$, summing Equation 19 from $t=1$ to T and dividing by $\mu\eta T/2$ yields

$$\begin{aligned}\mathcal{G}_T \le \frac{2\Delta}{\mu\eta T} + \frac{2L\eta h_+^2\bar{\sigma}^2}{\mu N} + \frac{2L\eta h_+^2 B_q G}{\mu(1-\beta_1)} \\ + \frac{2(G+LD_\eta)V_{1,T}}{\mu(1-\beta_1)} + \frac{2L\eta V_{2,T}}{\mu(1-\beta_1)^2},\end{aligned} \tag{20}$$

where $\Delta = F(\theta_1) - F_*$ and $\mathcal{G}_T = T^{-1}\sum_{t=1}^{T}\mathbb{E}\|\nabla F(\theta_t)\|^2$. The third term uses $\eta LD_\eta h_+ G = L\eta^2 h_+^2 B_q G/(1-\beta_1)$. At no step do we move a current-sample-dependent H_t outside a conditional expectation. □

Proof of Theorem 5.1. With $L, \Delta, \mu, h_+, \beta_1, q, G, N$, and $\bar{\sigma}^2$ independent of T, the state bound gives $B_q = O(1)$, and Equation 16 gives $V_{2,T} = O(1)$. A constant stepsize within each run chosen as $\eta = \Theta(T^{-1/2})$ gives $D_\eta = O(T^{-1/2})$. The five terms of Equation 20 then have orders

$$O((\eta T)^{-1}), \quad O(\eta), \quad O(\eta), \quad O((G+LD_\eta)V_{1,T}), \quad O(\eta V_{2,T}).$$

Under the additional condition $V_{1,T} = O(T^{-1/2})$, all five terms are $O(T^{-1/2})$. This proves the claimed rate for $\mathcal{G}_T$. If R is independent of the run and uniform on $\{1, \dots, T\}$, then

$$\mathbb{E}\|\nabla F(\theta_R)\|^2 = \mathcal{G}_T = O(T^{-1/2}), \qquad \mathbb{E}\|\nabla F(\theta_R)\| \le \sqrt{\mathcal{G}_T} = O(T^{-1/4}).$$

These conclusions concern stationarity of a uniform random iterate and do not assert convergence of the parameter sequence or of the last iterate. □

E.6 A SHARPER RESULT WITH FIXED PRECONDITIONING

A fixed deterministic preconditioner eliminates the need to assume adaptive alignment or drift decay. The objective, oracle, and aggregate contraction assumptions remain in force for the result below.

Proposition E.10 (Fixed deterministic preconditioner)**.** *Suppose Assumptions E.1–E.3 hold, and replace the adaptive update by $\theta_{t+1} = \theta_t - \eta H m_t$, where H is a fixed deterministic symmetric positive-definite matrix with $h_- I \preceq H \preceq h_+ I$ and $0 < h_- \le h_+ < \infty$. Define*

$$\Delta = F(\theta_1) - F_*, \qquad D_\eta = \frac{\eta h_+ \tau G}{1-\tau}.$$

If $\eta \le h_-/(2Lh_+^2)$, then

$$\mathcal{G}_T \le \frac{2\Delta}{\eta h_- T} + \frac{L\eta h_+^2 \bar{\sigma}^2}{N h_-} + \frac{2L^2 h_+^2 D_\eta^2}{h_-^2}. \tag{21}$$

In particular, if $H = I$, $\Delta > 0$, $\bar{\sigma}^2 > 0$, and

$$\eta = \sqrt{\frac{2\Delta N}{L\bar{\sigma}^2 T}}, \qquad T \ge \frac{8\Delta N L}{\bar{\sigma}^2},$$

then

$$\mathcal{G}_T \le 2\sqrt{\frac{2\Delta L \bar{\sigma}^2}{NT}} + \frac{4\Delta L N \tau^2 G^2}{\bar{\sigma}^2 (1-\tau)^2 T}. \tag{22}$$

Proof. With $H_t = H$, the virtual recursion reduces to

$$y_{t+1} = y_t - \eta H \bar{g}_t, \qquad y_t = \theta_t - \frac{\eta}{1-\beta_1} H s_{t-1}.$$

The state bound gives $\|y_t - \theta_t\| \le \eta h_+ B_q/(1-\beta_1) = D_\eta$. Set $a_t = \nabla F(\theta_t)$ and $\gamma_t = \nabla F(y_t) - a_t$, so $\|\gamma_t\| \le L D_\eta$. All of H, a_t, and γ_t are $\mathcal{F}_t$-measurable. By Equation 12,

$$\begin{aligned}
\mathbb{E}_t \langle \nabla F(y_t), H\bar{g}_t \rangle &= \langle a_t + \gamma_t, H a_t \rangle, \\
\mathbb{E}_t \|H\bar{g}_t\|^2 &= \|H a_t\|^2 + \mathbb{E}_t \|H(\bar{g}_t - a_t)\|^2 \\
&\le h_+^2 \|a_t\|^2 + \frac{h_+^2 \bar{\sigma}^2}{N}.
\end{aligned}$$

Applying smoothness to the virtual update therefore gives

$$\begin{aligned}
\mathbb{E}_t F(y_{t+1}) \le F(y_t) &- \eta h_- \|a_t\|^2 + \eta L h_+ D_\eta \|a_t\| \\
&+ \frac{L\eta^2 h_+^2}{2} \|a_t\|^2 + \frac{L\eta^2 h_+^2 \bar{\sigma}^2}{2N}.
\end{aligned}$$

Young's inequality gives

$$L h_+ D_\eta \|a_t\| \le \frac{h_-}{4} \|a_t\|^2 + \frac{L^2 h_+^2 D_\eta^2}{h_-}.$$

The stepsize condition implies $L\eta h_+^2/2 \le h_-/4$. Hence

$$\mathbb{E}_t F(y_{t+1}) \le F(y_t) - \frac{\eta h_-}{2} \|a_t\|^2 + \frac{\eta L^2 h_+^2 D_\eta^2}{h_-} + \frac{L\eta^2 h_+^2 \bar{\sigma}^2}{2N}.$$

Taking total expectations, telescoping from $y_1 = \theta_1$, using the lower bound F_*, and dividing by $\eta h_- T/2$ proves Equation 21.

For $H = I$, the stated restriction on T ensures $\eta \le 1/(2L)$. The first two terms in Equation 21 each equal $\sqrt{2\Delta L \bar{\sigma}^2/(NT)}$, and

$$2L^2 D_\eta^2 = \frac{4\Delta L N \tau^2 G^2}{\bar{\sigma}^2 (1-\tau)^2 T},$$

which proves Equation 22. This argument uses neither the adaptive alignment assumption nor a preconditioner-drift condition. A fixed H in fact satisfies the alignment inequality with $\mu = h_-$ and has $V_{1,T} = V_{2,T} = 0$, but the direct argument above gives the sharper displacement term of order η^2. □

Corollary E.11 (Fixed-preconditioner comparison)**.** *Under the $H = I$ specialization of Proposition E.10, Equation 22 holds. When the problem-dependent constants are uniform in N and T, this bound is $O((NT)^{-1/2}) + O(N/T)$.*

Proof of Corollary E.11. Apply the $H = I$ specialization of Proposition E.10. When the other factors in Equation 22 are uniform in N and T, its two terms are $O((NT)^{-1/2})$ and $O(N/T)$, respectively. For fixed N, the latter is lower order. □

E.7 Scope and the delayed-mask implementation

Under dense communication, $P_t = I$ gives $q = 0$, $\bar{e}_t = 0$, and $\tau = \beta_1$, so the state $s_t = \beta_1 m_t$ records momentum. If $\beta_1 = 0$, then $s_t = 0$ and the actual and virtual iterates coincide. The results above concern the averaged expected squared-gradient norm. For adaptive AdamS, their rate additionally requires positive conditional alignment and the specified decay of $V_{1,T}$; neither follows from fresh mask selection or from the denominator formula alone.

Remark E.12 (Delayed masks as an empirical observation). Algorithm 1 uses a mask from the preceding step to overlap mask communication with computation. The adjacent-step mask overlap observed in Section 3.2 and Figure 3 and the mask-staleness ablation in Section 6.3 and Figure 8b provide empirical support for this practical choice. The theorem above analyzes fresh masks. With the other idealizations retained, a common delayed coordinate projection still gives disjoint transmitted and residual supports, so both orthogonality and the algebraic error-feedback identities survive. The missing guarantee is contraction of the *current* averaged proposal. To see the distinction, let P_t be a hypothetical fresh projection formed from the current proposals and let $P_{t-1} = \mathrm{Diag}(M_{t-1})$ be the reused projection that Algorithm 1 applies. Then

$$\|(I - P_{t-1})\bar{h}_t\| \le \|(I - P_t)\bar{h}_t\| + \|(P_t - P_{t-1})\bar{h}_t\|. \tag{23}$$

The second term depends on the current proposal's energy on changed coordinates, which support-overlap counts alone do not control. The fresh-selector agreement lemma bounds only the first term under its stated condition. Observed overlap therefore motivates reuse without establishing the contraction needed for a delayed-mask convergence guarantee. Active clipping, weight decay, changing learning rates or densities, and time-dependent bias correction require corresponding changes to the analyzed recursion.

F Model Architecture and Hyperparameters

F.1 GPT-345M

We pre-train GPT-345M model using the same model architecture definition described in (Radford et al., 2019) on OpenWebText dataset (Gokaslan et al., 2019) for 100,000 steps. We set the global batch size to 512 and the sequence length to 1024. As for the hyperparameters of AdamW, AdamS, and SCAPE, we set $(\beta_1, \beta_2) = (0.9, 0.999)$, the peak learning rate $\eta = 1.5 \times 10^{-4}$, the minimum learning rate $\eta_{\text{min}} = 10^{-5}$, the learning rate scheduler to cosine, the learning rate warmup steps to 5,000 (5% of the pre-training steps), the weight decay $\lambda = 0.01$, AdamS $\epsilon_{\text{AdamS}} = 10^{-5}$, and gradient clip to 1. Similar to Llama-500M training, mixed-precision training is used: the model parameters are in BF16, and the gradient is accumulated in FP32. Similarly, we use density warmup to gradually decrease d from 1 to 0.1 or 0.01 within 10,000 steps. The LayerNorm and bias layers in GPT-345M were not compressed.

For the comparison study to LEGACY-L, we use the same model architecture but different hyperparameters. Specifically, we change the total training steps to 15,000 steps with the total training tokens of 7.86B, following the Chinchilla rule, and the learning rate warmup steps is changed to 150 steps (i.e., 1% of the total training steps). Moreover, the learning rate scheduler is changed from cosine decay to constant. For experiment in ablation studies, we use constant learning rate with 5,000 as the warmup steps.

F.2 Llama-500M

We use the same model architecture of H2O-Danube3-500M (Pfeiffer et al., 2024) to define our Llama-500M model and pre-trained it on SlimPajama-6B, a subdataset sampled from SlimPajama (Soboleva

et al., 2023). We used the same tokenizer as Llama-7B (Touvron et al., 2023) from HuggingFace. We pre-train Llama-500M for 100,000 steps and use a global batch size of 1024 and a sequence length of 4096 for each step. The hyperparameters for the optimizers are: peak learning rate $\eta = 3 \times 10^{-4}$, the minimum learning rate $\eta_{\text{min}} = 3 \times 10^{-5}$, cosine learning rate decay, learning rate warmup for 2,000 steps, $(\beta_1, \beta_2) = (0.9, 0.95)$, RMSNorm $\epsilon_{\text{RMS}} = 10^{-5}$, weight decay $\lambda = 0.1$, AdamS $\epsilon_{\text{AdamS}} = 10^{-5}$, and the gradient clip of 1. We use BF16 for parameters and FP32 for the gradient. For SCAPE, we use sparsity warmup to exponentially decrease density d from 1 to 0.1 or 0.01 within 10,000 steps, and we do not compress RMSNorm layers.

F.3 LEGACY

In the comparison study, we use LEGACY-L to represent the LEGACY experiment, which assigns different compression rates to layers of different sizes. To achieve effective 10% density for gradient compression, we set LEGACY-L to follow the formula below

$$d(W_\ell) = \begin{cases} 1.0, & W_\ell < 10^3 \\ 0.205787762, & 10^3 \leq W_\ell < 10^6 \\ 0.102893881, & 10^6 \leq W_\ell < 10^7 \\ 0.082317993, & W_\ell \geq 10^7 \end{cases} \tag{24}$$

Here, $d(W_\ell)$ maps the number of parameters in a layer (w) to its gradient density (d).

F.4 DGC

We follow the DGC in (Lin et al., 2018) to adapt it to AdamS optimizer to compress the communication in gradient synchronization and accumulating the residuals into momentum EF buffer. The density scheduling is shown in Equation 25.

$$d(t) = \begin{cases} 0.630957, & 1 \leq t \leq 750, \\ 0.398107, & 751 \leq t \leq 1500, \\ 0.251189, & 1501 \leq t \leq 2250, \\ 0.158489, & 2251 \leq t \leq 3000, \\ 0.1, & t \geq 3001. \end{cases} \tag{25}$$

F.5 DeMo

We follow the default hyperparameters of DeMo and set the block size to 64×64 and decay factor of 0.999. DeMo relies on *all-gather*, a collective communication whose volume is proportional to the number of workers, for synchronizing the compressed DCT information. Suppose the number of workers in DDP is N and each of them has communication volume of W. The total communication volume can be expressed as $W(N-1)$. Therefore, we use compression top-k to 8, so that it matches the volume as SCAPE under 32-GPU setting.

G Additional Pretraining Results

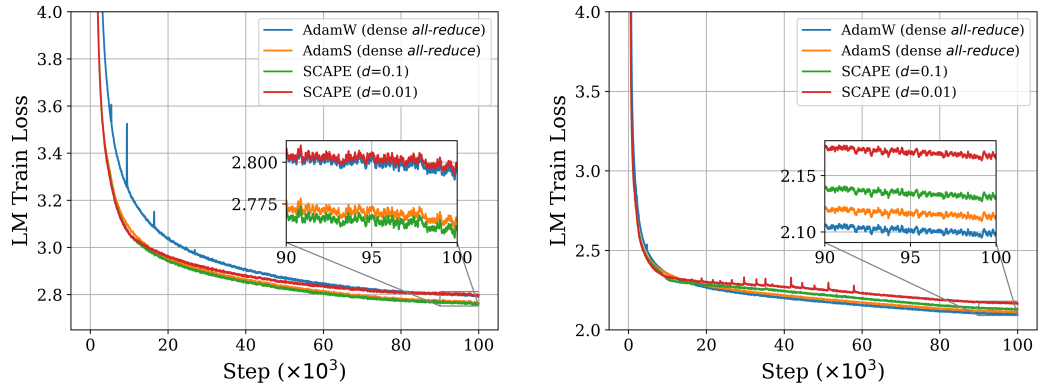
The additional pretraining results for GPT-345M and Llama-500M are shown in Figure 13.

H Additional Ablation Results

The training-loss curves corresponding to the validation-loss ablations in Figure 8 are shown in Figure 14.

I Downstream Task Performance

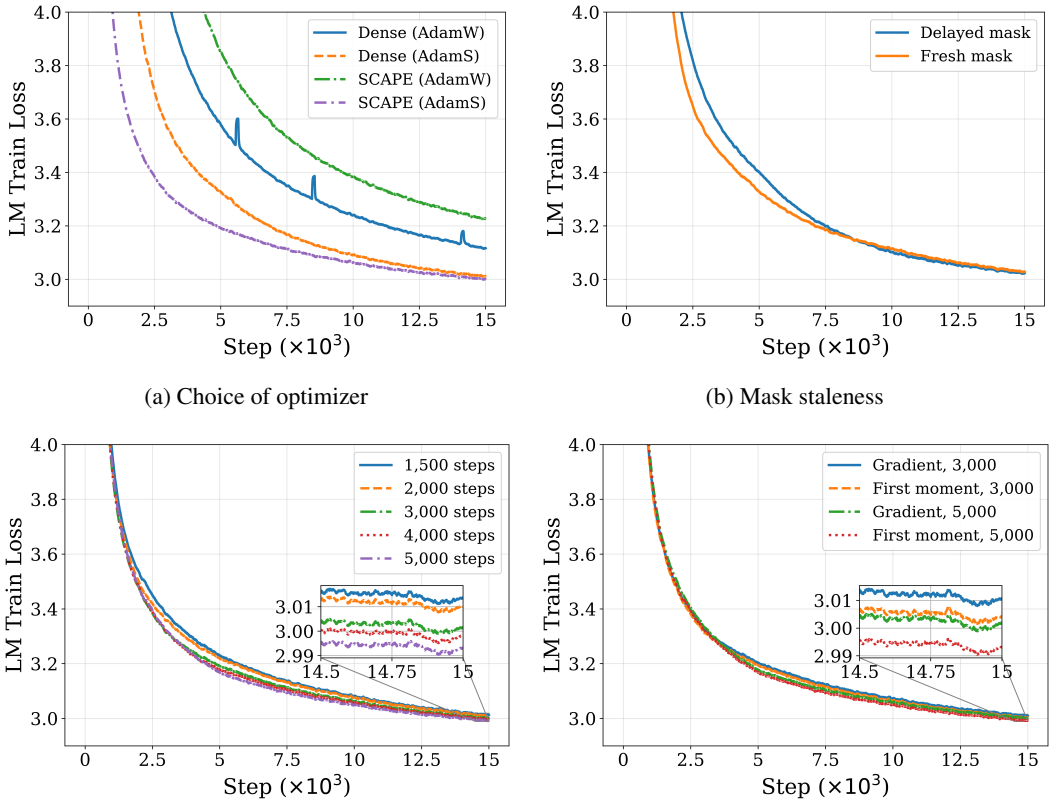
The downstream task scores for GPT-345M and Llama-500M models pretrained using different optimizers are shown in Table 3 and Table 4, respectively, including ARC (Easy and Challenge) (Clark



(a) GPT-345M train loss

(b) Llama-500M train loss

Figure 13: Training-loss curves for pretraining GPT-345M and Llama-500M.



(a) Choice of optimizer

(b) Mask staleness

(c) Sparsity warmup steps

(d) Mask score source

Figure 14: Training-loss curves for the SCAPE ablation studies on GPT-345M.

et al., 2018), LAMBADA (Paperno et al., 2016), HellaSwag (Zellers et al., 2019), MathQA (Amini et al., 2019), MMLU (Hendrycks et al., 2021), PIQA (Bisk et al., 2019), WinoGrande (Sakaguchi et al., 2020), OpenBookQA (Mihaylov et al., 2018), and SuperGLUE (Wang et al., 2019).

Table 3: Zero-shot evaluation results for GPT-345M pretrained using different methods.

Method	ARC-C (ACC)	ARC-E (ACC)	HellaSwag (ACC)	LAMBADA (ACC)	MMLU (ACC)	OpenBookQA (ACC)	PIQA (ACC)	WinoGrande (ACC)
AdamW (dense)	20.39	**48.82**	30.63	39.57	22.88	17.00	64.91	51.38
AdamS (dense)	19.88	47.77	**31.45**	39.51	**22.94**	18.00	**65.18**	**51.54**
SCAPE ($d = 0.1$)	**20.99**	**48.82**	31.28	**39.71**	22.91	17.80	64.80	51.38
SCAPE ($d = 0.01$)	19.71	46.93	31.06	39.06	22.93	**18.60**	64.42	50.67

Method	SuperGLUE								
	BoolQ (ACC)	WiC (ACC)	RTE (ACC)	CB (ACC)	ReCoRD (EM)	WSC (ACC)	MultiRC (ACC)	COPA (ACC)	Avg.
AdamW (dense)	53.12	49.53	52.71	28.57	76.24	**59.62**	**57.24**	**74.00**	**56.38**
AdamS (dense)	55.72	50.00	**53.43**	**50.00**	**78.06**	36.54	54.02	70.00	55.97
SCAPE ($d = 0.1$)	**59.36**	**51.72**	52.35	32.14	77.38	41.35	55.18	73.00	55.31
SCAPE ($d = 0.01$)	56.42	49.84	50.18	35.71	76.70	36.54	54.83	69.00	53.65

Table 4: Zero-shot evaluation results for Llama-500M pre-trained using different approaches.

Method	ARC-C (ACC)	ARC-E (ACC)	HellaSwag (ACC)	LAMBADA (ACC)	MMLU (ACC)	OpenBookQA (ACC)	PIQA (ACC)	WinoGrande (ACC)
AdamW (dense)	**22.53**	48.86	**34.70**	**38.60**	24.08	18.80	**67.30**	50.67
AdamS (dense)	20.31	46.97	34.57	36.66	**25.77**	18.80	65.94	**53.51**
SCAPE ($d = 0.1$)	20.90	**48.95**	34.38	38.39	23.28	**19.40**	66.76	52.33
SCAPE ($d = 0.01$)	21.93	47.47	33.93	36.64	23.28	18.60	66.32	51.07

Method	SuperGLUE								
	BoolQ (ACC)	WiC (ACC)	RTE (ACC)	CB (ACC)	ReCoRD (EM)	WSC (ACC)	MultiRC (ACC)	COPA (ACC)	Avg.
AdamW (dense)	57.65	49.69	53.43	35.71	69.81	**41.35**	**57.03**	65.00	**53.71**
AdamS (dense)	57.55	**50.63**	53.07	**41.07**	69.54	34.62	49.86	**68.00**	53.04
SCAPE ($d = 0.1$)	51.90	49.06	**57.40**	28.57	**70.00**	39.42	53.11	**68.00**	52.18
SCAPE ($d = 0.01$)	**57.92**	50.00	52.71	32.14	69.66	36.54	56.48	**68.00**	52.93

J PER-STEP TIME

We profile the time per iteration for pretraining Llama-500M (global batch 1024 and sequence length 4K) and Llama-1.8B (global batch size 1024 and sequence length 2K) with eight configurations, including AdamS, AdamS with DO, SCAPE (d = 0.1) with DDP, SCAPE (d = 0.1) with DO, SCAPE (d = 0.1) with DO and CPU offloading, SCAPE (d = 0.01) with DDP, SCAPE (d = 0.01) with DO, and SCAPE (d = 0.01) with DO and CPU offloading. We define the model architecture of Llama-1.8B by following the definition of H2O-Danube3-1.8B (Pfeiffer et al., 2024). The profiling results ranging from 4 to 64 GH200 GPUs are plotted in Figure 6.

As we can see from the profiling results, SCAPE can efficiently reduce the time per iteration for pretraining both Llama-500M and Llama-1.8B under different training configurations. For Llama-500M training, the per-step time reduction using SCAPE ($d = 0.1$) and SCAPE ($d = 0.01$) is less pronounced than Llama-1.8B. For Llama-500M on 64 GPUs, SCAPE with d = 0.1 and d = 0.01 reduces the per-step time of AdamS with DDP from 1753.66 ms under dense AdamS to 1077.40 ms and 908.19 ms, corresponding to 1.63× and 1.93× speedup, respectively. For Llama-1.8B, SCAPE reduces the per-step time from dense AdamS (with DO)'s 3890.17 ms to 1950.84 ms and 1216.38 ms, yielding 1.99× and 3.20× speedup, respectively. The reason is that with such a high number of tokens per step and a smaller model size than Llama-1.8B, the per-step training time for Llama-500M is dominated by computation rather than communication, which can be found from Figure 15.

We notice that when the DO is used, AdamS's per-step time decreases, while SCAPE's is slightly increased for Llama-500M. The reason is that by sharding the model and optimizer states to all workers for AdamS, each worker's computation time for updating the model and states is reduced. Moreover, the composition of *reduce-scatter* for synchronizing gradient and *all-gather* for gathering the model shards in the DO setting has the same cost as $T_{\text{ring-}\textit{all-reduce}}$ (see Equation 1). Therefore, using DO reduces the computation for each worker without introducing additional communication overhead. However, for SCAPE with DO, since each worker still needs to update the full model, the

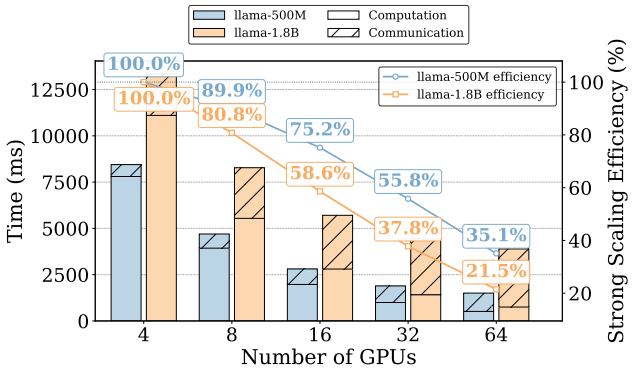


Figure 15: Scaling bottleneck for pretraining Llama-500M (sequence length 4K) and Llama-1.8B (sequence length 2K) using Megatron-LM with distributed optimizer (DO) on GH200 GPUs.

computation is not reduced. Furthermore, the use of *all-reduce* introduces additional communication overhead compared to *reduce-scatter* (see Equation 1), although its communication volume is significantly reduced. Hence, using SCAPE with DO has slightly higher per-step time than SCAPE when communication does not dominate the per-step time, such as training Llama-500M on four GPUs.

From Figure 6, we can see that adding CPU offload to SCAPE with DO does not add overhead for transmitting between CPU and GPU for each step. Thanks to the high bandwidth provided by GH200's NVLink-C2C of 900 GB/s, our double-buffering scheme for CPU offload only has minimal impact on the per-step time. Nevertheless, this could add extra overhead for systems that use PCIe for connecting CPU and GPU, which has a lower bandwidth than NVLink-C2C.

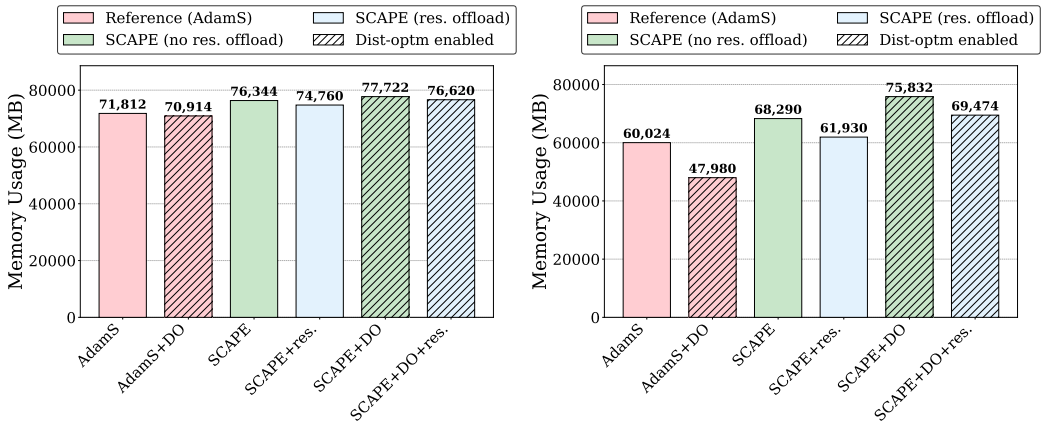
K Strong Scaling Efficiency

The strong scaling efficiency results for training Llama-500M and Llama-1.8B using all aforementioned methods on 4, 8, 16, 32, and 64 GH200 GPUs are plotted in Figure 6. Using the four-GPU setup as the baseline, the strong scaling efficiency improves with SCAPE for both models: for Llama-500M at 64 GPUs, efficiency is improved from AdamS's 30.98% to 46.11% for SCAPE (d = 0.1) and 54.56% for SCAPE (d = 0.01), and up to 63.69% when using SCAPE (d = 0.01) with DO and CPU offloading; for Llama-1.8B on 64 GPUs which has communication dominating the per-step time (see Figure 15), the same pattern is more obvious, improving from 17.58% for AdamS and 21.48% for AdamS with DO to 37.15% for SCAPE (d = 0.1) with DDP, 37.06% for SCAPE (d = 0.1) with DO, 47.83% for SCAPE (d = 0.01) with DDP, 56.88% for SCAPE (d = 0.01) with DO, and 55.04% for SCAPE (d = 0.01) with DO and CPU offloading.

L Memory Consumption

To reduce the memory overhead of keeping full model parameters on every worker, we offload the full-parameter replica to CPU memory. During parameter updates, we use a double-buffered pipeline that overlaps asynchronous CPU-to-GPU prefetch and sparse payload communication with local non-top-k weight-decay updates, then writes back the updated buffer and offloads it to the host memory for the next iteration.

We profile the memory consumption for training Llama-500M and Llama-1.8B under different settings, and the results are shown in Figure 16. Since the memory overhead of residual buffer and full model parameters is determined by the number of model parameters and does not depend on SCAPE's d or the number of workers, only the memory usage for d = 0.1 on 4 GPUs is reported. We can see that with residual offloading enabled, the memory usage for training Llama-1.8B is reduced from 68,290 MB to 61,930 MB and from 75,832 MB to 69,474 MB before and after DO is enabled. For Llama-500M training, because its model size is small and most of GPU memory is consumed by long-sequence activations, using residual offloading does not significantly reduce its memory usage.



(a) Llama-500M (b) Llama-1.8B

Figure 16: Memory usage for training Llama-500M (sequence length 4K) and Llama-1.8B (sequence length 2K).

M Communication Volume

Let $N \geq 2$ be the number of data-parallel GPUs, W_ℓ the parameter count of layer ℓ, $W = \sum_\ell W_\ell$, and $k_{\text{tot}} = \sum_\ell k_\ell$ the number of coordinates selected in the current step. The effective density is $d_{\text{eff}} = k_{\text{tot}}/W$; $k_\ell = W_\ell$ for uncompressed normalization or bias tensors. We count total bytes $\mathcal{V}$ *sent across all GPUs* under an ideal ring model. The average sent volume per GPU is $\mathcal{V}/N$. The current SCAPE implementation uses $B_v = 4$ bytes (FP32) per communicated value for both contributions and updated parameters.

Let B_M^{DDP} and B_M^{DO} denote the total packed-mask input sizes across ranks for the mask exchanges launched in the step. These sizes include padding. For one refresh, the implementation pads each rank's mask input in buffer b to the largest input length len_b, giving $B_M = N \sum_b \text{len}_b$. The approximation $B_M \simeq W/8$ assumes balanced mask ownership and negligible rounding, and the actual padding can differ between DDP and the distributed optimizer.

SCAPE with DDP. With replicated optimizer states, SCAPE uses one *all-reduce* of selected proposal values and one packed-mask *all-gather* per step in sparse steady state. Every worker applies the same optimizer update, so there is no post-update parameter exchange. Consequently,

$$\mathcal{V}_{\text{SCAPE,DDP}} = \underbrace{2(N-1)k_{\text{tot}}B_v}_{\text{selected-value } \textit{all-reduce}} + \underbrace{(N-1)B_M^{\text{DDP}}}_{\text{mask } \textit{all-gather}}. \tag{26}$$

SCAPE with a Distributed Optimizer. SCAPE first averages selected local contributions using *all-reduce-mean*; workers then reconstruct gradients and update their optimizer shards. A second *all-reduce-sum* synchronizes the workers' updated selected parameter values. Both value payloads contain k_{tot} entries per worker; the second contains zeros at selected coordinates owned by other ranks. Unselected parameters receive weight decay locally. Thus,

$$\mathcal{V}_{\text{SCAPE,DO}} = \underbrace{2(N-1)k_{\text{tot}}B_v}_{\text{contribution } \textit{all-reduce-mean}} + \underbrace{2(N-1)k_{\text{tot}}B_v}_{\text{parameter } \textit{all-reduce-sum}} + (N-1)B_M^{\text{DO}}. \tag{27}$$

This path replaces the dense gradient *reduce-scatter* and parameter *all-gather* with two sparse *all-reduce*. The mask *all-gather* remains.