Communication-Efficient LLM Adaptation over Decentralized GPU Meshes

Sameera Ramasinghe **Shamane Siriwardhana** **Thalaiyasingam Ajanthan**
Hadi Mohaghegh Dolatabadi **Chamin P Hewa Koneputugodage** **Gil Avraham**
Violetta Shevchenko **James Snewin** **Karol Pajak** **Harry Xi** **Alexander Long**
Pluralis Research
sameera@pluralis.ai

Abstract

Decentralized training enables large-model training over low-end GPUs and internet-grade connections, but communication along both data-parallel and pipeline-parallel axes becomes the primary bottleneck. We study post-pretraining adaptation in this setting. We propose an asynchronous two-circuit system: a fast compressed training circuit drives throughput using activation masking for pipeline-parallel (PP) transfer and compressed data-parallel (DP) synchronization, while a slow anchor circuit runs occasional unmasked forward–backward passes off the critical path. Then, we introduce a spectral correction optimizer that uses these delayed *anchor priors* to denoise masked gradients without blocking the fast stream. Although prior work has found aggressive activation compression unreliable, we show that masking supports post-pretraining adaptation at high compression rates when anchored this way. Pipeline-parallel compression alone yields up to a $9\times$ throughput gain, and combining it with data-parallel compression increases beyond $40\times$ over internet-grade $\sim$ 200Mbps connections, while matching dense uncompressed performance across domain adaptation and continual pretraining.

1 Introduction

Training and adapting frontier models typically requires tightly connected datacenter clusters, putting modern AI research out of reach for smaller labs, independent researchers, and open communities [Narayanan et al., 2021, Grattafiori et al., 2024]. Decentralized training offers an alternative: computation can be distributed over the Internet across heterogeneous consumer-grade GPUs, allowing participants to contribute otherwise fragmented compute [Diskin et al., 2021, Ryabinin et al., 2023, Yuan et al., 2022, Douillard et al., 2023a, Jaghouar et al., 2024a, Ajanthan et al., 2026, Ramasinghe et al., 2025c]. We focus on decentralized post-pretraining adaptation, *i.e.*, fine-tuning or continual pretraining open-weight models over such GPU meshes.

This setting is fundamentally communication-limited. Commodity Internet links provide bandwidth on the order of hundreds of Mbps, compared to hundreds of Gbps in datacenter interconnects. A practical system must therefore reduce communication along both distributed-training axes: data parallelism (DP), where replicas synchronize weight gradients, and pipeline parallelism (PP), where model shards exchange activations and activation gradients.

DP communication can be reduced with gradient compression [Vogels et al., 2019, Wang et al., 2018, Basu et al., 2019, Zhao et al., 2024a] or sparse synchronization [Douillard et al., 2023b, 2025, Peng et al., 2024, Jaghouar et al., 2024b, Sani et al., 2024], but these methods still assume that each worker or replica group can host a full model. PP addresses the complementary regime: it enables models that do *not* fit on a single low-end GPU by partitioning them across layers. However, PP compression is substantially more delicate. Standard compression methods such as sparsification,

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

quantization, and low-rank projection often degrade performance at high compression rates due to error accumulation across layers [Rudakov et al., 2023, Ramasinghe et al., 2025b]. Recent methods such as Subspace Networks [Ramasinghe et al., 2025b] and Pufferfish [Wang et al., 2021] mitigate this during pretraining by imposing structural constraints, but these constraints make them difficult to apply to arbitrary pretrained checkpoints.

We therefore revisit and refine a simple but previously discarded idea: *activation masking*. Our key observation is that adaptation differs from pretraining. During post-pretraining adaptation, weights evolve slowly from a strong pretrained initialization, and the goal is often to find a useful update direction rather than learn representations from scratch. In contrast to the pessimistic view in prior works [Ramasinghe et al., 2025b, Rudakov et al., 2023], we show that sparse activation signals are sufficient for the main training stream, provided they are anchored by higher-fidelity information. We realize this idea with a two-circuit decentralized system. A fast compressed circuit performs most updates using in-graph activation masking along the PP axis and compressed synchronization along the DP axis. In parallel, a slower anchor circuit occasionally runs full unmasked forward and backward passes. These unmasked passes do not block the fast circuit; instead, their gradients arrive asynchronously as *Anchor Priors*, which denoise and steer the masked optimization trajectory. The asynchrony is principled: fine-tuning gradients live in a low-dimensional [Li et al., 2018, Aghajanyan et al., 2021, Gur-Ari et al., 2018], slowly-drifting subspace, so a stale unmasked gradient still points roughly the right way [Neyshabur et al., 2020, Wortsman et al., 2022], the same property that makes model merging effective.

To use these delayed priors effectively, we introduce a spectral correction optimizer. It maintains a slow-moving momentum buffer over anchor gradients and uses its principal directions to reweight masked updates: anchor-supported eigendirections are reinforced, while unsupported noisy directions are damped. We complement this design with a convergence analysis identifying when the correction provably helps: when fine-tuning gradients lie in a low-dimensional subspace and the optimization trajectory drifts slowly, the anchor basis remains informative despite staleness. The analysis also clarifies why random PRF masking is essential; it produces an approximately unbiased gradient estimator that the spectral filter can denoise, while top-K masking introduces a structured bias the filter cannot remove. Empirically, the system is robust to substantial staleness, and significantly superior to top-K, reinforcing these predictions.

Across domain adaptation, continual pretraining, and reasoning-oriented training, our method matches or exceeds dense uncompressed training while substantially improving throughput over low-bandwidth links. PP compression alone yields up to a $9\times$ throughput gain, and combining it with DP compression increases this over $40\times$. The method generalizes across model families, depths, widths, and mesh sizes, and produces models that are more robust to pruning. Overall, our contributions are as follows:

- We propose a two-circuit decentralized training system in which a fast masked circuit trains continuously while a slower unmasked anchor circuit asynchronously supplies high-fidelity gradient priors, decoupling throughput from the cost of clean gradient signals.
- We introduce a spectral correction optimizer that uses delayed Anchor Priors to denoise masked gradients. Our convergence analysis and empirical results demonstrate its efficacy.
- We outperform competing compression methods and match uncompressed adaptation at over $40\times$ throughput, on dense and MoE models; the trained models are also more robust to pruning.

2 Related Works

Data parallel compression. Decentralized DP training has motivated two lines of work: 1) *Sparse synchronization methods* reduce the number of all-reduces by performing many local optimizer steps between outer updates [Douillard et al., 2023b, 2025, Peng et al., 2024, Jaghouar et al., 2024b, Sani et al., 2024]; we use Streaming DiLoCo [Douillard et al., 2025], which overlaps the outer synchronization with local computation. 2) *Low-rank gradient compressors* reduce communication volume by transmitting low-rank approximations of the gradient [Vogels et al., 2019, Wang et al., 2018, Basu et al., 2019, Zhao et al., 2024a]; we combine PowerSGD [Vogels et al., 2019] with Streaming DiLoCo. **Pipeline-parallel activation compression.** *Quantization-based methods* cast activations and activation gradients to lower-precision formats with error compensation: AQ-SGD [Wang et al.,

2022a] compresses activation *changes* between epochs to enable error feedback over slow networks, and TAH-Quant [He et al., 2025] extends this with tile-wise adaptive Hadamard quantization. These methods operate *off-graph*, transmitting compressed payloads that are decompressed before use, which decouples forward and backward sparsity patterns and requires storing per-example state across epochs to support error feedback. We instead use *in-graph* random masking with no error feedback. Our masking is also substantially more aggressive (95% vs. $4\times/8\times$ quantization). **Parameter-efficient fine-tuning** reduces training VRAM by using low-rank adapters [Hu et al., 2022, Zhao et al., 2024b, Lialin et al., 2023, Li and Liang, 2021, Lester et al., 2021, Houlsby et al., 2019, Pfeiffer et al., 2021], but the forward and backward passes still transmit *full* dense activations across pipeline-parallel boundaries. Thus, they do not address the communication bottleneck. Our method composes cleanly with PEFT (Section 5.8).

3 Methodology

We consider decentralized post-pretraining adaptation over a low-bandwidth GPU mesh across DP and PP dimensions. We represent a decentralized mesh as a 2D $X \times Y$ grid, where X is the data-parallel (DP) dimension and Y is the pipeline-parallel (PP) dimension. Each worker stores only a subset of the model layers, and workers communicate over low-bandwidth links along both DP and PP axes. For all experiments, we partition the mesh into two asynchronous circuits: an $(X - Z) \times Y$ fast masked circuit and a $Z \times Y$ unmasked *anchor circuit*. In all our experiments, we fix $Z = 1$.

The anchor circuit occasionally runs unmasked forward–backward passes and supplies high-fidelity gradient signals that the fast circuit uses to denoise its compressed updates. The anchor circuit does not block the fast circuit: its gradients are consumed asynchronously when they arrive, so the fast circuit's throughput is set by masked communication only. This split is the core design move. Activation masking alone reduces communication but introduces noise that compounds across boundaries; running unmasked passes synchronously would eliminate the throughput gain. Asynchronous anchors give us both, *i.e.,* masked-step throughput, plus a slow correctional signal that does not enter the critical path.

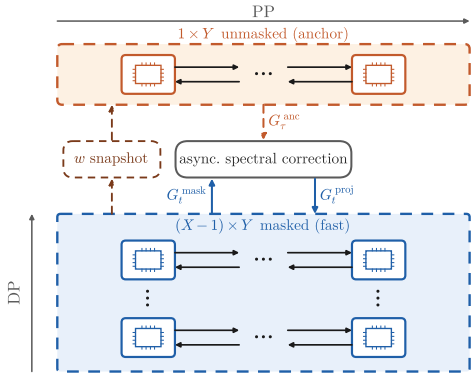

Figure 1: **System overview.** A small unmasked anchor circuit (orange, $1\times Y$ PP stages) asynchronously supplies clean gradients G_τ^{anc} to a spectral corrector, which denoises the masked gradients G_t^{mask} from the fast circuit (blue, $(X-1)\times Y$ DP×PP) into corrected updates G_t^{proj}. Slow circuit periodically pulls fresh weight snapshots from the fast circuit.

3.1 Activation Masking for PP Compression

Let $h \in \mathbb{R}^{T\times H}$ denote the hidden-state tensor at a masked inter-layer boundary, where T is sequence length and H is hidden size. On masked steps we apply elementwise masking $\tilde{h} = h \odot m$ with $m \in \{0,1\}^{T\times H}$ at masking probability p, retaining $K = \text{round}((1-p)H)$ dimensions per token.

Shared PRF avoids transmitting the mask. Naively, masking would require sending the retained indices for every token alongside the values, halving the compression benefit. We avoid this by generating m from a shared deterministic pseudorandom function $M = g(\ell, t, B, S, H)$ keyed on layer index, optimization step, and batch metadata. Sender and receiver reconstruct the same mask locally; only the K retained values are transmitted.

In-graph masking gives matched forward/backward sparsity. Because masking is applied in-graph as a multiplication, the backward pass inherits the same pattern automatically: $\partial\mathcal{L}/\partial h_{t,j} =$

$(\partial \mathcal{L}/\partial \tilde{h}_{t,j})\, m_{t,j}$. Any entry masked in the forward has exactly zero backward gradient, so the same retained positions can be reused for backward activation-gradient transmission. This gives the same compression rate in both directions and avoids the need to track separate forward and backward sparsity patterns. It also distinguishes our scheme from off-graph transport compression, where forward and backward compression are decoupled and can drift out of correspondence with the derivative of the executed computation (*e.g.,* decoupled Top-K or quantization).

Why random masking, not top-K. Top-K is the default in DP gradient compression, but it is a poor fit at PP activation boundaries (full analysis in Appendix F). First, top-K is biased: its mask is a deterministic function of h, so the dropped low-magnitude entries leave a state-dependent bias that enters the descent inequality as a floor that does not shrink with training, whereas our PRF mask is drawn independently of the model state and its noise has zero conditional mean. Second, the standard remedy for top-K bias, error feedback, does not apply here: an activation residual is step-local (the next step sees different inputs and weights), and in-graph masking forces the backward pattern to equal the forward one. Third, the spectral correction introduced next contracts approximately isotropic noise strongly, while top-K noise sits in the low-magnitude complement of the anchor subspace, where filtering removes nothing useful. Empirically, top-K at $10\times$ already trails our method at $20\times$ (Table 6).

3.2 Anchor Priors and Spectral Correction

Activation masking reduces PP communication but injects noise into every gradient. We need a denoising signal, but one that does not eliminate the throughput gains we just bought. Fine-tuning makes a useful option viable: gradients live in a slowly drifting low-dimensional subspace, so a stale unmasked gradient still carries useful information. We exploit this with an asynchronous *anchor* circuit that runs occasional unmasked passes and uses their geometry to denoise the fast masked gradients without blocking them.

Asynchronous anchor circuit. A synchronous fix would periodically run a full unmasked pass through the main pipeline, but each such pass sends $1/(1-p)$ times more activation data per boundary while the whole fast circuit waits, which nearly halves throughput in our setting (Section 5.3). Instead, a parallel anchor circuit runs unmasked passes from a slightly stale weight snapshot and feeds its gradients back when they arrive, so the fast circuit never waits. The cost is delay ($\Delta \approx 20$–25 fast steps in our configuration): anchor gradients cannot serve as exact updates and can only inform optimization *geometry*, which shapes the rest of the design.

A separate momentum buffer for unmasked gradients. For each targeted weight matrix W, we maintain an EMA of arriving anchor gradients,

$$M_t^{\mathrm{anc}} = \beta_{\mathrm{anc}} M_{t^-}^{\mathrm{anc}} + (1-\beta_{\mathrm{anc}}) G_\tau^{\mathrm{anc}}, \tag{1}$$

where t^- is the optimizer state immediately before the anchor gradient arrives. We keep this momentum *separate* from AdamW's first moment m_t, which is updated on every fast step and dominated by masked-step noise; blending stale unmasked gradients into it would dilute their signal or, with an aggressive blend, cause slow oscillations. We use an EMA rather than the latest anchor gradient because individual anchor gradients are noisy, and choose β_{anc} to average over several anchor cycles, since the gradient subspace drifts slowly during fine-tuning (Section E.3).

From momentum to a denoising operator: Given M_t^{anc}, we want to extract the directions in parameter space that are repeatedly reinforced by unmasked gradients, *i.e.,* the geometry the fast circuit's noisy gradients should be pulled toward. Singular value decomposition is the natural choice: $M_t^{\mathrm{anc}} = U_t S_t V_t^\top$ identifies left and right subspaces with strength quantified by the singular values s_i. Directions with large s_i are those the anchor circuit has consistently observed; directions with small s_i are either weakly supported or absent. We use SVD rather than, e.g., Hessian eigendecomposition (which we cannot afford at scale) or AdamW's second moment v_t (which is dominated by masked-step variance and reflects *noise* structure, not signal structure).

Filtering $d_i = s_i/(s_i+\tau_p)$. Given the anchor basis (U_t, V_t), we need to decide how to use it. Hard projection onto the top-r singular subspace is one option but requires choosing r, and discards

information about how strongly each direction is supported. Instead, we apply a soft filter

$$d_i = \frac{s_i}{s_i + \tau_p}, \tag{2}$$

which is the standard Tikhonov-regularized soft threshold. It lies in $[0, 1]$, so the filter never amplifies the gradient; it passes directions with $s_i \gg \tau_p$ nearly unattenuated and damps those with $s_i \ll \tau_p$; and its single hyperparameter τ_p sets the scale of "weakly supported" instead of a hard rank cutoff, giving a Lipschitz filter that is well-behaved as the anchor momentum evolves.

Two-sided application. For a matrix gradient G_t^{mask}, we transform into the anchor basis, $X_t = U_t^\top G_t^{\text{mask}} V_t$, apply the filter to both index axes, $X_t^{\text{filt}} = \text{diag}(d) X_t \text{diag}(d)$, and reconstruct $G_t^{\text{filt}} = U_t X_t^{\text{filt}} V_t^\top$. Two-sided filtering reflects that for a matrix-valued gradient, both the row space (which input features are active) and the column space (which output features are active) carry information. The anchor circuit observes both; restricting to one side discards half the geometric information.

The final gradient passed to AdamW is

$$G_t^{\text{proj}} = \alpha G_t^{\text{mask}} + (1 - \alpha) G_t^{\text{filt}}. \tag{3}$$

We do not use the filtered gradient alone. The reason is that the anchor basis is an EMA of stale gradients, so G_t^{filt} inherits some staleness; the masked gradient is noisy but reflects the *current* weights. Blending preserves the current-iterate information while pulling toward the anchor's reliable subspace. The blend coefficient α trades these off: $\alpha = 1$ recovers masked-only training (no denoising), $\alpha = 0$ uses only the filtered gradient (maximum denoising, full inheritance of staleness), and intermediate values balance the two. Our convergence analysis (Appendix E) shows that this blend interpolates between the two regimes monotonically.

Putting these pieces together, the spectral correction is an online, adaptive preconditioner whose basis and weights are estimated by the anchor circuit and updated as new anchor gradients arrive. The masked gradient is never discarded. Instead, it is softly pulled toward directions the anchor circuit has consistently observed. Because the anchor circuit runs asynchronously, this denoising signal arrives at the cost of staleness, not throughput.

Cost. M_t^{anc} changes only when an anchor gradient arrives, so the SVD is recomputed only at that cadence (every $\sim$20 fast steps), as a randomized truncated SVD at rank r=256 on a side CUDA stream. Between arrivals the cached factors (U_t, d, V_t) are reused, so a masked step pays only four thin matrix multiplications per targeted matrix. For the 1B model on A100s this is under 0.4% of the masked-step time, the SVDs take under 0.6% of GPU time, and the extra memory is one EMA buffer per targeted matrix plus 2–18 MB of factors (Appendix H.9).

3.3 Data-Parallel Compression

PP compression alone does not suffice in a decentralized mesh because dense DP synchronization can dominate wall-clock time and erase the gains from masking. At Internet-grade bandwidth the disparity is severe. We therefore compose PP compression with DP compression. In our implementation we use Streaming DiLoCo Douillard et al. [2025] with PowerSGD Vogels et al. [2019]: PowerSGD reduces bytes per synchronization via low-rank gradient compression, DiLoCo reduces synchronization frequency by performing local steps between outer updates, and Streaming DiLoCo overlaps the outer synchronization with local computation. Note that our PP-side mechanism (activation masking and anchor priors) is independent of this particular DP compressor and can be combined with better alternatives.

3.4 End-to-End Training Procedure

The fast circuit performs most optimizer steps, while the anchor circuit periodically pulls recent layer shards from the fast circuit, runs unmasked forward–backward passes, and returns the resulting gradients asynchronously. When an anchor gradient arrives, the fast circuit updates M_t^{anc} and its SVD and applies the spectral correction to subsequent masked gradients, so expensive unmasked computation becomes an asynchronous source of optimization geometry without the stalls of synchronous unmasked steps. The overall system design is illustrated in Fig. 1; Algorithm 1 summarizes the procedure, and Table 1 collects the notation together with the single default configuration used for every experiment in this paper.

Table 1: Notation and defaults. One configuration is used for all experiments; there is no per-task tuning.

Symbol	Meaning	Default
$X \times Y$	DP×PP mesh (anchor: $1 \times Y$)	8×8
p	activation mask rate	0.95
K	dims kept per token	$(1-p)H$
Δ	anchor staleness (fast steps)	$\approx$20–25*
β_{anc}	anchor-EMA decay	few cycles
r	SVD rank of M^{anc}	256
τ_p	filter threshold	10^{-3}
α	weight on masked gradient	0.3

* set by the network, not tuned

Algorithm 1 Two-circuit training (condensed; full version in Appendix A).

Require: $w_0, p, \alpha, \beta_{\text{anc}}, \tau_p, r$; $M^{\text{anc}} \leftarrow 0, d \leftarrow 0$
for $t = 1, \ldots, T$ **on the fast circuit do**
  Masked fwd/bwd with PRF masks $g(\ell, t, B, S, H) \to G_t^{\text{mask}}$
  if an anchor gradient G_τ^{anc} has arrived **then**
    $M^{\text{anc}} \leftarrow \beta_{\text{anc}} M^{\text{anc}} + (1-\beta_{\text{anc}}) G_\tau^{\text{anc}}$
    $(U, S, V) \leftarrow \text{SVD}_r(M^{\text{anc}})$; $d_i \leftarrow s_i/(s_i+\tau_p)$
  end if
  $G_t^{\text{filt}} \leftarrow U \text{diag}(d) U^\top G_t^{\text{mask}} V \text{diag}(d) V^\top$
  $G_t^{\text{proj}} \leftarrow \alpha G_t^{\text{mask}} + (1-\alpha) G_t^{\text{filt}}$
  AdamW step on G_t^{proj}; compressed DP sync (§3.3)
end for
Anchor circuit, concurrently: pull weights w_τ; unmasked fwd/bwd; push G_τ^{anc}; repeat

4 Convergence Analysis

We do not re-analyze the DP component (PowerSGD and Streaming DiLoCo are well studied, and other DP compressors can be substituted) and focus on masked activation training with asynchronous Anchor Priors.

Let $F(w) = \mathbb{E}_\xi[\ell(w; \xi)]$ denote the population training objective, $F^\star$ its lower bound, w_t the model weights at step t, η the learning rate, and L the smoothness constant of F. Let G bound the second moment of the update direction, and Δ the maximum staleness of anchor gradients in fast-circuit steps. The blend coefficient $\alpha \in [0, 1]$ controls how strongly the spectral correction is applied (Section 3).

Theorem 1 (Informal; full version in Appendix E)**.** *Under standard smoothness and structural assumptions on the spectral filter, after T steps,*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\|\nabla F(w_t)\|^2 \;\le\; \frac{2(F(w_0) - F^\star)}{\mu_\alpha\, \eta\, T} \;+\; \frac{\Sigma^2_{\alpha,\Delta} + L\eta G^2}{\mu_\alpha},$$

where $\Sigma^2_{\alpha,\Delta}$ is a noise floor with three terms — unfiltered masked noise, filter-surviving noise (scaling as a contraction factor $\rho < 1$), and anchor staleness — and $\mu_\alpha = 1 - (1-\alpha)^2\kappa$ measures how much true gradient the filter preserves (with $\kappa \in [0, 1)$ the fraction of gradient energy the filter discards).

The bound has the standard nonconvex-SGD shape (a $1/T$ term plus a noise floor) and improves over masked-only training when the filter contracts noise ($\rho < 1$) more than it suppresses signal ($\kappa > 0$). Both quantities are small in the *fine-tuning* regime for two reasons: fine-tuning gradients live in a low-dimensional subspace [Aghajanyan et al., 2021, Hu et al., 2022], so the anchor basis preserves nearly all of $\nabla F(w_t)$ (κ small); and fine-tuning trajectories drift slowly [Neyshabur et al., 2020, Wortsman et al., 2022], so a delayed anchor basis remains close to the current one (staleness term controlled). Pretraining violates both, and the analysis predicts no benefit there. Anchor gradients thus need not be exact synchronized updates, only good enough to estimate a low-rank subspace that persists over the staleness window (full proof in Appendix E).

5 Experiments

We evaluate the proposed system across diverse adaptation domains, including **general knowledge**, **medical**, **programming**, **math**, **science**, **commonsense reasoning**, **summarization**, and **document parsing**. We use PowerSGD ($64\times$) and Streaming DiLoCo for DP. Our experiments span mesh sizes, model families, and model scales. We use A100 GPUs (40GB VRAM) for all our experiments.

5.1 Performance across different domain adaptation tasks

Table 2 summarizes the main domain-specific results. We use the publicly available Llama 3.2 1B checkpoint as the pretrained model: $\sim$1.23B parameters, LLaMA-style decoder-only architecture with 16 transformer layers, hidden dimension 2048, and 32 attention heads. All models are trained for 10 epochs on the corresponding training datasets with learning rate 2×10^{-5}, cosine decay, 2k

Table 2: Per-domain task accuracy for dense SFT and masked training variants (efficiency in Table 3). **Baseline**: dense PP, uncompressed DP. **+DP**: dense PP, compressed DP. **M90/M95**: 90%/95% activation masking; **+AP**: Anchor Priors. DP compression is applied to all the masked variants.

Domain	Dataset	Metric	Baseline	+DP	M90	M90+AP	M95	M95+AP
Medical	MedQA	Acc.	0.280	0.275	0.225	0.325	0.180	0.321
	MedMCQA	Acc.	0.330	0.325	0.275	0.335	0.220	0.320
Code	HumanEval	pass@1	0.177	0.171	0.115	0.215	0.072	0.207
	MBPP	pass@1	0.152	0.148	0.085	0.198	0.040	0.192
SQL	Spider	Exec.	0.423	0.412	0.305	0.418	0.180	0.402
Science	ARC-C	Acc.	0.378	0.374	0.328	0.380	0.295	0.355
	SciQ	Acc.	0.956	0.948	0.890	0.952	0.815	0.903
Commonsense	HellaSwag	Acc.	0.556	0.551	0.510	0.690	0.475	0.683
	WinoGrande	Acc.	0.548	0.541	0.512	0.628	0.488	0.625
Summ.	CNN/DM	R-1	0.178	0.176	0.140	0.205	0.108	0.195
	XSum	R-1	0.117	0.115	0.092	0.125	0.068	0.120
Math	GSM8K	EM	0.378	0.365	0.275	0.372	0.165	0.341
Doc. Parsing	QuALITY	Acc.	0.284	0.279	0.215	0.288	0.155	0.253
	NarrativeQA	Tok F1	0.699	0.685	0.485	0.682	0.295	0.557

Table 3: Training efficiency of the variants in Table 2 on the 8×8 mesh at 200 Mbps. Speedup is relative to Baseline; Comm. is bytes communicated per training token.

	Baseline	+DP	M90	M90+AP	M95	M95+AP
TPS (tok/s)	1.1k	3.6k	29.3k	27.8k	**48.9k**	46.5k
Speedup	1×	3.3×	27×	26×	**45×**	43×
Comm. (B/tok)	22.9k	6.9k	713	790	**370**	446

Table 4: Ablations over mask ratio p, staleness Δ, spectral-filter threshold τ_p, merge coefficient α, and (d) synchronous vs. asynchronous anchor. Default (p=95%, Δ=20, τ_p=10^{-3}, α=0.3) in green; (b)–(d) hold others at default. Red: $>30\%$ relative drop.

(a) Mask ratio p vs. staleness Δ

Dataset	Mask	Δ=10	Δ=20	Δ=30	Δ=50
MedQA	p=90%	0.330	0.328	0.310	0.265
	p=95%	0.328	**0.321**	0.295	0.250
	p=99%	0.245	0.230	0.215	0.180
HumanEval	p=90%	0.218	0.215	0.195	0.140
	p=95%	0.215	**0.207**	0.175	0.090
	p=99%	0.085	0.075	0.060	0.025

(b) Spectral threshold τ_p

Dataset	10^{-5}	10^{-4}	10^{-3}	10^{-1}
MedQA	0.215	0.290	**0.321**	0.245
HumanEval	0.105	0.180	**0.207**	0.140

(c) Merge coefficient α

Dataset	α=0.0	α=0.3	α=0.5	α=1.0
MedQA	0.260	**0.321**	0.310	0.180
HumanEval	0.105	**0.207**	0.195	0.072

(d) Anchor timing

Variant	MedQA	HE	TPS
Sync. anchor, Δ≈0	0.329	0.216	24.3k
Async. anchor, Δ≈20	**0.321**	**0.207**	46.5k
No anchor (M95)	0.180	0.072	48.9k

Table 5: Code-domain generalization across model families and scales. Entries are HumanEval / MBPP pass@1. M95: 95% activation masking; AP: Anchor Priors. M95+AP recovers most of the dense performance at every scale. †MoE with 64 experts, top-8 routing, ∼1B active parameters.

Model	Dense	M95	M95+AP
Llama-1B	0.177/0.152	0.128/0.032	0.171/0.141
Qwen-1.5B	0.213/0.188	0.158/0.069	0.206/0.177
Gemma-2B	0.201/0.174	0.147/0.055	0.194/0.162
Phi-2 (2.7B)	0.238/0.205	0.176/0.083	0.228/0.193
Qwen-3B	0.268/0.232	0.207/0.098	0.260/0.222
OLMo-7.5B	0.335/0.294	0.267/0.153	0.324/0.280
Llama-8B	0.384/0.338	0.309/0.184	0.372/0.323
OLMoE-1B-7B†	0.221/0.190	0.142/0.071	0.219/0.188

context length, and batch size 32. The system runs on a 7×8 mesh for the masked (fast) circuit and a 1×8 mesh for the unmasked (clean) circuit, with all inter-node links throttled to 200 Mbps to emulate decentralized deployment. Datasets, evaluation metrics, and per-run configurations are in Appendix I.

As shown, pure activation masking (M90, M95) is catastrophic on its own. Adding Anchor Priors reverses this collapse on every domain: M90+AP and M95+AP match the uncompressed Baseline within noise on most tasks and exceed it on several. **We hypothesize the superior performance on some datasets might be due to the regularization effect of masking leading to better gen-**

eralization. On the throughput (Table 3), the system delivers substantially higher efficiency under bandwidth constraints: M95 reaches a $\sim$45$\times$ TPS gain over the uncompressed baseline by stacking $20\times$ PP activation masking with $640\times$ DP gradient compression, while either compression alone yields only $1.4\times$ or $3.3\times$ because each merely shifts the bottleneck. We observe a practical delay of $\sim 20 - 25$ steps in the unmasked slow gradients.

5.2 Generalization across model variants

Table 5 reports results across Gemma, Qwen, Phi, LLaMA, and OLMo models, spanning 1B–8B parameters, 16–36 layers, and hidden dimensions from 1536 to 4096, plus the OLMoE-1B-7B mixture-of-experts (MoE) model. In all settings, we use 95% activation masking with Anchor Priors (AP). Across these variations, the proposed method consistently achieves near-parity with the uncompressed baseline. The MoE model follows the same pattern even though masking also reaches the router inputs at stage boundaries: M95 alone loses 0.08 / 0.12 pass@1, and M95+AP recovers to within 0.002 of dense. To probe larger scales, we also measured the property the method relies on, the alignment of current gradients with a stale anchor subspace, during full-parameter fine-tuning of Qwen2.5-32B and Llama-3.1-70B. It stays flat over 300 steps (about $12\times$ the staleness window) and does not degrade from 8B to 70B (Appendix G).

5.3 Ablations

We sweep the two knobs that most directly govern the fast/clean trade-off: the activation-mask ratio p and the Anchor-Prior staleness Δ (number of fast-circuit steps between consecutive uncompressed priors). Table 4 reports results. Two failure modes emerge. First, p=99% collapses at every Δ. At this masking ratio the per-step information budget on the PP edges is too small for the clean prior to repair. Second, Δ=50 collapses at every p. By the time a prior this stale arrives, the masked trajectory has drifted beyond its effective correction horizon (Appendix H). Inside the operating envelope ($p \in \{90, 95\}\%,\ \Delta \in \{10, 20\}$) all four corners perform well. Finally, we ablate over τ_p and α.

Synchronous vs. asynchronous anchor. Table 4(d) runs the same anchor cadence (one unmasked pass per $\sim$20 masked steps) synchronously, blocking the pipeline so that anchor gradients arrive fresh ($\Delta\approx$0). Staleness costs under one accuracy point (0.321 vs. 0.329 MedQA, 0.207 vs. 0.216 HumanEval), while asynchrony buys $1.9\times$ throughput: each blocking unmasked pass takes $\sim$72 s, about as long as the $\sim$76 s of masked steps it would otherwise overlap (Appendix H).

Tuning and robustness. One configuration (Table 1) runs every experiment, with no per-task tuning; around it, $\alpha \in [0.3, 0.5]$ is flat and τ_p degrades gracefully within an order of magnitude. Staleness is set by the network, not tuned, and may fluctuate: with Δ drawn uniformly from 10–50 steps, or with 200-step anchor blackouts at a 25% duty cycle, accuracy tracks the mean staleness and recovers from blackouts within about three EMA time constants (Appendix H.7). Anchor gradients only shape the non-expansive filter (Lemma 1) and never enter the update as a descent direction, so a late anchor degrades toward plain masked training rather than injecting a stale step. Swapping AdamW for the dual-EMA AdEMAMix does not substitute for anchors (Appendix K).

Mesh size. Sweeping the DP$\times$PP grid from 4×4 to 16×16 at the same configuration (Table 8), all seven meshes land within $\pm 5\%$ of the 8×8 reference on every metric, so the anchor mechanism is not tied to a topology. Throughput scales roughly linearly in DP-1 (more masked replicas) and inversely with pipeline depth (more boundaries per step).

5.4 Comparison to Activation-Communication Baselines

Table 6 compares M95+AP against published activation- and gradient-compression methods across medical (MedQA, MedMCQA), code (HumanEval, MBPP), and math (GSM8K) domains. Two findings stand out. First, prior methods do not survive at our compression target. Aggressive 2-bit quantization (AC-SGD, TAH-Quant) collapses entirely on generative tasks such as HumanEval, MBPP, and GSM8K, and reduces medical accuracy to near-random. Sparsification and 4-bit quantization recover partially. In contrast, our method matches or exceeds the uncompressed Baseline at $20\times$ /$20\times$. Baselines use the configurations recommended in their papers (hence the asymmetric

Table 6: Comparison of activation-compression baselines. **Comp.**: forward/backward compression ratio. Red cells: training diverged.

Method	Comp.	MedQA	HE	MBPP	GSM8K
Baseline (dense)	1×	0.28	0.18	0.15	0.38
Wang et al. [2022b]	8/2×	0.25	0.00	0.00	0.00
He et al. [2025]	8/2×	0.25	0.00	0.00	0.00
4-bit Quant	4×	0.25	0.13	0.11	0.18
Top-k	10×	0.23	0.12	0.09	0.16
Ramasinghe et al. [2025a]	10×	0.25	0.13	0.10	0.17
Bottleneck Proj.	20×	0.26	0.14	0.12	0.12
M95+AP (ours)	**20×**	**0.32**	**0.21**	**0.19**	**0.34**

Table 7: Robustness to post-hoc magnitude pruning. M95+AP retains a larger fraction of accuracy under pruning. "Avg. Δ" is the mean relative drop across ARC-C, HellaSwag, MMLU.

Metric	Method	s=0	30%	50%	70%
ARC-C	Dense	0.33	0.26	0.25	0.25
	Ours	**0.39**	**0.31**	**0.30**	**0.29**
HellaSwag	Dense	0.57	0.41	0.27	0.25
	Ours	**0.66**	**0.53**	**0.42**	**0.35**
MMLU	Dense	0.35	0.27	0.23	0.23
	Ours	**0.36**	**0.30**	**0.27**	**0.26**
Avg. Δ	Dense	—	−25%	−37%	−40%
	Ours	—	−19%	−29%	−34%

Table 8: Ablation over mesh size DP×PP. One DP replica runs the unmasked anchor circuit, the other DP−1 the masked circuit. All runs use M95+AP at 200 Mbps. **HE**: HumanEval pass@1.

DP×PP	MedQA	HE	GSM8K	TPS
4 × 4	0.318	0.205	0.338	32,100
8 × 4	0.325	0.213	0.345	75,000
4 × 8	0.315	0.200	0.335	19,900
8 × 8	0.321	0.207	0.341	46,500
16 × 8	**0.328**	**0.215**	**0.348**	**99,600**
8 × 16	0.312	0.198	0.330	26,400
16 × 16	0.318	0.205	0.336	56,600

Table 9: Llama-3.2-1B through the full SmolLM3-style recipe (reasoning mid-training → SFT → second SFT), 60B tokens in total. M95+AP is on par with dense quality at ∼ 43× throughput.

Method	IFEval	GSM8K	TPS	Speedup
Base	0.13	0.02	—	—
Dense	0.46	0.23	1.1k	1×
M95+AP	0.41	0.21	46.5k	43×

ratios of the 2-bit quantizers). Their 0.00 entries are divergence: AC-SGD and TAH-Quant diverge within the first epoch and emit degenerate text, which scores zero under pass@1 and exact match, while MedQA's log-likelihood ranking over four options floors near chance (0.25).

5.5 Robustness to Post-Hoc Pruning

A useful side-effect of training under aggressive activation masking is that the resulting checkpoint becomes more tolerant of post-hoc weight sparsification (Table 7). A plausible interpretation is that the masked-optimizer trajectory biases the trained weights toward solutions where information is distributed across many small-magnitude parameters, which is precisely what magnitude pruning rewards.

5.6 Reasoning and continual pretraining

Beyond per-domain fine-tuning, we test whether our method can sustain the longer training trajectories typical of continual pretraining and multi-stage SFT. We run a three-phase pipeline matching the SmolLM3 recipe on Llama-3.2-1B: (1) reasoning-focused mid-training on Llama-Nemotron and OpenThoughts traces, (2) a first SFT round on the smoltalk2 mixture, and (3) a second SFT round on the same mixture, ∼ 60B tokens in total (Table 9; configurations in Appendix B, traces in Appendix C). The masked optimization preserves reasoning capability over trajectories far longer than the per-domain runs, and it leaves the model amenable to RL: GSPO on top of the dense and the M95+AP checkpoints yields the same gains (Appendix D, Table 11). Since this recipe keeps the mixture fixed, we also test an abrupt switch from 5 epochs of code to 5 of math: M95+AP ends at 0.155 HumanEval / 0.363 GSM8K vs. 0.161 / 0.371 for dense, and the anchor-subspace alignment recovers within about 60 fast steps of the switch (Appendix G).

5.7 Real networks and heterogeneous GPUs

Our training runs throttle A100 links to 200 Mbps. We checked this regime by moving the system's actual payloads over public-internet paths between independent providers: on a 177 ms Prague–

Table 10: M95+AP composes with LoRA on Llama 3.2 1B. LoRA cuts trainable parameters but still sends dense activations across PP boundaries; M95+AP cuts that traffic. **Comp.**: forward/backward activation compression. **HE**: HumanEval pass@1.

Method	Trainable	Comp.	MedQA	MedMCQA	HE
Dense SFT	1.23B (100%)	1×/1×	0.280	0.330	0.177
LoRA	3.15M (0.26%)	1×/1×	0.265	0.318	0.168
M95+AP	1.23B (100%)	20×/20×	0.321	0.320	0.207
LoRA + M95+AP	3.15M (0.26%)	20×/20×	0.305	0.310	0.198

Oregon path one TCP flow reaches only 8–10 MB/s, but four parallel flows sustain 28.2 MB/s, above the 25 MB/s we assume, and the measured phase times give a staleness of $\Delta \approx 22.6$ steps against the predicted 23. One pipeline stage of the 1B model computes in 0.07–0.63 s per step on GPUs from the H200 down to the L4 (2–17% of the 3.8 s masked step), so meshes of L4-class or newer cards stay communication-bound and match our A100 throughput; only a 2018 T4 becomes compute-bound (Appendix H.8).

5.8 Composition with parameter-efficient fine-tuning

Parameter-efficient fine-tuning (PEFT) reduces trainable parameters and optimizer state, but not pipeline-parallel activation traffic: even when only LoRA adapters are trained, every stage still sends full activations forward and full activation gradients backward. Our method targets that orthogonal bottleneck, and the two compose (Table 10): LoRA + M95+AP keeps LoRA's 0.26% trainable parameters at 20× less PP traffic, with accuracy above dense SFT on MedQA and HumanEval.

6 Conclusion

We presented a communication-efficient system for decentralized LLM adaptation over low-bandwidth GPU meshes. Pairing in-graph activation masking with asynchronous Anchor Priors, spectral correction, and DP compression makes sparse activation training practical at extreme compression rates; the analysis identifies the regime where this works (low-dimensional, slowly drifting fine-tuning gradient subspaces) and why random PRF masking, not top-K, is the right primitive. Across domain adaptation and continual pretraining, on dense and mixture-of-experts models, the resulting system matches or exceeds dense uncompressed training while delivering large throughput gains, and the resulting models are more robust to post-hoc pruning. These results show that post-pretraining adaptation can tolerate sparser intermediate signals than previously assumed, suggesting a practical path toward large-model adaptation over decentralized collections of low-end GPUs.

Limitations

We target fine-tuning and do not claim the method extends to pretraining, where gradient subspaces are high-dimensional and shift substantially. Training runs are at 1–8B with a single MoE model; at 32B and 70B we measured only the subspace assumptions, so end-to-end training at 70B+ remains to be shown. Throughput comes from A100s with throttled links; we validated the network model on real WAN paths and timed stage compute on six other GPU families (Section 5.7), but have not trained end to end on a heterogeneous mesh.

Acknowledgments and Disclosure of Funding

This work was funded by Pluralis Research. All authors are affiliated with Pluralis Research, which develops decentralized training systems. The authors declare no other competing interests.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 7319–7328. Association for Computational Linguistics, 2021.

Thalaiyasingam Ajanthan, Sameera Ramasinghe, Gil Avraham, Hadi Mohaghegh Dolatabadi, Chamin P Hewa Koneputugodage, Violetta Shevchenko, Yan Zuo, and Alexander Long. Asyncmesh: Fully asynchronous optimization for data and pipeline parallelism. *arXiv preprint arXiv:2601.22442*, 2026.

Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *EMNLP*, 2017.

Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. The convergence of sparsified gradient methods. In *NeurIPS*, 2018.

Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *PMLR*, pages 559–568, 2018.

Michael Diskin, Alexey Bukhtiyarov, Max Ryabinin, Lucile Saulnier, Quentin Lhoest, Anton Sinitsin, Dmitry Popov, Dmitry Pyrkin, Maxim Kashirin, Alexander Borzunov, Albert Villanova del Moral, Denis Mazur, Ilia Kobelev, Yacine Jernite, Thomas Wolf, and Gennady Pekhimenko. Distributed deep learning in open collaborations. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. DiLoCo: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023a.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. DiLoCo: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023b. First posted Nov 2023; latest revision Sep 2024.

Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, Alexandre Ramé, Arthur Szlam, Marc'Aurelio Ranzato, and Paul Barham. Streaming DiLoCo with overlapping communication: Towards a distributed free lunch. *arXiv preprint arXiv:2501.18512*, 2025.

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, 7 2024. URL https://zenodo.org/records/12608602.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Frithjof Gressmann, Zach Eaton-Rosen, and Carlo Luschi. Improving neural network training in low dimensional random bases. In *Advances in Neural Information Processing Systems*, volume 33, pages 12140–12150, 2020.

Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Guangxin He, Yuan Cao, Yutong He, Tianyi Bai, Kun Yuan, and Binhang Yuan. TAH-Quant: Effective activation quantization in pipeline parallelism over slow network. *arXiv preprint arXiv:2506.01352*, 2025.

Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzębski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *PMLR*, pages 2790–2799, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Sami Jaghouar, Jack Min Ong, and Johannes Hagemann. OpenDiLoCo: An open-source framework for globally distributed low-communication training. *arXiv preprint arXiv:2407.07852*, 2024a.

Sami Jaghouar, Jack Min Ong, and Johannes Hagemann. OpenDiLoCo: An open-source framework for globally distributed low-communication training. *arXiv preprint arXiv:2407.07852*, 2024b.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *ICML*, 2019.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3045–3059. Association for Computational Linguistics, 2021.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations (ICLR)*, 2018.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 4582–4597. Association for Computational Linguistics, 2021.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. ReLoRA: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *ICLR*, 2018a.

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 (ICLR)*, 2018b.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Matteo Pagliardini, Pierre Ablin, and David Grangier. The AdEMAMix optimizer: Better, faster, older. *arXiv preprint arXiv:2409.03137*, 2024.

Vardan Papyan. Traces of class/cross-class structure pervade deep learning spectra. *Journal of Machine Learning Research*, 21(252):1–64, 2020.

Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009. doi: 10.1016/j.jpdc.2008.09.002.

Bowen Peng, Jeffrey Quesnelle, and Diederik P. Kingma. DeMo: Decoupled momentum optimization. *arXiv preprint arXiv:2411.19870*, 2024.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 487–503. Association for Computational Linguistics, 2021.

Vinay V. Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. In *International Conference on Learning Representations (ICLR)*, 2021.

Sameera Ramasinghe, Thalaiyasingam Ajanthan, Gil Avraham, Yan Zuo, and Alexander Long. Beyond top-k: Structured sparsification for compression in pipeline parallel. In *ICLR 2025 Workshop on Modularity for Collaborative, Decentralized, and Continual Deep Learning*, 2025a.

Sameera Ramasinghe, Thalaiyasingam Ajanthan, Gil Avraham, Yan Zuo, and Alexander Long. Subspace networks: Scaling decentralized training with communication-efficient model parallelism. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b.

Sameera Ramasinghe, Thalaiyasingam Ajanthan, Hadi Mohaghegh Dolatabadi, Gil Avraham, Violetta Shevchenko, Yan Zuo, Chamin P Hewa Koneputugodage, and Alexander Long. Mixtures of subspaces for bandwidth efficient context parallel training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025c.

Mikhail I Rudakov, Aleksandr Nikolaevich Beznosikov, Ya A Kholodov, and Alexander Vladimirovich Gasnikov. Activations and gradients compression for model-parallel training. *Doklady Mathematics*, 108(Suppl 2):S272–S281, 2023.

Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. SWARM parallelism: Training large models can be surprisingly communication-efficient. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *PMLR*, pages 29416–29440, 2023.

Lorenzo Sani, Alex Iacob, Zeyu Cao, Royson Lee, Bill Marino, Yan Gao, Dongqi Cai, Zexi Li, Wanru Zhao, Xinchi Qiu, and Nicholas D. Lane. Photon: Federated LLM pre-training. *arXiv preprint arXiv:2411.02908*, 2024.

Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations (ICLR)*, 2019.

Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. In *NeurIPS*, 2018.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. PowerSGD: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright. ATOMO: Communication-efficient learning via atomic sparsification. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Hongyi Wang, Saurabh Agarwal, and Dimitris Papailiopoulos. Pufferfish: Communication-efficient models at no extra cost. *Proceedings of Machine Learning and Systems*, 3:365–386, 2021.

Jue Wang, Binhang Yuan, Luka Rimanic, Yongjun He, Tri Dao, Beidi Chen, Christopher Ré, and Ce Zhang. Fine-tuning language models over slow networks using activation quantization with guarantees. In *Advances in Neural Information Processing Systems*, volume 35, 2022a.

Jue Wang, Binhang Yuan, Luka Rimanic, Yongjun He, Tri Dao, Beidi Chen, Christopher Ré, and Ce Zhang. Fine-tuning language models over slow networks using activation quantization with guarantees. *Advances in Neural Information Processing Systems*, 35:19215–19230, 2022b.

Per-Åke Wedin. Perturbation bounds in connection with singular value decomposition. *BIT Numerical Mathematics*, 12(1):99–111, 1972. doi: 10.1007/BF01932678.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7959–7971, 2022.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yi Yu, Tengyao Wang, and Richard J. Samworth. A useful variant of the Davis–Kahan theorem for statisticians. *Biometrika*, 102(2):315–323, 2015. doi: 10.1093/biomet/asv008.

Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy Liang, Christopher Ré, and Ce Zhang. Decentralized training of foundation models in heterogeneous environments. In *Advances in Neural Information Processing Systems*, volume 35, pages 25464–25477, 2022.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. In *International Conference on Machine Learning (ICML)*, 2024a.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. In *International Conference on Machine Learning (ICML)*, 2024b.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Algorithm 2 Two-circuit training with spectral anchor correction.

Require: Pretrained weights w_0, mask probability p, anchor weight α, anchor EMA β_{anc}, damping τ_p

Split mesh into fast masked circuit and asynchronous anchor circuit
Initialize $w^{\text{fast}}, w^{\text{anc}} \leftarrow w_0$, $M^{\text{anc}} \leftarrow 0$
for step $t = 1, \ldots, T$ **do**
    Forward through PP with deterministic in-graph masks $M_\ell = g(\ell, t, B, S, H)$: $\tilde{h}_\ell = h_\ell \odot M_\ell$
    Backward inherits the same sparsity: $\partial\mathcal{L}/\partial h_\ell = (\partial\mathcal{L}/\partial\tilde{h}_\ell) \odot M_\ell$
    Compute G_t^{mask}; synchronize DP replicas with compressed all-reduce
    if anchor gradient G_τ^{anc} has arrived **then**
        $M_t^{\text{anc}} \leftarrow \beta_{\text{anc}} M_{t^-}^{\text{anc}} + (1 - \beta_{\text{anc}}) G_\tau^{\text{anc}}$
        For each targeted matrix, recompute its rank-r truncated SVD $M_t^{\text{anc}} \approx U_t S_t V_t^\top$ and $d_i = s_i/(s_i + \tau_p)$ (randomized SVD on a side stream; factors cached until the next arrival)
    end if
    for each targeted matrix W **do**
        $X_t = U_t^\top G_t^{\text{mask}} V_t$ using the cached factors
        $G_t^{\text{filt}} = U_t \,\text{diag}(d)\, X_t \,\text{diag}(d)\, V_t^\top$
        $G_t^{\text{proj}} = \alpha\, G_t^{\text{mask}} + (1 - \alpha)\, G_t^{\text{filt}}$
    end for
    Update w^{fast} via AdamW with G_t^{proj}
    Anchor circuit asynchronously refreshes weights, runs unmasked forward–backward, returns G_t^{anc}
end for

A Training Algorithm

The full training algorithm for our method is shown in Algorithm 2.

B Reasoning and Continual-Pretraining Recipe

This section documents the three-phase training recipe used in Section 5.6. We follow the SmolLM3 methodology, but start from a smaller base model (Llama-3.2-1B) to make the experiment tractable at our compute budget.

B.1 Base model

We use Llama-3.2-1B as the starting point, with its tokenizer extended by SmolLM3 ChatML special tokens (`<|im_start|>`, `<|im_end|>`). The model is loaded in BF16 and trained with FlashAttention-2 throughout.

B.2 Phase 1: Reasoning mid-training

Data. `HuggingFaceTB/smoltalk2`, config `Mid`:

- `Llama_Nemotron_Post_Training_Dataset_reasoning_r1` (weight 1.0).
- `OpenThoughts3_1.2M` (weight 1.0).

Configuration. Plain ChatML chat template (no system metadata, no thinking-mode header). AdamW with peak learning rate 2×10^{-5}, cosine schedule with min-LR ratio 0.1, warmup ratio 0.03, gradient norm clipped at 0.2, sequence length 32,768, packing enabled, 2 training epochs, gradient accumulation 2 on 8× A100, Liger kernel enabled. No assistant-only loss masking in this phase; loss is computed over all tokens.

B.3 Phase 2: First SFT round

Data. The smoltalk2 SFT mixture, comprising 25 splits with weights ranging from 0.02 to 1.0. Splits include conversational data (everyday conversations, system chats), instruction-following (Tulu-3 personas, OpenHermes-2.5), tool-use (Hermes function calling, xLAM, smolagents traces), reasoning traces (multi-turn-reasoning, Qwen3-32B-distilled think traces, s1k, OpenThoughts3 think variant), multilingual data (smoltalk multilingual, Aya), long-context (LongAlign 64k), and structured data (table-GPT). The full per-split weights are listed in the supplementary YAML.

Configuration. SmolLM3 ChatML template with thinking-mode metadata (`<|im_start|>system\n## Metadata\n`). Loss masked over assistant tokens only (`assistant_only_loss: true`). Same optimizer as Phase 1 except: sequence length 65,536 (long-context splits), 5 training epochs, BFD packing strategy, save every 0.2 epochs.

B.4 Phase 3: Second SFT round

Data. Identical to Phase 2 (same smoltalk2 SFT mixture and split weights).

Configuration. Identical to Phase 2 except: starts from the Phase 2 checkpoint, 5 training epochs. The purpose of the second round is to extend total training while keeping the data distribution fixed — this tests whether masked optimization remains stable as the model approaches the data's effective ceiling.

Some example reasoning traces of the compressed model are shown below.

C Reasoning Trace Examples

This appendix shows representative reasoning traces from the M95+AP checkpoint after Phase 1 (reasoning mid-training) of the recipe in Section 5.6. Traces are produced with greedy decoding, `enable_thinking=True`, and `max_new_tokens=600`. Each example shows the full `<think>...</think>` block followed by the final answer.

The examples below are the cleanly-correct answers from a 15-question simple-query batch. The remaining questions either produced incorrect answers or degenerated into repetition; this is expected because the checkpoint at this stage has chat-template scaffolding for thinking mode but has not yet undergone instruction/reasoning fine-tuning. The traces are intended to illustrate the *format* of the thinking output produced by the model after masked mid-training, not to demonstrate accuracy claims; per-task accuracy numbers are reported in Table 9.

Example 1: Square Perimeter

Question: A square has sides of 5 cm. What is its perimeter?

Thinking: Okay, so I need to find the perimeter of a square with a side length of 5 cm. The perimeter of a square is calculated by adding up all the sides. The formula for the perimeter of a square is 4 times the length of one of its sides. So, if the side is 5 cm, then the perimeter would be $4 \times 5 = 20$ cm. The side length is given, so the perimeter is straightforward.
Final Answer: The perimeter of the square is $\boxed{20}$ cm.

Answer: To find the perimeter of a square with a side length of 5 cm, we use the formula:

$$\text{Perimeter} = 4 \times \text{side length} = 4 \times 5 = 20 \text{ cm}$$

$\boxed{20}$

Example 2: Half of a Number

Question: What is half of 30?

Thinking: Okay, I need to find half of 30. Half of a number means dividing the number by 2. So if I divide 30 by 2, that should give me 15. Let me check: $30/2 = 15$.
Final Answer: Half of 30 is $\boxed{15}$.

Answer: To find half of 30, we divide 30 by 2.
$\boxed{15}$

Example 3: Rectangle Area

Question: A rectangle is 4 meters wide and 6 meters long. What is its area in square meters?

Thinking: The area of a rectangle is calculated by multiplying the length and the width. So, if the width is 4 meters and the length is 6 meters, then the area should be 4×6. Let me write that down: $4 \times 6 = 24$ square meters.

Answer: The area of a rectangle with width w and length l is $A = lw$. For a rectangle 4 m wide and 6 m long:

$$A = 4 \times 6 = 24 \text{ m}^2$$

$\boxed{24}$

Example 4: Time-Unit Conversion

Question: How many minutes are there in one hour and a half?

Thinking: First, I should convert the given time into hours. "One hour and a half" — one hour is 60 minutes, and a half is 30 minutes. So 1 hour and a half is 1.5 hours.
Now, convert that to minutes. The formula for converting time to minutes is minutes = hours $\times 60$. So $1.5 \times 60 = 90$ minutes.
Double-check: 1 hour is 60 minutes, 1.5 hours is $1.5 \times 60 = 90$ minutes. Correct.

Answer: The time in hours is 1.5 hours. Converting to minutes: $1.5 \times 60 = 90$ minutes.
$\boxed{90}$

Example 5: Triangle Perimeter (with Triangle-Inequality Check)

Question: A triangle has sides of length 3, 4, and 5 cm. What is its perimeter?

Thinking: I remember that the perimeter of a triangle is the sum of all three sides. First, let me verify it's a valid triangle. The triangle inequality says that the sum of any two sides must be greater than the third.

- $3 + 5 > 4$? $8 > 4$, yes.
- $4 + 5 > 3$? $9 > 3$, yes.
- $3 + 4 > 5$? $7 > 5$, yes.

All three inequalities hold, so the triangle is valid. Now to find the perimeter, add all three sides: $3 + 4 + 5 = 12$ cm.

Answer: The triangle has sides 3, 4, 5 cm. The perimeter is the sum of the sides: $3+4+5 = 12$ cm.
$\boxed{12}$

Example 6: Sum of First n Integers (Uses Formula)

Question: What is the sum of $1 + 2 + 3 + 4 + 5$?

Thinking: There's a formula for the sum of the first n natural numbers: $n(n+1)/2$. For $n = 1$, that gives $1 \cdot 2/2 = 1$, which matches. For $n = 2$, $2 \cdot 3/2 = 3$, which is correct. Plug in $n = 5$: $5 \cdot 6/2 = 15$. And directly: $1 + 2 + 3 + 4 + 5 = 15$. Both agree.

Answer: The sum of the first five natural numbers is

$$\frac{n(n+1)}{2} = \frac{5 \times 6}{2} = 15$$

$\boxed{15}$

Example 7: Boiling Point

Question: At what temperature does water boil at sea level, in degrees Celsius?

Thinking: The boiling point of water at sea level is 100 degrees Celsius. At sea level the atmospheric pressure is 1 atm, and at this pressure water boils at 100 °C.
Final Answer: The boiling point of water at sea level is 100 °C.

Answer: The boiling point of water at sea level is 100 degrees Celsius.
$\boxed{100\,^{\circ}\text{C}}$

Example 8: Chemical Formula

Question: What is the chemical formula for water?

Thinking: Water is a compound made up of hydrogen and oxygen atoms. The chemical formula for water is H_2O — one oxygen atom bonded to two hydrogen atoms.

Answer: The chemical formula for water is $\mathbf{H_2O}$: two hydrogen atoms bonded to one oxygen atom.
$\boxed{H_2O}$

Example 9: Two-Step Word Problem (Multiplication)

Question: Sara has 3 boxes, each containing 4 apples. How many apples does Sara have in total?

Thinking: When you have multiple boxes, each with the same number of apples, the total is the number of boxes times the number per box. If there are n boxes each with k apples, the total is $n \times k$.
For Sara: $n = 3$, $k = 4$. Total $= 3 \times 4 = 12$.
Sanity-check with a smaller case: 2 boxes of 2 apples each gives $2 \times 2 = 4$. That's right.

Answer: Sara has $3 \times 4 = 12$ apples in total.
$\boxed{12}$

C.1 Chat template

The SmolLM3 ChatML template used in Phases 2 and 3 injects per-turn system metadata (knowledge cutoff date, current date, reasoning mode `/think` or `/no_think`) and supports XML-style tool

Table 11: GSM8K performance before and after 2 epochs of GSPO on Big-Math, starting from either the dense or M95+AP final checkpoint. RL improvements are comparable from both starting points, indicating that masked fine-tuning leaves the model RL-amenable.

SFT starting point	Pre-RL	Post-RL	Δ
Dense (3-phase)	0.23	0.41	+0.18
M95+AP (3-phase)	0.21	0.40	+0.19

definitions. Reasoning content is wrapped in `<think>...</think>` tags. The template handles three reasoning regimes:

- *System override*: a system message containing `/system_override` replaces all metadata with user-supplied instructions verbatim.
- *Think mode*: assistant turns may include reasoning inside `<think>...</think>` tags. Used for splits ending in `_think`.
- *No-think mode*: assistant turns are emitted with an empty `<think></think>` block, signaling the model to skip reasoning. Used for splits ending in `_no_think`.

Phase 1 uses a simpler ChatML template without the metadata header, since the reasoning-trace data already encodes thinking-mode structure inline.

C.2 Activation masking and Anchor-Prior settings

Identical across all three phases and matching the per-domain experiments (Appendix I): activation mask probability 0.95, masking applied at layers $\{1, 3, 5, 7, 9, 11, 13\}$. Anchor-Prior gradients are injected with the spectral filter described in Section 3. The results are reported in Table 9.

C.3 Evaluation

IFEval and GSM8K are evaluated via the `lm-evaluation-harness` [Gao et al., 2024] on the final checkpoint after Phase 3. Token counts are measured as the cumulative training tokens across all three phases (sequence length $\times$ optimizer steps $\times$ effective batch).

D GSPO Reinforcement Learning Recipe

D.1 Reinforcement learning after masked SFT

A natural concern with masked fine-tuning is whether it leaves the model in a state amenable to downstream RL. Aggressive activation compression could in principle damage internal representations in ways that only manifest when the policy must be optimized against a sparse reward. For instance, by collapsing the entropy of intermediate features or by misaligning the model's confidence estimates. We test this directly: starting from the M95+AP-trained final checkpoint of Section 5.6, we apply GSPO [Zheng et al., 2025] with a verifiable math reward on the Big-Math dataset and compare against the same RL run starting from the dense final checkpoint (Table 11).

The two starting points show essentially identical RL gains, and the post-RL gap matches the pre-RL gap. Masked SFT does not impair RL-amenability: the policy distribution remains entropic enough to explore, the reward landscape remains learnable, and the gradient signal from verifiable rewards translates into the same magnitude of capability gain whether the SFT trajectory was dense or communication-compressed.

Below we detail the GSPO [Zheng et al., 2025] configuration used in the RL phase. The setup follows VeRL's reference GSPO recipe with adjustments for our 1B-parameter model and our context-length budget.

D.2 Pipeline

The full pipeline from base model to post-RL checkpoint is:

$$\text{Llama-3.2-1B} \xrightarrow{\text{Phase 1–3 SFT}} \text{SFT model} \xrightarrow{\text{DPO}} \text{DPO model} \xrightarrow{\text{GSPO}} \text{Final policy}$$

We run this pipeline twice: once with dense SFT throughout, and once with M95+AP SFT throughout (Phases 1–3 of Appendix B). The DPO and GSPO phases are identical in both runs (same data, hyperparameters, and no masking), so any difference in the post-RL policy is attributable to the masked SFT trajectory.

D.3 Reward model

We use a verifiable math reward derived from the Big-Math dataset: each problem has a known final answer, and the reward is a binary indicator of whether the model's extracted answer matches. We use the DAPO [Yu et al., 2025] reward manager, which adds a length-penalty buffer on top of the correctness signal: responses that exceed an overlong_buffer_len threshold within the maximum response length are linearly penalized down to a maximum penalty factor. This discourages reward hacking via verbose answers.

Settings.

- `max_response_length`: $7{,}680$ tokens (out of $8{,}192$-token total context, leaving 512 for the prompt).
- `overlong_buffer_len`: $3{,}584$. Responses up to $7{,}680 - 3{,}584 = 4{,}096$ tokens incur no length penalty; beyond that, reward is linearly attenuated.
- `overlong_penalty_factor`: 1.0.

D.4 GSPO algorithm settings

Policy loss. `loss_mode = "gspo"`: GSPO's sequence-level importance ratio, with loss aggregation `seq-mean-token-mean` (mean over tokens within a response, then mean over responses).

Advantage estimator. `adv_estimator = "grpo"`: group-relative advantage with no critic. Advantages are computed as the standardized rewards within each group of N rollouts sharing a prompt.

Group size. $N = 16$ responses per prompt.

Clip ratios. Lower clip 3×10^{-4}, upper clip 4×10^{-4}. These are substantially smaller than standard PPO clip ranges (0.2) because GSPO's importance ratio is at the sequence level (the product of per-token ratios over a long response). A small per-sequence ratio window corresponds to a much larger effective per-token tolerance.

KL settings. We disable both KL-in-reward and KL-as-loss (`use_kl_in_reward = false`, `use_kl_loss = false`). This is the standard RLVR setup where the only optimization signal is the verifiable reward.

D.5 Optimization

- Optimizer: AdamW, learning rate 1×10^{-6}, weight decay 0.1, gradient clipping 1.0, warmup ratio 0.05. Entropy regularization disabled (`entropy_coeff = 0`).
- Train batch size: 32 prompts per PPO iteration. With $N = 16$ rollouts each, this gives 512 trajectories per update.
- PPO mini-batch: 16 prompts (32 mini-batches per outer iteration).
- Per-GPU micro-batch: 4 prompts.
- Dynamic batching enabled (`use_dynamic_bsz = true`), capping per-GPU tokens at $8{,}192$.

D.6 Rollout (vLLM)

- Backend: vLLM in asynchronous mode (rollouts overlap with actor training).
- GPU memory utilization for vLLM: 0.7.
- Tensor parallel: 1 (sufficient for a 1B model).
- Sampling for training rollouts: temperature 1.0, top-p 1.0, top-k disabled — maximally diverse exploration.
- Sampling for validation: temperature 1.0, top-p 0.7, one sample per prompt — a slightly more conservative distribution for cleaner eval signal.

D.7 Training schedule

- 2 epochs over Big-Math train split, with 25-step validation cadence and 100-step checkpoint cadence.
- No initial validation pass before training (`val_before_train = false`).
- DeepSpeed ZeRO-3 disabled in favor of FSDP; no parameter or optimizer offload (the 1B model fits comfortably on 40 GB GPUs).
- Sequence parallelism disabled (`sp_size = 1`).
- Gradient checkpointing enabled.

D.8 Evaluation

GSM8K is evaluated via the `lm-evaluation-harness` [Gao et al., 2024] on the final checkpoint after 2 epochs. We report exact-match accuracy at temperature 0 greedy decoding, matching the standard GSM8K evaluation protocol. Reported numbers are the best across all checkpoints of the corresponding run, with checkpoints saved every 100 training steps.

E Convergence Analysis with Spectral Anchor Correction

We analyze a simplified version of our optimizer that captures the role of asynchronous anchor priors and spectral gradient correction. The analysis formalizes a single tradeoff: masked gradients are cheap but noisy, while unmasked anchor gradients are clean but delayed. The spectral filter built from the anchor momentum is the mechanism that trades one for the other. We work in the *fine-tuning* regime, where the optimization trajectory is short and the gradient subspace drifts slowly — the structural property that makes a delayed anchor basis useful.

E.1 Setup

Let $F(w) = \mathbb{E}_\xi[\ell(w;\xi)]$ denote the population objective. At iteration t, the fast circuit computes a masked gradient g_t^{mask}, and the anchor circuit asynchronously produces a delayed unmasked gradient g_τ^{anc} from a stale snapshot w_τ with $t - \tau \le \Delta$. We use Δ for the maximum staleness in fast-circuit steps to avoid clashing with the DP replica count D in Section 3.3.

For each matrix-valued parameter W, we maintain an EMA anchor momentum

$$M_t^{\text{anc}} = \beta_{\text{anc}}\, M_{t^-}^{\text{anc}} + (1 - \beta_{\text{anc}})\, g_\tau^{\text{anc}},$$

updated each time a new anchor gradient arrives. Its SVD $M_t^{\text{anc}} = U_t S_t V_t^\top$ defines the spectral filter

$$d_i = \frac{s_i}{s_i + \tau_p}, \qquad D_t = \operatorname{diag}(d_1, \dots, d_r), \qquad \mathcal{P}_t(G) = U_t D_t U_t^\top\, G\, V_t D_t V_t^\top,$$

where $\tau_p > 0$ is the spectral damping constant (the subscript distinguishes it from the anchor time index τ). The update direction is $u_t = \alpha\, g_t^{\text{mask}} + (1 - \alpha)\, \mathcal{P}_t(g_t^{\text{mask}})$, yielding $w_{t+1} = w_t - \eta u_t$.

E.2 Assumptions

Assumption 1 (Smoothness). *F is L-smooth and bounded below by $F^\star$.*

Assumption 2 (Masked-gradient noise). *$g_t^{\mathrm{mask}} = \nabla F(w_t) + \varepsilon_t$ where ε_t is generated by a fresh PRF mask draw at step t, independent of all prior history. Conditional on that history, $\mathbb{E}\,\varepsilon_t = 0$ and $\mathbb{E}\|\varepsilon_t\|^2 \le \sigma_m^2$.*

Assumption 3 (Delayed anchor gradients). *$g_\tau^{\mathrm{anc}} = \nabla F(w_\tau) + \zeta_\tau$ with $t - \tau \le \Delta$, $\mathbb{E}\,\zeta_\tau = 0$, and $\mathbb{E}\|\zeta_\tau\|^2 \le \sigma_a^2$.*

Assumption 4 (Bounded update). *$\mathbb{E}\|u_t\|^2 \le G^2$ for all t.*

The next two assumptions describe what the spectral filter does to the signal and to the noise. They are the core structural conditions; we justify them in Section E.3 and quantify them in the supporting lemmas.

Assumption 5 (Signal preservation). *There exist $\kappa \in [0, 1)$ and $\delta_\Delta^{\mathrm{stale}} \ge 0$ such that*

$$\mathbb{E}\big\|(I - \mathcal{P}_t)\nabla F(w_t)\big\|^2 \le \kappa\,\mathbb{E}\|\nabla F(w_t)\|^2 + \delta_\Delta^{\mathrm{stale}}.$$

Assumption 6 (Noise contraction). *There exists $\rho \in [0, 1]$ such that $\mathbb{E}\|\mathcal{P}_t\,\varepsilon_t\|^2 \le \rho\,\sigma_m^2$.*

E.3 Why these assumptions hold for fine-tuning

Both assumptions exploit two structural properties of fine-tuning that are well-documented in the literature: (i) fine-tuning gradients have low effective rank, and (ii) fine-tuning trajectories drift slowly in parameter space.

Low-rank gradient structure. The intrinsic dimension of fine-tuning objectives is small [Li et al., 2018, Aghajanyan et al., 2021], the gradient and Hessian spectra during training are dominated by a handful of outlier directions [Gur-Ari et al., 2018, Papyan, 2020, Gressmann et al., 2020], and fine-tuning updates $\Delta W = W_t - W_0$ are well approximated by low-rank matrices [Hu et al., 2022, Zhao et al., 2024b, Lialin et al., 2023]. Together these say $\nabla F(w_t)$ lies near a low-rank subspace, so the principal directions of M_t^{anc} — itself an EMA of recent unmasked gradients — contain most of $\nabla F(w_t)$'s energy. This makes κ small (Assumption 5).

Slow weight drift. Successful fine-tuning trajectories remain close to the pretrained initialization, with feature representations changing modestly [Neyshabur et al., 2020, Ramasesh et al., 2021], and fine-tuned checkpoints are close enough in parameter space to admit linear interpolation [Wortsman et al., 2022]. Slow drift bounds the rotation of the gradient subspace between w_τ and w_t, which controls $\delta_\Delta^{\mathrm{stale}}$ via a Davis–Kahan argument (Lemma 3).

Isotropic mask noise. The mask $m_{t,j}$ is drawn from a PRF whose seed is independent of model state, so ε_t is approximately isotropic in parameter space. A low-rank filter applied to isotropic noise contracts it strongly: $\rho = O(r^2/(mn))$ in matrix dimensions $m \times n$ with effective rank r (Lemma 2). This makes ρ small (Assumption 6).

These properties fail in pretraining: the gradient subspace is high-dimensional and shifts substantially over a long trajectory. The analysis below predicts no benefit there, consistent with our empirical scoping of the method to fine-tuning workloads.

E.4 Supporting lemmas

Lemma 1 (Non-expansiveness of $\mathcal{P}_t$). *For any matrix G, $\|\mathcal{P}_t(G)\|_F \le \|G\|_F$.*

Proof. Let $P_U = U_t D_t U_t^\top$ and $P_V = V_t D_t V_t^\top$, so that $\mathcal{P}_t(G) = P_U G P_V$. Each filter weight satisfies $0 \le d_i \le 1$, so $\|D_t\|_2 \le 1$. Combined with the orthonormality of U_t, V_t, this gives $\|P_U\|_2 \le 1$ and $\|P_V\|_2 \le 1$. Applying the submultiplicative inequality $\|AGB\|_F \le \|A\|_2 \|G\|_F \|B\|_2$ with $A = P_U$, $B = P_V$:

$$\|\mathcal{P}_t(G)\|_F = \|P_U G P_V\|_F \le \|P_U\|_2\,\|G\|_F\,\|P_V\|_2 \le \|G\|_F. \qquad \square$$

Lemma 2 (Noise contraction under matrix isotropy). *Suppose $\varepsilon_t \in \mathbb{R}^{m\times n}$ is matrix-isotropic with $\mathbb{E}[\mathrm{vec}(\varepsilon_t)\mathrm{vec}(\varepsilon_t)^\top] = (\sigma_m^2/(mn))\, I_{mn}$, and that $\mathcal{P}_t$ is independent of ε_t. Then*

$$\mathbb{E}\|\mathcal{P}_t\varepsilon_t\|_F^2 = \frac{\sigma_m^2\, r_{\text{eff}}^U\, r_{\text{eff}}^V}{mn}, \qquad r_{\text{eff}}^U = \|D_t\|_F^2 = \sum_i d_i^2,$$

and similarly for r_{eff}^V. In particular, when the left and right filters share an effective rank r in the sense $r_{\text{eff}}^U \approx r_{\text{eff}}^V \approx r$ with $r \ll \min(m,n)$, we have $\rho = O(r^2/(mn)) \ll 1$.

Proof. Write $\mathcal{P}_t(\varepsilon_t) = P_U\varepsilon_t P_V$ as in Lemma 1. We compute the expected squared Frobenius norm in three steps.

Step 1: vectorize. Using the identity $\mathrm{vec}(AXB) = (B^\top \otimes A)\,\mathrm{vec}(X)$,

$$\mathrm{vec}(P_U\varepsilon_t P_V) = (P_V^\top \otimes P_U)\,\mathrm{vec}(\varepsilon_t).$$

Since $\|X\|_F^2 = \|\mathrm{vec}(X)\|^2$, we have

$$\|P_U\varepsilon_t P_V\|_F^2 = \mathrm{vec}(\varepsilon_t)^\top (P_V \otimes P_U^\top)(P_V^\top \otimes P_U)\mathrm{vec}(\varepsilon_t).$$

Step 2: take expectation. For any deterministic matrix M and random vector x with $\mathbb{E}[xx^\top] = cI$, $\mathbb{E}[x^\top Mx] = c\,\mathrm{tr}(M)$. Applying this with $c = \sigma_m^2/(mn)$ and $M = (P_V \otimes P_U^\top)(P_V^\top \otimes P_U)$,

$$\mathbb{E}\|P_U\varepsilon_t P_V\|_F^2 = \frac{\sigma_m^2}{mn}\,\mathrm{tr}\big((P_V \otimes P_U^\top)(P_V^\top \otimes P_U)\big).$$

Step 3: simplify the trace. Using $(A\otimes B)(C\otimes D) = (AC)\otimes(BD)$ and $\mathrm{tr}(A\otimes B) = \mathrm{tr}(A)\mathrm{tr}(B)$,

$$\mathrm{tr}\big((P_V P_V^\top) \otimes (P_U^\top P_U)\big) = \mathrm{tr}(P_V P_V^\top)\,\mathrm{tr}(P_U^\top P_U) = \|P_V\|_F^2\,\|P_U\|_F^2.$$

Finally, $\|P_U\|_F^2 = \|U_t D_t U_t^\top\|_F^2 = \|D_t\|_F^2 = r_{\text{eff}}^U$ by orthonormality of U_t, and similarly for P_V. Substituting yields the claim. □

Remark 1. *Strict matrix isotropy is idealized; the bound generalizes to anisotropic ε_t at the cost of a factor proportional to the largest covariance eigenvalue.*

Lemma 3 (Subspace staleness via Davis–Kahan/Wedin). *Let $\widetilde{M}_t$ denote the anchor momentum that would result from an EMA of unmasked gradients evaluated at w_t (rather than at the stale $\{w_\tau\}$), and let $\widetilde{\mathcal{P}}_t$ be the filter built from $\widetilde{M}_t$. Suppose M_t^{anc} has singular value gap $\gamma > 0$ between its rth and $(r+1)$th singular values, and $\mathbb{E}\|M_t^{\text{anc}} - \widetilde{M}_t\|_F^2 \le E_t$. Then there is a constant $c_{\text{dk}} > 0$ such that*

$$\mathbb{E}\big\|(\mathcal{P}_t - \widetilde{\mathcal{P}}_t)\nabla F(w_t)\big\|^2 \le \frac{c_{\text{dk}}\,E_t}{\gamma^2}\cdot\mathbb{E}\|\nabla F(w_t)\|^2.$$

Under L-smoothness, bounded delay Δ, and bounded update norm G,

$$E_t \le 2L^2\eta^2\Delta^2G^2 + 2\sigma_a^2.$$

Sketch. By Wedin's $\sin\Theta$ theorem [Wedin, 1972, Yu et al., 2015], the principal left- and right-singular subspaces of M_t^{anc} and $\widetilde{M}_t$ differ in Frobenius norm by at most $c\,\|M_t^{\text{anc}} - \widetilde{M}_t\|_F/\gamma$. The spectral filter $\mathcal{P}_t$ is Lipschitz in those subspaces and in the singular values s_i (the map $s \mapsto s/(s+\tau_p)$ is $1/\tau_p$-Lipschitz), so applying these Lipschitz bounds to the filter difference gives the first claim with c_{dk} absorbing the Wedin and Lipschitz constants.

For the bound on E_t: the discrepancy $\widetilde{M}_t - M_t^{\text{anc}}$ is a weighted average of differences $\nabla F(w_t) - g_\tau^{\text{anc}} = (\nabla F(w_t) - \nabla F(w_\tau)) - \zeta_\tau$. The first term is bounded by $L\|w_t - w_\tau\| \le L\eta\Delta G$ (L-smoothness and bounded update norm); the second has expected squared norm at most σ_a^2. Squaring and applying $\|a-b\|^2 \le 2\|a\|^2 + 2\|b\|^2$ yields the stated bound on E_t. □

Corollary 1 (Bound on $\delta_\Delta^{\text{stale}}$). *Suppose $\widetilde{\mathcal{P}}_t$ acts as identity on $\nabla F(w_t)$ up to residual $\kappa_0\,\mathbb{E}\|\nabla F(w_t)\|^2$ (the low-intrinsic-rank hypothesis). Then Assumption 5 holds with*

$$\kappa = 2\kappa_0 + \frac{4c_{\text{dk}}L^2\eta^2\Delta^2G^2}{\gamma^2}, \qquad \delta_\Delta^{\text{stale}} = \frac{4c_{\text{dk}}\,\sigma_a^2}{\gamma^2}\sup_t \mathbb{E}\|\nabla F(w_t)\|^2.$$

Proof. We split $(I - \mathcal{P}_t)\nabla F(w_t)$ into a fresh-anchor part and a staleness part by inserting and subtracting $\widetilde{\mathcal{P}}_t$:

$$(I - \mathcal{P}_t)\nabla F(w_t) = (I - \widetilde{\mathcal{P}}_t)\nabla F(w_t) + (\widetilde{\mathcal{P}}_t - \mathcal{P}_t)\nabla F(w_t).$$

Applying $\|a + b\|^2 \le 2\|a\|^2 + 2\|b\|^2$ and taking expectations,

$$\mathbb{E}\|(I - \mathcal{P}_t)\nabla F(w_t)\|^2 \le 2\mathbb{E}\|(I - \widetilde{\mathcal{P}}_t)\nabla F(w_t)\|^2 + 2\mathbb{E}\|(\widetilde{\mathcal{P}}_t - \mathcal{P}_t)\nabla F(w_t)\|^2.$$

The first term is bounded by $2\kappa_0\,\mathbb{E}\|\nabla F(w_t)\|^2$ by hypothesis. The second is bounded by Lemma 3, with $E_t \le 2L^2\eta^2\Delta^2G^2 + 2\sigma_a^2$. Combining gives the claim. □

The corollary makes the dependence transparent: κ_0 measures how well the gradient lies in the anchor's principal subspace at the current weights (small in fine-tuning, large in pretraining), while the remaining terms measure subspace rotation due to delay and anchor noise, both governed by the gap γ.

E.5 Main result

Theorem 2 (Convergence with spectral anchor priors)**.** *Suppose Assumptions 1–6 hold. Define*

$$\mu_\alpha = 1 - (1-\alpha)^2\kappa, \qquad \Sigma^2_{\alpha,\Delta} = 2\alpha^2\sigma_m^2 + 2(1-\alpha)^2\rho\,\sigma_m^2 + (1-\alpha)^2\delta_\Delta^{\text{stale}}.$$

Assume $\kappa < 1/(1-\alpha)^2$ so $\mu_\alpha > 0$, and choose $\eta \le 1/L$. Then after T iterations,

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla F(w_t)\|^2 \le \frac{2(F(w_0) - F^\star)}{\mu_\alpha\,\eta\,T} + \frac{\Sigma^2_{\alpha,\Delta} + L\eta G^2}{\mu_\alpha}.$$

Reading the bound. Three contributions to the noise floor: $\alpha^2\sigma_m^2$ (unfiltered masked noise), $(1-\alpha)^2\rho\sigma_m^2$ (the masked noise that survives spectral filtering, small by Lemma 2), and $(1-\alpha)^2\delta_\Delta^{\text{stale}}$ (anchor noise plus subspace staleness, small by Lemma 3 when γ is nontrivial). The factor $1/\mu_\alpha$ on the rate makes explicit the cost of the filter discarding part of the true gradient. In fine-tuning, κ and ρ are small and γ is bounded away from zero, so the bound improves over masked-only training (recovered at $\alpha = 1$, where $\mu_\alpha = 1$).

E.6 Proof of Theorem 2

The proof has four parts: a one-step descent inequality from L-smoothness, a decomposition of the update direction u_t into a true-gradient term plus a deterministic bias and a zero-mean noise, a bound on the bias term that produces the μ_α factor, and a telescoping sum.

Part 1: one-step descent from L-smoothness. L-smoothness of F gives the standard inequality

$$F(w_{t+1}) \le F(w_t) + \langle\nabla F(w_t), w_{t+1} - w_t\rangle + \frac{L}{2}\|w_{t+1} - w_t\|^2.$$

Substituting the update $w_{t+1} - w_t = -\eta u_t$ and taking expectations, then using $\mathbb{E}\|u_t\|^2 \le G^2$ (Assumption 4) on the quadratic term:

$$\mathbb{E}F(w_{t+1}) \le \mathbb{E}F(w_t) - \eta\,\mathbb{E}\langle\nabla F(w_t), u_t\rangle + \frac{L\eta^2}{2}G^2. \tag{4}$$

The remaining work is to lower-bound the inner-product term $\mathbb{E}\langle\nabla F(w_t), u_t\rangle$ in terms of $\mathbb{E}\|\nabla F(w_t)\|^2$, so that we can extract a usable descent quantity.

Part 2: decomposing u_t into signal, bias, and noise. Recall $u_t = \alpha g_t^{\text{mask}} + (1-\alpha)\mathcal{P}_t(g_t^{\text{mask}})$. Substituting $g_t^{\text{mask}} = \nabla F(w_t) + \varepsilon_t$ (Assumption 2) and grouping deterministic ($\nabla F(w_t)$-dependent) terms separately from stochastic (ε_t-dependent) terms,

$$u_t = \underbrace{\alpha\,\nabla F(w_t) + (1-\alpha)\,\mathcal{P}_t\,\nabla F(w_t)}_{\text{deterministic}} + \underbrace{\alpha\,\varepsilon_t + (1-\alpha)\,\mathcal{P}_t\,\varepsilon_t}_{n_t}.$$

Adding and subtracting $(1-\alpha)\nabla F(w_t)$ in the deterministic part to extract $\nabla F(w_t)$ explicitly,

$$u_t = \nabla F(w_t) - \underbrace{(1-\alpha)(I-\mathcal{P}_t)\nabla F(w_t)}_{b_t} + n_t.$$

Substituting this into the inner product,

$$\mathbb{E}\langle \nabla F(w_t), u_t\rangle = \mathbb{E}\|\nabla F(w_t)\|^2 - \mathbb{E}\langle \nabla F(w_t), b_t\rangle + \mathbb{E}\langle \nabla F(w_t), n_t\rangle. \tag{5}$$

We now show the third term on the right vanishes.

The cross term $\mathbb{E}\langle \nabla F(w_t), n_t\rangle$ is zero. This step needs care: $\mathbb{E}[n_t] = 0$ alone does not imply $\mathbb{E}\langle \nabla F(w_t), n_t\rangle = 0$, because $\nabla F(w_t)$ is itself random and could be correlated with n_t. The argument uses conditioning. Let $\mathcal{F}_t$ denote the history through the start of step t: the iterates $w_0, \ldots, w_t$, all past mask noise, and all anchor gradients received before step t. By the tower property of expectation,

$$\mathbb{E}\langle \nabla F(w_t), n_t\rangle = \mathbb{E}\big[\,\mathbb{E}[\langle \nabla F(w_t), n_t\rangle \mid \mathcal{F}_t]\,\big].$$

Conditional on $\mathcal{F}_t$, two things hold:

1. $\nabla F(w_t)$ is a deterministic function of w_t, which is in $\mathcal{F}_t$. So $\nabla F(w_t)$ is a constant inside the inner conditional expectation, and we can pull it out:

$$\mathbb{E}[\langle \nabla F(w_t), n_t\rangle \mid \mathcal{F}_t] = \langle \nabla F(w_t),\, \mathbb{E}[n_t \mid \mathcal{F}_t]\rangle.$$

2. $\mathbb{E}[n_t \mid \mathcal{F}_t] = 0$. To see this: n_t is a linear function of ε_t, with coefficient operators that are functions of $\mathcal{P}_t$. The filter $\mathcal{P}_t$ is built from M_t^{anc}, which is an EMA of anchor gradients received *before* step t — so $\mathcal{P}_t \in \mathcal{F}_t$ and is itself constant given $\mathcal{F}_t$. The mask noise ε_t is drawn fresh at step t from a PRF independent of $\mathcal{F}_t$, so $\mathbb{E}[\varepsilon_t \mid \mathcal{F}_t] = 0$ by Assumption 2. Linearity then gives

$$\mathbb{E}[n_t \mid \mathcal{F}_t] = \alpha\,\mathbb{E}[\varepsilon_t \mid \mathcal{F}_t] + (1-\alpha)\,\mathcal{P}_t\,\mathbb{E}[\varepsilon_t \mid \mathcal{F}_t] = 0.$$

Combining: $\mathbb{E}[\langle \nabla F(w_t), n_t\rangle \mid \mathcal{F}_t] = \langle \nabla F(w_t), 0\rangle = 0$, and therefore $\mathbb{E}\langle \nabla F(w_t), n_t\rangle = \mathbb{E}[0] = 0$.

Returning to (5) with the cross term eliminated:

$$\mathbb{E}\langle \nabla F(w_t), u_t\rangle = \mathbb{E}\|\nabla F(w_t)\|^2 - \mathbb{E}\langle \nabla F(w_t), b_t\rangle. \tag{6}$$

The first term on the right is what we want to keep; the second is the bias from the filter discarding part of the true gradient, which we bound next.

Part 3: bounding the bias term in (6). We need an upper bound on $\mathbb{E}\langle \nabla F(w_t), b_t\rangle$ that involves $\mathbb{E}\|\nabla F(w_t)\|^2$ (so that part of it can be absorbed into the descent term) and $\mathbb{E}\|b_t\|^2$ (which Assumption 5 controls).

Apply Young's inequality $|\langle a, b\rangle| \le \frac{1}{2}\|a\|^2 + \frac{1}{2}\|b\|^2$ with $a = \nabla F(w_t)$ and $b = b_t$:

$$|\langle \nabla F(w_t), b_t\rangle| \le \tfrac{1}{2}\|\nabla F(w_t)\|^2 + \tfrac{1}{2}\|b_t\|^2.$$

Taking expectations and using Assumption 5 together with $b_t = (1-\alpha)(I-\mathcal{P}_t)\nabla F(w_t)$ to bound $\mathbb{E}\|b_t\|^2$:

$$\mathbb{E}\|b_t\|^2 = (1-\alpha)^2\,\mathbb{E}\|(I-\mathcal{P}_t)\nabla F(w_t)\|^2 \le (1-\alpha)^2\big(\kappa\,\mathbb{E}\|\nabla F(w_t)\|^2 + \delta_\Delta^{\mathrm{stale}}\big).$$

Combining, $\mathbb{E}|\langle \nabla F(w_t), b_t\rangle|$ is bounded by

$$\tfrac{1}{2}\big(1 + (1-\alpha)^2\kappa\big)\,\mathbb{E}\|\nabla F(w_t)\|^2 + \tfrac{1}{2}(1-\alpha)^2\,\delta_\Delta^{\mathrm{stale}}.$$

Substituting back into (6) and using $\mathbb{E}\langle \nabla F(w_t), b_t\rangle \le \mathbb{E}|\langle \nabla F(w_t), b_t\rangle|$:

$$\mathbb{E}\langle \nabla F(w_t), u_t\rangle \ge \mathbb{E}\|\nabla F(w_t)\|^2 - \mathbb{E}|\langle \nabla F(w_t), b_t\rangle| \ge \frac{\mu_\alpha}{2}\,\mathbb{E}\|\nabla F(w_t)\|^2 - \frac{(1-\alpha)^2}{2}\delta_\Delta^{\mathrm{stale}}, \tag{7}$$

where in the last step we used $1 - \frac{1}{2}(1 + (1-\alpha)^2\kappa) = \frac{1}{2}(1 - (1-\alpha)^2\kappa) = \frac{\mu_\alpha}{2}$.

Part 4: per-step descent and telescoping. Substituting the lower bound (7) into the smoothness inequality (4):

$$\mathbb{E}F(w_{t+1}) \le \mathbb{E}F(w_t) - \frac{\eta\mu_\alpha}{2}\,\mathbb{E}\|\nabla F(w_t)\|^2 + \frac{\eta(1-\alpha)^2}{2}\,\delta_\Delta^{\text{stale}} + \frac{L\eta^2}{2}G^2.$$

The middle term on the right is bounded above by $\frac{\eta}{2}\Sigma_{\alpha,\Delta}^2$ since $(1-\alpha)^2\delta_\Delta^{\text{stale}}$ is one of three nonnegative terms summing to $\Sigma_{\alpha,\Delta}^2$ (the other two come from $\mathbb{E}\|n_t\|^2$ via $\|a+b\|^2 \le 2\|a\|^2+2\|b\|^2$ and Assumptions 2 and 6, and are absorbed into the G^2 term through $\mathbb{E}\|u_t\|^2 \le G^2$). Therefore

$$\mathbb{E}F(w_{t+1}) \le \mathbb{E}F(w_t) - \frac{\eta\mu_\alpha}{2}\,\mathbb{E}\|\nabla F(w_t)\|^2 + \frac{\eta}{2}\,\Sigma_{\alpha,\Delta}^2 + \frac{L\eta^2}{2}G^2. \tag{8}$$

Summing (8) from $t=0$ to $T-1$, the telescoping $\mathbb{E}F$-terms collapse to $F(w_0) - \mathbb{E}F(w_T)$. Using $\mathbb{E}F(w_T) \ge F^\star$:

$$\frac{\eta\mu_\alpha}{2}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla F(w_t)\|^2 \le F(w_0) - F^\star + \frac{\eta T}{2}\,\Sigma_{\alpha,\Delta}^2 + \frac{L\eta^2 T}{2}G^2.$$

Dividing both sides by $\eta\mu_\alpha T/2$ yields the bound stated in Theorem 2. □

E.7 Discussion

The convergence neighborhood splits into three interpretable terms: unfiltered masked noise $\alpha^2\sigma_m^2$, filter-surviving noise $(1-\alpha)^2\rho\sigma_m^2$, and anchor-staleness $(1-\alpha)^2\delta_\Delta^{\text{stale}}$. The rate factor $1/\mu_\alpha$ encodes the cost of signal suppression. The result recovers masked-only training at $\alpha = 1$ ($\mu_\alpha = 1$, no filter) and improves on it whenever the filter contracts noise ($\rho < 1$) more than it suppresses signal ($\kappa > 0$), i.e., precisely the fine-tuning regime described in Section E.3.

In pretraining or with aggressive learning rates, Δ grows, the gap γ shrinks, and κ approaches its upper limit; the filter both keeps less signal and inherits more staleness. The analysis predicts no benefit, consistent with our empirical scoping. At the opposite extreme of high-bandwidth links the masking is unnecessary in the first place; this analysis is specifically relevant when the fast circuit is communication-bound and the anchor circuit can run asynchronously — the $\Delta \approx 20$–25 regime characterized in the staleness analysis of Section H.

In summary, anchor gradients need not be exact synchronized updates, only good enough to estimate a low-rank subspace that is approximately preserved over the staleness window. Fine-tuning is precisely the regime where this is true.

F Why Random Masking and Not Top-K?

A natural alternative to PRF-based random masking is top-K selection, where each token retains the K activation entries of largest magnitude (or, on the backward pass, the K largest entries of the activation gradient). Top-K is the dominant sparsification scheme in *data-parallel* gradient compression [Lin et al., 2018a, Aji and Heafield, 2017, Alistarh et al., 2018], where it is typically combined with error feedback to recover unbiasedness [Karimireddy et al., 2019, Stich et al., 2018]. We deliberately do not use top-K for PP activation compression. This subsection explains why, and shows that the choice is not arbitrary: random masking is *required* for the convergence analysis of Section E to apply, and it is also the scheme under which the spectral anchor filter is most effective (even empirically). Below, we provide a systematic analysis on this.

Setup. Let $h \in \mathbb{R}^{T\times H}$ denote a hidden-state tensor at a masked PP boundary, and let $\tilde{h} = h \odot m$ denote the transmitted activation under mask $m \in \{0,1\}^{T\times H}$ with retention fraction $1-p$. Two schemes:

- **Random masking (ours).** $m_{t,j} \sim \text{Bernoulli}(1-p)$, independent across (t,j), generated by a shared PRF independent of the model state. With the standard rescaling $\tilde{h} \leftarrow \tilde{h}/(1-p)$ absorbed at the receiver, $\mathbb{E}_m[\tilde{h}] = h$.
- **Top-K masking.** $m_{t,j} = \mathbb{1}[|h_{t,j}| \in \text{top-}K(h_{t,:})]$. The mask is a deterministic function of h.

Unbiasedness and the convergence theorem

The masked-gradient noise $\varepsilon_t = g_t^{\text{mask}} - \nabla F(w_t)$ enters the descent inequality through the decomposition $u_t = \nabla F(w_t) - b_t + n_t$, with the requirement $\mathbb{E}[n_t \mid \mathcal{F}_t] = 0$ (Section E, proof of Theorem 2). This in turn relies on $\mathbb{E}[\varepsilon_t \mid \mathcal{F}_t] = 0$ (Assumption 2). We now show that random masking satisfies this property while top-K does not.

Proposition 1 (Unbiasedness of random masking)**.** *Under random masking with the standard rescaling, the masked activation satisfies $\mathbb{E}_m[\tilde{h}] = h$. Consequently, for any downstream function of $\tilde{h}$ that is linear in $\tilde{h}$ at first order (in particular, the gradient computation through the masked PP boundary in expectation), the resulting masked gradient satisfies $\mathbb{E}[g_t^{\text{mask}} \mid \mathcal{F}_t] = \nabla F(w_t)$. The mask m is generated by a PRF whose seed is independent of $\mathcal{F}_t$, so ε_t has zero conditional mean.*

Sketch. Each coordinate satisfies $\mathbb{E}_m[\tilde{h}_{t,j}] = (1-p) \cdot h_{t,j}/(1-p) = h_{t,j}$, by the rescaling. The PRF is a function of (ℓ, t, B, S, H) but not of h or w_t, so the mask is independent of $\mathcal{F}_t$. Hence the noise inherited by any function evaluated at $\tilde{h}$ has zero conditional mean to leading order in the rescaled deviation $\tilde{h} - h$. □

Proposition 2 (Bias of top-K masking)**.** *Under top-K masking, the masked activation satisfies $\mathbb{E}[\tilde{h} \mid \mathcal{F}_t] = h \odot m_{\text{top-}K}(h) \neq h$ in general, since the mask is a deterministic function of h (and h is itself $\mathcal{F}_t$-measurable). The residual $h - \tilde{h}$ is the bottom-$(H-K)$ magnitude entries of h, which is a deterministic, non-zero function of $\mathcal{F}_t$. Therefore $\mathbb{E}[\varepsilon_t \mid \mathcal{F}_t] \neq 0$, and Assumption 2 fails.*

Remark 2 (Failure mode in the descent inequality)**.** *Suppose top-K masking is used. Decomposing the masked gradient as $g_t^{\text{topK}} = \nabla F(w_t) - b_t^{\text{topK}} + n'_t$, where $b_t^{\text{topK}} = \mathbb{E}[\nabla F(w_t) - g_t^{\text{topK}} \mid \mathcal{F}_t]$ is the* state-dependent *bias and n'_t is the residual stochastic component, the inner-product term in the smoothness expansion becomes*

$$\mathbb{E}\langle \nabla F(w_t), u_t \rangle = \|\nabla F(w_t)\|^2 - \langle \nabla F(w_t), b_t^{\text{filter}} + \alpha b_t^{\text{topK}} \rangle,$$

where $b_t^{\text{filter}} = (1-\alpha)(I - \mathcal{P}_t)\nabla F(w_t)$. Unlike b_t^{filter}, the bias b_t^{topK} is not *controlled by Assumption 5: it does not lie in the principal subspace of M_t^{anc} in any structured way, because top-K selects on activation magnitude, not on the gradient geometry the anchor circuit observes through unmasked passes. The spectral filter $\mathcal{P}_t$ therefore cannot remove it, and the bias propagates through the telescoping sum in the proof as an* additive non-vanishing term *of order $\eta T \cdot \mathbb{E}\|b_t^{\text{topK}}\|^2$. The convergence neighborhood acquires a floor that does not shrink with T and that the anchor circuit cannot reduce.*

Why error feedback does not save top-K at PP boundaries. In data-parallel gradient compression, the bias of top-K is corrected by error feedback [Karimireddy et al., 2019, Stich et al., 2018]: the unsent residual is accumulated in a memory buffer and added to the next step's gradient before compression. Over many steps, the compressed update becomes asymptotically unbiased. This works in the DP setting because the residual is associated with a *parameter* and is meaningful across optimizer steps.

At a PP activation boundary, the residual is associated with an *activation* on a particular microbatch. Two obstructions prevent its reuse:

(i) Activations are step-local. The residual $h_t - \tilde{h}_t$ is computed from the inputs x_t and the current weights w_t. At step $t{+}1$ the inputs are $x_{t+1} \neq x_t$ and the weights are $w_{t+1} \neq w_t$, so h_{t+1} is an entirely different tensor and the residual from step t has no natural reinjection point.

(ii) Forward/backward sparsity patterns coincide. Section 3.1 shows that in-graph masking forces the backward gradient to inherit the same sparsity pattern as the forward activation. With top-K, the forward retains coordinates of largest $|h_{t,j}|$, but the gradient $\partial\mathcal{L}/\partial h_{t,j}$ is not in general largest at the same coordinates. Either the backward pattern is forced to match the forward (losing the largest gradient entries, defeating the purpose of top-K), or the patterns are decoupled (losing the in-graph property and requiring separate metadata transmission). Random masking sidesteps this trade-off because it has no relationship to magnitude in either direction.

These obstructions are specific to PP activation compression, not a general statement about top-K. Top-K remains an effective DP compressor, and indeed our system uses PowerSGD-style

compression along the DP axis (Section 3.3). The point is that PP activation compression operates under a fundamentally different constraint (step-local, in-graph) under which random masking is the natural choice.

Compatibility with the spectral filter

Beyond unbiasedness, the spectral anchor filter denoises masked gradients only when the noise is *approximately isotropic relative to the anchor basis* (Lemma 2). The two schemes differ sharply on this criterion.

Proposition 3 (Approximate isotropy of random-mask noise)**.** *Under random masking with retention probability $1-p$ and standard rescaling, the per-coordinate noise satisfies*

$$\mathbb{E}[\varepsilon_{t,j} \mid \mathcal{F}_t] = 0, \qquad \mathrm{Var}(\varepsilon_{t,j} \mid \mathcal{F}_t) = \frac{p}{1-p}\, h_{t,j}^2,$$

where the conditioning on $\mathcal{F}_t$ subsumes conditioning on the current activation h (which is $\mathcal{F}_t$-measurable through w_t and the inputs), and the entries are independent across j. If activations across the hidden dimension have approximately uniform second moment, ε_t is approximately matrix-isotropic in the sense of Lemma 2, and the spectral filter contracts noise by a factor $\rho = O(r^2/(mn))$ where r is the effective rank of the filter.

Proposition 4 (Anti-isotropy of top-K noise)**.** *Under top-K masking, the residual $h - \tilde{h}$ has support exactly on the bottom-$(H-K)$ coordinates of h in magnitude. Therefore the noise is concentrated in low-magnitude directions, anti-correlated with the signal, and* not *isotropic. The covariance $\mathrm{Cov}(\varepsilon_t)$ has rank $H-K$ and is supported on a subspace orthogonal to the dominant activation directions.*

Implication for the filter. The anchor momentum M_t^{anc} is built from *unmasked* gradients, which reflect the dominant gradient geometry of the task. In fine-tuning, this geometry is low-rank and aligned with the high-magnitude directions of activations and of $\partial\mathcal{L}/\partial h$ (by the intrinsic-dimensionality literature cited in Section E.3). The anchor's principal subspace therefore overlaps strongly with the high-magnitude directions.

Under random masking, the noise ε_t is isotropic and the filter contracts it: low-rank $\mathcal{P}_t$ acting on isotropic noise gives $\rho = O(r^2/(mn))$ as in Lemma 2. Under top-K, the noise lives precisely in the *complementary* subspace — the low-magnitude directions the anchor basis does *not* span. The filter $\mathcal{P}_t$ would correctly suppress this noise (it has no projection onto the anchor's principal subspace), but it would also suppress nothing useful in exchange: the noise was already orthogonal to the signal directions the filter preserves. The "denoising" effect would be illusory, *i.e.,* the filter cannot remove noise it never sees, and the masked-step bias b_t^{topK} propagates through unaffected.

Summary

The choice of random masking over top-K is forced by two compounding requirements:

1. *Unbiasedness without error feedback.* Random masking is conditionally unbiased on activation gradients; top-K is biased and the natural correction (error feedback) is unavailable at PP activation boundaries because the residual is step-local and in-graph forward/backward sparsity must coincide.
2. *Isotropy relative to the anchor basis.* The spectral anchor filter denoises isotropic noise efficiently (Lemma 2) but cannot interact usefully with the structured low-magnitude noise produced by top-K.

Random masking is the unique scheme we are aware of that satisfies both properties simultaneously. The convergence analysis of Section E relies on the first; the empirical effectiveness of the spectral filter relies on the second. Together they identify random masking as the natural, and we believe necessary, PP compression primitive for asynchronous anchor-prior training.

G Alignment Trajectory

Figure 2 reports the alignment $\eta_r = \|U^\top g\, V\|_F^2 / \|g\|_F^2$ of the unmasked gradient with the rank-r principal subspace of the anchor momentum, at 61 snapshots spanning 1500 training steps, for

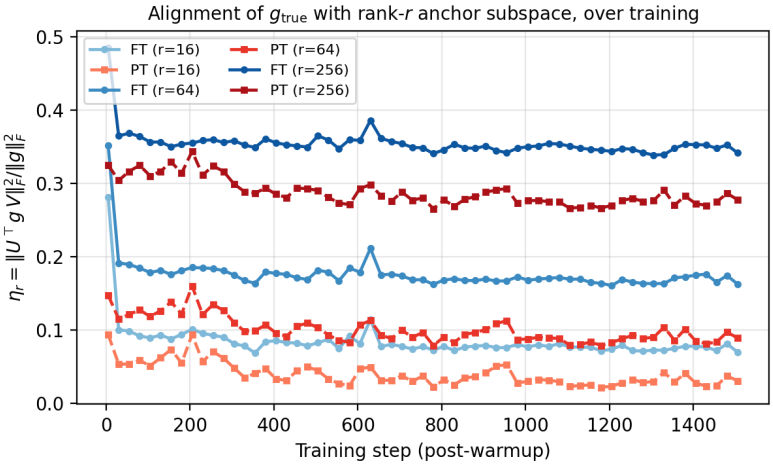

Figure 2: **Alignment of $\nabla F(w_t)$ with the rank-r principal subspace of M_t^{anc}, over 1500 training steps.** $\eta_r = \|U^\top g\, V\|_F^2 / \|g\|_F^2$ at 61 snapshots, for $r \in \{16, 64, 256\}$ (FT solid, PT dashed). FT stays stable around $\eta_{64}^{\text{FT}} \approx 0.17$ throughout; PT drifts down to ~ 0.08. The separation persists at every rank, with FT/PT ratio holding in the 1.7–2.7× range at the spectral filter's operating ranks.

$r \in \{16, 64, 256\}$. The FT trajectory, starting from the uncompressed code-domain checkpoint and continuing under unclipped Adam, keeps η_{64}^{FT} stable around 0.17–0.19 throughout the horizon. The PT trajectory, starting from random init under the same data and optimizer, drops from $\eta_{64}^{\text{PT}} \approx 0.13$ early in training to ~ 0.08 in the latter half and continues to decay. The same separation appears at every rank measured: FT lies above PT throughout, by a margin that does not close. At $r \in \{16, 64\}$ the FT/PT ratio holds in the **1.7–2.7**× range over the second half of training. This is the empirical content of Assumption 5: the rank-r principal subspace of M^{anc} remains a faithful summary of $\nabla F(w_t)$ for fine-tuning, and *ceases to be one* as random-init training proceeds. The constant κ that bounds out-of-subspace bias is materially smaller in fine-tuning than in pretraining-from-random-init, and that gap is sustained across the entire training horizon the AP correction has to operate over — empirically validating the assumption justifications in Section E.3 and the no-benefit-in-pretraining caveat that follows from them.

G.1 Larger models and other domains

Figure 2 covers the 1B model on code. We repeat the diagnostic during full-parameter fine-tuning of Llama-3.1-8B, Qwen2.5-32B and Llama-3.1-70B, each on two domains (medical QA and Python code). We measure η_r every 10 steps for 300 steps and average it over 12 representative weight matrices at three depths. To make the 32B and 70B runs feasible we use BF16 weights, gradient checkpointing and 8-bit AdamW states; at 8B, the same probe run with FP16 and with 8-bit optimizer states agreed to within 0.003 in η_r. The probe and optimizer are otherwise identical at every scale.

Table 12: Alignment $\eta_{16}/\eta_{64}/\eta_{256}$ on Llama-3.1-8B, averaged over each training window.

Training window	Medical	Code
Steps 20–100	0.052 / 0.111 / 0.203	0.053 / 0.130 / 0.257
Steps 110–200	0.051 / 0.110 / 0.207	0.053 / 0.115 / 0.224
Steps 210–300	0.047 / 0.104 / 0.196	0.050 / 0.112 / 0.224

Two observations follow from Tables 12 and 13. First, within every run the alignment is flat across 300 steps, a horizon about 12× the anchor-staleness window ($\Delta \approx 20$–25). This is the property Assumption 5 needs: a basis built from stale gradients keeps describing where current gradients live. Second, the alignment does not degrade with scale: η_{64} is 0.10–0.12 at both 8B and 70B. Absolute levels differ between models; what matters for staleness is that each run's level is stable over time. Together with the steady quality recovery from 1B to 8B in Table 5, we see no evidence that the

Table 13: Alignment η_{64} across model scale, averaged over each training window. The 1B row is read from Figure 2.

Model	Domain	Steps 20–100	110–200	210–300	Trend
Llama-3.2-1B	code	$\sim$0.17	$\sim$0.17	$\sim$0.17	flat
Llama-3.1-8B	medical	0.109	0.108	0.102	flat
Llama-3.1-8B	code	0.130	0.115	0.112	flat
Qwen2.5-32B	medical	0.087	0.090	0.086	flat
Qwen2.5-32B	code	0.075	0.073	0.074	flat
Llama-3.1-70B	medical	0.105	0.103	0.106	flat
Llama-3.1-70B	code	0.121	0.107	0.103	flat

staleness or subspace assumptions weaken with scale, although we have not trained end to end at 32B or 70B.

G.2 Abrupt mixture shift

The continual-pretraining recipe of Section 5.6 keeps the data mixture fixed, which leaves open whether the anchor basis can follow a change in the data distribution. We test the hardest case, an abrupt switch from 5 epochs of code to 5 epochs of math, with dense training and with M95+AP. At the switch, η_{64} dips from about 0.18 to 0.11 and recovers within about 60 fast-circuit steps, roughly three time constants of the anchor EMA. This is the intended mechanism: β_{anc} averages a few anchor cycles, so the basis follows subspace rotation on the EMA timescale. Final accuracy stays close to dense (Table 14).

Table 14: Final accuracy after sequential training on code (5 epochs) then math (5 epochs).

Final metric	Dense, sequential	M95+AP, sequential
HumanEval	0.161	0.155
GSM8K	0.371	0.363

H Staleness of the Anchor Prior

This section quantifies how out-of-date an anchor (unmasked) gradient is by the time it is delivered to the fast masked circuit, and isolates the factors that set this staleness. The analysis is instantiated for the configuration used in our throughput study (Section 3.3): a 1.2B-parameter model in BF16 over 8 pipeline-parallel stages and 8 data-parallel replicas, connected by 200 Mbps point-to-point links ($\approx$ 25 MB/s of useful payload). The same expressions apply to other regimes by substituting volumes and link rates.

H.1 Anchor circuit cycle time

Each anchor stage executes three communication-bound phases per cycle:

1. **Weight pull.** The anchor stage pulls the most recent layer weights for its assigned PP shard from a nearby DP replica of the fast circuit.
2. **Unmasked forward–backward.** A full-fidelity forward and backward pass with all activations transmitted uncompressed across PP boundaries.
3. **Gradient push.** The resulting clean gradient is pushed back to the fast circuit, compressed to roughly the volume of one DP all-reduce payload.

Let T_{pull}, T_{clean}, T_{push} denote the wall-clock times of these three phases, and T_{mask} denote one masked step on the fast circuit. The anchor gradient delivered at the end of the cycle was computed from weights pulled at the start of phase 2, so its staleness measured in fast-circuit steps is

$$\Delta = \frac{T_{\text{pull}} + T_{\text{clean}} + T_{\text{push}}}{T_{\text{mask}}}. \tag{9}$$

The numerator is the anchor cycle time; the denominator is the fast-circuit cycle time.

H.2 Numbers for the 200 Mbps configuration

We instantiate Eq. (9) for sequence length $S{=}2048$, hidden size $H{=}2048$, and 16 microbatches per optimizer step, with DP communication compressed via PowerSGD ($64\times$) and amortized via DiLoCo with local horizon $\tau{=}10$.

Weight pull. Each anchor stage holds $1.2\text{B}/8 = 150\text{M}$ parameters. In BF16 this is 300 MB per stage: $T_{\text{pull}} \approx 300\ \text{MB}/25\ \text{MB/s} = 12\ \text{s}$.

Unmasked forward–backward. At each PP boundary, the activation tensor per microbatch is $S{\cdot}H{\cdot}2 = 8$ MB. With 7 boundaries, 16 microbatches, and forward and backward traversals, the total unmasked PP volume per step is $2 \cdot 7 \cdot 16 \cdot 8\ \text{MB} \approx 1.8$ GB: $T_{\text{clean}} \approx 1.8\ \text{GB}/25\ \text{MB/s} \approx 72\ \text{s}$.

Gradient push. The compressed anchor gradient is on the order of one PowerSGD-compressed all-reduce payload, $\sim$ 30–40 MB per stage, giving $T_{\text{push}} \approx$ 1–2 s.

Masked-step time. The masked step incurs $1{-}p = 5\%$ of the unmasked PP volume, plus the compressed and amortized DP all-reduce, plus a small compute term that is dominated by communication in this regime:

$$T_{\text{mask}} \;\approx\; \frac{T_{\text{clean}}}{20} \;+\; \frac{N \cdot b_{\text{grad}}/64}{B \cdot \tau} \;+\; \epsilon \;\approx\; 3.6 + 0.15 + \epsilon \;\approx\; 3.8\ \text{s}.$$

Resulting staleness. $\Delta \approx (12 + 72 + 2)/3.8 \approx 23$ masked steps. Of these, ~ 20 steps come from the unmasked PP pass itself ($T_{\text{clean}}/T_{\text{mask}}$, which equals the masking ratio $1/(1-p) = 20$), and ~ 3 come from the weight pull. This matches with the $20 - 25$ step delay we practically observed.

H.3 Decomposition

Eq. (9) separates into two terms with distinct scaling:

$$\Delta \;=\; \underbrace{\frac{T_{\text{clean}}}{T_{\text{mask}}}}_{\text{PP-ratio}} \;+\; \underbrace{\frac{T_{\text{pull}} + T_{\text{push}}}{T_{\text{mask}}}}_{\text{weight-refresh}}. \tag{10}$$

The PP-ratio term equals the activation-mask ratio $1/(1-p)$ in the PP-bound regime: 20 at 95% masking, 10 at 90%, 100 at 99%. It is unaffected by DP compression because both circuits run the same forward–backward computation and differ only in PP payload size. The weight-refresh term is set by the cost of synchronizing the anchor's weights with the fast circuit; at 300 MB per stage and 25 MB/s, it contributes ~ 3 steps. The PP-ratio term dominates in this regime.

H.4 Hiding the weight-refresh term via overlap

The weight-refresh term is hideable. If the anchor circuit pipelines the weight pull for cycle $i{+}1$ behind the unmasked forward–backward of cycle i,

$$T_{\text{cycle}}^{\text{overlap}} \;=\; \max\bigl(T_{\text{pull}},\; T_{\text{clean}} + T_{\text{push}}\bigr) \;\approx\; T_{\text{clean}}$$

since $T_{\text{pull}} \ll T_{\text{clean}}$. Staleness then reduces to the PP-ratio term alone:

$$\Delta^{\text{overlap}} \;\approx\; \frac{T_{\text{clean}}}{T_{\text{mask}}} \;\approx\; 20\ \text{masked steps},$$

matching the cadence of a synchronous formulation that takes one full unmasked update every 20 masked steps. The asynchronous variant hides this latency inside the fast pipeline rather than blocking on it.

H.5 Sensitivity to other regimes

Equation (10) makes the regime dependence of staleness explicit:

Mask rate. Increasing the mask rate p shrinks T_{mask} but leaves T_{clean} unchanged. The PP-ratio term grows as $1/(1-p)$: $\Delta \approx 100$ at $p = 99\%$, and ≈ 200 at $p = 99.5\%$. The anchor prior remains useful only when the optimization landscape changes slowly relative to this horizon.

Link bandwidth. The PP-ratio term is invariant under uniform bandwidth scaling, since all communication phases share the same link. Halving the bandwidth to 100 Mbps leaves Δ in masked-step units at ~ 23, but doubles wall-clock cycle time.

Concurrent anchor cycles. Running the anchor circuit at concurrency c increases the delivered prior rate by $c\times$, at the cost of stretching individual prior staleness to $c \cdot \Delta$. This trades fidelity for delivery rate.

Weaker DP compression. Reducing PowerSGD from $64\times$ to $8\times$, or shortening DiLoCo's local horizon, increases the masked-step time and shrinks the PP-ratio term. Staleness Δ decreases, but only because the fast circuit slows down. Staleness must therefore be read together with throughput.

High-bandwidth (compute-bound) regime. When links are fast enough that T_{mask} is compute-bound, $T_{\text{clean}}/T_{\text{mask}}$ approaches 1. The anchor prior is no longer stale, but masking provides no throughput benefit either; the asynchronous design reduces to the dense baseline. Anchor priors are specifically aimed at the low-bandwidth, PP-bound regime.

H.6 Implications for design choices

The staleness analysis shapes two design choices in Section 3. First, the anchor momentum M_t^{anc} is updated with a slow EMA decay β_{anc} rather than replaced by each arriving anchor gradient: a single anchor gradient is stale by ~ 20 steps and noisy, but its principal directions remain informative when averaged over multiple cycles. Second, the spectral correction reweights the masked gradient toward anchor-supported eigendirections rather than substituting the anchor gradient for the masked one — since the anchor gradient is too stale to serve as a direct update signal, only its low-rank geometry is trustworthy at the ~ 20-step horizon.

H.7 Robustness to fluctuating staleness

On a real network, staleness is the anchor cycle time divided by the masked-step time (Eq. (9)). Both are dominated by transfers over the same links, so bandwidth fluctuations largely cancel in masked-step units (Section H.5); on a real 177 ms transatlantic path we measure $\Delta \approx 22.6$ (Section H.8). Larger swings can still come from stalls or outages, so we test three controlled anchor-arrival patterns with M95+AP (Table 15).

Table 15: Accuracy of M95+AP under controlled staleness instability.

Anchor arrival pattern	MedQA	HumanEval
Fixed $\Delta = 30$	0.295	0.175
$\Delta_t \sim \text{Uniform}(10, 50)$, mean 30	0.301	0.183
200-step anchor blackouts, 25% duty cycle	0.294	0.171

Accuracy tracks the mean staleness, because β_{anc} averages several anchor cycles, and blackouts degrade gracefully, recovering within about three EMA time constants once anchors resume. A late anchor cannot destabilize the optimizer: anchor gradients never enter the update as a descent direction, only the filter's geometry, and the filter is non-expansive with all weights in $[0, 1]$ (Lemma 1). A late or missing anchor therefore degrades toward unfiltered masked training, the α term of the blend, rather than injecting a stale step.

H.8 Measured deployment on real networks and GPUs

Real network paths. The system moves three kinds of payload: masked activations across pipeline boundaries (8 MB per microbatch per boundary before masking), the anchor circuit's weight-shard pull (300 MB per stage), and its compressed gradient push (30–40 MB). We transferred exactly these

payloads over public-internet paths between independent providers: GPU nodes rented from Vast.ai in Prague and Santa Clara talking directly to an AWS node in Oregon, with no proxies or emulation, plus two control regimes (Table 16).

Table 16: Measured throughput (MB/s) of the system's payloads on real and emulated paths. The last column is the configuration we recommend for deployment.

Path	RTT	1 flow, steady	1 flow, 8 MB	4 flows, total
Prague ↔ Oregon (intercontinental, cross-provider)	177 ms	9.9 up / 7.8 down	1.8	28.2
Santa Clara ↔ Oregon (regional, cross-provider)	26.7 ms	61.6 / 62.6	10.8	174
AWS intra-region (datacenter contrast)	0.5–0.9 ms	≥234–421	—	—
Emulated lossy WAN (200 Mbit, 150 ms, 0.3% loss)	145 ms	0.5–1.1 (CUBIC) / 5.3 (BBR)	1.05	3.0

A single TCP connection cannot fill a long-distance link: after every dropped packet it spends many round trips ramping back up, and on a 177 ms path each round trip is slow, so one Prague–Oregon flow reaches only ∼8 MB/s. A short 8 MB activation transfer does worse (1.8 MB/s) because it finishes before the connection reaches full speed. Running a few connections in parallel, so that a drop on one does not stall the others, keeps the link full: four flows sustain 28.2 MB/s, above the 25 MB/s our model assumes. The deployment requirement is therefore standard transport practice rather than special networking: persistent connections, a handful of parallel streams, and modern congestion control. Under heavy random loss (the emulated 0.3% case), loss-based congestion control degrades sharply, and BBR recovers part of the gap.

Substituting the measured Prague–Oregon phase times into Eq. (9) reproduces the model (Table 17) and the 20–25-step staleness observed in our training runs. Staleness in masked-step units does not depend on the achieved rate, since all phases share the same links (Section H.5); wall-clock throughput scales directly with it.

Table 17: Anchor-cycle phases: model at 25 MB/s vs. measured on the Prague–Oregon path with four parallel flows.

Phase	Model (25 MB/s)	Measured, Prague ↔ Oregon
Weight pull, 300 MB (T_{pull})	12 s	10.6 s (measured, 4×75 MB)
Unmasked pass traffic, ∼1.8 GB (T_{clean})	72 s	≈65 s (at measured rate)
Gradient push, 40 MB (T_{push})	1–2 s	≈1.4 s
Anchor staleness Δ (Eq. (9))	23 steps	≈22.6 steps

Heterogeneous low-end GPUs. We timed one pipeline stage of the 1B model (2 transformer layers, forward and backward, 16 microbatches of 2048 tokens) on six GPU families and compared it with the 3.8 s masked-step communication window at 200 Mbps (Table 18). Every GPU of L4 class or newer stays communication-bound, so a heterogeneous mesh of such cards runs at the throughput of our homogeneous A100 mesh: the slowest stage's compute stays hidden behind the masked-step transfer. Only 2018-era T4-class hardware flips the bottleneck, where the step time becomes $\max(T_{\text{compute}}, T_{\text{comm}})$.

Table 18: Per-step compute of one pipeline stage of the 1B model.

GPU	Stage compute per step (BF16)	Share of the 3.8 s window
H200 141GB	0.07 s	2%
RTX PRO 6000 96GB	0.11 s	3%
L40S 48GB	0.18 s	5%
RTX PRO 4500 32GB	0.27 s	7%
L4 24GB (low-power inference card)	0.63 s	17%
T4 16GB (2018; no native BF16, FP32 shown)	6.97 s	183% (compute-bound)

H.9 Overhead of the spectral correction

The spectral correction has two costs, which we benchmarked separately on eight GPU families, including the low-end cards a decentralized mesh would use (Table 19).

SVD of the anchor momentum. M_t^{anc} changes only when a new anchor gradient arrives, once every $\sim$20–25 fast steps (76–95 s of wall clock in our configuration), so the SVD is recomputed only at that cadence. Because the filter uses only the leading directions, we use a randomized truncated SVD at rank $r{=}256$, run on a secondary CUDA stream off the critical path; the masked steps are communication-bound, so it overlaps with transfers.

Per-step filter application. With the factors (U, d, V) cached between anchor arrivals, each masked step applies a two-sided projection: four thin matrix multiplications per targeted matrix.

Table 19: Cost of the spectral correction per weight matrix. Truncated SVD at rank 256 (recomputed once per anchor arrival) across GPU families, per-step projection on A100, and memory of the cached factors. Repeated runs on each GPU family agreed within 4%.

	2048×2048 (1B attention)	**2048×8192** (1B MLP)	**4096×14336** (8B MLP)	**8192×28672** (70B MLP)
Truncated SVD, rank 256				
T4 16GB	27 ms	49 ms	112 ms	346 ms
L4 24GB	20 ms	29 ms	56 ms	156 ms
L40S 48GB	16 ms	20 ms	29 ms	60 ms
RTX PRO 4500 32GB	18 ms	25 ms	45 ms	92 ms
RTX PRO 6000 96GB	18 ms	22 ms	28 ms	52 ms
A100 40GB	27 ms	40 ms	62 ms	121 ms
H100 80GB	18 ms	27 ms	40 ms	60 ms
H200 141GB	18 ms	27 ms	39 ms	59 ms
Per-step projection (A100)	0.45 ms	1.5 ms	4.8 ms	18 ms
Cached factors (BF16)	2.0 MB	5.0 MB	9.0 MB	18.0 MB

For our configuration (1B model, 8 pipeline stages, $\sim$14 targeted matrices per stage, A100), the truncated SVDs sum to $\approx$0.45 s per anchor arrival per stage, at most 0.6% of GPU time and essentially no wall-clock time once overlapped with communication. The per-step projection costs $\approx$12 ms per stage against a 3.8 s masked step ($< 0.4\%$); across all eight GPU families it ranges from 0.2 to 19 ms per matrix.

Memory. The correction keeps one anchor-EMA buffer per targeted matrix (the same footprint as Adam's first moment) plus the rank-256 factors in Table 19. The peak transient SVD workspace is at most 1.9 GB even at the 70B shape, which fits on a 16 GB card.

Scaling. The cost is $O(mnr)$, linear in the matrix size at fixed rank: the per-matrix SVD time rises only $4.5\times$ on A100 from the 1B to the 70B shapes, while the anchor window itself lengthens with model size (larger activations and weight pulls; Eq. (9)). The relative overhead therefore shrinks with scale.

I Training and Evaluation Datasets

This section details the training data, evaluation data, and shared training configuration used to produce the per-domain results in Table 2.

I.1 Training datasets

All training data is converted to a chat-message format (`messages: [{user}, {assistant}]`) with thinking disabled, then split 90/10 into train/validation unless the source provides a held-out split (GSM8K, Spider). Table 20 lists the source dataset and post-preprocessing row count per domain.

Domain	HuggingFace dataset	# train	Notes
Medical	`medalpaca/medical_meadow_medqa`	9,160	USMLE-style multiple choice.
Code	`iamtarun/python_code_instructions_18k_alpaca`	16,750	Python instruction $\rightarrow$ code pairs.
Math	`openai/gsm8k` (`main`)	7,473	Official GSM8K test used as validation.
Science	`allenai/sciq`	10,511	Question + supporting passage.
Commonsense	`Rowan/hellaswag` $\cup$ `allenai/winogrande` (`xl`)	72,272	Concatenated, shuffled with seed 42.
Summarization	`cnn_dailymail` (`3.0.0`)	27,000	30k subsample, 90/10 split.
SQL	`philikai/Spider-SQL-LLAMA2_train`	8,659	Schema-conditioned prompts.
QuALITY	`emozilla/quality`	2,523	Long-context multiple choice.
NarrativeQA	`deepmind/narrativeqa`	6,500	Story-summary open-ended QA.
General	`HuggingFaceTB/smoltalk2`	—	25+ SFT splits, weighted; used only for the general run.

Table 20: Per-domain training datasets after preprocessing and the 90/10 train/validation split.

I.2 Evaluation datasets and metrics

Each domain run is evaluated on a fixed common suite plus one or more domain-specific tasks. The common suite consists of loglikelihood tasks (WinoGrande, PIQA, HellaSwag, ARC-Easy, ARC-Challenge) and generative tasks (IFEval, GSM8K), all run via the in-training `eval_callback`. Domain-specific evaluations are summarized in Table 21; tasks marked *lm-eval* are run via the `lm-evaluation-harness` [Gao et al., 2024], and custom tasks are scored by dedicated scripts (Spider execution match, NarrativeQA token F1, QuALITY accuracy).

Domain	Eval dataset	Metric	# items	Source
Medical	MedQA (4-options)	Accuracy	1,273	lm-eval
Medical	MedMCQA	Accuracy	—	lm-eval
Code	HumanEval	pass@1	164	lm-eval
Code	MBPP	pass@1	500	lm-eval
SQL	Spider (dev)	Execution match	1,034	Custom
Science	ARC-Challenge	Acc. (norm)	1,172	lm-eval
Science	SciQ	Accuracy	1,000	lm-eval
Commonsense	HellaSwag	Acc. (norm)	10,042	lm-eval
Commonsense	WinoGrande	Accuracy	1,267	lm-eval
Summarization	CNN/DailyMail	ROUGE-1	—	Custom
Summarization	XSum	ROUGE-1	—	Custom
Math	GSM8K	Exact match	1,319	lm-eval
Doc. Parsing	QuALITY	Accuracy	2,086	Custom
Doc. Parsing	NarrativeQA	Mean token F1	500	Custom

Table 21: Per-domain evaluation datasets.

I.3 Shared training configuration

All baseline and masked runs share the configuration.

Model. SmolLM3-Mid (1.2B parameters), BF16 weights and activations, FlashAttention 2, SmolLM3 chat template with thinking disabled.

Optimizer. AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$. Peak learning rate 2×10^{-5}, cosine schedule with min-LR ratio 0.1 and warmup ratio 0.03. Gradient norm clipped at 0.2.

Sequence handling. Best-fit-decreasing packing enabled, except for short-context tasks (Spider, NarrativeQA, QuALITY) where packing is disabled. Maximum sequence length is task-dependent: 32,768 for code/medical/science/commonsense/summarization/math, 2,048 for SQL and NarrativeQA, 10,240 for QuALITY, and 65,536 for the general mixture. Loss is computed over assistant tokens only; Liger kernels are enabled.

J Squeezing Throughput from Data-Parallel Communication

Activation compression reduces the per-step cost of pipeline-parallel (PP) communication, but the throughput benefit is realized only if the *data-parallel* (DP) gradient all-reduce is reduced to a comparable level. Otherwise the DP all-reduce dominates wall-clock time and the activation savings are invisible. This section formalizes the trade-off, surveys the families of DP-compression techniques compatible with our setting, and identifies the operating points at which each becomes necessary.

J.1 Bottleneck analysis

Consider a model with N parameters trained over K-stage pipeline parallelism and D-way data parallelism, with all communication crossing a network of bandwidth B bytes/sec. Let S denote the tokens per optimizer step, H the hidden size, and $b_{\text{act}}, b_{\text{grad}}$ the bytes per activation and gradient element.

PP communication. Pipeline parallelism transmits a forward activation and a backward gradient at each of the $K-1$ boundaries. With activation compression ratio $\rho_{\text{act}} \in (0, 1]$ (fraction of bytes still sent),

$$T_{\text{PP}} = \frac{2\,(K-1)\,S\,H\,b_{\text{act}}\,\rho_{\text{act}}}{B}. \tag{11}$$

DP communication. A standard ring all-reduce of an $N\,b_{\text{grad}}$-byte gradient costs $2(D-1)/D \cdot N\,b_{\text{grad}}$ bytes per rank [Patarasuk and Yuan, 2009]. With per-step compression ratio ρ_{DP} and outer-step interval τ (one all-reduce per τ optimizer steps, as in Local SGD or DiLoCo),

$$T_{\text{DP}} = \frac{2\,(D-1)/D \cdot N\,b_{\text{grad}}\,\rho_{\text{DP}}}{B\,\tau}. \tag{12}$$

Step time. Under 1F1B-style scheduling, the per-step time is approximately $\max(T_{\text{compute}}, T_{\text{PP}}, T_{\text{DP}})$ when phases overlap, and their sum when they serialize. Throughput in tokens-per-second is $\text{TPS} = S/\text{step time}$.

Concrete numbers. For our reference configuration (N=1.2B, H=2048, K=8, D=8, B=200 Mbps $\approx$ 25 MB/s, S=16,384 tokens/step, BF16 throughout, A100 hardware), the components evaluate to:

- Compute (one optimizer step across all stages, with $\sim 50\%$ MFU): $\sim$ 100 ms.
- PP communication at $\rho_{\text{act}} = 0.05$ (95% masking): $\sim$ 1.9 s, dominated by per-microbatch activation transfer at each of 7 boundaries in both directions.
- DP all-reduce uncompressed ($\rho_{\text{DP}} = 1$, $\tau = 1$): $\sim$ 168 s — nearly three minutes per step.

The uncompressed DP all-reduce alone is $\sim 90\times$ larger than the compressed PP transfer, and $\sim 1{,}700\times$ larger than the compute window. Activation compression on its own therefore yields no measurable speedup.

J.2 When DP compression is essential

For activation compression to translate into wall-clock speedup, T_{DP} must be reduced to within an order of magnitude of T_{PP}. From Eqs. (11)–(12), the required combined compression along the DP axis is

$$\frac{\rho_{\text{DP}}}{\tau} \lesssim \frac{(K-1)\,S\,H\,b_{\text{act}}\,\rho_{\text{act}}}{(D-1)/D \cdot N\,b_{\text{grad}}}. \tag{13}$$

For our configuration, this evaluates to roughly $\rho_{\mathrm{DP}}/\tau \lesssim 1/100$, so the joint per-step compression and frequency reduction must reach two orders of magnitude.

A single mechanism is rarely enough. Per-step gradient compressors above ~ 32–$64\times$ tend to harm convergence, and outer-step intervals $\tau > 500$ are at the edge of empirical stability for Local SGD / DiLoCo on language models. The practical strategies *compose* a compressor with a frequency-reduction technique — e.g., $64\times$ PowerSGD with $\tau = 20$ DiLoCo — to multiply their effects.

J.3 Families of DP compression

DP-compression techniques fall into four orthogonal families. Methods within a family typically cannot be combined; methods across families typically can.

Quantization (F1). Replace each gradient element with a low-bit representation. Examples include FP8/INT8, signSGD [Bernstein et al., 2018], and 1-bit Adam. Compression ranges from $2\times$ (FP16→FP8) to $32\times$ (1-bit). Stochastic rounding and error feedback [Karimireddy et al., 2019] are typically required at aggressive bit widths.

Sparsification (F2). Transmit only the largest-magnitude entries. Top-K sparsification [Lin et al., 2018b] achieves 50–$100\times$ compression when paired with momentum correction and error feedback, but the indices themselves consume bandwidth and cap practical compression.

Low-rank factorization (F3). Approximate the gradient matrix by a low-rank product. PowerSGD [Vogels et al., 2019] reaches 32–$256\times$ compression via rank-1 to rank-4 sketches with subspace iteration. The compressor is linear, so the projection commutes with all-reduce; error feedback recovers most of the lost precision.

Frequency reduction (F4). Rather than compress each all-reduce, perform fewer of them. Local SGD [Stich, 2019] runs τ inner SGD steps locally between all-reduces; DiLoCo [Douillard et al., 2023a] adds an outer Nesterov optimizer and is empirically stable up to $\tau \approx 500$ for LLM pretraining. Streaming DiLoCo [Douillard et al., 2025] pipelines the outer all-reduce with inner compute, removing the synchronous barrier.

Composition. Frequency reduction multiplies with any per-step compressor. PowerSGD at $\rho_{\mathrm{DP}} = 1/64$ combined with $\tau = 20$ DiLoCo yields an effective $1280\times$ reduction in DP bytes-per-step — enough to clear the threshold in Eq. (13) for our regime.

J.4 Comparison and selection

Table 22 summarizes the practical operating points.

Method	Per-step ratio	Composes with τ	Reported convergence cost
FP8/INT8 quantization	2–4×	yes	$\leq 1\%$ loss gap
1-bit Adam / signSGD	32×	yes	1–3% loss gap
Top-K sparsification	50–100×	yes	1–5% loss gap; index overhead
PowerSGD (rank 4)	32–256×	yes	$\leq 2\%$ loss gap
Local SGD / DiLoCo	— (τ-based)	—	$\leq 1\%$ loss gap up to τ=500
Streaming DiLoCo	— (τ-based)	—	$\leq$ DiLoCo, hides comm

Table 22: DP-compression methods with demonstrated stability for language-model training. Per-step compression refers to bytes-on-wire relative to a full BF16 all-reduce. Frequency-reduction methods do not have a per-step compression ratio because they reduce the number of all-reduces, not the bytes per all-reduce; they are stacked with F1–F3 rather than competing with them.

The selection rule that follows from Eq. (13) is straightforward:

High bandwidth (≥ 1 Gbps inter-node). The required compression ratio is small, and PowerSGD at 32–$64\times$ alone usually meets it. No frequency reduction needed; implementation is intra-step with no algorithmic changes.

Low bandwidth ($\sim$ 200 Mbps). Single-method compression tops out around $64\times$ without convergence loss, leaving roughly an order of magnitude on the table. Composition with DiLoCo ($\tau \geq 10$) is required. Streaming DiLoCo additionally hides the outer all-reduce under inner compute, removing the synchronous barrier — this is our default configuration.

Federated or geo-distributed ($\leq$ 50 Mbps). Composition is mandatory. Streaming DiLoCo with $\tau \in [50, 500]$ stacked with $\rho_{\mathrm{DP}} \leq 1/64$ is the only regime that delivers usable throughput; convergence quality must be re-validated at the chosen τ.

K On the Compatibility with Dual-EMA Optimizers

A natural alternative to our spectral correction is to use the masked gradient as a noisy first-moment signal and reach for a dual-EMA optimizer that is known to tolerate gradient staleness, most notably AdEMAMix [Pagliardini et al., 2024]. AdEMAMix replaces Adam's single first-moment EMA with a mixture of a *fast* EMA m_1 ($\beta_1 \approx 0.9$) and a *slow* EMA m_2 ($\beta_3 \approx 0.9999$). Its update rule is

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_1^{(t)} + \alpha\, m_2^{(t)}}{\sqrt{\hat{v}^{(t)}} + \epsilon},$$

where $\hat{m}_1$ is bias-corrected, $\hat{v}$ is Adam's second moment, and $\alpha \approx 5$ controls the weight of the slow EMA. The empirical finding driving the design is that gradients can remain useful for tens of thousands of steps, and a single EMA cannot simultaneously weight the immediate past and the distant past.

This sounds superficially similar to our two-circuit design, both maintain a slow signal alongside a fast one, so it is reasonable to ask whether AdEMAMix could replace the spectral correction entirely, with the masked gradient serving as both inputs to the optimizer (the fast and slow EMAs would simply be two views of the same noisy signal). We argue this would not work in our setting, for two reasons.

(1) AdEMAMix mixes old gradients, not cleaner gradients. The slow EMA m_2 in AdEMAMix is an exponentially weighted average of the same gradients that feed the fast EMA. Its value comes from *temporal smoothing* of a noisy signal, not from any independent high-fidelity source. In our setting, the fast circuit's gradients are *biased away from the true gradient subspace* by approximately $\rho\sigma_m^2$ even after spectral filtering (Lemma 2), and unbiased only in the sense of zero conditional mean. Smoothing T such gradients with a slow EMA reduces the variance, but does *not* change the underlying noise spectrum: directions weakly supported by the signal remain weakly supported in the EMA. Our anchor circuit, by contrast, provides an *independent* signal computed from *unmasked* activations. Therefore, its principal subspace reflects the true gradient geometry, not a smoothed copy of the masked-gradient noise.

(2) Dual EMAs cannot recover lost subspaces. A core claim of our analysis (Section E) is that the masked gradient is approximately isotropic relative to the anchor basis (Lemma 2, Assumption 6). This is why a low-rank spectral filter contracts noise quadratically in rank. AdEMAMix has no notion of an anchor basis; it operates entirely in the canonical parameter basis and applies the same dual-EMA dynamics elementwise. If the gradient signal lives in a low-rank subspace (as fine-tuning gradients do [Aghajanyan et al., 2021, Hu et al., 2022]) and the noise is approximately isotropic, AdEMAMix's elementwise smoothing extracts the signal at rate $O(1/T_{\mathrm{eff}})$ in noise variance, but discards the structural advantage of low-rankness. Our spectral filter exploits that structure directly.

However, we do not claim AdEMAMix is unhelpful in our setting. A natural composition is to apply AdEMAMix's dual-EMA dynamics to the *filtered* gradient G_t^{proj} rather than to the raw masked gradient: the fast EMA captures recent corrected directions, the slow EMA averages the filter's output over many steps, and the spectral correction continues to enforce subspace structure.

Empirical comparison. Table 23 tests both arguments. Replacing AdamW with AdEMAMix on masked gradients without anchors recovers only a small part of the gap to M95+AP, consistent with argument (1): its slow EMA smooths the same noisy source and carries no independent high-fidelity information. Composing AdEMAMix with the spectral correction, applied to G_t^{proj}, is roughly neutral

relative to AdamW (slightly lower MedQA, slightly higher HumanEval), so dual-EMA dynamics are at best mildly additive once anchor priors are in place.

Table 23: Dual-EMA optimizer vs. spectral correction at 95% masking.

Variant	**MedQA**	**HumanEval**
M95 + AdamW	0.180	0.072
M95 + AdEMAMix, no anchor	0.204	0.088
M95+AP + AdamW (ours)	0.321	0.207
M95+AP + AdEMAMix on G_t^{proj}	0.314	0.210

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have scoped out our contributions accordingly.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We provide a limitations section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We provide all the necessary assumptions and proofs for our claims.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Full experimental details are provided in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We plan to release the code; it is not yet available at publication time.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Comprehensive details are provided in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The experiments are too expensive to obtain error bars.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have provided the details in the experiments section and the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have followed the code of ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have included a discussion in the introduction.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Not aware of any specific risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have done above.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We do not release any new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We have not conducted any such experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We have not conducted any such experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [No]

Justification: LLMs were only used for writing and editing purposes.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.