DOBB: SHARD-LOCAL PRECONDITIONING IN BLOCK-DIAGONAL EIGENBASES

Anonymous authors
Paper under double-blind review

ABSTRACT

Training language models whose parameters and optimizer state exceed a single node's memory requires sharding these tensors across devices. Under tensor parallelism (TP), matrix-aware optimizers such as Muon introduce communication to perform global matrix operations across shards. We propose Dobb (Distributed Optimizer with Block-diagonal Bases), a matrix-aware optimizer designed to compute updates locally within parameter shards. Dobb approximates the input-activation second moment with diagonal blocks and tracks a complete basis within each block. It projects momentum into these bases, applies sign normalization to the coefficients, and reconstructs the update, retaining every direction within each block. When each block lies within one shard under TP or fully sharded data parallelism (FSDP), Dobb applies its matrix update locally, requiring no communication beyond data-parallel synchronization of the activation second moments. We benchmark Dobb against Dion, a distributed matrix-aware optimizer designed to retain the benefits of orthonormalized updates while reducing communication. On GPT-2 models with 124M, 354M, and 772M parameters, Dobb with 64 blocks achieves validation losses of 3.037, 2.742, and 2.600, respectively, compared with 3.265, 2.879, and 2.696 for Dion at rank fraction 1/64. In a Qwen3-32B timing benchmark with FSDP shard degree 32 and TP degree 2, Dobb reduces the amortized optimizer-call time from Dion's 1307.7 ms to 168.0 ms, a 7.8× speedup.

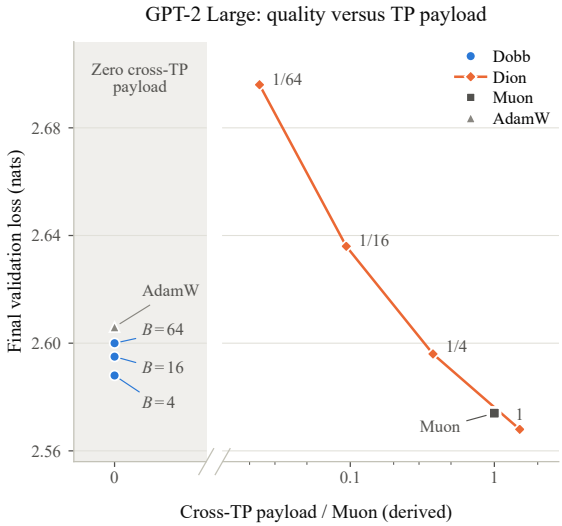


Figure 1: GPT-2 Large validation loss and optimizer-specific cross-TP payload. Losses are measured under the shared-RMS protocol (Table 3); payloads are derived for $P = 4$ with shard-aligned blocks. Labels give block count B or rank fraction ρ. Relative to Muon's dense count, Dion's leading logical payload per optimizer step is $\sum_W r_W(m_W + n_W)/\sum_W m_W n_W = 1.5\rho$, summing over transformer-block matrices with QKV split; Dobb and AdamW require zero cross-TP payload. Counts omit collective constants, element sizes, data-parallel synchronization, and forward/backward communication.

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

1 INTRODUCTION

Language-model scaling increases the memory requirements for parameters, gradients, and optimizer state (Kaplan et al., 2020; Rajbhandari et al., 2020). When these quantities exceed a single GPU's capacity, training distributes them across devices through tensor parallelism (TP) or fully sharded data parallelism (FSDP) (Shoeybi et al., 2019; Zhao et al., 2023). These layouts make matrix-aware optimization a distributed problem: an update defined on a complete weight matrix may require information held by several devices. Muon orthogonalizes momentum (Jordan et al., 2024), which introduces dense optimizer-specific communication in conventional sharded implementations (Ahn et al., 2025; Khaled et al., 2026).

One approach to reducing this communication is to decrease the frequency of global orthogonalization. MuonBP performs local Newton–Schulz orthogonalization between periodic global steps, so its local optimizer steps require no cross-block exchange. However, global steps still require full cross-block communication. The global-orthogonalization period controls a tradeoff between communication cost and optimization quality, with the 280M experiments showing greater sensitivity to this choice at higher TP degrees (Khaled et al., 2026). Another approach is to reduce the size of each exchange through low-rank approximation. Dion follows this approach by approximating momentum with low-rank factors and retaining the residual through error feedback (Ahn et al., 2025). It constructs a matrix-aware update from these factors without reconstructing the full momentum matrix, so communication scales with the update rank. In this work, we take a different approach to reducing optimizer communication. We approximate the activation second moment by a block-diagonal matrix and retain a complete eigenbasis within every block. This construction discards cross-block correlations without imposing a reduced-rank constraint on the update. When these blocks align with TP shards, the optimizer step requires no cross-shard communication.

Dion derives its low-rank factors from momentum; we instead construct our basis from an exponentially averaged activation second moment. This choice is motivated by the outer-product structure of a layer's mini-batch gradient, in which each term pairs an output gradient with an input activation. The activation second moment therefore identifies input directions in which to normalize momentum coefficients. The spectral diagnostics in Figure 2 support this choice, showing greater spectral concentration in activation factors than in several corresponding output-gradient factors.

We call the resulting method Dobb (Distributed Optimizer with Block-diagonal Bases). Dobb uses the activation-side basis and element-wise sign normalization. When each of the B blocks of a weight matrix lies within one shard, it introduces zero optimizer-specific cross-TP bytes in the optimizer step.

We make four contributions:

- We introduce Dobb, which normalizes momentum coefficients in block-diagonal activation second-moment eigenbases while retaining every direction within each block.
- We retain every direction within each second-moment block and require only that each block lie within one shard. Under this condition, the optimizer step introduces zero optimizer-specific cross-TP bytes.
- We compare Dobb with Muon and Dion on GPT-2 Small, Medium, and Large under a shared update-RMS protocol. Across these scales, Dobb exhibits less validation-loss degradation under block partitioning than Dion under corresponding rank reductions.
- We benchmark Dobb against Dion in distributed training with FSDP and tensor parallelism. Across the measured Qwen3 configurations, Dobb achieves lower optimizer-call and total iteration times.

2 RELATED WORK

Matrix-aware optimization. Matrix-aware optimizers use the intrinsic two-dimensional structure of weight matrices to construct parameter updates rather than treating each entry independently. Muon orthogonalizes momentum using Newton–Schulz iterations, while RMNP uses row normalization (Jordan et al., 2024; Deng et al., 2026). K-FAC builds a Kronecker-factored approximation to layerwise curvature (Martens & Grosse, 2015). Shampoo accumulates gradient-derived matrix

second moments, and SOAP applies Adam in their eigenbases (Gupta et al., 2018; Vyas et al., 2025).

Distributed matrix updates. While element-wise optimizer updates parallelize naturally, matrix-aware updates introduce dependencies among optimizer state, matrix computations, and the underlying parameter layout. Dion reduces communication through low-rank factor operations and error feedback, avoiding reconstruction of the full momentum matrix while retaining communication that scales with the update rank (Ahn et al., 2025). Distributed Shampoo takes a different approach. It distributes preconditioner state and computation by parameter block within data-parallel process groups, then all-gathers the computed search directions within each group at every optimizer step (Shi et al., 2023). These designs must also account for how training systems distribute parameters and computation: FSDP shards parameters and gradients across data-parallel workers, while Megatron-LM uses tensor parallelism to partition matrix operations across devices (Zhao et al., 2023; Shoeybi et al., 2019). The resulting parameter layout determines which operands are local and which require communication during the optimizer step.

Periodic cross-block synchronization. Periodic synchronization mitigates communication bottlenecks by decoupling local matrix updates from less frequent global aggregation steps. MuonBP orthogonalizes shards locally between periodic global orthogonalizations (Khaled et al., 2026). The local steps require no optimizer-specific cross-block communication, but the global steps still gather information across blocks. With synchronization period K, their leading dense exchange for an $m \times n$ matrix is amortized as $\mathcal{O}(mn/K)$. Their 280M period sweep reports worsening validation loss as K increases, most pronounced at the highest sharding degrees. Thus extending the communication interval has an observed quality cost in that setting.

3 Gradient structure and motivation

Our goal is to construct a basis in which to normalize momentum coefficients while keeping the optimizer step local to parameter shards. The gradient's outer-product structure suggests deriving this basis from either activation or output-gradient second moments. We use activation second moments and approximate them by shard-aligned diagonal blocks, retaining a complete eigenbasis within each block. This section motivates these choices through the gradient structure, spectral diagnostics, and distributed parameter layout.

A mini-batch gradient has two sources of structure. Consider a layer $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$ with $\boldsymbol{W} \in \mathbb{R}^{m\times n}$. For the mini-batch $\mathcal{B}_t$ at optimizer step t, containing N contributing examples or token positions, let $f_{\mathcal{B}_t}$ denote its mean loss and let $\boldsymbol{g}_i \in \mathbb{R}^m$ denote the output gradient associated with its ith contribution before the $1/N$ averaging. With inputs $\boldsymbol{x}_i \in \mathbb{R}^n$, the layer's stochastic gradient is

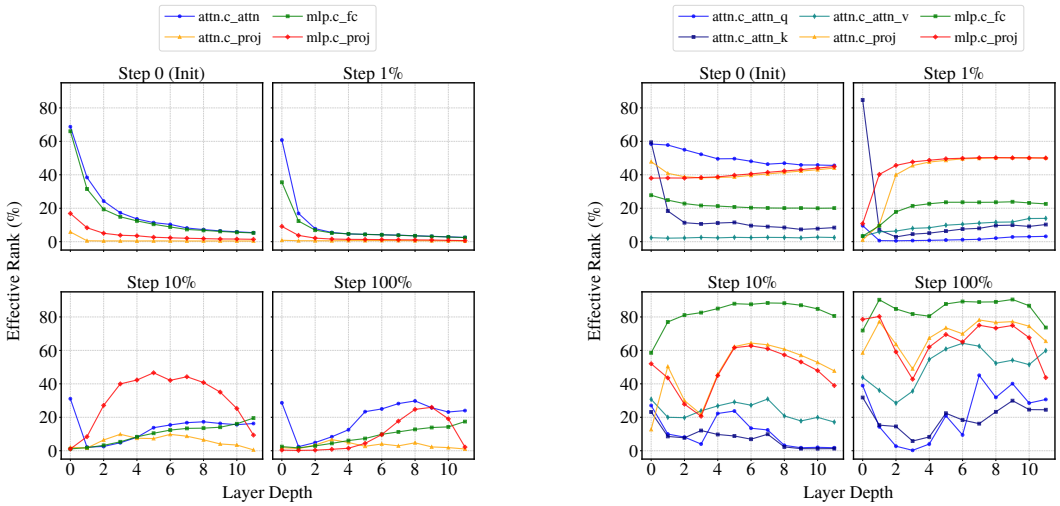
$$\nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W}_t) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{g}_i \boldsymbol{x}_i^\top = \frac{1}{N}\boldsymbol{D}_t \boldsymbol{X}_t^\top, \tag{1}$$

where $\boldsymbol{X}_t = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{n\times N}$ and $\boldsymbol{D}_t = [\boldsymbol{g}_1, \ldots, \boldsymbol{g}_N] \in \mathbb{R}^{m\times N}$. The batch subscript distinguishes this estimate from the full-objective gradient $\nabla_{\boldsymbol{W}} f$. The corresponding second moments are

$$\boldsymbol{A}_t = \frac{1}{N}\boldsymbol{X}_t\boldsymbol{X}_t^\top, \qquad \boldsymbol{G}_t = \frac{1}{N}\boldsymbol{D}_t\boldsymbol{D}_t^\top. \tag{2}$$

These are uncentered second moments, with dimensions $n \times n$ and $m \times m$, respectively. They correspond to the activation and output-gradient factors in K-FAC-style curvature approximations (Martens & Grosse, 2015). Neither factor requires materializing per-example weight gradients.

Spectral concentration motivates the input-side choice. We seek a basis that captures the low-rank structure of model gradients. Equation 1 expresses the gradient as a product of input-activation and output-gradient matrices, suggesting their second moments as two sources of basis information. We compare these factors using entropy-based effective rank, normalized by the factor dimension, across layer types and training checkpoints (Figure 2). A lower effective rank indicates that the



(a) Input-activation second moment $\boldsymbol{A}_t$

(b) Output-gradient second moment $\boldsymbol{G}_t$

Figure 2: Effective rank of the second moments from GPT-2 Small runs on OpenWebText. Within each plot, four panels show transformer layer depth at the indicated fraction of training completed; curves distinguish weight-matrix types. Effective rank is $\exp(-\sum_i p_i \log p_i)$ as a percentage of the factor dimension, n for $\boldsymbol{A}_t$ and m for $\boldsymbol{G}_t$, where p_i is the ith normalized eigenvalue. The second-moment matrices are averages over accumulated mini-batches. The output-gradient plot separates the query, key, and value blocks of the fused attention projection.

spectrum is concentrated in fewer eigendirections. In these diagnostic runs, several activation factors exhibit greater spectral concentration than the corresponding output-gradient factors. This observation suggests that activation second moments provide a more informative basis for identifying the dominant input directions underlying the gradient's low-rank structure. We therefore use their eigenvectors as coordinates for normalizing momentum coefficients.

Temporal stability motivates basis reuse. K-FAC exploits slowly changing second-moment factors and relatively stable curvature to amortize inverse computations across optimizer steps (Martens & Grosse, 2015, Appendix G). Following this rationale, we reuse the activation-side basis between periodic refreshes, reducing the amortized cost of basis computation. Appendix E.3 evaluates the resulting tradeoff between iteration time and optimization quality.

Sharding motivates a block-diagonal basis. We write P for TP degree and B for the number of second-moment blocks. In the Megatron-LM layout, a column-parallel layer shards the output axis m and receives a replicated input, whereas a row-parallel layer shards the input axis n and reduces partial outputs (Shoeybi et al., 2019). These layouts motivate a basis that can be constructed and applied using the operands available on each rank. For row-parallel layers, we partition the activation second moment into diagonal blocks within the input shards, so each block basis acts on local momentum columns. For column-parallel layers, the replicated input allows each rank to construct the same block bases and apply them to its local momentum rows. This construction accommodates both sharding axes without requiring cross-TP communication during the optimizer step. Second moments still require synchronization across data-parallel ranks, which process different data.

Block partitioning retains a complete basis. The activation spectrum motivates the choice of basis, while tensor parallelism constrains how we construct and apply it. Dion reduces communication through low-rank updates, restricting each update to a lower-dimensional subspace while retaining the approximation residual through error feedback (Ahn et al., 2025). A rank-r update has at least $\min(m, n) - r$ zero singular values, which limits its orthogonality over the full matrix space even when its factors have orthonormal columns. We instead partition the activation second moment into blocks and retain all eigenvectors within each block. This construction discards cross-block cor-

Algorithm 1 Dobb for one logical matrix. Appendix B gives the capture policy and the synchronization of statistics.

Require: $\boldsymbol{W} \in \mathbb{R}^{m \times n}$; blocks $\{\mathcal{S}_b\}_{b=1}^{B}$; period T; decays β_C, μ; learning rate γ_t; weight decay λ; target τ

Initialize $\boldsymbol{M} = 0$
for each optimizer step $t = 0, 1, \ldots$ **do**
  if $t \bmod T = 0$ **then**
    Capture input activations $\boldsymbol{X}_t$ and update each $\boldsymbol{C}_{t,b}$ by Equation 3
    Refresh each block basis $\boldsymbol{U}_b$ from $\boldsymbol{C}_{t,b}$
  end if
  $\boldsymbol{M} \leftarrow \mu \boldsymbol{M} + \nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W})$
  for $b = 1, \ldots, B$ **do**
    $\boldsymbol{O}_{:,\mathcal{S}_b} \leftarrow \big(\operatorname{sign}(\boldsymbol{M}_{:,\mathcal{S}_b}\boldsymbol{U}_b)/\sqrt{m}\big)\boldsymbol{U}_b^\top$
  end for
  $\boldsymbol{W} \leftarrow (1 - \gamma_t\lambda)\boldsymbol{W} - \gamma_t\tau\sqrt{m}\,\boldsymbol{O}$
end for

relations without imposing a reduced-rank constraint on the update and allows basis operations to remain local when blocks align with shard boundaries.

4 Distributed Optimizer with Block-diagonal Bases

Dobb normalizes momentum in a second-moment eigenbasis whose blocks can be assigned to individual shards. All eigenvectors within each block are retained; B controls the block partition rather than a truncated update rank.

An exponentially averaged block-diagonal second moment. Partition the input coordinates into B equal-sized contiguous sets $\mathcal{S}_1, \ldots, \mathcal{S}_B$. On each capture window, form

$$\boldsymbol{S}_{t,b} = \frac{1}{N}\boldsymbol{X}_{t,\mathcal{S}_b,:}\boldsymbol{X}_{t,\mathcal{S}_b,:}^\top, \qquad \boldsymbol{C}_{t,b} = \beta_C \boldsymbol{C}_{\mathrm{prev},b} + \boldsymbol{S}_{t,b}. \tag{3}$$

The subscript $\mathcal{S}_b, :$ selects the rows indexed by $\mathcal{S}_b$, and $:, \mathcal{S}_b$ selects the columns. The accumulator $\boldsymbol{C}_{\mathrm{prev},b}$ is the value from the previous capture, and β_C is its decay. The first capture initializes $\boldsymbol{C}_{t,b} = \boldsymbol{S}_{t,b}$. This is an exponentially weighted second moment; multiplying the entire accumulator by a positive scalar leaves its eigenvectors unchanged. The block-diagonal approximation discards cross-block entries of the second moment. We refresh the orthonormal basis $\boldsymbol{U}_{t,b}$ every T optimizer steps and reuse it between refreshes. The first refresh takes all eigenvectors of $\boldsymbol{C}_{t,b}$ from a symmetric eigendecomposition. Each later refresh applies one step of orthogonal iteration warm-started from the previous basis: it orthonormalizes $(\boldsymbol{C}_{t,b} + \sigma\boldsymbol{I})\boldsymbol{U}_{\mathrm{prev},b}$ by a QR factorization, where the shift $\sigma > 0$ leaves the eigenvectors of $\boldsymbol{C}_{t,b}$ unchanged. Our GPT-2 training runs use Householder QR, and the Qwen3 iteration-time measurements use Cholesky QR, which requires a larger shift. Appendix B states the refresh with both routines, and Appendix E.5 compares the two routines in GPT-2 training. The capture schedule and basis refresh are coupled, with $T = 9$ as the principal setting.

Projection, normalization, and reconstruction. We maintain momentum with decay μ,

$$\boldsymbol{M}_t = \mu \boldsymbol{M}_{t-1} + \nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W}_t). \tag{4}$$

For the activation-side update, each block computes

$$\boldsymbol{Z}_{t,b} = \boldsymbol{M}_{t,:,\mathcal{S}_b}\boldsymbol{U}_{t,b}, \qquad \boldsymbol{O}_{t,:,\mathcal{S}_b} = \mathcal{N}(\boldsymbol{Z}_{t,b})\boldsymbol{U}_{t,b}^\top. \tag{5}$$

Dobb uses $\mathcal{N}(\boldsymbol{Z}) = \operatorname{sign}(\boldsymbol{Z})/\sqrt{m}$. The normalization determines the coefficient magnitudes; the basis determines the directions in which it acts. Dobb uses the second moment's eigenvectors, without applying an inverse-eigenvalue rescaling.

Update orthogonality under block partitioning. We examine the effect of block partitioning on update orthogonality by applying each configuration to the same momentum and second-moment

Table 1: Update orthogonality under block partitioning and rank truncation. The orthogonality metric is $\|\boldsymbol{O}\|_*/(\sqrt{\min(m,n)}\,\|\boldsymbol{O}\|_F)$ for a nonzero $m \times n$ update $\boldsymbol{O}$, where $\|\cdot\|_*$ and $\|\cdot\|_F$ denote the nuclear and Frobenius norms. A value of 1 indicates equal singular values. Values average 72 logical matrices using the same momentum from a GPT-2 Small $B = 1$, $T = 16$ checkpoint at optimizer step 5,000 with second-moment capture at every step. Dobb uses exact eigendecompositions of second-moment sub-blocks, with bases cast to bf16; Dion uses one cold-start basis update without error feedback.

Dobb blocks B	Dobb orthogonality	Dion rank r	Dion orthogonality
1	0.849	768	0.804
4	0.834	192	0.427
16	0.815	48	0.208

state. The orthogonality diagnostic in Table 1 uses one checkpoint and simulates larger block counts from the $B = 1$ state; Dion is evaluated from a cold start without error feedback. It characterizes individual updates rather than trained trajectories. For the orthogonality metric defined in Table 1, a rank-r update is bounded above by $\sqrt{r/\min(m,n)}$. Increasing Dobb's block count imposes no corresponding rank-dependent ceiling. In this diagnostic, increasing B from 1 to 16 reduces Dobb's mean orthogonality from 0.849 to 0.815, whereas reducing Dion's rank from 768 to 48 reduces its orthogonality from 0.804 to 0.208. Section 6.1 reports validation loss under block partitioning.

Appendices E.1 and E.2 evaluate the choices of sign normalization and an activation-side basis through controlled ablations.

Update magnitude and weight decay. Define $\mathrm{RMS}(\boldsymbol{O}) = \|\boldsymbol{O}\|_F/\sqrt{mn}$ before learning-rate multiplication and weight decay. For complete orthonormal block bases and nonzero coefficient entries, Dobb's raw update has squared Frobenius norm n, independent of B. Thus, with learning rate γ_t and decoupled weight decay λ, the nominal common-RMS target $\tau = 0.2$ gives

$$\boldsymbol{W}_{t+1} = (1 - \gamma_t\lambda)\boldsymbol{W}_t - \gamma_t s\,\boldsymbol{O}_t, \qquad s = \tau\sqrt{m}. \tag{6}$$

This factor depends on logical matrix shape, not the local shard shape, and introduces no norm collective. Appendix D.1 derives the factor and states the effects of numerical orthogonality and zero coefficients. The shared-RMS experiment uses this convention.

Locality. Dobb requires only that each block lie within one shard; projection and reconstruction then use locally owned momentum slices, and the block update needs no cross-shard communication. When the basis axis is replicated, each rank can form and retain the bases needed for its local update. The sign normalization is element-wise in either case. Data-parallel second-moment synchronization remains necessary even when the TP update is local. Section 5 separates these costs.

5 Communication and computation

We count optimizer-specific communication separately on TP and data-parallel groups. All communication counts in this section, including Table 2, describe logical payloads. For TP comparisons we assume $P > 1$ and the Megatron-LM layout of Section 3, with each block within one shard. At $P = 1$, cross-TP communication is zero for every optimizer.

Blocks within TP shards. For Dobb on a row-parallel layer, each input block and its momentum columns reside on the same rank. On a column-parallel layer the input is replicated, so each rank can form every input-side block basis and apply it to its own momentum rows. The logical m in the sign divisor and RMS multiplier is known from the parameter shape. No cross-TP statistic is needed for either update. Thus, when each block lies within one shard, Dobb introduces zero optimizer-specific cross-TP bytes, including basis construction.

For a square 4096×4096 attention projection of Qwen3-8B in fp32, one dense logical payload is 67.11 MB, and two factors at $r = 64$ contain 2.10 MB. With each block within one shard, sign normalization contributes 0 MB across TP.

Table 2: Leading optimizer-specific cross-TP payload for one $m \times n$ matrix. The counts apply when either the input or output axis is sharded. For Dobb, each block lies within one shard. Constant collective factors and element sizes are omitted. Second-moment synchronization across data-parallel ranks is additional.

Method	Cross-TP payload
Muon, dense exchange	$\mathcal{O}(mn)$
Dion, rank r	$\mathcal{O}(r(m+n))$
Dobb	0

FSDP has a separate synchronization requirement. FSDP ranks process different data, so consistent second moments require aggregation across the data-parallel group even when each block lies within one shard. Dobb captures and synchronizes its block second moments only on refresh iterations, once every T optimizer steps; the other optimizer steps issue no optimizer collective. Under FSDP alone, the measured implementation reduce-scatters the block second moments across the data-parallel group, refreshes each rank's share of the block bases, and all-gathers the refreshed bases. Under FSDP with TP, it all-reduces the block second moments across the data-parallel group, and each rank refreshes the bases of the blocks its shard covers, so no bases are exchanged. Each refresh synchronizes n^2/B second-moment entries per matrix for equal blocks (Appendix C), separately from the cross-TP count above. FSDP parameter all-gathers and gradient reduce-scatters remain part of the complete iteration, and our iteration times include them together with the capture hooks.

Arithmetic and memory. For Dobb, projection and reconstruction cost $\mathcal{O}(mn^2/B)$, while one orthogonal iteration per refresh costs $\mathcal{O}(n^3/(B^2T))$ amortized. Capturing N_c token positions once per refresh adds $\mathcal{O}(N_c n^2/(BT))$. Storing the second moment and full block basis requires $2n^2/B$ scalars, in addition to momentum. Small products, basis orthogonalization, and communication overlap determine the realized cost.

Comparison with a rank budget. We pair B with Dion rank fraction $1/B$ in the quality comparison. For a square $n \times n$ matrix with $r = n/B$, Dobb's block basis and Dion's persistent factor each contain n^2/B entries. Including momentum and the block second moment, Dobb stores $n^2 + 2n^2/B$ scalars, while Dion stores $n^2 + n^2/B$. Appendix C gives the rectangular-matrix counts and distinguishes persistent state from transient workspace.

6 Experiments

We evaluate whether block-diagonal second moments retain optimization quality as their distributed cost decreases. The principal quality comparison covers GPT-2 Small, Medium, and Large (Radford et al., 2019), with 124M, 354M, and 772M parameters excluding position embeddings. The quality runs use OpenWebText (Gokaslan & Cohen, 2019), sequence length 1024, a global batch of 480 sequences, and bfloat16 computation. Table 9 records the scale-specific training budgets, and Table 7 records the selected learning rates. All configurations within a scale use the same token budget, data order, evaluation procedure, and learning-rate schedule.

Baselines and parameter groups. We compare Dobb at $B \in \{1, 4, 16, 64\}$ against AdamW, Muon, RMNP, and Dion at rank fractions $1, 1/4, 1/16, 1/64$ (Loshchilov & Hutter, 2019; Jordan et al., 2024; Deng et al., 2026; Ahn et al., 2025). A Dion rank fraction ρ specifies $r = \max(1, \lfloor \rho \min(m, n) \rfloor)$ for each logical matrix, subject to documented distributed-kernel constraints. Matrix optimizers update the two-dimensional weights inside transformer blocks; embeddings, the language-model head, biases, and normalization parameters use AdamW. Fused QKV parameters are treated as three logical matrices for matrix-update normalization, with each matrix's own m, n. The quality implementation follows the RMNP/nanoGPT training framework (Deng et al., 2026; Karpathy, 2022). The timing implementation builds on the open-source Dion codebase (Ahn et al., 2025) and is described separately below.

Table 3: Final validation loss in nats under the shared-RMS protocol, with common learning rate and weight decay within each model scale. Small, Medium, and Large train for 10,000, 20,000, and 30,000 steps. Dion ranks are fractions of each logical matrix's full rank.

Optimizer	Blocks / rank fraction	Small	Medium	Large
AdamW	n/a	3.044	2.761	2.606
Muon	full	3.004	2.725	2.574
RMNP	n/a	3.083	2.778	2.618
Dion	1	3.002	2.720	2.568
	$1/4$	3.038	2.754	2.596
	$1/16$	3.108	2.802	2.636
	$1/64$	3.265	2.879	2.696
Dobb	$B = 1$	3.017	2.735	2.579
	$B = 4$	3.024	2.741	2.588
	$B = 16$	3.026	2.746	2.595
	$B = 64$	3.037	2.742	2.600

A common update-RMS protocol. Following Liu et al. (2025), we calibrate matrix updates to a nominal per-element RMS of $\tau = 0.2$ before learning-rate multiplication and weight decay. For a full-rank orthogonalized $m \times n$ update, their Lemma 1 gives raw RMS $1/\sqrt{\max(m, n)}$, yielding the multiplier $0.2\sqrt{\max(m, n)}$. This targets the scale of their observed AdamW update RMS of 0.2–0.4. For Dobb, RMNP, and low-rank Dion we use the corresponding shape- and rank-dependent factors in Appendix D.1. We select AdamW's learning rate by grid search at Small and Medium and extrapolate a power-law fit of the sweeps to Large (Appendix D.2), then transfer the selected rate and schedule to all other optimizers. All optimizers use weight decay 0.1, following the pretraining setting of Liu et al. (2025).

Evaluation. We report final held-out cross-entropy in nats to three decimals and absolute loss differences. Table 3 reports the results under this protocol. The distributed timing study uses 80 iterations from random initialization on Qwen3 shapes, with model compilation disabled.

6.1 Quality under block and rank compression

Table 3 reports the cross-scale comparison under the shared-RMS protocol. For Dobb, the change from $B = 1$ to $B = 64$ measures the quality cost of block diagonality. For Dion, the change from full rank to $1/64$ rank measures the cost of restricting the update rank. The table gives absolute losses at every block count and rank fraction.

From $B = 1$ to $B = 64$, Dobb's validation loss rises from 3.017 to 3.037 on Small, from 2.735 to 2.742 on Medium, and from 2.579 to 2.600 on Large, increases of 0.020, 0.007, and 0.021 nats. Reducing Dion's rank fraction from 1 to $1/64$ raises its loss from 3.002 to 3.265, from 2.720 to 2.879, and from 2.568 to 2.696, increases of 0.263, 0.159, and 0.128 nats. At $B = 1$, Dobb has higher loss than Muon and full-rank Dion at each scale; at $B \in \{4, 16, 64\}$, it has lower loss than Dion at the corresponding rank fraction $1/B$.

6.2 Measured iteration time under FSDP and tensor parallelism

We measured iteration time on randomly initialized models with the matrix shapes of dense Qwen3-8B and Qwen3-32B (Yang et al., 2025), using a timing implementation built on the open-source Dion codebase (Ahn et al., 2025). Qwen3-8B ran under FSDP alone, with $P = 1$ and FSDP shard degrees $F \in \{8, 16, 32, 64\}$; Qwen3-32B ran under FSDP with TP at $(F, P) \in \{(32, 2), (64, 2), (64, 4)\}$. Dobb used $B = F$ blocks per logical matrix, and Dion's rank fraction was matched to a block width of $1024/F$ (Appendix F.1). In the TP runs, each block lies within one shard, with $B = F \geq 16P$.

Table 4 reports means over complete $T = 9$ cycles, so the Dobb entries include their refresh iterations. Dobb's optimizer-call means range from 15.1 to 86.2 ms on Qwen3-8B and from 69.9 to 168.0 ms on Qwen3-32B, against 114.8 to 159.5 ms and 797.3 to 1307.7 ms for Dion. We report

Table 4: Measured iteration times in milliseconds on randomly initialized Qwen3-8B under FSDP and Qwen3-32B under FSDP with TP, with $B = F$ blocks per logical matrix. Each entry is a mean over seven complete refresh cycles of $T = 9$ iterations. Optimizer call is the amortized mean of the optimizer call; Iteration is the total iteration time, including forward/backward. Tables 19 and 20 split the optimizer call into refresh and non-refresh iterations and report forward/backward time.

			Optimizer call		Iteration	
Model	F	P	Dion	Dobb	Dion	Dobb
Qwen3-8B	8	1	159.5	86.2	419.3	347.3
	16	1	146.1	60.1	5348.6	5176.3
	32	1	114.8	27.9	5631.2	5464.6
	64	1	138.6	15.1	6039.2	5830.0
Qwen3-32B	32	2	1307.7	168.0	18,435.0	16,316.8
	64	2	1287.5	83.4	18,825.6	17,169.7
	64	4	797.3	69.9	14,282.4	13,915.3

Dion's iteration time divided by that of the compared method as the ratio. On Qwen3-8B at $F = 8$, where the run occupies one host, Dobb takes 347.3 ms per iteration against 419.3 ms for Dion, a ratio of 1.21. At $F \geq 16$, the runs span hosts, forward/backward takes 5.1 to 5.9 s, Dion's optimizer call is at most 2.7% of its iteration, and the ratios lie between 1.03 and 1.04. On Qwen3-32B, the ratios lie between 1.10 and 1.13 at $P = 2$. At $P = 4$, Dobb takes 13,915.3 ms against 14,282.4 ms for Dion, a ratio of 1.03, and Dion's optimizer call is at most 7.1% of its iteration across the Qwen3-32B runs.

The methods ran in separate allocations, and their forward/backward means also differ (Appendix F.1). The total-iteration ratios therefore include forward/backward variation as well as optimizer-call differences.

7 Conclusion

We introduce Dobb, which normalizes momentum coefficients in block-diagonal activation second-moment eigenbases. The central design choice is to discard cross-block correlations while retaining every basis direction within each block. Together with element-wise sign normalization, this construction introduces zero optimizer-specific cross-TP bytes when each block lies within one shard. It demonstrates how the structure of a matrix-aware update can be designed around the parameter layout used in distributed training.

Across GPT-2 Small, Medium, and Large, Dobb exhibits less validation-loss degradation under block partitioning than Dion under corresponding rank reductions. The Qwen3 benchmarks separately show lower optimizer-call and total iteration times in the measured distributed configurations.

Further work could investigate how block size and basis-refresh frequency should vary across layers and training stages. Understanding how discarded cross-block correlations and coefficient normalization theoretically affect optimization could also guide block partitions that preserve useful structure while respecting shard boundaries.

AI use statement

We used generative AI tools to assist with implementing the methods. We did not use them to develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design or give feedback on research methodology or experiments, generate synthetic datasets, clean or reformat datasets, assist with translation, or interpret results. Every reported loss and timing measurement comes from an executed training run. The paper contains no proofs and no qualitative data, so the remaining tasks that require disclosure do not apply. In addition, we used generative AI tools to write and edit other code (including plotting scripts), to draft parts of the paper, and to polish text for readability and grammar. We reviewed all AI-assisted code, figures, and text. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Kwangjun Ahn, Byron Xu, Natalie Abreu, Ying Fan, Gagik Magakyan, Pratyusha Sharma, Zheng Zhan, and John Langford. Dion: Distributed orthonormalized updates. *arXiv preprint arXiv:2504.05295*, 2025.

Johan Bjorck, Alon Benhaim, Vishrav Chaudhary, Furu Wei, and Xia Song. Scaling optimal LR across token horizons. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2409.19913.

DeepSeek-AI. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024. URL https://arxiv.org/abs/2401.02954.

Shenyang Deng, Zhuoli Ouyang, Tianyu Pang, Zihang Liu, Ruochen Jin, Shuhua Yu, and Yaoqing Yang. RMNP: Row-momentum normalized preconditioning for scalable matrix-based optimization. In *International Conference on Machine Learning*, 2026.

Aaron Gokaslan and Vanya Cohen. OpenWebText corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1842–1850, 2018.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Andrej Karpathy. nanoGPT. https://github.com/karpathy/nanoGPT, 2022.

Ahmed Khaled, Kaan Ozkara, Tao Yu, Mingyi Hong, and Youngsuk Park. MuonBP: Faster Muon via block-periodic orthogonalization. In *International Conference on Learning Representations*, 2026.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025. URL https://arxiv.org/abs/2502.16982.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2408–2417, 2015.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–16, 2020. doi: 10.1109/SC41405.2020.00024.

Hao-Jun Michael Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat. A distributed data-parallel PyTorch implementation of the Distributed Shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*, 2023.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and stabilizing Shampoo using Adam for language modeling. In *International Conference on Learning Representations*, 2025.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs V: Tuning large neural networks via zero-shot hyperparameter transfer. In *Advances in Neural Information Processing Systems*, volume 34, pp. 17084–17097, 2021. URL https://arxiv.org/abs/2203.03466.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12): 3848–3860, 2023. doi: 10.14778/3611540.3611569.

APPENDIX CONTENTS

A Notation **13**

B Complete algorithm and implementation settings **13**

C Communication, arithmetic, and state accounting **14**

D Training protocol and update-RMS calibration **16**

D.1 Normalization factors 16

D.2 Learning-rate scaling 17

E Ablations **19**

E.1 Sign versus ℓ_2 normalization 20

E.2 Input-activation versus output-gradient bases 20

E.3 Basis-refresh period 21

E.4 Orthogonal iterations per refresh 21

E.5 Householder versus Cholesky QR 22

E.6 First-micro-batch capture 22

E.7 Second-moment and basis precision 23

E.8 Gradient-derived second moments 23

E.9 Nesterov momentum 24

F Details of the distributed iteration-time experiments **25**

F.1 Qwen3 FSDP and tensor-parallel series 25

A NOTATION

Table 5 lists the symbols that recur across sections, grouped by topic, with the location of each definition. Symbols confined to one paragraph, figure, algorithm, or appendix are defined where they appear. Bold capitals denote matrices, and bold lowercase letters denote vectors. Calligraphic letters denote index sets, the mini-batch, the coefficient normalization, the fitted loss, and asymptotic order $\mathcal{O}(\cdot)$. For a matrix $\boldsymbol{M}$, $\boldsymbol{M}_{:,\mathcal{S}}$ selects the columns indexed by $\mathcal{S}$ and $\boldsymbol{M}_{\mathcal{S},:}$ the rows. Matrix dimensions refer to the logical matrix after the QKV split and before sharding. Algorithms 1 and 3 omit the step index t from the momentum, bases, coefficients, and update, writing $\boldsymbol{M}$, $\boldsymbol{U}_b$, $\boldsymbol{Z}_b$, and $\boldsymbol{O}$.

Table 5: Notation. Parenthesized values are the principal settings.

Symbol	Meaning	Defined in
Layer and gradient statistics		
m, n	Output and input dimensions of a weight matrix	Section 3
$\boldsymbol{W}_t$, t	Weight matrix of a layer $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$ at optimizer step t	Section 3
$\mathcal{B}_t$, N	Mini-batch at step t, and its number of examples or token positions	Section 3
f, $f_{\mathcal{B}_t}$	Full training objective and mean loss on $\mathcal{B}_t$	Equation 1
$\boldsymbol{x}_i$, $\boldsymbol{g}_i$	Input activation and output gradient of the ith contribution	Equation 1
$\boldsymbol{X}_t$, $\boldsymbol{D}_t$	Stacked activations $[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ and output gradients $[\boldsymbol{g}_1, \ldots, \boldsymbol{g}_N]$	Equation 1
$\boldsymbol{A}_t$, $\boldsymbol{G}_t$	Activation and output-gradient second moments, $n \times n$ and $m \times m$	Equation 2
A, G	Basis-source labels for $\boldsymbol{A}_t$ and $\boldsymbol{G}_t$, in variant names and Algorithm 3	Section 4
Blocks and bases		
B, b	Number of blocks per logical matrix, and block index	Section 3
$\mathcal{S}_b$	Coordinates of block b on the input axis for A, output axis for G	Equation 3
$\boldsymbol{S}_{t,b}$	Block second moment of one capture window	Equation 3
$\boldsymbol{C}_{t,b}$, β_C	Exponentially averaged block second moment, and its decay (0.95)	Equation 3
T	Capture and basis-refresh period in optimizer steps (9)	Section 4
N_c	Token positions captured per refresh	Section 5
σ	Shift added to a block second moment before orthogonal iteration	Section 4
ε	Shift for the cold-start eigendecomposition and Householder QR (10^{-6})	Algorithm 2
ω	Relative shift for Cholesky QR, $\sigma \geq \omega\lambda_{\max}$ (10^{-2})	Algorithm 2
$\boldsymbol{U}_{t,b}$	Orthonormal eigenbasis of block b in use at step t	Section 4
Update		
$\boldsymbol{M}_t$, μ	Momentum and its decay (0.95)	Equation 4
$\boldsymbol{Z}_{t,b}$	Coefficients of the momentum slice in the block basis	Equation 5
$\mathcal{N}$, $\widehat{\boldsymbol{Z}}_b$	Coefficient normalization, and normalized coefficients $\widehat{\boldsymbol{Z}}_b = \mathcal{N}(\boldsymbol{Z}_b)$	Equation 5
ℓ_∞, ℓ_2	Sign normalization $\operatorname{sign}(\boldsymbol{Z})/\sqrt{d}$, and unit-norm columns	Algorithm 3
d	Length of the non-basis axis: m for A, n for G	Algorithm 3
$\boldsymbol{O}_t$	Raw update assembled from the blocks, before scaling	Equation 5
$\operatorname{RMS}(\boldsymbol{O})$	Per-element root mean square, $\|\boldsymbol{O}\|_F/\sqrt{mn}$	Section 4
τ, s	Target update RMS (0.2), and update multiplier $\tau\sqrt{d}$	Appendix D.1
γ_t, λ	Learning rate at step t, and decoupled weight decay (0.1)	Equation 6
Baselines and parallelism		
r, ρ	Dion rank and rank fraction, $r = \max(1, \lfloor \rho \min(m, n) \rfloor)$	Section 6
P	Tensor-parallel degree	Section 3
F	FSDP shard degree	Section 6.2

B COMPLETE ALGORITHM AND IMPLEMENTATION SETTINGS

Algorithm 3 specifies the four Dobb variants under the common-RMS convention. Algorithm 2 maintains a complete orthonormal basis within each block. Each warm-started refresh orthonormalizes $(\boldsymbol{C}+\sigma\boldsymbol{I})\boldsymbol{U}$, and we use two QR routines for this step. The GPT-2 training runs use Householder QR with $\sigma = \varepsilon$, except in the ablation of Appendix E.5. The Qwen3 iteration-time measurements of Section 6.2 and Appendix F use the Cholesky QR of Algorithm 4. Cholesky QR replaces the sequence of Householder reflections with matrix products, a Cholesky factorization of the Gram

matrix $\boldsymbol{Q}^\top \boldsymbol{Q}$, and a triangular solve. The Gram matrix has the squared condition number of $\boldsymbol{Q}$, so its factorization can fail in fp32 when $\boldsymbol{Q}$ is ill-conditioned. For an orthonormal $\boldsymbol{U}$, the condition number of $(\boldsymbol{C}+\sigma\boldsymbol{I})\boldsymbol{U}$ equals that of $\boldsymbol{C}+\sigma\boldsymbol{I}$, which the shift $\sigma \geq \omega\lambda_{\max}(\boldsymbol{C})$ bounds by $(1+\omega)/\omega$, or 101 at $\omega = 10^{-2}$. The shift leaves the eigenvectors unchanged but replaces each eigenvalue ratio λ_i/λ_j by $(\lambda_i+\sigma)/(\lambda_j+\sigma)$, which slows the convergence of orthogonal iteration. The distributed implementation may assign basis factorizations to different ranks, batch matrices with equal block shapes, and distribute refreshed bases to the ranks that consume them. Those assignments affect the collective schedule, not the logical block update.

Algorithm 2 Full block-basis refresh. The first call uses an exact eigendecomposition; subsequent calls apply one step of warm-started orthogonal iteration, orthonormalized by Householder QR or by Cholesky QR (Algorithm 4). The GPT-2 training runs use Householder QR, and the Qwen3 iteration-time measurements use Cholesky QR. Equal-sized blocks can be processed in a batch.

function EIGENVECTORS($\boldsymbol{C}, \boldsymbol{U}, \varepsilon, \omega$)
  if $\boldsymbol{U}$ is empty **then**
    $\boldsymbol{U} \leftarrow$ eigenvectors from EIGH($\boldsymbol{C} + \varepsilon\boldsymbol{I}$)
    Discard the eigenvalues and retain every column of $\boldsymbol{U}$
  else if the routine is Householder QR **then**
    $\boldsymbol{U} \leftarrow$ orthonormal factor of HOUSEHOLDERQR$\big((\boldsymbol{C} + \varepsilon\boldsymbol{I})\boldsymbol{U}\big)$
  else
    $\sigma \leftarrow \omega \max_i \sum_j |C_{ij}|$ ▷ upper bound on $\omega\lambda_{\max}(\boldsymbol{C})$
    $\boldsymbol{U} \leftarrow$ CHOLESKYQR$\big((\boldsymbol{C} + \sigma\boldsymbol{I})\boldsymbol{U}\big)$ ▷ Algorithm 4
  end if
  return $\boldsymbol{U}$
end function

For an axis of length n on the activation side or m on the output side, use at most that many nonempty blocks. The first remainder blocks have ceiling width and the rest have floor width, so the block count remains the specified count rather than a layer-dependent width convention. A local update requires only that each block lie within one shard. The block count is defined per logical matrix; a fused QKV parameter is split into its three logical matrices before the update, and each receives its own blocks. No equality of block width across differently shaped layers is assumed.

The principal capture policy uses only the first micro-batch on refresh iterations. With all-micro-batch capture, form a token-weighted second moment across the captured micro-batches before applying the window-level decay. This normalization makes the second moment an average over captured tokens for either capture policy in Appendix E.6.

Second-moment capture uses forward hooks for side A and backward hooks for side G. The hooks are inactive outside training mode and on non-refresh iterations, and they accumulate the block outer products in place, so captured activations or output gradients are not retained after the hook returns. Column normalization in the ℓ_2 variants sums squares over a full coefficient column; when that axis is sharded, the sum requires one reduction across the shards, which Section 5 counts separately from the local update. The principal settings are $\beta_C = 0.95$, $\mu = 0.95$, $T = 9$, one warm orthogonal iteration, fp32 second-moment accumulation, and bf16 basis storage. Appendix E.7 compares these two precision choices against their alternatives. Appendix E.4 compares the single orthogonal iteration with two and three iterations and with an exact eigendecomposition. The shift for the cold-start eigendecomposition and Householder QR is $\varepsilon = 10^{-6}$ in the stated accumulation convention, and Cholesky QR uses $\omega = 10^{-2}$. The common-RMS target is $\tau = 0.2$, and weight decay is $\lambda = 0.1$. Table 7 lists the selected learning rates.

C COMMUNICATION, ARITHMETIC, AND STATE ACCOUNTING

Communication for the ablation configurations. Appendix E defines three alternative configurations that change the basis source or coefficient normalization of Dobb. With each block within one shard, sign normalization requires no cross-TP reduction for either basis source. Column normalization changes this result only when its reduction axis is sharded. Dobb-ℓ_2 reduces one squared norm for each of n coefficient columns when the output axis is sharded. Dobb-G-ℓ_2 reduces m

Algorithm 3 Complete update for the four Dobb variants on one matrix under the common-RMS protocol. The side and norm select the variant: (A, ℓ_∞) gives Dobb, (G, ℓ_∞) Dobb-G, (A, ℓ_2) Dobb-ℓ_2, and (G, ℓ_2) Dobb-G-ℓ_2. Appendices E.1 and E.2 compare the four variants. Every 2D parameter runs this update independently. The gradient uses the full mini-batch $\mathcal{B}_t$; capture may use a subset of its micro-batches. Logical operations are shown; synchronization follows the actual TP/FSDP mesh.

Require: $\boldsymbol{W} \in \mathbb{R}^{m \times n}$; side A or G; norm ℓ_∞ or ℓ_2; blocks $\{\mathcal{S}_b\}_{b=1}^{B}$ of the input coordinates for side A, of the output coordinates for side G
Require: $\beta_C, \mu, T, \varepsilon, \omega, \tau, \lambda, \gamma_t$; capture policy; QR routine
$d \leftarrow m$ for side A, otherwise n
Initialize $\boldsymbol{M} = 0$; block statistics and bases are empty
for each optimizer step $t = 0, 1, \ldots$ **do**
  if $t \bmod T = 0$ **then**
    Capture $\boldsymbol{X}_t$ for side A or $\boldsymbol{D}_t$ for side G from the micro-batches selected by the capture policy
    $\boldsymbol{S}_{t,b} \leftarrow$ mean second moment of the captured positions for each block b, as in Equation 3
    Synchronize $\boldsymbol{S}_{t,b}$ across data-parallel ranks when required
    $\boldsymbol{C}_{t,b} \leftarrow \boldsymbol{S}_{t,b}$ on the first capture, otherwise $\beta_C \boldsymbol{C}_{\mathrm{prev},b} + \boldsymbol{S}_{t,b}$
    $\boldsymbol{U}_b \leftarrow \textsc{Eigenvectors}(\boldsymbol{C}_{t,b}, \boldsymbol{U}_b, \varepsilon, \omega)$ for each block b
  end if
  $\boldsymbol{M} \leftarrow \mu \boldsymbol{M} + \nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W})$
  for each locally needed block b **do**
    $\boldsymbol{V}_b \leftarrow \boldsymbol{M}_{:,\mathcal{S}_b}$ for side A, otherwise $\boldsymbol{M}_{\mathcal{S}_b,:}^\top$
    $\boldsymbol{Z}_b \leftarrow \boldsymbol{V}_b \boldsymbol{U}_b$
    if norm is ℓ_∞ **then**
      $\widehat{\boldsymbol{Z}}_b \leftarrow \operatorname{sign}(\boldsymbol{Z}_b)/\sqrt{d}$
    else
      $\widehat{\boldsymbol{Z}}_b \leftarrow \boldsymbol{Z}_b$ with each nonzero column scaled to unit ℓ_2 norm; zero columns stay zero
    end if
    $\widehat{\boldsymbol{V}}_b \leftarrow \widehat{\boldsymbol{Z}}_b \boldsymbol{U}_b^\top$
    Store $\widehat{\boldsymbol{V}}_b$ in $\boldsymbol{O}_{:,\mathcal{S}_b}$ for side A, otherwise its transpose in $\boldsymbol{O}_{\mathcal{S}_b,:}$
  end for
  $\boldsymbol{W} \leftarrow (1 - \gamma_t \lambda)\boldsymbol{W} - \gamma_t \tau \sqrt{d}\, \boldsymbol{O}$
end for

Algorithm 4 Cholesky QR with two passes, used by the Qwen3 iteration-time measurements. Algorithm 2 applies it to $(\boldsymbol{C} + \sigma \boldsymbol{I})\boldsymbol{U}$ with $\sigma \geq \omega \lambda_{\max}(\boldsymbol{C})$, which bounds the condition number of the input by $(1 + \omega)/\omega$. The timing implementation falls back to Householder QR if a Cholesky factorization fails.

function $\textsc{CholeskyQR}(\boldsymbol{Q})$
  for pass $= 1, 2$ **do**
    $\boldsymbol{R} \leftarrow \textsc{Cholesky}(\boldsymbol{Q}^\top \boldsymbol{Q})$ ▷ upper triangular, $\boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{R}^\top \boldsymbol{R}$
    $\boldsymbol{Q} \leftarrow \boldsymbol{Q} \boldsymbol{R}^{-1}$ ▷ triangular solve
  end for
  return $\boldsymbol{Q}$
end function

such values when the input axis is sharded. Independent norm reductions can be packed to reduce collective-launch overhead. For a square 4096×4096 matrix in fp32, these 4096 scalars contain 0.0164 MB.

Object size and wire traffic. For equal input-axis blocks, $\sum_b |\mathcal{S}_b|^2 = n^2/B$. For output-axis blocks the corresponding sum is m^2/B. Multiplying by the bytes per stored scalar gives the second-moment object size. Communicating this object once per T updates gives an amortized size of $n^2/(BT)$ or $m^2/(BT)$ scalars. For a 4096×4096 matrix at $B = 16$, $T = 9$, and fp32 accumulation, the second-moment object contains 4,194,304 bytes per refresh, or approximately 0.466 MB per optimizer step. A basis exchange is a separate object whose size depends on its storage precision. For a ring all-reduce over R ranks, each rank sends $2(R-1)/R$ times the input-buffer size; reduce-scatter and all-gather have different accounting.

Table 6: Leading optimizer-specific cross-TP payload for the basis and normalization ablations on one $m \times n$ matrix, with each block within one shard. The ℓ_2 entries are scalars reduced per update; constant collective factors and element sizes are omitted. Second-moment synchronization across data-parallel ranks is additional.

Configuration	Output axis m sharded	Input axis n sharded
Dobb	0	0
Dobb-G	0	0
Dobb-ℓ_2	$\mathcal{O}(n)$	0
Dobb-G-ℓ_2	0	$\mathcal{O}(m)$

Projection and basis maintenance. For input block widths n_b, projection and reconstruction require work proportional to $m \sum_b n_b^2$. For B equal input blocks of width n/B, projection and reconstruction each cost $\mathcal{O}(B\, m(n/B)^2) = \mathcal{O}(mn^2/B)$. Orthogonal iteration and QR require work proportional to $\sum_b n_b^3$, giving n^3/B^2 per iteration. An exact cold-start eigendecomposition has the same cubic order and occurs once. Captured outer products cost $N_c \sum_b n_b^2$. These counts exclude data movement, launch overhead, and overlap with forward/backward. For output-side blocks of widths m_b, use $n \sum_b m_b^2$, $\sum_b m_b^3$, and $N_c \sum_b m_b^2$, respectively.

State versus workspace. The four Dobb variants retain momentum of size mn, a block second moment, and a block basis. Their element counts are $mn + 2n^2/B$ on the activation side and $mn + 2m^2/B$ on the output side, before sharding and precision are applied. Captured activations, projected coefficients, and factorization buffers are additional transient memory. Dion retains momentum of size mn and a factor of size $r \min(m, n)$; its leading factor-communication payload is $r(m + n)$. Error feedback may be represented within an existing momentum buffer rather than a separate dense allocation. Dion also uses a second factor and QR workspace during an update. Storage precision and the device mesh determine the byte count per rank for each implementation.

D Training protocol and update-RMS calibration

AdamW learning-rate selection. The principal comparison transfers the AdamW-selected learning rate to every optimizer at a given scale. Table 7 lists the selected rate, evaluation, and seeds at each scale, and Table 9 lists the grids and training budgets. Selection uses the same validation procedure for every AdamW candidate; the final benchmark uses the selected rate without optimizer-specific retuning. Weight decay is fixed at 0.1 for all candidates and benchmark configurations. Appendix D.2 fits the optimal learning rate as a power law in model size and extrapolates it to Large.

Table 7: Training protocol for the shared-RMS quality suite. Table 9 gives the architecture, step count, tokens per step, and AdamW grid at each size. After the linear warmup stated there, the learning rate follows a cosine decay to 1% of its peak at the final step. Grid indices j refer to Equation 7. Validation loss is computed on a fixed set of held-out sequences.

Field	Small	Medium	Large
Selected learning rate	3.54×10^{-3}	3.54×10^{-3}	2.5×10^{-3}
Grid index j	-1	-1	-2
Evaluation interval (steps)	1,000	1,000	1,000
Evaluation sequences	3,000	3,000	3,000
Seeds	5000	5000	5000

D.1 Normalization factors

The scale factors below multiply the raw update before learning-rate multiplication and weight decay. They replace, rather than compound, any earlier shape correction. All matrix dimensions refer to the logical matrix before sharding and after the declared QKV split.

Table 8: Nominal scaling to update RMS $\tau = 0.2$ under the stated exact-arithmetic assumptions. Approximate orthogonalization and zero columns can lower or perturb the attained RMS.

Raw update	$\|\boldsymbol{O}\|_F^2$	Multiplier
Full-rank orthogonalized Muon	$\min(m, n)$	$\tau\sqrt{\max(m, n)}$
Dion, r orthonormal left / unit-norm right columns	r	$\tau\sqrt{mn/r}$
RMNP, unit-norm rows	m	$\tau\sqrt{n}$
Dobb, Dobb-ℓ_2	n	$\tau\sqrt{m}$
Dobb-G, Dobb-G-ℓ_2	m	$\tau\sqrt{n}$

The Muon factor is the full-rank result of Liu et al. (2025). For an activation-side Dobb block, orthonormality gives $\|\widehat{\boldsymbol{Z}}_b \boldsymbol{U}_b^\top\|_F^2 = \|\widehat{\boldsymbol{Z}}_b\|_F^2$. Unit-norm coefficient columns give $|\mathcal{S}_b|$, and summing over blocks gives n. Sign coefficients divided by $\sqrt{m}$ give the same count when every coefficient is nonzero. Transposing the construction gives m on the output side. RMNP's m unit-norm rows give squared norm m. For Dion's raw factor product $\boldsymbol{L}\boldsymbol{R}^\top$, $\boldsymbol{L}^\top\boldsymbol{L} = \boldsymbol{I}_r$ and unit-norm columns of $\boldsymbol{R}$ imply $\|\boldsymbol{L}\boldsymbol{R}^\top\|_F^2 = \mathrm{tr}(\boldsymbol{R}^\top\boldsymbol{R}) = r$. Mutual orthogonality of the right columns is not required.

These formulas avoid a new global norm reduction during the update. The factors target RMS $\tau = 0.2$; finite-precision orthogonalization, normalization epsilons, and zero coefficients can change the attained RMS.

D.2 Learning-rate scaling

Let γ denote the peak of the learning-rate schedule γ_t and $\gamma^\star$ its optimal value. We sweep γ at four smaller model sizes, fit a power law for $\gamma^\star$, and extrapolate the fit to 772M. The 124M, 354M, and 772M models have the architectures of GPT-2 Small, Medium, and Large (Section 6). Table 9 lists the sizes and sweeps, and Figure 3 shows the fits.

Table 9: Model sizes and learning-rate sweeps. All sizes use head dimension 64, 491,520 tokens per optimizer step, 10% linear warmup, and 0.41B training tokens per layer; N_{param} is the parameter count, including token embeddings and excluding position embeddings, and D is the training token budget. Sweeps use the grid of Equation 7, and grid ranges are inclusive. For swept sizes, $\widehat{\gamma}^\star$ is the optimum of a quadratic in j fitted to the five grid points centered on the sweep's best run, with both seeds at 62M; its 68% interval comes from a parametric bootstrap at the measured seed noise. For 772M, $\widehat{\gamma}^\star$ is the extrapolation of Equation 11 with a 68% residual-bootstrap interval; the model is trained at $\gamma = 2.5 \times 10^{-3}$ ($j = -2$).

N_{param}	Layers	Width	Heads	Steps	D	Grid j	Runs	$\widehat{\gamma}^\star$ (10^{-3})
Swept sizes								
62M	6	640	10	5,000	2.46B	$-4 \ldots 4$	$9 + 5^a$	4.16 [3.74, 4.45]
124M	12	768	12	10,000	4.92B	$-4 \ldots 4$	9	3.56 [3.42, 3.72]
219M	18	896	14	15,000	7.37B	$-3 \ldots 1$	5	3.56 [3.31, 3.83]
354M	24	1024	16	20,000	9.83B	$-3 \ldots 1$	5	3.13 [2.43, 3.60]
Extrapolated size								
772M	36	1280	20	30,000	14.75B	n/a	n/a	2.94 [2.70, 3.20]

[a] Second seed over $j = -2 \ldots 2$.

Optimizer and update scale. The sweeps train all parameters with AdamW (Loshchilov & Hutter, 2019) at a fixed weight decay, so γ is the only swept quantity. In the standard parametrization, AdamW keeps the per-element update magnitude fixed as width grows. The maximal update parametrization (μP) instead scales the Adam learning rate of hidden weights inversely with fan-in, which keeps the optimal learning rate stable across widths (Yang et al., 2021). We therefore measure how $\gamma^\star$ depends on model size instead of assuming that it transfers.

Grid search. We sweep γ over a geometric grid with ratio $\sqrt{2}$,

$$\gamma = 5 \times 10^{-3} \cdot 2^{j/2}, \qquad j \in \mathbb{Z}, \tag{7}$$

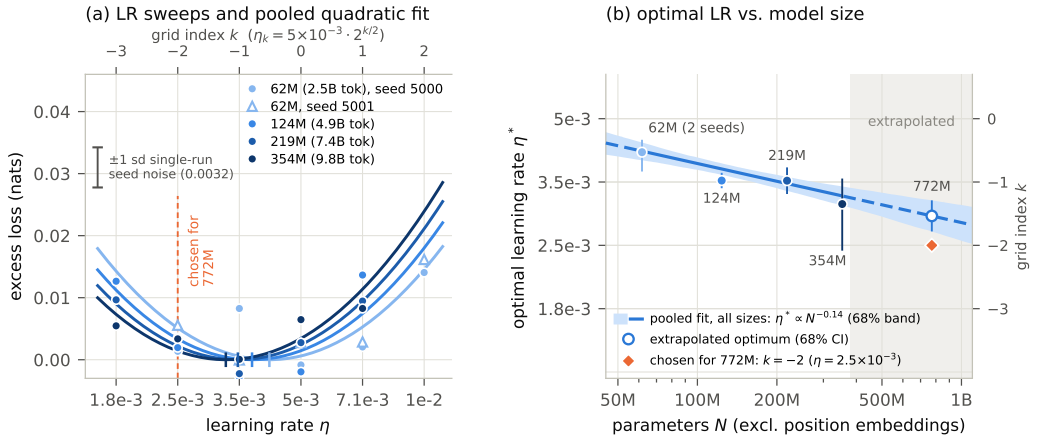


Figure 3: Learning-rate scaling. (a) Final validation loss relative to the fitted per-size minimum, with the shared-curvature parabolas of Equation 8; ticks at zero mark the fitted optima. Both seeds at 62M are shown, and the bar marks the estimated run-to-run standard deviation. (b) Optimal learning rate against parameter count. Filled points are the per-size optima of Table 9 with 68% intervals, and the line and band are the fitted law of Equation 11 with its 68% bootstrap band. The open circle is the extrapolated optimum at 772M, and the diamond marks the learning rate used, $\gamma = 2.5 \times 10^{-3}$ ($j = -2$). The shaded region lies beyond the largest swept size. The figure writes the learning rate as η, the parameter count as N, and the grid index as k.

so adjacent grid points differ by a factor of $\sqrt{2}$ and points two steps apart by a factor of 2. The sweep covers four sizes whose parameter counts N_{param}, including token embeddings and excluding position embeddings, are 62M, 124M, 219M, and 354M. The 62M and 124M sweeps use $j = -4, \ldots, 4$, from 1.25×10^{-3} to 2×10^{-2}. The 219M and 354M sweeps use $j = -3, \ldots, 1$, from 1.77×10^{-3} to 7.07×10^{-3}, because the two smaller sweeps had placed the optimum inside that window. A second 62M sweep over $j = -2, \ldots, 2$ with a different seed measures run-to-run noise. Within a size, only γ varies.

Training recipe. All sizes follow one recipe: head dimension 64, 491,520 tokens per optimizer step (480 sequences of 1024 tokens), and 0.41B training tokens per layer, so the token budget D grows in proportion to depth. The learning rate warms up linearly over the first 10% of steps and then follows a cosine decay to 1% of its peak; the weight decay λ is 0.1. Each run is scored by its final validation loss. The 33 sweep runs use 2.0×10^{20} FLOPs in total, counting $6N_{\text{param}}D$ per run.

Noise floor. Across the five learning rates run with both seeds at 62M, the final losses differ by 0.005 nats RMS, giving an estimated run-to-run standard deviation of $\sigma \approx 0.003$ nats. At 62M, the best grid point changes from $j = 0$ to $j = -1$ between the two seeds. We estimate $\gamma^\star$ by fitting the sweeps, with this noise estimate used for the per-size bootstrap intervals.

Joint fit. Let $\mathcal{L}(j; N_{\text{param}})$ denote the final validation loss at grid index j for a model with N_{param} parameters. Near its minimum, $\mathcal{L}$ is well approximated by a quadratic in $\log \gamma$, that is, in j. We fit all swept sizes jointly with a shared curvature a and an optimum that follows a power law in N_{param}:

$$\mathcal{L}(j; N_{\text{param}}) = \mathcal{L}^\star(N_{\text{param}}) + a\left(j - j^\star(N_{\text{param}})\right)^2, \tag{8}$$

$$j^\star(N_{\text{param}}) = j^\star_{\text{ref}} - 2\alpha \log_2\left(N_{\text{param}}/N_{\text{ref}}\right), \tag{9}$$

where each size has a free offset $\mathcal{L}^\star$, and the reference size N_{ref} is 354M, the largest swept size, which anchors the fit inside the data. The fit also determines $j^\star_{\text{ref}}$, the optimal grid index at N_{ref}, and the exponent α. Because $j = 2\log_2(\gamma/5 \times 10^{-3})$, Equation 9 is equivalent to the power law

$$\gamma^\star(N_{\text{param}}) = \gamma^\star_{\text{ref}}\left(N_{\text{param}}/N_{\text{ref}}\right)^{-\alpha}, \tag{10}$$

where $\gamma^\star_{\mathrm{ref}} = 5 \times 10^{-3} \cdot 2^{j^\star_{\mathrm{ref}}/2}$ is the optimal learning rate at N_{ref}. The parabola is a local approximation, and the sweeps depart from it beyond about two grid steps from the optimum. Each size therefore contributes only the five grid points centered on its best run, with both seeds at 62M, which gives 25 points for 7 parameters. We fit by nonlinear least squares and derive confidence intervals from 600 residual-bootstrap resamples of the joint fit.

Fitted law. The joint fit gives

$$\gamma^\star(N_{\mathrm{param}}) = 3.28 \times 10^{-3} \left(\frac{N_{\mathrm{param}}}{354\,\mathrm{M}} \right)^{-0.140}, \tag{11}$$

with $\alpha = 0.140$ (68% interval $[0.096, 0.186]$; 95% interval $[0.051, 0.224]$) and a 68% interval of $[3.11, 3.45] \times 10^{-3}$ for $\gamma^\star_{\mathrm{ref}}$. The shared curvature is $a = 0.0023$ nats per squared grid step, so a learning rate off by a factor of $\sqrt{2}$ raises the final loss by about 0.002 nats, and one off by a factor of 2 raises it by about 0.009 nats. The residual standard deviation of the fit is 0.003 nats, which matches the measured seed noise, so the model describes the sweeps to within noise. Pooling the sizes also keeps a near-tie within one sweep from deciding the fit; at 124M, for example, $j = -1$ and $j = 0$ differ by less than 0.001 nats.

Compute coordinate. Because the recipe grows width, depth, and data together, N_{param} and the training compute $C = 6N_{\mathrm{param}}D$ are interchangeable coordinates within this family. Fitting against C instead of N_{param} gives $\gamma^\star \propto C^{-0.079}$, with a 68% interval of $[0.053, 0.106]$ for the exponent. This exponent is smaller than the 0.125 that DeepSeek-AI (2024) fit for AdamW under a different recipe. Their fit also scales the batch size with compute and measures compute as the number of training tokens times the non-embedding FLOPs per token, a count that includes attention. Our C instead charges $6N_{\mathrm{param}}$ FLOPs per token, including the token embeddings, so the two exponents are not directly comparable.

Extrapolation to 772M. The 772M model has 2.2 times the parameters of the largest swept size, 354M. Equation 11 predicts $\gamma^\star = 2.94 \times 10^{-3}$ at 772M (68% interval $[2.70, 3.20] \times 10^{-3}$; $j^\star = -1.54$). We round the predicted $j^\star$ to the nearest grid value, $j = -2$, and train the model at $\gamma = 2.5 \times 10^{-3}$. Rounding moves j by at most half a grid step, so by the fitted curvature it raises the loss by at most $a/4$, less than 0.001 nats. At 772M, $j^\star$ lies almost midway between $j = -2$ and $j = -1$, and the 68% interval spans less than one grid step and contains neither value; by the fitted curvature, both choices lie within 0.001 nats of the optimum.

Limitations. The 772M prediction extrapolates beyond the swept range. Because width, depth, and data grow together, Equation 11 describes this recipe and cannot attribute the change in $\gamma^\star$ to any single factor. In particular, it combines the effect of model size with that of the training horizon, which also shifts the optimal learning rate (Bjorck et al., 2025). Batch size, warmup fraction, and schedule are held fixed, and the fit applies only to these settings.

E Ablations

Configurations for the basis and normalization ablations. We use three alternative configurations to isolate the choices of activation-side basis and sign normalization in Dobb. Dobb-ℓ_2 retains the activation-side basis and normalizes every coefficient column to unit Euclidean norm, with a zero column mapped to zero. For an output-side basis, we partition the output coordinates and construct $\boldsymbol{C}_{t,b}$ from $\boldsymbol{D}_{t,\mathcal{S}_b,:}$. We project the transposed momentum slice $\boldsymbol{M}^\top_{t,\mathcal{S}_b,:}$, normalize its coefficients, reconstruct, and transpose back. Dobb-G uses this basis with sign normalization and denominator $\sqrt{n}$; Dobb-G-ℓ_2 instead normalizes each coefficient column to unit Euclidean norm. The four configurations are summarized below.

Basis source	Sign normalization	Column normalization
Input second moment $\boldsymbol{A}_t$	Dobb	Dobb-ℓ_2
Output-gradient second moment $\boldsymbol{G}_t$	Dobb-G	Dobb-G-ℓ_2

The common-RMS multipliers are $\tau\sqrt{m}$ for the activation-side configurations and $\tau\sqrt{n}$ for the output-side configurations, as derived in Appendix D.1.

Each ablation changes its named component while holding the shared-RMS scale, weight decay, token budget, and evaluation procedure fixed. Each ablation also uses the principal comparison's learning rate at its model size, selected as described in Appendix D. Unless a subsection states otherwise, each ablation trains GPT-2 Small with $B = 1$ for 10,000 iterations with seed 5000, using data parallelism on four H100 GPUs, which replicates rather than shards the model.

E.1 Sign versus ℓ_2 normalization

Dobb and Dobb-ℓ_2 share the activation-side basis and differ only in the coefficient normalization $\mathcal{N}$ of Equation 5: Dobb applies the sign, and Dobb-ℓ_2 scales each coefficient column to unit ℓ_2 norm. Both use the shared-RMS scale of Equation 6, so the comparison holds the nominal update RMS fixed. This ablation departs from the default setting above: it uses the principal comparison's runs of both variants on GPT-2 Small, Medium, and Large at $B \in \{1, 4, 16, 64\}$, with the training budgets of Table 3.

Table 10 shows that sign normalization gives the lower final loss in all twelve matched comparisons, by 0.008 to 0.026. On Small, the margin decreases from 0.026 at $B = 1$ to 0.008 at $B = 64$: Dobb's loss rises by 0.020 over this range, while Dobb-ℓ_2's loss stays between 3.036 and 3.045. On Medium and Large, the margin stays between 0.014 and 0.019 and between 0.019 and 0.021, respectively. At these two scales, the two variants change by similar amounts from $B = 1$ to $B = 64$: by 0.007 and 0.006 on Medium and by 0.021 and 0.020 on Large. Against Dion at rank fraction $1/B$ in Table 3, Dobb-ℓ_2 has lower loss at $B = 16$ and $B = 64$ on all three scales. At $B = 4$, Dobb-ℓ_2 is 0.001 below Dion on Small and 0.006 and 0.013 above it on Medium and Large.

The two normalizations also differ in derived communication under TP. Sign normalization acts on each coefficient independently, so when each block lies within one shard, Dobb adds no optimizer-specific cross-TP payload. Column normalization sums squares over each coefficient column, so Dobb-ℓ_2 reduces n scalars per matrix when the output axis is sharded (Table 6).

Table 10: Normalization ablation for the activation-side basis: final validation loss in nats for Dobb (sign) and Dobb-ℓ_2 (column ℓ_2), with the training budgets of Table 3. The Dobb columns repeat Table 3; Δ is the Dobb-ℓ_2 loss minus the Dobb loss.

	Small			Medium			Large		
B	Dobb	Dobb-ℓ_2	Δ	Dobb	Dobb-ℓ_2	Δ	Dobb	Dobb-ℓ_2	Δ
1	3.017	3.043	0.026	2.735	2.753	0.018	2.579	2.599	0.020
4	3.024	3.037	0.013	2.741	2.760	0.019	2.588	2.609	0.021
16	3.026	3.036	0.010	2.746	2.760	0.014	2.595	2.615	0.020
64	3.037	3.045	0.008	2.742	2.759	0.017	2.600	2.619	0.019

E.2 Input-activation versus output-gradient bases

The four Dobb variants separate the basis source from the coefficient normalization. Appendix E.1 compares the activation-side variants across scales; this ablation adds Dobb-G and Dobb-G-ℓ_2 at $B \in \{1, 4, 16, 64\}$ with $T = 9$. We compare A versus G at each fixed normalization and block count, then compare the normalizations at each fixed basis source.

Table 11 shows that the activation basis gives the lower final loss in all eight matched comparisons. Dobb-G exceeds Dobb by 0.035 to 0.054, and Dobb-G-ℓ_2 exceeds Dobb-ℓ_2 by 0.034 to 0.039. The two groups do not overlap: the highest activation-side loss, 3.045, is 0.022 below the lowest output-gradient loss, 3.067. Every normalization difference, at most 0.026, is smaller than every basis-source difference. With the activation basis, sign normalization gives the lower loss at every B, as Appendix E.1 reports for all three scales. With the output-gradient basis, sign normalization gives lower loss by 0.006 at $B = 1$ and 0.011 at $B = 64$, while at $B = 4$ and $B = 16$ the two normalizations differ by at most 0.004.

Table 11: Basis-source and normalization ablation on GPT-2 Small with $T = 9$: final validation loss in nats after 10,000 iterations. The activation-side columns repeat the Small entries of Table 10.

	Activation basis		Output-gradient basis	
B	Dobb	Dobb-ℓ_2	Dobb-G	Dobb-G-ℓ_2
1	3.017	3.043	3.071	3.077
4	3.024	3.037	3.067	3.071
16	3.026	3.036	3.076	3.075
64	3.037	3.045	3.072	3.083

E.3 Basis-refresh period

The principal period is $T = 9$. We compare final loss and iteration time at $T = 1, 4, 9, 16, 100$, keeping the block partition and capture policy fixed. Because the capture policy records second moments only on refresh iterations, T sets both the age of the basis and the number of micro-batches that enter the second-moment average. Iteration times include second-moment capture.

Table 12 shows that refreshing at every iteration gives the lowest final loss, 3.005, which is 0.012 below the 3.017 of the principal $T = 9$. From $T = 4$ to $T = 100$, final loss stays between 3.014 and 3.017 without a monotone trend. Relative to $T = 1$, the principal $T = 9$ reduces the amortized iteration time from 762.5 to 403.6 ms, or by 47%. With θ_{R} and θ_{N} the median times of iterations with and without a refresh, the amortized time $(\theta_{\mathrm{R}} + (T-1)\theta_{\mathrm{N}})/T = \theta_{\mathrm{N}} + (\theta_{\mathrm{R}} - \theta_{\mathrm{N}})/T$ is a constant non-refresh term plus a refresh term that decays as $1/T$, so each increase in T saves less time than the previous one. At $T = 9$ the refresh term is 43.8 ms, which bounds the saving available from any longer period. Beyond $T = 9$, longer periods lower the amortized iteration time by at most 40.2 ms, to 363.4 ms at $T = 100$, and change final loss by at most 0.002.

Table 12: Refresh-period ablation for Dobb on GPT-2 Small with $B = 1$. Final loss is validation loss in nats after 10,000 iterations. Full iteration is the amortized time in milliseconds, $(\theta_{\mathrm{R}} + (T-1)\theta_{\mathrm{N}})/T$, where θ_{R} and θ_{N} are the median times of iterations with and without an eigenbasis refresh.

T	Final loss	Full iteration (ms)
1	3.005	762.5
4	3.014	459.3
9	3.017	403.6
16	3.017	384.8
100	3.015	363.4

E.4 Orthogonal iterations per refresh

Let k denote the number of warm-started orthogonal iterations per refresh; Algorithm 2 uses $k = 1$. We compare this principal setting with $k = 2$ and $k = 3$ and with an exact eigendecomposition at every refresh. Each additional iteration repeats the product $\widetilde{\boldsymbol{C}}_b \boldsymbol{U}_b$, where $\widetilde{\boldsymbol{C}}_b = \boldsymbol{C}_{t,b} + \varepsilon \boldsymbol{I}$, and its QR orthonormalization, so the derived refresh arithmetic grows linearly in k. The exact variant computes the eigendecomposition of $\widetilde{\boldsymbol{C}}_b$ in fp32 at every refresh and discards the warm start. The block update is invariant to the sign and order of the basis columns, so the sign and ordering conventions of the eigensolver do not affect it. All four settings hold $T = 9$, the block partition, the capture policy, the basis precision, and the shared-RMS scale fixed.

Iteration times include second-moment capture.

Table 13 shows that final loss varies by 0.004 across the four settings, from 3.015 for the exact eigendecomposition to 3.019 at $k = 2$, and does not decrease monotonically with k. The extra cost falls on refresh iterations. Non-refresh iterations take 358.5 to 360.1 ms in all four settings, while each additional orthogonal iteration adds 375 to 381 ms to a refresh iteration and the exact eigendecomposition adds 632 ms. At $T = 9$ these refresh costs raise the amortized iteration time by 10.6%, 20.6%, and 17.5% for $k = 2$, $k = 3$, and the exact variant. The principal $k = 1$ setting has

the lowest amortized iteration time, 403.6 ms; exact eigendecomposition takes 474.4 ms and lowers loss from 3.017 to 3.015.

Table 13: Orthogonal-iteration ablation for Dobb at $T = 9$ on GPT-2 Small with $B = 1$. Final loss is validation loss in nats after 10,000 iterations. Non-refresh and Refresh are median full-iteration times in milliseconds on iterations without and with an eigenbasis refresh; Amortized time is $(\text{Refresh} + 8 \cdot \text{Non-refresh})/9$.

		Full iteration (ms)		
Basis refresh	Final loss	Non-refresh	Refresh	Amortized
$k = 1$ (principal)	3.017	359.5	756.5	403.6
$k = 2$	3.019	359.9	1137.8	446.4
$k = 3$	3.017	358.5	1512.3	486.7
Exact eigendecomposition	3.015	360.1	1388.9	474.4

E.5 Householder versus Cholesky QR

The GPT-2 training runs orthonormalize each warm-started refresh with Householder QR, while the Qwen3 iteration-time measurements use the Cholesky QR of Algorithm 4. This ablation replaces Householder QR with Cholesky QR at $\omega = 10^{-2}$ in Dobb at $B = 1$ and $B = 4$. It holds $T = 9$, the cold-start eigendecomposition, the capture policy, the precision, and the shared-RMS scale fixed.

Table 14 shows that the QR routine changes final loss by at most 0.002: Cholesky QR gives 3.016 against 3.017 for Householder QR at $B = 1$, and 3.022 against 3.024 at $B = 4$. At iteration 1000, Cholesky QR has 0.012 higher validation loss at both block counts; from iteration 2000 onward, the two routines differ by at most 0.005. Cholesky QR lowers the iteration time. The saving falls primarily on refresh iterations: their median time decreases from 760.3 to 486.3 ms at $B = 1$ and from 604.7 to 431.8 ms at $B = 4$, while the non-refresh medians change by at most 3.5 ms. At $T = 9$, Cholesky QR lowers the amortized iteration time from 409.1 to 375.6 ms, a reduction of 33.5 ms (8.2%), at $B = 1$ and from 417.6 to 397.5 ms, a reduction of 20.1 ms (4.8%), at $B = 4$.

Table 14: QR-routine ablation for Dobb at $T = 9$ on GPT-2 Small. Final loss is validation loss in nats after 10,000 iterations. Non-refresh and Refresh are median full-iteration times in milliseconds on iterations without and with an eigenbasis refresh; Amortized time is $(\text{Refresh}+8\cdot\text{Non-refresh})/9$.

			Full iteration (ms)		
B	QR routine	Final loss	Non-refresh	Refresh	Amortized
1	Householder (principal)	3.017	365.2	760.3	409.1
	Cholesky	3.016	361.7	486.3	375.6
4	Householder	3.024	394.2	604.7	417.6
	Cholesky	3.022	393.2	431.8	397.5

E.6 First-micro-batch capture

The principal policy, first-micro-batch capture, forms the second moment on each refresh iteration from the first micro-batch alone. The alternative, all-micro-batch capture, forms a token-weighted second moment from every micro-batch of that iteration. Both policies use the full mini-batch stochastic gradient for momentum, so they differ only in the data from which the second moment is estimated. We compare quality and complete iteration time at fixed $T = 9$, global batch, and accumulation count. Each iteration accumulates gradients over eight micro-batches of 15 sequences on each of the four GPUs, so first-micro-batch capture pools 60 of the iteration's 480 sequences across GPUs. We run the comparison for Dobb and Dobb-G.

Table 15 shows that capturing every micro-batch does not lower final loss. For Dobb it gives 3.021 against 3.017 with first-micro-batch capture, and for Dobb-G both policies give 3.071. Capturing every micro-batch raises the median refresh iteration time by 132.4 ms for Dobb and by 59.6 ms for

Dobb-G, while the non-refresh medians differ by at most 0.9 ms. At $T = 9$ this raises the amortized iteration time by 15.5 ms (3.8%) for Dobb and by 6.5 ms (1.5%) for Dobb-G.

Table 15: Capture-policy ablation for Dobb and Dobb-G at $T = 9$ on GPT-2 Small with $B = 1$. Final loss is validation loss in nats after 10,000 iterations. Non-refresh and Refresh are median full-iteration times in milliseconds on iterations without and with an eigenbasis refresh; Amortized is $(\text{Refresh} + 8 \cdot \text{Non-refresh})/9$.

			Full iteration (ms)		
Method	Capture policy	Final loss	Non-refresh	Refresh	Amortized
Dobb	First micro-batch	3.017	360.9	755.7	404.8
	All micro-batches	3.021	361.8	888.1	420.3
Dobb-G	First micro-batch	3.071	370.4	835.1	422.0
	All micro-batches	3.071	370.2	894.7	428.5

E.7 Second-moment and basis precision

The principal setting accumulates the block second moments in fp32 and stores the bases in bf16. The implementation forms each captured outer product in the activation dtype and casts it to the accumulator dtype; at each refresh it casts the accumulator to fp32 before adding the shift and factorizing. Accumulation precision therefore affects the exponential average and its storage, not the factorization arithmetic, while basis precision affects the projection and reconstruction in Equation 5. For side A the accumulator and the basis each hold n^2/B entries per matrix, so each bf16 choice halves that buffer. We compare the four combinations of accumulation and basis precision at fixed B, T, capture policy, and shared-RMS scale.

Table 16 shows that fp32 accumulation gives lower final loss than bf16 accumulation at both basis precisions: 3.023 against 3.028 with fp32 bases, and 3.017 against 3.027 with bf16 bases. Basis precision changes final loss by 0.006 at fp32 accumulation and by 0.001 at bf16 accumulation, in both cases in favor of bf16 bases. The principal combination, fp32 accumulation with bf16 bases, has the lowest final loss of the four. Relative to the principal setting, bf16 accumulation reduces the derived state from $6n^2/B$ to $4n^2/B$ bytes per matrix and raises final loss by 0.010.

Table 16: Precision ablation for Dobb at $T = 9$ on GPT-2 Small with $B = 1$. Final loss is validation loss in nats after 10,000 iterations. Median iteration is the median full-iteration time in milliseconds on iterations without an eigenbasis refresh. The state column is derived from buffer sizes, not measured.

Accumulation	Basis	Final loss	Median iteration (ms)	State per matrix (bytes)
fp32	fp32	3.023	361.2	$8n^2/B$
fp32	bf16 (principal)	3.017	360.4	$6n^2/B$
bf16	fp32	3.028	361.0	$6n^2/B$
bf16	bf16	3.027	360.9	$4n^2/B$

E.8 Gradient-derived second moments

We also consider a hook-free basis constructed from the mini-batch weight gradient itself. The input- and output-side gradient Gram matrices are

$$\begin{aligned} \boldsymbol{S}_t^{\text{in}} &= \big[\nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W}_t)\big]^\top \big[\nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W}_t)\big], \\ \boldsymbol{S}_t^{\text{out}} &= \big[\nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W}_t)\big] \big[\nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W}_t)\big]^\top. \end{aligned} \tag{12}$$

These gradient Gram matrices are the type of statistics accumulated by Shampoo and by SOAP, which runs Adam in the Shampoo eigenbasis (Gupta et al., 2018; Vyas et al., 2025). A one-sided block eigenbasis uses sign or column normalization. The input-side construction stores a block-diagonal second moment and its basis, $2n^2/B$ scalars, and each refresh factorizes B blocks of

width n/B. The ablation holds B, T, and nominal update RMS fixed, runs each source under both normalizations, and compares final loss. The two kinds of source also differ in sampling: each gradient Gram matrix enters its exponential average at every iteration and uses the full mini-batch gradient, whereas the capture hooks record only the first micro-batch on refresh iterations. The comparison changes both the statistic and its sampling schedule.

Table 17 shows that every input-side source has lower final loss than every output-side source under both normalizations. Under ℓ_∞ normalization, final loss rises from 3.017 for the activation second moment to 3.027 and 3.055 for the input- and output-side gradient Gram matrices, and to 3.071 for the output-gradient second moment. Under ℓ_2 normalization the ordering is the same, except that the activation second moment and the input-side gradient Gram matrix differ by 0.001 (3.043 and 3.042). On the input side, the activation second moment has 0.010 lower final loss than the gradient Gram matrix under ℓ_∞ normalization, although it is updated less often and from fewer sequences. On the output side, the gradient Gram matrix has lower final loss than the output-gradient second moment by 0.016 under ℓ_∞ and by 0.013 under ℓ_2 normalization; this difference combines the statistic with its denser sampling.

Table 17: Second-moment-source ablation at $T = 9$ on GPT-2 Small with $B = 1$. Final loss is validation loss in nats after 10,000 iterations.

Norm	Source	Final loss
ℓ_∞	Activation second moment (Dobb)	3.017
	Gradient Gram, input side	3.027
	Gradient Gram, output side	3.055
	Output-gradient second moment (Dobb-G)	3.071
ℓ_2	Activation second moment (Dobb-ℓ_2)	3.043
	Gradient Gram, input side	3.042
	Gradient Gram, output side	3.064
	Output-gradient second moment (Dobb-G-ℓ_2)	3.077

E.9 Nesterov momentum

The principal Dobb definition uses ordinary momentum with Nesterov disabled. The ablation compares the update input $\boldsymbol{M}_t$ with the Nesterov input $\nabla_{\boldsymbol{W}} f_{\mathcal{B}_t}(\boldsymbol{W}_t) + \mu \boldsymbol{M}_t$ after the same momentum accumulation. It tests whether the additional gradient contribution changes final quality under matched update RMS. The principal comparison disables Nesterov momentum for Dobb, Muon, and RMNP at every model size. We test Dobb at $B = 1$, 4, and 16 with $T = 9$, and the Muon and RMNP baselines.

Table 18 shows that the effect of Nesterov momentum on Dobb changes sign with B. It lowers final loss by 0.003 at $B = 1$ (3.017 to 3.014) and by 0.001 at $B = 4$, and raises it by 0.004 at $B = 16$ (3.026 to 3.030). For the baselines, Nesterov momentum leaves the final loss of Muon at 3.004 and lowers that of RMNP by 0.011, from 3.083 to 3.072. Nesterov momentum changes the amortized iteration time of Dobb by at most 0.7%.

Table 18: Nesterov ablation on GPT-2 Small: final validation loss in nats after 10,000 iterations. Dobb uses $T = 9$.

Method	B	Off	On
Dobb	1	3.017	3.014
Dobb	4	3.024	3.023
Dobb	16	3.026	3.030
Muon	n/a	3.004	3.004
RMNP	n/a	3.083	3.072

F Details of the distributed iteration-time experiments

This appendix gives the settings and the full timing breakdown of the two Qwen3 series summarized in Table 4, Qwen3-8B under FSDP and Qwen3-32B under FSDP with TP. It also reports timings for the output-gradient basis with column normalization, Dobb-G-ℓ_2, defined in Appendix E. All entries are measured iteration times; the communication counts in Section 5 are analytical, and neither series instrumented bytes. Both series use the fixed-count partition of Appendix B, with $B = F$ blocks per logical matrix. Each series pairs Dion's rank fraction with a width of $1024/F$, following the width–rank match of Section 5. At each refresh, the timing implementation orthonormalizes the orthogonal-iteration product with the Cholesky QR of Algorithm 4 at $\omega = 10^{-2}$, falling back to Householder QR if a Cholesky factorization fails. Model compilation is disabled, and timings are logged on one rank. Total iteration time is the comparison metric, because the hooks place second-moment capture, and in one implementation its reduction, inside the forward/backward timer.

F.1 Qwen3 FSDP and tensor-parallel series

Setup. The Qwen3 series used the matrix shapes of the dense Qwen3-8B and Qwen3-32B models with random initialization (Yang et al., 2025). Qwen3-8B has 36 layers of width 4096 and MLP width 12288; Qwen3-32B has 64 layers of width 5120 and MLP width 25600. Both models have eight key-value heads of dimension 128, so the key and value projections have 1024 outputs. Training read OpenWebText under the GPT-2 tokenizer, whose token indices lie within the Qwen3 vocabulary. Each device processed one 1024-token sequence per iteration, and each run lasted 80 iterations. Means use the last 63 of 69 timed iterations, seven complete $T = 9$ cycles. The timed window excludes first-iteration startup: Dion's optimizer compilation and the exact cold-start eigendecompositions of Dobb and Dobb-G-ℓ_2. Dobb and Dobb-G-ℓ_2 used fp32 second-moment accumulation and one warm orthogonal iteration per refresh. The runs at $F = 8$ occupied one host, and the runs at larger F spanned hosts; the GPU type is $---$. Dion ran in an earlier campaign than Dobb and Dobb-G-ℓ_2, with separate allocations; the source of their forward/backward timing differences is unidentified.

Block layout. With $B = F$, the blocks on the 1024-wide key and value axes have width $1024/F$: 128, 64, 32, and 16 for Qwen3-8B at $F \in \{8, 16, 32, 64\}$, and 32 and 16 for Qwen3-32B at $F \in \{32, 64\}$. Longer basis axes receive proportionally wider blocks. In the Qwen3-8B series, $B = F$ makes each output-axis block coincide with one rank's FSDP shard, and Dobb-G-ℓ_2 requires each rank to hold whole blocks. Under TP, the timing implementation applies FSDP to the input axis of column-parallel weights and to the output axis of row-parallel weights, so the TP and FSDP shards of each weight lie on different axes. With $B = F$, each block then lies within one shard. In the Qwen3-32B TP runs, B is 32 or 64 on every matrix, so $B \geq 16P$. Dobb issues no optimizer collective on the TP group. Dobb-G-ℓ_2 completes its column norms across TP for row-parallel weights, the $\mathcal{O}(m)$ entry of Table 6, and across FSDP for column-parallel weights. The TP series compares Dobb and Dobb-G-ℓ_2 with Dion; the timing implementation does not support Muon under TP.

FSDP on Qwen3-8B. Table 19 gives the three-way comparison for Qwen3-8B. At $F = 8$, Dobb and Dobb-G-ℓ_2 take 347.3 and 371.6 ms per iteration against 419.3 ms for Dion. At $F \geq 16$, forward/backward takes 5.1 to 5.9 s, and the Dobb and Dobb-G-ℓ_2 iteration means are 89.7 to 242.7 ms below Dion's. At these degrees, the Dobb and Dobb-G-ℓ_2 forward/backward means are 6.9 to 157.6 ms below Dion's, comparable to the 83.4 to 123.5 ms gap in optimizer-call means. At $F = 16$, Dion's iteration mean minus its entire optimizer call is 5202.5 ms, still above the Dobb iteration mean of 5176.3 ms. The total-time differences at $F \geq 16$ therefore combine optimizer-call and forward/backward differences. Across the Qwen3-8B series, refresh calls take 92.1 to 331.7 ms for Dobb and 134.1 to 336.3 ms for Dobb-G-ℓ_2, and both peak at $F = 16$.

Tensor parallelism on Qwen3-32B. Table 20 reports iteration times at $(F, P) \in \{(32, 2), (64, 2), (64, 4)\}$. Across Dobb and Dobb-G-ℓ_2, optimizer-call means range from 69.9 to 260.2 ms, against Dion's 797.3 to 1307.7 ms, while forward/backward takes 13.4 to 17.4 s; Dion's optimizer call is at most 7.1% of its iteration. At $P = 2$, the Dobb and Dobb-G-ℓ_2 iteration means are 1655.9 to 2381.7 ms below Dion's. At $P = 4$, Dobb and Dobb-G-ℓ_2 take 13,915.3 and 13,729.5 ms

Table 19: Measured Qwen3-8B FSDP iteration times in milliseconds, with $P = 1$ and $B = F$ blocks per logical matrix. Each row is a mean over the last 63 of 69 timed iterations, seven complete $T = 9$ cycles. Non-refresh and Refresh give the mean optimizer-call time on iterations without and with an eigenbasis refresh; Amortized is the mean over all 63 iterations. Dion performs the same update on every iteration, so its split entries are empty. Total adds the optimizer phase, which also includes gradient-norm computation, the scheduler, and gradient zeroing, to Fwd/bwd.

		Optimizer				
F	Method	Non-refresh	Refresh	Amortized	Fwd/bwd	Total
8	Dion	–	–	159.5	254.3	419.3
	Dobb	75.7	170.0	86.2	255.9	347.3
	Dobb-G-ℓ_2	88.7	219.5	103.2	262.5	371.6
16	Dion	–	–	146.1	5195.6	5348.6
	Dobb	26.1	331.7	60.1	5109.9	5176.3
	Dobb-G-ℓ_2	28.5	336.3	62.7	5188.7	5258.9
32	Dion	–	–	114.8	5509.4	5631.2
	Dobb	10.7	165.0	27.9	5430.1	5464.6
	Dobb-G-ℓ_2	11.0	179.2	29.7	5351.8	5388.5
64	Dion	–	–	138.6	5890.7	6039.2
	Dobb	5.5	92.1	15.1	5806.3	5830.0
	Dobb-G-ℓ_2	5.5	134.1	19.8	5819.3	5849.0

Table 20: Measured Qwen3-32B FSDP and TP iteration times in milliseconds, with $B = F$ blocks per logical matrix. Columns and averaging follow Table 19.

			Optimizer				
F	P	Method	Non-refresh	Refresh	Amortized	Fwd/bwd	Total
32	2	Dion	–	–	1307.7	17,050.2	18,435.0
		Dobb	32.8	1249.2	168.0	16,106.7	16,316.8
		Dobb-G-ℓ_2	53.0	1917.9	260.2	15,710.5	16,053.3
64	2	Dion	–	–	1287.5	17,414.2	18,825.6
		Dobb	14.5	634.9	83.4	17,021.5	17,169.7
		Dobb-G-ℓ_2	27.9	1007.3	136.7	16,722.5	16,921.1
64	4	Dion	–	–	797.3	13,391.9	14,282.4
		Dobb	8.5	560.6	69.9	13,798.5	13,915.3
		Dobb-G-ℓ_2	25.7	876.8	120.2	13,521.4	13,729.5

against 14,282.4 ms for Dion. At $P = 2$, the Dobb and Dobb-G-ℓ_2 forward/backward means are 392.7 to 1339.7 ms below Dion's. At $P = 4$, both exceed Dion's, by 406.6 ms for Dobb and 129.5 ms for Dobb-G-ℓ_2. The TP implementation of Dobb reduces its second moments inside the optimizer call rather than from the capture hook. Refresh calls take 560.6 to 1249.2 ms for Dobb and 876.8 to 1917.9 ms for Dobb-G-ℓ_2.