FULL-STACK NVFP4: STABLE, ROLE-AWARE 4-BIT LLM PRETRAINING BEYOND LINEAR PROJECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Most NVFP4 pretraining recipes stop at Transformer projections, leaving optimizer states, optimizer computation, and attention in higher precision. Those remaining pieces fail for different reasons, so a single FP4 rule is a poor fit. We introduce **Full-Stack NVFP4**, four recipes that assign precision by numerical role. **Subspace-Protected LoRA-SVD** keeps a compact BF16 projection subspace inside full-shape NVFP4 computation; gradient decoupling and periodic SVD realignment keep the low-rank factors stable and cut the linear-only loss gap from **1.40%** to **0.61%**. **NVFP4-State AdamW** transforms persistent momentum before storage, **NVFP4 Root** runs Newton–Schulz products directly in NVFP4, and **Selective FP4 Attention** leaves softmax-sensitive products in BF16. A 3B model trained for 64B tokens reaches loss **2.286** versus **2.267** for BF16, a **0.838%** gap, with similar aggregate zero-shot scores. On four decoder blocks of one RTX 5090, Root is 2.50–2.83× faster than optimized BF16, and AdamW peak memory drops by 37.9–42.5%.

1 INTRODUCTION

LLM pretraining has moved from FP32 to BF16 and FP8 to cut memory and compute. Native NVFP4 Tensor Cores on Blackwell GPUs make 4-bit training a practical target, and recent work already keeps Transformer projections stable in NVFP4 (Abecassis et al., 2025; Chmiel et al., 2025; Castro et al., 2025; Dong et al., 2026; Cook et al., 2025; Chen et al., 2025a;b). Optimizer states, optimizer computation, and attention, however, usually remain in higher precision. Low-bit Muon has started to close part of this gap (Wu et al., 2026), but a projection-only recipe still leaves most of the training stack untouched.

The remaining modules fail in different ways. Projection error becomes more consequential as gradients weaken relative to quantization noise (Chmiel et al., 2025), AdamW's second moment is a persistent, heavy-tailed quantity that later appears in a denominator, and Root compounds small errors through repeated Newton–Schulz products. Attention adds another split: some matrix products tolerate FP4, while softmax-sensitive paths and reused forward–backward tensors do not. One FP4 rule does not cover this mix.

Full-Stack NVFP4 assigns precision by numerical function. It protects a compact projection subspace, reshapes persistent optimizer states, keeps Newton–Schulz iteration in NVFP4, and retains higher precision on probability-sensitive attention paths. The four recipes can be enabled separately or together.

Our technical contributions are: (1) a **role-aware NVFP4 framework** that distinguishes residual projection error, persistent heavy-tailed states, iterative error amplification, and softmax-sensitive attention paths; (2) four corresponding recipes—**Subspace-Protected LoRA-SVD**, **NVFP4-State AdamW** (Q-AdamW), **NVFP4 Root** (Q-Root), and **Selective FP4 Attention** (Q-Attn)—including projection stabilization that reduces the linear-only loss gap from **1.40%** to **0.61%**; and (3) modular and joint evaluation at 1B and 3B, together with four-block native measurements against optimized BF16 and NVIDIA Transformer Engine (TE).

Table 1 positions this module-wise coverage against representative low-precision training methods; Section 5 gives detailed comparisons.

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.

Table 1: **Training-module coverage of representative low-precision methods.** Wt/Act/G denotes projection weights, activations, and gradients; Train. Attn denotes a low-precision attention forward–backward path. "Pretrain Eval." indicates that the covered components are evaluated in pretraining. SageAttn3 uses INT8 for its training path and FP4 for inference. ✓ = addressed; — = not addressed; ○ = partial or higher-precision coverage.

Method	Prec.	Wt/Act/G	Opt. State	Opt. Comp.	Train. Attn	Pretrain Eval.
COAT (Xi et al., 2024)	FP8	✓	✓	—	—	✓
NVFP4 (Abecassis et al., 2025)	FP4	✓	—	—	—	✓
FP4-All-the-Way (Chmiel et al., 2025)	FP4	✓	—	—	—	✓
Quartet (Castro et al., 2025)	FP4	✓	—	—	—	✓
CHON (Dong et al., 2026)	FP4	✓	—	—	—	✓
4/6 (Cook et al., 2025)	FP4	✓	—	—	—	✓
TetraJet-v2 (Chen et al., 2025b)	FP4	✓	—	—	—	✓
Metis (Cao et al., 2025)	FP4	✓	—	—	—	✓
Low-bit Muon (Wu et al., 2026)	FP4	—	✓	✓	—	—
SageAttn3 (Zhang et al., 2025)	INT8/FP4	—	—	—	○	—
Ours	**FP4**	✓	✓	✓	✓	✓

2 PRELIMINARIES

NVFP4 format. NVFP4 (Abecassis et al., 2025) uses E2M1 encoding (1 sign, 2 exponent, 1 mantissa) with *block-wise* scaling: each 16×16 or 1×16 block shares an FP8 E4M3 scale factor s, and an additional FP32 global scale s_g: $\hat{x} = x_{\mathrm{E2M1}} \times s \times s_g$. This is finer-grained than MXFP4's block-32 E8M0 scaling (Rouhani et al., 2023).

Linear projection. A transformer linear layer computes $Y = XW^\top$ where W is the weight and X is the input activation. Training involves three GEMMs: forward, activation-gradient ($\nabla_X = \nabla_Y W$), and weight-gradient ($\nabla_W = \nabla_Y^\top X$).

AdamW optimizer. AdamW maintains first- and second-order momentum states, $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ and $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$. The update rule requires both states to persist throughout training.

Root optimizer. Root (He et al., 2025) improves Muon (Jordan et al., 2024) with soft-thresholding for first-order momentum outlier removal and a Newton–Schulz (NS) orthogonalization iteration, $X_{k+1} = aX_k + bX_k(X_k^\top X_k) + cX_k(X_k^\top X_k)^2$, with optimized coefficients a, b, and c. Root/Muon is not applied to every parameter: embeddings, `lm_head`, normalization weights, biases, LoRA factors, and narrow parameters use AdamW. The dense matmuls per iteration (typically 5 cycles) dominate Root's compute cost.

Attention operator. Standard multi-head attention computes $S = QK^\top/\sqrt{d}$, $P = \mathrm{softmax}(S)$, $O = PV$ for each head, where $Q, K, V \in \mathbb{R}^{N \times d}$. FlashAttention (Dao et al., 2022; Dao, 2023) uses I/O-aware tiling to avoid materializing the full $N \times N$ matrix. The backward pass recomputes $S = QK^\top/\sqrt{d}$ and P, and computes $dV = P^\top dO$, $dP = dO \cdot V^\top$, $dS = P \odot (dP - \mathrm{rowsum}(dO \odot O))$, $dQ = dS \cdot K/\sqrt{d}$, $dK = dS^\top \cdot Q/\sqrt{d}$. Overall, there are 2 matmuls in the forward pass and 5 in the backward pass, and the activation tensors (Q, K, V) dominate the memory and I/O cost.

Rounding and transforms. Round-to-nearest (RTN) is deterministic but may introduce bias; stochastic rounding (SR) is unbiased but random. Hadamard transforms redistribute localized outliers through low-cost butterfly operations. We use RTN primarily in forward and SR with Hadamard transforms in backward (Abecassis et al., 2025; Tseng et al., 2025; Cook et al., 2025). FP4-All-the-Way (Chmiel et al., 2025) identifies a threshold $\|\mathbf{g}\| < \sqrt{3}\sigma_q$ below which the optimization signal falls beneath approximately constant quantization noise.

3 METHOD

3.1 Role-Aware Recipe Overview

Module	Numerical difficulty	Recipe
Linear projection (§3.2)	residual projection error	NVFP4 residual + BF16 subspace
AdamW state (§3.3)	persistent, heavy-tailed moments	NVFP4-State AdamW
Root (§3.4)	iterative low-precision error	NVFP4 Root
Attention (§3.5)	softmax-sensitive paths	Selective FP4 Attention

Each recipe below targets one of these difficulties; together they define Full-Stack NVFP4. All of them reuse standard primitives: RTN, SR, random Hadamard transforms, and adaptive 4/6 scaling.

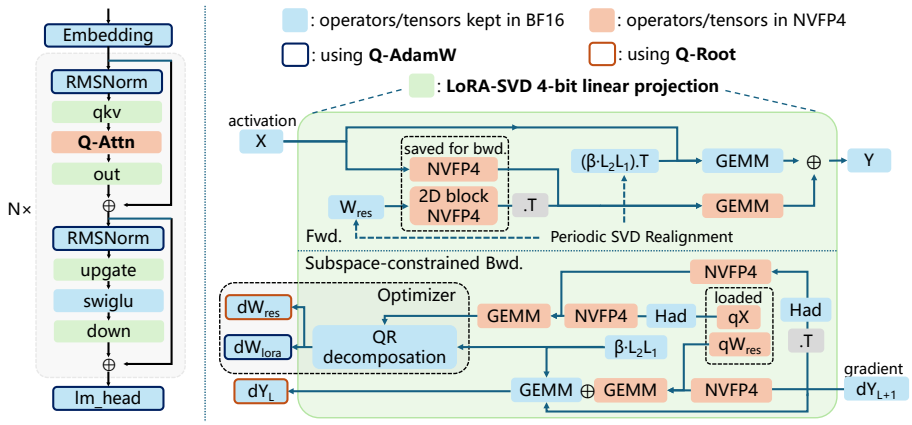


Figure 1: **Full-Stack NVFP4 overview.** Subspace-Protected LoRA-SVD protects a low-rank projection subspace; NVFP4-State AdamW compresses persistent momentum states; NVFP4 Root executes eligible matrix updates through NVFP4 Newton–Schulz iterations; and Selective FP4 Attention assigns NVFP4 and BF16 to different attention paths.

3.2 Subspace-Protected LoRA-SVD Projections

As gradients weaken relative to quantization noise (Chmiel et al., 2025), residual projection error starts to limit NVFP4 training. Subspace-Protected LoRA-SVD cuts that error while leaving the bulk of the GEMM in NVFP4.

LoRA-SVD parameterizes a training-time weight as $W = W_{\text{res}} + \beta L_2 L_1$, where $\beta = \beta_0(32/r)$, $\beta_0 = 0.5$, and r is the low-rank dimension. It preserves a compact principal subspace and its activation path in BF16 while computing the full-shape residual in NVFP4. It is a drop-in replacement for `nn.Linear`, leaves the Transformer topology unchanged, and can be merged into a standard dense weight after training. The fusion strategy follows the low-rank traffic-reuse principle demonstrated by SVDQuant (Li et al., 2025); implementation details are discussed in Appendix C.

Subspace-constrained backward pass. To maintain alignment with the principal subspace, we use a Cholesky-QR constraint. The forward and backward computations are

$$Y = \underbrace{Q_{1\text{x}16}(X)\, Q_{16\text{x}16}(W_{\text{res}})^\top}_{\text{NVFP4 GEMM}} + \beta \cdot \underbrace{(X L_1^\top) L_2^\top}_{\text{BF16}}$$

$$\mathrm{d}X = Q_{1\text{x}16}(\mathrm{d}Y)\, \underbrace{Q_{16\text{x}16}(W_{\text{res}})}_{\text{reused from fwd.}} + \beta \cdot (\mathrm{d}Y\, L_2)\, L_1 \tag{1}$$

$$\mathrm{d}W = Q_{1\text{x}16}(\mathrm{d}Y^\top H_{16}^r)\, Q_{16\text{x}1}\left(H_{16}^{r\top} Q_{1\text{x}16}(X)\right) \tag{2}$$

$$Q_{L_2} R_{L_2} = L_2, \quad Q_{L_1} R_{L_1} = L_1^\top \tag{3}$$

$$\Lambda := Q_{L_2}^\top \mathrm{d}W Q_{L_1}, \qquad \mathrm{d}W_{\text{LoRA}} = Q_{L_2} \Lambda Q_{L_1}^\top$$

$$\mathrm{d}W_{\text{res}} = \mathrm{d}W - \mathrm{d}W_{\text{LoRA}}$$

$$\mathrm{d}L_1 = \beta L_2^\top \mathrm{d}W_{\text{LoRA}}, \quad \mathrm{d}L_2 = \beta \mathrm{d}W_{\text{LoRA}} L_1^\top. \tag{4}$$

H_{16}^{r} is a fresh 16×16 random Hadamard transform. Components aligned with the maintained principal subspace update the BF16 low-rank branch, while the complementary residual updates W_{res}. Equations 3–4 are deferred to the optimizer to avoid repeated work during gradient accumulation. Quantized activations are reused when forming LoRA gradients to avoid an additional high-precision activation read and large GEMM; Appendix A gives the derivation.

Figure 2 shows lower projection-level error for outputs, input gradients, and weight gradients across 1B BF16 checkpoints.

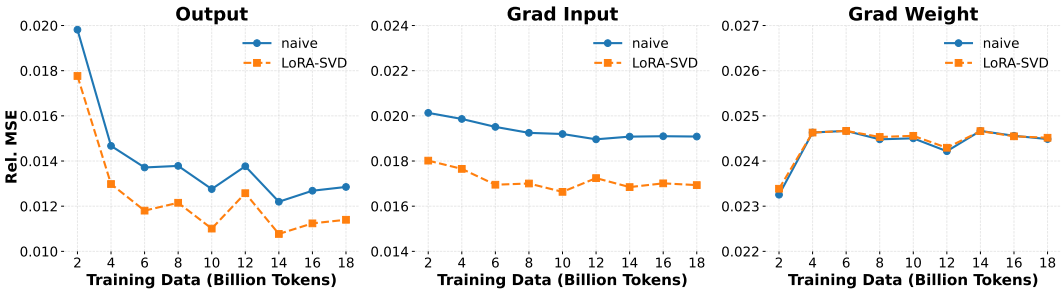


Figure 2: **Linear-projection error.** Each point averages relative MSE across sampled linear projections from the same 1B BF16 checkpoint. LoRA-SVD reduces error for outputs, input gradients, and weight gradients by approximately 10% compared with direct NVFP4, with no increasing trend over this training trajectory.

Factor-norm gradient decoupling. The bilinear factorization makes each factor's gradient scale with the norm of the other. Its unconstrained dynamics can therefore exhibit

$$\frac{\mathrm{d}\|L_1\|}{\mathrm{d}t} \propto \|L_2\|, \qquad \frac{\mathrm{d}\|L_2\|}{\mathrm{d}t} \propto \|L_1\|.$$

so factor and global gradient norms may grow while the effective dense update remains well behaved. We decouple the reconstructed gradients as

$$\widetilde{\mathrm{d}L_1} = \frac{c}{\max(\|L_2\|_F, c)}\,\mathrm{d}L_1, \qquad \widetilde{\mathrm{d}L_2} = \frac{c}{\max(\|L_1\|_F, c)}\,\mathrm{d}L_2, \tag{5}$$

with $c = 1$. The scaling is inactive below c and bounds cross-factor amplification above it, reducing exponential feedback to at most linear norm growth without changing the Cholesky-QR subspace assignment. Appendix B gives the optimizer ordering.

Periodic realignment. After a two-period warmup, randomized SVD (Halko et al., 2011) realigns W_{res}, L_2, L_1 every 2,048 successful updates. The first reset occurs at step 4,096 and restores a balanced minimum-norm factorization of the selected rank-r component. Appendix F reports its amortized cost.

3.3 NVFP4-State AdamW

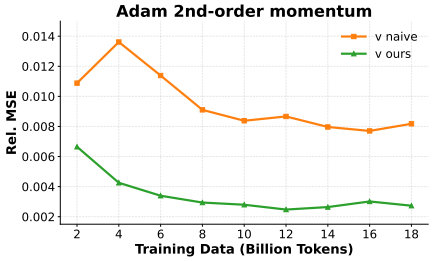
AdamW's second moment $v_t = \mathrm{EMA}(g_t^2)$ is non-negative and heavy-tailed. It persists for the entire run and later appears in a denominator, so direct NVFP4 storage distorts small values and destabilizes updates. NVFP4-State AdamW, abbreviated Q-AdamW in figures and tables, adapts established FP4 primitives—Hadamard transforms, stochastic rounding, and block/tensor scaling (Castro et al., 2025; Hu et al., 2025; NVIDIA, 2026)—to the structure of optimizer states. The first moment m_t uses Hadamard mixing, SR, and adaptive 4/6 quantization. The optimizer-state pipeline contains no SVD.

For v_t, an ordered square-root, tile-mean, and Hadamard pipeline precedes NVFP4 storage. Square root shortens the positive tail; a BF16 tile mean removes the common component; and the Hadamard transform spreads residual energy. Centering before the transform avoids constructive accumulation of the non-negative common mode. Reconstruction reverses these operations and squares the result.

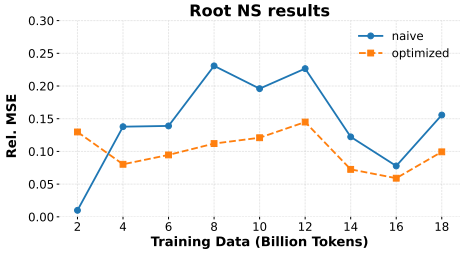
Component reconstruction analysis. At one 500M checkpoint, Hadamard reduces first-moment reconstruction error from 0.0074 to 0.0062. For the second moment, the complete pipeline reduces

error from 0.0116 to 0.0053; removing Hadamard, square root, or tile mean yields 0.0071, 0.0085, and 0.0130. In separate 1B native-state runs, removing any required stage caused rapid instability or divergence. To our knowledge, this is the first stable integrated 1B pretraining run with NVFP4 AdamW momentum-state storage.

The second-moment state occupies **5.0625 bits/element**. It is reconstructed rather than consumed by NVFP4 MMA, so BF16 scales improve persistent-state accuracy without constraining Tensor Core input format. Master weights remain BF16; no FP32 master copy is retained.



(a) NVFP4-State AdamW



(b) Quantized NS iteration

Figure 3: **Optimizer representation error.** (a) Across the 1B training trajectory, the complete transformed-state pipeline reduces second-moment reconstruction error by approximately 60% relative to direct quantization. Component ablations at one 500M checkpoint are reported in the text. (b) Shape-optimized coefficients reduce error in NVFP4 Newton–Schulz iterations.

3.4 NVFP4 Root

Root's Newton–Schulz (NS) update applies repeated matrix products and is sensitive to low-precision iteration error. GRASP/low-bit Muon (Wu et al., 2026) uses a mixed-precision route and a per-step SVD-based subspace operation. NVFP4 Root, abbreviated Q-Root in figures and tables, instead combines Root's shape-dependent coefficients with p99 outlier clipping and executes the NS matrix products directly in NVFP4. In our numerical comparison, the relative L_2 error drops from about 10% on a plain all-FP4 route to about 3%.

The first-order momentum follows the first-moment storage recipe in §3.3. Q-Root is applied only to eligible matrix parameters. Embeddings, `lm_head`, normalization and bias parameters, LoRA factors, and narrow parameters use Q-AdamW. Other hyperparameters follow Root (He et al., 2025). The NVFP4 NS iteration is

$$qX_k = Q_{1\text{x}16}(XH_{16}^r),\ A_k = qX_k\, qX_k{}^\top$$
$$qA_k = Q_{1\text{x}16}(A_kH_{16}^r),\ qA_k^{16\text{x}1} = Q_{16\text{x}1}(H_{16}{}^\top A_k),\ B_k = bA_k + cqA_kqA_k^{16\text{x}1}$$
$$qB_k = Q_{1\text{x}16}(B_kH_{16}^r),\ qX_k^{16\text{x}1} = Q_{16\text{x}1}(H_{16}^r{}^\top X_k),\ X_{k+1} = aX_k + qB_kqX_k^{16\text{x}1}.$$

All quantizations use RTN and adaptive 4/6 scaling. Figure 3b shows that the Root coefficients (a, b, c) reduce iteration error relative to standard Muon coefficients. Their aspect-ratio specialization limits error amplification across successive NS steps and avoids an auxiliary mixed-precision or SVD path.

3.5 Selective FP4 Attention

Existing FP4 attention systems are built mainly for inference and do not supply a consistent NVFP4 forward–backward operator for pretraining from scratch. In training, PV and $dOV^\top$ are sensitive to low precision, and reused $Q/K/V$ representations must stay consistent between forward and backward. Selective FP4 Attention, abbreviated Q-Attn in figures and tables, therefore assigns precision per matrix product.

Table 2 assigns NVFP4 to the $QK^\top$ and dS paths and retains the softmax-sensitive PV, $P^\top dO$, and $dOV^\top$ products in BF16. Quantized views are reused between forward and backward to avoid precision mismatch and repeated memory traffic. Figure 4 shows that quantizing PV or $dOV^\top$

degrades accuracy for limited additional savings. Appendix D analyzes these paths, and Appendix E gives the tiled algorithm.

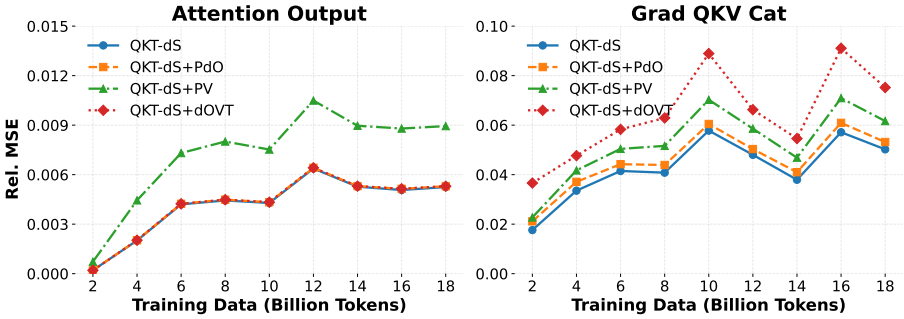


Figure 4: **Attention precision analysis.** Quantizing PV or $dOV^\top$ produces substantially larger error than the selected mixed-precision route, so we keep those products in BF16.

4 Full-Stack NVFP4 Pretraining

4.1 Experimental Setup

We evaluate module-level accuracy, whether the recipes compose, and native resource gains. Quality runs train a 3B SwiGLU/GQA Transformer for 64B tokens on a high-quality subset of Nemotron-CC v2 (NVIDIA et al., 2025). BF16 and Full-Stack NVFP4 match data, global batch size, schedule, steps, and token budget; every NVFP4 block uses all selected recipes. The runs use two and four eight-GPU A800 nodes, respectively, because fake quantization is slower but does not change the optimization settings. Appendix G gives full configurations.

Native speed and memory are measured separately on four target-shaped decoder blocks of one RTX 5090 (SM120). The 3B quality runs themselves use fake quantization on A800 nodes.

4.2 RQ1: Module-Level Accuracy and Composability

We evaluate module-level ablations on a 1B model trained for 20B tokens. NVFP4+4/6 (Cook et al., 2025) is the matched linear-only quality baseline. All quantized rows apply the selected recipes to every block rather than reserving BF16 blocks. "Full-Stack w/o Root" combines LoRA-SVD, Q-Attn, and Q-AdamW; in the Root rows, Q-Root handles eligible matrices and Q-AdamW handles the remaining parameters.

Table 2: **Precision assignment in Q-Attn.** Hats denote quantized views reused across passes.

Matmul	Pass	Precision
$\hat{Q}\hat{K}^\top$	Fwd	NVFP4
$P\hat{V}$	Fwd	BF16
$\hat{Q}\hat{K}^\top$	Bwd	NVFP4
$P^\top dO$	Bwd	BF16
$dO\,\hat{V}^\top$	Bwd	BF16
$d\hat{S}\,\hat{K}$	Bwd	NVFP4
$d\hat{S}^\top\hat{Q}$	Bwd	NVFP4

Table 3: **Ablations on 1B/20B-token pretraining.** All quantized recipes are applied to every block.

Configuration	Loss ↓	ΔBF16 ↓
AdamW only		
BF16	2.6415	—
BF16 + LoRA	**2.6351**	−0.24%
NVFP4 4/6	2.6785	+1.40%
4/6 + plain LoRA	2.6711	+1.12%
4/6 + LoRA-SVD	**2.6575**	**+0.61%**
4/6 + Q-AdamW	2.6674	+0.98%
4/6 + Q-Attn	2.6539	+0.47%
Full-Stack w/o Root	**2.6563**	**+0.56%**
Root + AdamW		
BF16 + Root	2.4819	—
Full-Stack	**2.4973**	**+0.62%**

Plain LoRA reduces the linear-only gap from 1.40% to 1.12%, whereas LoRA-SVD reaches 0.61%. The BF16+LoRA control improves loss by only 0.24%, indicating that added capacity explains a smaller share of the gain than SVD initialization and subspace-constrained routing.

LoRA-SVD increases parameters by 4.32% at 1B, from 1,212,663,808 to 1,265,092,608, and by 3.12% at 3B. Both model sizes use rank 128 only for `down_proj` and rank 64 for `gate_up_proj`, concatenated `qkv_proj`, and `out_proj`; rank does not scale with model size. The progressive rows in Table 3 also show that the recipes can be enabled independently and composed in one training run.

4.3 RQ2: Combined Pretraining Quality

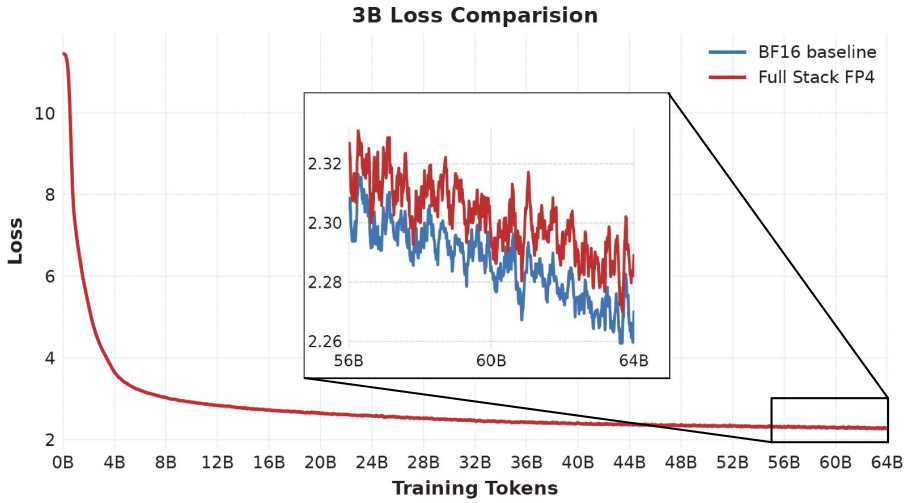


Figure 5: **Training loss on 3B/64B-token pretraining.** Under matched data and optimization budgets, BF16 and Full-Stack NVFP4 reach losses of 2.267 and 2.286, respectively, a 0.838% gap.

The matched loss trajectories in Figure 5 remain close over 64B tokens, ending at a **0.838%** relative gap.

Final checkpoints are evaluated zero-shot with `max_length=2048`. ARC-C and HellaSwag use normalized accuracy. Table 4 shows that Full-Stack NVFP4 has slightly lower average perplexity (26.665 versus 26.675) and 0.10 percentage points lower average accuracy. The individual accuracy differences range from 0.03 to 0.16 percentage points.

Table 4: **Zero-shot evaluation of the 3B checkpoints.** The vertical rule separates perplexity and accuracy metrics.

Model	Wiki PPL ↓	LAMBADA PPL ↓	Avg. PPL ↓	ARC-E ↑	ARC-C Norm. ↑	HellaSwag Norm. ↑	LAMBADA ↑	PIQA ↑	Avg. Acc. ↑
BF16	26.24	**27.11**	26.675	**.6272**	**.3020**	**.4695**	**.3929**	**.7016**	**.49864**
Full-Stack NVFP4	**26.09**	27.24	**26.665**	.6259	.3017	.4686	.3913	.7007	.49764

4.4 RQ3: Native-Hardware Resource Benefits

We measure four decoder blocks on one RTX 5090 (SM120) with $B = 2$, $L = 8192$ (16,384 tokens), excluding embeddings and the LM head. The $d = 4096/6144$ shapes use $d_{ff} = 11008/13824$, 32/48 query heads, and 8 KV heads, representing approximately 0.7B/1.4B dense block parameters. Each model row uses five warmups and 20 measurements in a fresh process; isolated optimizer rows use 20 warmups and 100 measurements. CUDA-event segments are synchronized. Accounting includes LoRA gradients in forward/backward, state transformations, packed AdamW updates, QR refresh, five Root NS iterations, and parameter updates; periodic fast-SVD is reported separately. SM120 lacks an efficient stochastic-rounding primitive on this path, so TE and ours both disable SR.

The BF16 baseline uses full-graph, max-autotune `torch.compile` with CODA-style epilogue fusion (Guo et al., 2026). TE 2.17 provides the optimized fused NVFP4 systems baseline (Abecassis

et al., 2025; NVIDIA, 2026); ours adds RHT, adaptive 4/6, and the complete LoRA-SVD path. Four Over Six remains the projection-quality baseline, but its public path lacks comparable fused-CUDA coverage.

Table 5: **Complete four-block local training step.** Each cell reports milliseconds / peak allocated GiB. Forward, backward, and optimizer are segments of the same step.

Optimizer/shape	BF16	TE	Ours
Root, $d = 4096$	760.16 / 12.35	556.25 / 12.75	**303.84 / 9.70**
Root, $d = 6144$	1750.27 / 18.55	1343.33 / 19.11	**618.87 / 14.27**
AdamW, $d = 4096$	406.26 / 16.31	**199.61** / 16.71	200.66 / **10.13**
AdamW, $d = 6144$	761.31 / 26.22	330.07 / 26.77	**326.36 / 15.08**

In the Root setting, BF16 and TE apply Muon to dense matrices, while ours applies Q-Root to base matrices and packed Q-AdamW to LoRA factors. Ours is 2.50×/2.83× faster than BF16 and 1.83×/2.17× faster than TE, with 21.5%/23.1% less peak memory than BF16. In the AdamW setting, ours is 0.5% slower/1.1% faster than TE and uses 37.9%/42.5% less memory than BF16. TE retains high-precision resident and optimizer states while adding NVFP4 views, scales, RHT workspace, and saved tensors; ours compresses the optimizer states and reuses workspaces despite including LoRA.

At the same single-GPU input shape, the Root speedups correspond to 60.0%/64.6% less time per token than BF16. Interpolating the 3B loss curves, FP4 needs about 7.8% more tokens to match BF16 loss, which would leave estimated matched-loss savings of 56.9%/61.9%. Component breakdowns and remaining kernel-fusion gaps are in Appendix F.

5 Related Work

FP8 and FP4 pretraining. Early low-bit training includes WAGE (Wu et al., 2018), FP8 training with E4M3/E5M2 (Micikevicius et al., 2022), and stable 8-bit vision–language training (Wortsman et al., 2023). FP8 systems later expanded coverage to activations and optimizer states (Fishman et al., 2024; Xi et al., 2024), and DeepSeek-V3 demonstrated large-scale FP8 pretraining (Liu et al., 2024). For 4-bit training, MXFP introduced the MXFP4 format (Rouhani et al., 2023), while NVIDIA's NVFP4 recipe combined fine-grained scaling, stochastic rounding, and Hadamard transforms (Abecassis et al., 2025). Transformer Engine provides NVIDIA's optimized implementation of this projection path (NVIDIA, 2026).

Subsequent work improves projection quality or training behavior. FP4-All-the-Way (Chmiel et al., 2025) studies fully quantized projection training and the decreasing gradient-signal-to-quantization-noise ratio over training. Quartet (Castro et al., 2025) and the FP4 design-space study (Hu et al., 2025) analyze scaling, Hadamard transforms, stochastic rounding, and implementation trade-offs. Four Over Six (Cook et al., 2025) introduces adaptive $M = 4/M = 6$ block scaling and is our matched projection-quality baseline. TetraJet and TetraJet-v2 (Chen et al., 2025a;b), CHON (Dong et al., 2026), and Metis (Cao et al., 2025) address weight oscillation, outliers, or anisotropic distributions. These methods primarily improve projection-side quantization. Our work instead extends NVFP4 recipes to persistent optimizer states, optimizer computation, and a trainable attention path while retaining Four Over Six for projection quality.

Low-bit optimizers. Block-wise INT8 Adam (Dettmers et al., 2021), FP8 optimizer states in COAT (Xi et al., 2024), and 4-bit Adam with rank-1 normalization (Li et al., 2023) reduce optimizer-state memory. Quantized Muon has been studied with INT8 momentum (Gupta et al., 2025) and with GRASP's grid-quantized mixed-precision stabilization and SVD-based subspace operation (Wu et al., 2026). Our NVFP4-State AdamW contribution is an NVFP4 representation for both persistent momentum states that remains stable in integrated pretraining. NVFP4 Root is a separate direct NVFP4 Newton–Schulz route based on Root's shape-dependent coefficients and clipping.

Low-bit attention. The SageAttention series (Zhang et al., 2024b;a; 2025) develops INT8 and FP4 attention kernels mainly for inference; SageAttention3's training path uses INT8 rather than NVFP4.

Attn-QAT (Zhang et al., 2026) trains models for quantized inference and identifies forward–backward recomputation consistency, but it does not provide an NVFP4 attention operator for pretraining from scratch. These methods build on the IO-aware tiling of FlashAttention (Dao et al., 2022; Dao, 2023); FlashAttention-4 (Zadouri et al., 2026) further introduces Blackwell-oriented pipeline optimizations. Our Selective FP4 Attention instead studies a trainable NVFP4/BF16 precision split and evaluates its forward, backward, memory, and convergence behavior.

Systems baselines. CODA rewrites Transformer blocks as fused GEMM–epilogue programs (Guo et al., 2026); we use the same systems principle with full-graph, max-autotune `torch.compile` for the BF16 baseline. NVIDIA TE 2.17 is the primary systems baseline because it provides NVIDIA's fused NVFP4 implementation. Four Over Six is the stronger projection-quality baseline, but its current public end-to-end training path has less complete CUDA fusion, especially in backward. We therefore use TE for systems measurements and Four Over Six for projection-quality comparisons.

6 Conclusion and Limitations

Full-Stack NVFP4 extends NVFP4 from projections to optimizer states, optimizer computation, and attention by assigning precision according to numerical role. On 3B/64B-token pretraining the combined recipe stays within **0.838%** of BF16 loss and matches aggregate zero-shot metrics closely. Four-block native measurements show 2.50–2.83$\times$ Root speedups over optimized BF16 and 37.9–42.5% lower AdamW peak memory.

Outlook. LoRA-SVD is aimed at the late-training regime where gradients sit closer to quantization noise (Chmiel et al., 2025). A natural next step is to test how far it can postpone a precision upshift, and more generally to schedule precision by module and training stage, restoring BF16 on sensitive paths only when needed.

Limitations. The main limitation is scale. Quality experiments reach 3B parameters and 64B tokens, short of 7B+ models and of substantially longer token budgets.

Efficiency numbers come from four decoder blocks on one RTX 5090. Combining those speedups with the 3B loss curves gives a matched-loss time estimate at fixed batch size. A full distributed 3B step, and any extra speedup from spending the saved memory on larger batches, is left for follow-up. Kernel fusion is also unfinished: LoRA backward and attention still launch extra kernels, and the attention path omits several FlashAttention-4 optimizations. The recipes are built for NVFP4.

AI USE STATEMENT

In this work, the authors used generative AI tools to assist with Chinese–English translation; language editing of author-written text; presentation-level restructuring; refinement of the title, terminology, and section headings; and LaTeX formatting and layout. The scientific ideas, method design, experiments, data, analysis, and conclusions were developed and validated by the authors. All AI-assisted changes were reviewed and revised by the authors, who verified the technical statements and citations. The authors take full responsibility for the final content of this work, including text or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections 2–3 specify the NVFP4 format, role-specific precision assignments, and optimization procedures. Appendices A–E provide the LoRA-SVD derivations, stabilized update procedure, kernel-fusion design, attention sensitivity analysis, and tiled attention algorithm. Section 4 reports the evaluation protocols and measurement scope, while Appendices F and G give component-level measurements, hardware qualifications, model architecture, and matched training hyperparameters.

REFERENCES

Felix Abecassis, Anjulie Agrusa, Dong Ahn, Jonah Alben, Stefania Alborghetti, Michael Andersch, Sivakumar Arayandi, Alexis Bjorlin, Aaron Blakeman, Evan Briones, et al. Pretraining large language models with nvfp4. *arXiv preprint arXiv:2509.25149*, 2025.

Hengjie Cao, Mengyi Chen, Yifeng Yang, Ruijun Huang, Fang Dong, Jixian Zhou, Anrui Chen, Mingzhi Dong, Yujiang Wang, Jinlong Hou, et al. Metis: Training llms with fp4 quantization. *arXiv preprint arXiv:2509.00404*, 2025.

Roberto L Castro, Andrei Panferov, Soroush Tabesh, Oliver Sieberling, Jiale Chen, Mahdi Nikdan, Saleh Ashkboos, and Dan Alistarh. Quartet: Native fp4 training can be optimal for large language models. In *Advances in Neural Information Processing Systems*, 2025.

Yuxiang Chen, Haocheng Xi, Jun Zhu, and Jianfei Chen. Oscillation-reduced mxfp4 training for vision transformers. *arXiv preprint arXiv:2502.20853*, 2025a.

Yuxiang Chen, Xiaoming Xu, Pengle Zhang, Michael Beyer, Martin Rapp, Jun Zhu, and Jianfei Chen. Tetrajet-v2: Accurate nvfp4 training for large language models with oscillation suppression and outlier control. *arXiv preprint arXiv:2510.27527*, 2025b.

Brian Chmiel, Maxim Fishman, Ron Banner, and Daniel Soudry. Fp4 all the way: Fully quantized training of llms. *arXiv preprint arXiv:2505.19115*, 2025.

Jack Cook, Junxian Guo, Guangxuan Xiao, Yujun Lin, and Song Han. Four over six: More accurate nvfp4 quantization with adaptive block scaling. *arXiv preprint arXiv:2512.02010*, 2025.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021.

Peijie Dong, Ruibo Fan, Yuechen Tao, Di Mou, Wenhu Hu, Zhenheng Tang, Yinghao Yu, Jiamang Wang, Wenbo Su, Guodong Yang, et al. Dissecting outlier dynamics in llm nvfp4 pretraining. *arXiv preprint arXiv:2602.02047*, 2026.

Maxim Fishman, Brian Chmiel, Ron Banner, and Daniel Soudry. Scaling fp8 training to trillion-token llms. *arXiv preprint arXiv:2409.12517*, 2024.

Han Guo, Jack Zhang, Arjun Menon, Driss Guessous, Vijay Thakkar, Yoon Kim, and Tri Dao. Coda: Rewriting transformer blocks as gemm-epilogue programs. *arXiv preprint arXiv:2605.19269*, 2026.

Aman Gupta, Rafael Celente, Abhishek Shivanna, DT Braithwaite, Gregory Dexter, Shao Tang, Hiroto Udagawa, Daniel Silva, Rohan Ramanath, and S Sathiya Keerthi. Effective quantization of muon optimizer states. *arXiv preprint arXiv:2509.23106*, 2025.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2): 217–288, 2011.

Wei He, Kai Han, Hang Zhou, Hanting Chen, Zhicheng Liu, Xinghao Chen, and Yunhe Wang. Root: Robust orthogonalized optimizer for neural network training. *arXiv preprint arXiv:2511.20626*, 2025.

Robert Hu, Carlo Luschi, and Paul Balanca. Elucidating the design space of fp4 training. *arXiv preprint arXiv:2509.17791*, 2025.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. *URL https://kellerjordan. github. io/posts/muon*, 6(3):4, 2024.

Bingrui Li, Jianfei Chen, and Jun Zhu. Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36:15136–15171, 2023.

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022. URL https://arxiv.org/abs/2209.05433.

NVIDIA. Transformer engine: Nvfp4 training recipe. https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/features/low_precision_training/nvfp4/nvfp4.html, 2026.

NVIDIA, :, Aarti Basant, Abhijit Khairnar, Abhijit Paithankar, Abhinav Khattar, Adithya Renduchintala, Aditya Malte, Akhiad Bercovich, Akshay Hazare, Alejandra Rico, Aleksander Ficek, Alex Kondratenko, Alex Shaposhnikov, Alexander Bukharin, Ali Taghibakhshi, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amy Shen, Andrew Tao, Ann Guan, Anna Shors, Anubhav Mandarwal, Arham Mehta, Arun Venkatesan, Ashton Sharabiani, Ashwath Aithal, Ashwin Poojary, Ayush Dattagupta, Balaram Buddharaju, Banghua Zhu, Barnaby Simkin, Bilal Kartal, Bita Darvish Rouhani, Bobby Chen, Boris Ginsburg, Brandon Norick, Brian Yu, Bryan Catanzaro, Charles Wang, Charlie Truong, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christian Munley, Christopher Parisien, Dan Su, Daniel Afrimi, Daniel Korzekwa, Daniel Rohrer, Daria Gitman, David Mosallanezhad, Deepak Narayanan, Dima Rekesh, Dina Yared, Dmytro Pykhtar, Dong Ahn, Duncan Riach, Eileen Long, Elliott Ning, Eric Chung, Erick Galinkin, Evelina Bakhturina, Gargi Prasad, Gerald Shen, Haifeng Qian, Haim Elisha, Harsh Sharma, Hayley Ross, Helen Ngo, Herman Sahota, Hexin Wang, Hoo Chang Shin, Hua Huang, Iain Cunningham, Igor Gitman, Ivan Moshkov, Jaehun Jung, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jian Zhang, Jiaqi Zeng, Jimmy Zhang, Jinze Xue, Jocelyn Huang, Joey Conway, John Kamalu, Jonathan Cohen, Joseph Jennings, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kari Briski, Katherine Cheung, Katherine Luna, Keith Wyss, Keshav Santhanam, Kezhi Kong, Krzysztof Pawelec, Kumar Anik, Kunlun Li, Kushan Ahmadian, Lawrence McAfee, Laya Sleiman, Leon Derczynski, Luis Vega, Maer Rodrigues de Melo,

Makesh Narsimhan Sreedhar, Marcin Chochowski, Mark Cai, Markus Kliegl, Marta Stepniewska-Dziubinska, Matvei Novikov, Mehrzad Samadi, Meredith Price, Meriem Boubdir, Michael Boone, Michael Evans, Michal Bien, Michal Zawalski, Miguel Martinez, Mike Chrzanowski, Mohammad Shoeybi, Mostofa Patwary, Namit Dhameja, Nave Assaf, Negar Habibi, Nidhi Bhatia, Nikki Pope, Nima Tajbakhsh, Nirmal Kumar Juluru, Oleg Rybakov, Oleksii Hrinchuk, Oleksii Kuchaiev, Oluwatobi Olabiyi, Pablo Ribalta, Padmavathy Subramanian, Parth Chadha, Pavlo Molchanov, Peter Dykas, Peter Jin, Piotr Bialecki, Piotr Januszewski, Pradeep Thalasta, Prashant Gaikwad, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Rabeeh Karimi Mahabadi, Rajen Patel, Ran El-Yaniv, Ranjit Rajan, Ria Cheruvu, Rima Shahbazyan, Ritika Borkar, Ritu Gala, Roger Waleffe, Ruoxi Zhang, Russell J. Hewett, Ryan Prenger, Sahil Jain, Samuel Kriman, Sanjeev Satheesh, Saori Kaji, Sarah Yurick, Saurav Muralidharan, Sean Narenthiran, Seonmyeong Bak, Sepehr Sameni, Seungju Han, Shanmugam Ramasamy, Shaona Ghosh, Sharath Turuvekere Sreenivas, Shelby Thomas, Shizhe Diao, Shreya Gopal, Shrimai Prabhumoye, Shubham Toshniwal, Shuoyang Ding, Siddharth Singh, Siddhartha Jain, Somshubra Majumdar, Soumye Singhal, Stefania Alborghetti, Syeda Nahida Akter, Terry Kong, Tim Moon, Tomasz Hliwiak, Tomer Asida, Tony Wang, Tugrul Konuk, Twinkle Vashishth, Tyler Poon, Udi Karpas, Vahid Noroozi, Venkat Srinivasan, Vijay Korthikanti, Vikram Fugro, Vineeth Kalluru, Vitaly Kurin, Vitaly Lavrukhin, Wasi Uddin Ahmad, Wei Du, Wonmin Byeon, Ximing Lu, Xin Dong, Yashaswi Karnati, Yejin Choi, Yian Zhang, Ying Lin, Yonggan Fu, Yoshi Suhara, Zhen Dong, Zhiyu Li, Zhongbo Zhu, and Zijia Chen. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025. URL https://arxiv.org/abs/2508.14444.

Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023.

Albert Tseng, Tao Yu, and Youngsuk Park. Training llms with mxfp4. *arXiv preprint arXiv:2502.20586*, 2025.

Mitchell Wortsman, Tim Dettmers, Luke Zettlemoyer, Ari Morcos, Ali Farhadi, and Ludwig Schmidt. Stable and low-precision training for large-scale vision-language models. *Advances in Neural Information Processing Systems*, 36:10271–10298, 2023.

Huaijin Wu, Bingrui Li, Yebin Yang, Yi Tu, Zhanpeng Zhou, Jianfei Chen, and Junchi Yan. Achieving low-bit muon through subspace preservation and grid quantization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=g2l9bg9DWx.

Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. In *International Conference on Learning Representations*, 2018.

Haocheng Xi, Han Cai, Ligeng Zhu, Yao Lu, Kurt Keutzer, Jianfei Chen, and Song Han. Coat: Compressing optimizer states and activation for memory-efficient fp8 training. *arXiv preprint arXiv:2410.19313*, 2024.

Ted Zadouri, Markus Hoehnerbach, Jay Shah, Timmy Liu, Vijay Thakkar, and Tri Dao. Flashattention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. *arXiv preprint arXiv:2603.05451*, 2026.

Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia Wei, Jun Zhu, and Jianfei Chen. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization. *arXiv preprint arXiv:2411.10958*, 2024a.

Jintao Zhang, Jia Wei, Haofeng Huang, Pengle Zhang, Jun Zhu, and Jianfei Chen. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. *arXiv preprint arXiv:2410.02367*, 2024b.

Jintao Zhang, Jia Wei, Pengle Zhang, Xiaoming Xu, Haofeng Huang, Haoxu Wang, Kai Jiang, Jianfei Chen, and Jun Zhu. Sageattention3: Microscaling fp4 attention for inference and an exploration of 8-bit training. *arXiv preprint arXiv:2505.11594*, 2025.

Peiyuan Zhang, Matthew Noto, Wenxuan Tan, Chengquan Jiang, Will Lin, Wei Zhou, and Hao Zhang. Attn-qat: 4-bit attention with quantization-aware training. *arXiv preprint arXiv:2603.00040*, 2026.

A Full Derivation of LoRA-SVD

The aligned forward and naive backward computations are

$$Y = \underbrace{Q_{1\text{x}16}(X)\, Q_{16\text{x}16}(W_{\text{res}})^\top}_{\text{NVFP4 GEMM}} + \beta \underbrace{(XL_1^\top)L_2^\top}_{\text{BF16}}$$
$$\mathrm{d}X = Q_{1\text{x}16}(\mathrm{d}Y)\, Q_{16\text{x}16}(W_{\text{res}}) + \beta(\mathrm{d}Y L_2)L_1$$
$$\mathrm{d}W = Q_{1\text{x}16}(\mathrm{d}Y^\top H_{16}^r)\, Q_{16\text{x}1}\left(H_{16}^{r\,\top} Q_{1\text{x}16}(X)\right)$$
$$\mathrm{d}W_{\text{res}} = \mathrm{d}W$$
$$\mathrm{d}L_1 = \beta L_2^\top(\mathrm{d}Y^\top X), \qquad \mathrm{d}L_2 = \beta(\mathrm{d}Y^\top X)L_1^\top.$$

The forward pass uses RTN and the backward pass uses SR; both use adaptive 4/6 scaling. H_{16}^r is a fresh 16×16 random Hadamard transform. Computing $\mathrm{d}L_1$ and $\mathrm{d}L_2$ naively repeats the large GEMM used for $\mathrm{d}W_{\text{res}}$. We instead reuse the already formed weight gradient:

$$\mathrm{d}L_1 = \beta L_2^\top \mathrm{d}W, \qquad \mathrm{d}L_2 = \beta \mathrm{d}W L_1^\top.$$

This approximation avoids another large GEMM over full-precision activations. The following subspace constraint separates the components assigned to the two branches.

Cholesky-QR gives

$$Q_{L_2}R_{L_2} = L_2, \qquad Q_{L_1}R_{L_1} = L_1^\top,$$

where $R = \mathrm{chol}(A^\top A)^\top \in \mathbb{R}^{r\times r}$ and $Q = AR^{-1} \in \mathbb{R}^{d\times r}$. For $r \ll d$, this operates on small $r \times r$ matrices. We then compute

$$\Lambda = Q_{L_2}^\top \mathrm{d}W Q_{L_1} \in \mathbb{R}^{r\times r}$$
$$\mathrm{d}W_{\text{LoRA}} = Q_{L_2}\Lambda Q_{L_1}^\top$$
$$\mathrm{d}W_{\text{res}} = \mathrm{d}W - \mathrm{d}W_{\text{LoRA}}$$
$$\mathrm{d}L_1 = \beta L_2^\top \mathrm{d}W_{\text{LoRA}}, \qquad \mathrm{d}L_2 = \beta \mathrm{d}W_{\text{LoRA}} L_1^\top.$$

B LoRA-SVD Factor-Norm Stabilization

The Cholesky-QR construction determines which component of $\mathrm{d}W$ is assigned to the low-rank branch, but it does not by itself control the scale ambiguity of the bilinear factors: $L_2L_1 = (aL_2)(L_1/a)$ for any nonzero scalar a. Moreover, the reconstructed gradients contain the opposite factor multiplicatively. Consequently, growth in $\|L_2\|_F$ enlarges $\mathrm{d}L_1$, and growth in $\|L_1\|_F$ enlarges $\mathrm{d}L_2$. Adam-style normalization can keep the effective update direction usable while the factor and global gradient norms continue to inflate.

We apply Equation 5 after the Cholesky-QR projection. Let G_1 and G_2 denote the reconstructed factor gradients before stabilization. The optimizer receives

$$\widetilde{G}_1 = s_1 G_1, \quad s_1 = \frac{c}{\max(\|L_2\|_F, c)}, \qquad \widetilde{G}_2 = s_2 G_2, \quad s_2 = \frac{c}{\max(\|L_1\|_F, c)}.$$

All norms and the Cholesky-QR operation are evaluated in FP32. The threshold $c = 1$ leaves small factors unchanged and caps the multiplicative cross-factor gain once either norm exceeds the threshold. The LoRA factors occupy native parameter groups with a $2\times$ learning-rate multiplier.

The custom backward pass accumulates only the raw main-weight gradient across microbatches. Cholesky-QR, subspace separation, factor-gradient reconstruction, and Equation 5 are then evaluated once per optimizer step. With distributed Adam, this transformation is applied before data-parallel reduce-scatter so that the resulting parameter gradients follow the optimizer's sharding path. With replicated Adam, the raw main-weight gradient is first all-reduced and the transformation is applied afterward.

Periodic randomized-SVD alignment is executed only after a successful Adam update. For distributed Adam, the BF16 model tensors are all-gathered to form the complete updated parameter, the rank-r alignment is computed on that parameter, and only the locally owned slices are copied back to the FP32 master shards. A base interval of 2,048 successful updates and warmup factor two place the first alignment at step 4,096; later alignments occur every 2,048 successful updates.

Algorithm 1 LoRA-SVD stabilized optimizer step

Accumulate the raw main-weight gradient G_W across microbatches
if replicated Adam **then**
    All-reduce G_W across the data-parallel group
end if
Compute FP32 Cholesky-QR factors Q_{L_2}, Q_{L_1}
$\Lambda \leftarrow Q_{L_2}^\top G_W Q_{L_1}$; $G_{\mathrm{LoRA}} \leftarrow Q_{L_2} \Lambda Q_{L_1}^\top$
$G_{\mathrm{res}} \leftarrow G_W - G_{\mathrm{LoRA}}$
$G_1 \leftarrow \beta L_2^\top G_{\mathrm{LoRA}}$; $G_2 \leftarrow \beta G_{\mathrm{LoRA}} L_1^\top$
$G_1 \leftarrow cG_1 / \max(\|L_2\|_F, c)$; $G_2 \leftarrow cG_2 / \max(\|L_1\|_F, c)$
if distributed Adam **then**
    Reduce-scatter $G_{\mathrm{res}}, G_1, G_2$ to their optimizer shards
end if
Apply the optimizer update, using the LoRA learning-rate multiplier
if the update succeeds and the periodic alignment is due **then**
    Form the complete updated parameter and compute its randomized rank-r SVD
    Reset $W_{\mathrm{res}}, L_2, L_1$; copy owned slices back to master shards
end if

C Kernel Fusion of LoRA-SVD

The Y and $\mathrm{d}X$ paths read high-precision X or $\mathrm{d}Y$ and write large outputs. Their low-rank operations can therefore share input and output traffic with adjacent quantization and NVFP4 GEMM kernels. The activation quantizer can be fused with normalization or SwiGLU, while output projections still require a dedicated quantization path. Similar reuse applies to $\mathrm{d}Y$.

SVDQuant (Li et al., 2025), originally developed for post-training quantization and inference, shows that a separate low-rank branch can add latency through repeated movement of 16-bit activations. Its Nunchaku engine fuses low-rank computation with quantization and 4-bit kernels to remove intermediate memory traffic and redundant launches. LoRA-SVD has the same opportunity in Y and $\mathrm{d}X$, but the current training backward is not fully fused. In Appendix F, the LoRA-SVD MLP is 11.2–17.2% slower than TE while remaining 2.87–3.30× faster than BF16. An operator model projects below-5% overhead after full fusion.

D Quantization Sensitivity in Attention

This section explains the empirical sensitivity of PV and $dOV^\top$ in Figure 4 through error terms induced by block quantization. Their magnitude depends on the attention distribution.

D.1 Sensitivity of PV

The attention matrix $P \in \mathbb{R}^{N \times N}$ satisfies $\sum_j P_{i,j} = 1$ and $P_{i,j} \geq 0$. In many heads, a small number of entries carry much of the probability mass while other entries are small. Under block quantization, a local maximum sets the scale, so smaller probabilities in the same block may be represented coarsely or rounded to zero.

Without explicit renormalization, this can change the row sum and perturb the convex combination. Writing $\Delta P = \hat{P} - P$, the output error is

$$\Delta O_i = \Delta P_i V, \qquad \|\Delta O_i\| \leq \|\Delta P_i\|_1 \max_j \|V_j\|.$$

The skewed distribution therefore makes PV sensitive in the evaluated setting, consistent with Figure 4.

D.2 Sensitivity of $dOV^\top$

In a FlashAttention-style backward pass,

$$dS_{i,j} = P_{i,j}(dP_{i,j} - D_i), \qquad dP_{i,j} = \langle dO_i, V_j \rangle.$$

where $D_i = \sum_k P_{i,k} dP_{i,k} = \langle O_i, dO_i \rangle$. If quantizing $dOV^\top$ introduces error $\Delta dP_{i,j}$ while D_i remains in higher precision, then

$$\widehat{dS}_{i,j} = dS_{i,j} + P_{i,j} \Delta dP_{i,j}. \tag{6}$$

Block scaling is set by the largest magnitude in each group, so smaller dP entries can receive larger relative error. The ordering of $|dP|$ need not match that of P. For an index c with appreciable probability but a sub-maximum $dP_{i,c}$,

$$\widehat{dS}_{i,c} = dS_{i,c} + P_{i,c} \Delta dP_{i,c}.$$

The perturbation can be large relative to $dS_{i,c}$ when $dP_{i,c} - D_i$ is small. Conversely, the factor P attenuates this term in the dense tail. Retaining $dOV^\top$ in BF16 avoids this error source in our evaluated precision split.

E Selective FP4 Attention Algorithm

Algorithm 2 summarizes the tiled forward and backward pass. Q, K, and V are stored as quantized views; sensitive products reconstruct their inputs and execute in BF16.

Algorithm 2 Selective FP4 Attention (one head)

Require $Q, K, V \in \mathbb{R}^{N \times d}$ (BF16), tile sizes B_q, B_k
Require NVFP4 quantizer $\phi(\cdot)$
// Forward pass
$\hat{Q} \leftarrow \phi(Q)$; $\hat{K} \leftarrow \phi(K)$; $\hat{V} \leftarrow \phi(V)$ {NVFP4 views}
Save $\hat{Q}, \hat{K}, \hat{V}$ to HBM {compact NVFP4 views with scales}
for $i = 1$ to T_q **do**
  $m_i \leftarrow -\infty$; $l_i \leftarrow 0$; $O_i \leftarrow 0$
  for $j = 1$ to T_k **do**
    $S_{ij} \leftarrow \hat{Q}_i \hat{K}_j^\top / \sqrt{d}$ {NVFP4 GEMM, FP32 accumulation}
    Apply causal mask
    $\tilde{P}_{ij} \leftarrow \mathrm{softmax}_{\mathrm{online}}(S_{ij})$
    $O_i \leftarrow \mathrm{diag}(\alpha) O_i + \tilde{P}_{ij} \hat{V}_j$ {dequantized $\hat{V}$, BF16 GEMM}
  end for
  $L_i \leftarrow m_i + \log l_i$
end for
// Backward pass (given dO in BF16)
$D \leftarrow \mathrm{rowsum}(dO \odot O)$
for $j = 1$ to T_k **do**
  for $i = 1$ to T_q **do**
    Recompute $S_{ij} \leftarrow \hat{Q}_i \hat{K}_j^\top / \sqrt{d}$
    Recompute P_{ij} from S_{ij} and L_i
    $dV_j \mathrel{+}= P_{ij}^\top dO_i$ {BF16}
    $dS_{ij} \leftarrow P_{ij} \odot (dO_i \hat{V}_j^\top - D_i)$ {dequantized $\hat{V}$, BF16 GEMM}
    $\widehat{dS}_{ij} \leftarrow \phi_{\mathrm{SR}}(dS_{ij})$
    $dQ_i \mathrel{+}= \widehat{dS}_{ij} \hat{K}_j / \sqrt{d}$ {NVFP4 GEMM}
    $dK_j \mathrel{+}= \widehat{dS}_{ij}^\top \hat{Q}_i / \sqrt{d}$ {NVFP4 GEMM}
  end for
end for
Return O, dQ, dK, dV

Distinction from Attn-QAT. Attn-QAT (Zhang et al., 2026) uses QAT to obtain quantized attention for inference rather than an NVFP4 forward–backward pretraining operator. Our design targets pretraining: it stores quantized Q, K, and V views, uses NVFP4 for $QK^\top$ and the dS products, and executes PV and $dOV^\top$ in BF16.

F NATIVE-HARDWARE EFFICIENCY DETAILS

Table 6 decomposes the complete-step measurements in Table 5. Component memory is incremental peak allocation rather than complete training memory. Both NVFP4 MLP implementations create quantized views, scales, RHT workspace, and saved tensors absent in BF16. Ours uses compact quantized intermediates and releases temporary buffers earlier than TE.

Table 6: **Native component measurements on one RTX 5090.** Each cell reports milliseconds / incremental peak allocated GiB.

Region/shape	BF16	Comparator	Ours
MLP, $d = 4096$	61.79 / 2.14	TE: **18.38** / 3.35	21.54 / **2.63**
MLP, $d = 6144$	115.92 / 2.83	TE: **31.60** / 4.44	35.13 / **3.24**
Attention, $d = 4096$	18.87 / 1.379	—	**14.90 / 0.675**
Attention, $d = 6144$	28.52 / 2.006	—	**22.49 / 0.945**
Root, $d = 4096$	344.55 / 4.42	GRASP: 474.16 / 6.21	**109.80 / 3.25**
Root, $d = 6144$	981.64 / 8.54	GRASP: 1239.20 / 11.85	**304.28 / 6.35**
AdamW, $d = 4096$	10.21 / 7.92	—	**7.28 / 3.48**
AdamW, $d = 6144$	19.73 / 15.33	—	**14.04 / 6.73**

LoRA-SVD MLP. The MLP rows exclude attention and optimizer computation but include LoRA gradients. Ours is 17.2%/11.2% slower than TE and 2.87×/3.30× faster than BF16. Adaptive 4/6 adds approximately 0.01%; the remaining TE gap is primarily the not-yet-fully-fused LoRA backward path. SVDQuant (Li et al., 2025) demonstrates that fusing low-rank computation with quantization and low-bit kernels removes redundant traffic. Our operator model projects below-5% overhead after full fusion.

Attention. The attention rows compare BF16 SDPA with the NVFP4 Q/K body; QKV and output projections are counted in the complete-step table. Ours is 1.27× faster and uses 51.1%/52.9% less incremental allocation. The attention training kernels are not fully fused, and the path omits several FlashAttention-4 pipeline optimizations (Zadouri et al., 2026).

Root. Root rows match shapes, p99 clipping, aspect-ratio coefficients, and five Newton–Schulz iterations. GRASP uses its published method and official implementation (Wu et al., 2026). Ours is 3.14×/3.23× faster than BF16 and 4.32×/4.07× faster than GRASP, with 26.4%/25.6% less allocation than BF16.

AdamW. The isolated rows use precomputed synchronized gradients and exclude LoRA, QR, and projection computation. Both implementations fuse decay and parameter update. BF16 stores a BF16 weight and gradient and two FP32 moments, totaling 96 bits/parameter. Ours uses a 5-bit first moment and a 5.0625-bit second moment with the same BF16 weight and gradient, totaling 42.0625 bits/parameter, with no FP32 master weight. AdamW updates are memory-bandwidth bound, so reduced moment traffic outweighs packing and reconstruction: ours is 1.40×/1.41× faster with 56.0%/56.1% less allocation.

Periodic fast-SVD. After a two-period warmup, the first realignment occurs at successful optimizer step 4,096 and subsequent realignments use a 2,048-step period. One realignment costs 160.00/226.96 ms at $d = 4096/6144$. In the steady state, this amortizes to 0.078/0.111 ms per step, or 0.026%/0.018% of the complete ours step. Added peak allocation is 0.031/0.039 GiB (0.3% in both settings).

Measurement scope. All numbers in this appendix use the same four-block, single-GPU RTX 5090 microbenchmark as §4.4. The same-token and matched-loss estimates in the main text hold that batch size fixed.

G Model Architecture and Training Configuration

Tables 7 and 8 summarize the 3B architecture and matched training settings.

Model architecture	Value
Number of layers	24
Hidden dimension	3072
FFN intermediate dimension	9216
Attention heads	24
Attention KV heads	12
Context window	2048 tokens

Table 7: 3B model architecture.

Training configuration	Value
Optimizer	Root+AdamW
Peak learning rate	1.5×10^{-4}
LoRA peak learning rate	3.0×10^{-4}
LoRA base scaling β_0	0.5
LoRA effective scaling β	$\beta_0(32/r)$
LoRA grad-decoupling threshold c	1.0
SVD interval / warmup factor	2,048 / 2
Weight decay	0.1
β_1/β_2	0.9 / 0.95
Gradient clipping	1.0
Warmup tokens	3B tokens (5%)
Learning-rate decay	Cosine
Global batch size	1M tokens
Root quantile ratio	0.99
Training tokens	64B

Table 8: Matched training hyperparameters.

Table 9: **LoRA-SVD ranks and 3B parameter increases.** The same rank policy is used at 1B and 3B.

Projection	Rank	3B parameter increase
`down_proj`	128	5.56%
`gate_up_proj`	64	2.43%
Concatenated `qkv_proj`	64	3.13%
`out_proj`	64	4.17%
Overall	—	3.12%