BiLoCo: Binary Low-Rank Corrections for Post-Training Quantization

David Jin* **Beshr IslamBouli*** **Tarushii Goel** **Han Guo** **Yoon Kim**

Massachusetts Institute of Technology

{jindavid,beshr,tarushii,hanguo,yoonkim}@mit.edu

Abstract

Post-training quantization (PTQ) to FP4 has emerged as a key technique for reducing large language model inference cost, especially on Blackwell GPUs, which offer native FP4 tensor core support. Direct PTQ to FP4, however, still leaves a noticeable accuracy gap. Recent work on SVDQuant [10] mitigates the errors introduced through FP4 quantization for diffusion models by adding high-precision low-rank corrections obtained via SVD. However, this high-precision low-rank correction requires many bits per rank, and it runs on the same tensor cores as the main FP4 GEMM. We introduce BiLoCo, a low-rank correction method whose left and right factors are stored as binary $\{\pm 1\}$ sign vectors rather than as BF16 numbers. The signed format uses $16\times$ less storage per rank-one component, and since the correction now consists of additions/subtractions, this can be run on CUDA cores concurrently with the FP4 tensor-core GEMM. On 4B–72B LLMs, BiLoCo matches or improves memory-matched SVDQuant for PTQ. We implement BiLoCo on B300 as a fused single kernel, where the FP4 GEMM runs on tensor cores, the bit-packed correction on the CUDA cores of the same launch, and the two are combined in the GEMM epilogue. With batch size 1, the fused kernel is at most 11% slower than the FP4-only GEMM for every rank up to 512. At rank 128, it is at most 20% slower up to batch size 8.

1 Introduction

LLM decode is dominated by weight movement [6, 8, 13]. Post-training quantization (PTQ) to FP4 on Blackwell GPUs can thus improve latency by reducing this traffic and moreover exposing native FP4 tensor-core GEMM. However, direct post-training quantization to FP4 still leaves an accuracy gap. A natural fix is to keep the residual weight matrix in lower bits and add a low-rank higher-precision correction, i.e., factorize the original 16-bit matrix W as $W \approx R + VU$ where R is a 4 bit matrix and VU is kept in higher precision. Such a correction approach was investigated for weight-only quantization in LQ-LoRA [9], LoftQ [11], and CALDERA [17]. SVDQuant [10] extends this approach to weight and activation quantization to FP4 with a BF16 rank-32 correction, where the correction absorbs weight outliers that the low-bit residual cannot represent.

The initial SVDQuant work focused on vision diffusion models. However, autoregressive LLM decode regime differs from the diffusion regime in ways that affect correction design. Concretely, with LLM decode shapes, the FP4 GEMM is weight-traffic bound, causing tensor cores to stall on HBM while CUDA cores sit idle. Tensor cores and CUDA cores are separate units that issue instructions independently, and recent attention kernels overlap work on the two [18, 25] (§2.3). A correction that uses CUDA cores rather than tensor cores could therefore run concurrently with the FP4 GEMM instead of competing for the same hardware. SVDQuant, however, computes its BF16

*Equal contribution.

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

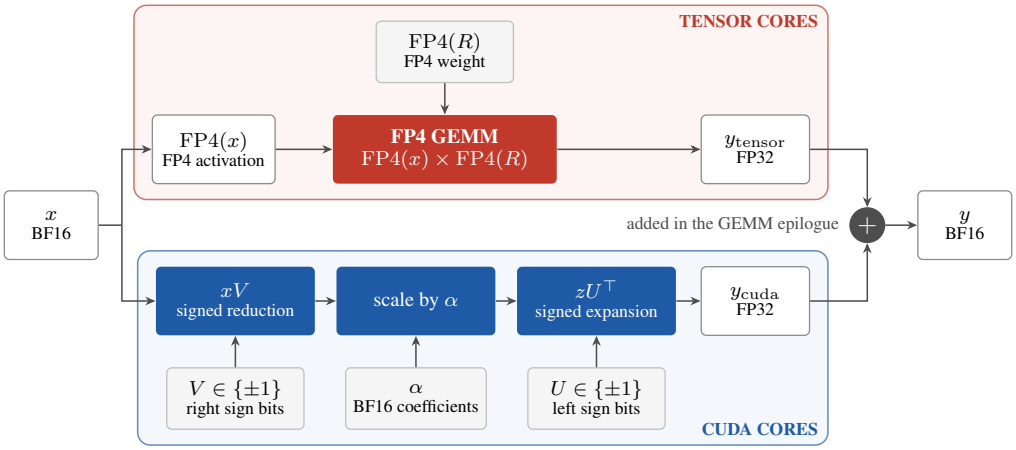


Figure 1: BiLoCo inference for performing $y = xR + xVU^\top$. A native-FP4 residual GEMM on tensor cores runs alongside a signed add/subtract correction on CUDA cores.

correction on tensor cores, the same units that run the FP4 GEMM. The challenge then, is to find a correction format whose arithmetic naturally fits the CUDA-core slot.

Could we use an alternative factorization of the weight matrix to achieve strong error correction while ensuring the correction component can make use of idle CUDA cores during inference? We introduce Binary Low-rank Correction (BiLoCo, Figure 1), where instead of storing BF16 left and right singular vectors, we store the correction factors as $\{\pm 1\}$ sign vectors, together with scalar coefficients. Each rank-one component then reduces to signed accumulation, replacing multiply operations with predicated add/subtract operations on CUDA cores. The representation also changes the storage trade-off: ignoring the per-term scalar coefficient, one binary vector pair costs $16\times$ fewer bits than a BF16 vector pair, so the same vector-storage budget as a BF16 rank-k branch can store a binary rank-$16k$ branch.

The main algorithmic challenge is that the best sign-vector decomposition is not given by the SVD. Following Riasanovsky and Kazdadi [16], we fit the sign vectors one rank-one component at a time. For each component, we start from random signs, alternate between updating u and v for a few steps, compute the best scale, and subtract the component from the residual. After all components are fit, we re-solve all scales jointly by least squares, which accounts for the overlap between components, and quantize the remaining residual to NVFP4.

We find that across various LLM families, BiLoCo is effective at the same storage budget as the BF16 correction used by SVDQuant. At the same vector-storage budget as a BF16 rank-32 branch, BiLoCo uses rank 512 binary factors and accounts for 47–53% of the Frobenius norm of LLM weight matrices across three model sizes, compared with 26–29% for BF16 rank 32. For perplexity, BiLoCo matches or improves the BF16 rank-32 correction on WikiText-2/C4 perplexity across five models. We implement BiLoCo on Blackwell GPUs as a *fused single kernel* with the factors bit-packed at runtime: the native FP4 GEMM runs on tensor cores, the binary correction is computed on the CUDA cores of the same launch, and the correction is added inside the GEMM epilogue. For batch-one inference, against the fastest FP4-only kernel we could build at the shape, BiLoCo adds 6–11% latency for ranks 16 to 512. For larger batches, the cost of the binary branch on CUDA cores grows with the batch size, while the FP4 GEMM time, which is dominated by reading the weights, does not change for batch sizes up to 128. The largest batch size at which the overhead stays within 20% therefore decreases with rank: 8 at r=128, 4 at r=256, and 2 at r=512. Our code is publicly available.[†]

2 Background

2.1 FP4 Quantization

Given the FP4 native tensor core GEMMs on the Blackwell generation of GPUs, there has been much recent work on investigating FP4 quantization. In this work we work with NVFP4 quantization, which

[†] `https://github.com/jindavid/BiLoCo`

partitions the input dimension of a tensor into blocks of 16 elements [15]. Each block is quantized to E2M1 (4-bit, 1 sign + 2 exponent + 1 mantissa) with a shared E4M3 (FP8) scale, and the whole tensor has one additional FP32 scale. For a weight matrix $W \in \mathbb{R}^{m \times n}$, the quantized representation is $\hat{W}_{ij} = g\, s_{i,\lfloor j/16 \rfloor}\, q_{ij}$ with $q_{ij} \in \{0, \pm 0.5, \pm 1, \pm 1.5, \pm 2, \pm 3, \pm 4, \pm 6\}$, s the per-block FP8 scale, and g the per-tensor FP32 scale. The on-disk cost is approximately 4.5 bits per weight (4 weight bits plus 0.5 scale bits).

NVFP4 is attractive because both storage and arithmetic are low precision. When the activation and weight operands are in the native block-scaled FP4 format, Blackwell tensor cores can execute the main GEMM directly. However, the E2M1 codebook is extremely coarse, and each block has only one local scale. A few large entries in a block can therefore increase error for the remaining entries, while aggressive activation quantization can also amplify the effect of hidden state outliers. Thus, direct PTQ to NVFP4 can still leave a measurable accuracy gap, especially when both weights and activations are quantized.

2.2 Low-Rank Corrections for Quantization Error

One way to reduce this gap is to decompose a weight matrix into a low-bit residual plus a higher-precision correction. For weight-only quantization, LQ-LoRA [9] and LoftQ [11] decompose pretrained weights into a quantized component and a high-precision low-rank component via SVD, then fine-tune the low-rank component.

SVDQuant applies this decomposition for 4-bit weight-and-activation quantization for diffusion models [10]. It first migrates activation outliers into the weights, then uses SVD to decompose a matrix W into a low-rank component $VU^\top$ and a residual R, which is quantized to 4 bits. Inference proceeds as

$$y = \text{FP4-GEMM}(\text{FP4}(x), \text{FP4}(R)) + xVU^\top. \tag{1}$$

This correction is effective because SVD captures directions of large residual energy that are poorly represented by the 4-bit residual. It also has a clear cost, since the correction stores BF16 left and right factors and is thus naturally implemented as one or more higher-precision GEMMs.

2.3 Tensor Cores and CUDA Cores

Each streaming multiprocessor (SM) of an NVIDIA GPU has two kinds of arithmetic units: tensor cores, which execute matrix multiply-accumulate instructions, and general-purpose CUDA cores, which execute scalar and vector instructions such as additions and comparisons. The two kinds of units issue instructions independently, so one group of threads can keep the tensor cores busy with a GEMM while another group uses the CUDA cores. FlashAttention-3 and FlashAttention-4 use this to overlap the softmax, computed on CUDA cores and special function units, with the attention GEMMs on tensor cores [18, 25]. A GPU kernel runs as a grid of thread blocks, or cooperative thread arrays (CTAs), each scheduled on one SM and divided into warps of 32 threads. In a CUTLASS GEMM on Blackwell, each CTA computes one tile of the output. Some of its warps load the operands and issue tensor-core instructions in the main loop, and the remaining epilogue warps convert the accumulators and write the output tile to memory. At LLM decode batch sizes the GEMM is limited by reading the weights from memory, so the tensor cores are underused and the epilogue warps wait during most of the main loop.

3 Method

3.1 Motivating Study: LLM Weights Favor Rank Multiplicity

For a given memory budget, should a correction use a few high-precision rank-one components or many low-precision ones? For an $m \times n$ matrix, a BF16 rank-k correction stores $2k$ vectors in 16 bits, for $16k(m+n)$ bits. A binary rank-r correction stores r sign vectors per side and r BF16 scales, for $r(m+n+16)$ bits. At equal storage, the binary correction can therefore use about 16 times as many components. More components help only if the spectrum of W is flat enough that many directions matter, which we test next. We test this on 189 projection matrices from Qwen3-4B, Qwen3-14B, and Llama-3-8B, including 9 decoder layers per model and 7 projection types per layer. For each

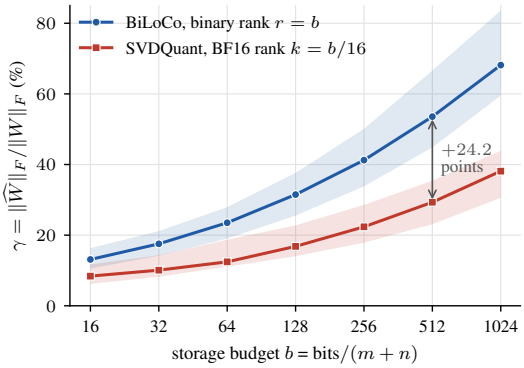


Figure 2: Fraction $\gamma = \|\widehat{W}\|_F / \|W\|_F$ of the Frobenius norm in the low-rank part versus the storage budget b, on 63 Qwen3-4B projection matrices (7 projection types in 9 decoder layers). At each budget, BiLoCo uses binary rank $r = b$ and SVDQuant uses BF16 rank $k = b/16$, so both use the same storage. Lines are medians and bands are interquartile ranges.

Table 1: Fraction γ at equal storage across three model families. Signed rank 512 and BF16 SVD rank 32 use the same vector-storage budget.

model	σ_1/σ_{32} med.	γ, SVD k=32	γ, sign r=512	ratio	Pearson
Qwen3-4B	1.93	29.3%	52.9%	1.92×	−0.65
Qwen3-14B	2.00	26.3%	47.3%	1.77×	−0.65
Llama-3-8B	2.24	27.9%	50.6%	1.76×	−0.81

matrix, we compare BF16 SVD rank 32 with a signed rank-512. We measure $\gamma = \|\widehat{W}\|_F / \|W\|_F$, the fraction of the Frobenius norm of W in the low-rank part $\widehat{W}$. Table 1 shows the results, where we find that the top spectrum is flat, with the median σ_1/σ_{32} ranging from 1.93 on Qwen3-4B to 2.24 on Llama-3-8B. At equal vector storage, signed rank 512 retains a median 47–53% of $\|W\|_F$, versus 26–29% for BF16 SVD rank 32. The ratio of the two values of γ is anti-correlated with σ_1/σ_{32} on every family, matching the rank-multiplicity prediction.

Figure 2 plots γ against vector-storage budget b. At each matched budget, SVD uses BF16 rank k=b/16, while binary factorization uses signed rank $r = b$. The binary factorization has a higher γ at every tested budget. The advantage is not isolated to one projection type as shown in Table 7. This motivates BiLoCo, where the same correction bytes can cover many more directions if the factors are one-bit signs. Since γ is only a proxy for model quality, Section 4 evaluates whether this storage advantage preserves model quality.

3.2 BiLoCo: Binary Low-Rank Corrections for Quantized Weights

BiLoCo uses binary low rank components to factorize the weight matrix and stores the residuals in NVFP4. Concretely, BiLoCo uses the same two-branch inference form as SVDQuant (Eq. 1), but changes the representation of the correction branch. Instead of a BF16 factorization, it stores signed rank-one terms and one scalar coefficient per term. At PTQ time, for each weight matrix W, we choose a correction $C = \sum_i \alpha_i u_i v_i^\top$ and store the residual $R = W - C$ in NVFP4. NVFP4 uses 16-element input-channel blocks, E2M1 4-bit values, and a shared FP8 scale, for approximately 4.5 bits per residual weight. We represent the correction as

$$W \approx \sum_{i=1}^{r} \alpha_i \, u_i v_i^\top + R, \qquad u_i \in \{\pm 1\}^m, \; v_i \in \{\pm 1\}^n, \; \alpha_i \in \mathbb{R} \tag{2}$$

The deployed forward pass, with BF16 coefficient α, is

$$y = \text{FP4-GEMM}(\text{FP4}(x), \text{FP4}(R)) + \sum_{i=1}^{r} \alpha_i \, u_i \, (v_i^\top x). \tag{3}$$

Because each $v_i \in \{\pm 1\}^n$, the inner product $v_i^\top x$ requires only additions and subtractions, i.e., $\sum_k v_{i,k} \, x_k$ where $v_{i,k} \in \{\pm 1\}$ selects between $+x_k$ and $-x_k$. This extends to the outer product with u_i. The low-rank term therefore needs no multiplications, and its factors take 16× less memory than BF16 factors.

Algorithm 1 Greedy binary peeling with global coefficient refit

Require: Weight matrix $W \in \mathbb{R}^{m\times n}$, rank r, alternation steps T
$R \leftarrow W$
for $i = 1, \ldots, r$ **do**
  initialize $v_i \in \{\pm 1\}^n$ at random
  for $t = 1, \ldots, T$ **do**
    $u_i \leftarrow \mathrm{sign}(Rv_i)$
    $v_i \leftarrow \mathrm{sign}(R^\top u_i)$
  end for
  $\alpha_i \leftarrow (u_i^\top R v_i)/(mn)$
  $R \leftarrow R - \alpha_i u_i v_i^\top$
end for
Form $A_{ij} = (u_i^\top u_j)(v_i^\top v_j)$ and $b_i = u_i^\top W v_i$
Solve $(A + \epsilon I)\alpha = b$ and set $R \leftarrow \mathrm{FP4}(W - \sum_i \alpha_i u_i v_i^\top)$
return sign vectors $\{u_i, v_i\}_{i=1}^r$, BF16 coefficients α, and FP4 residual R

To find the binary low-rank corrections, we avoid an explicit SVD and incorporate a modified signed-cut decomposition algorithm [7, 16] to build $\{u_i, v_i, \alpha_i\}_{i=1}^r$ greedily. We first initialize $R \leftarrow W$. Then for each rank-one component, we draw a random sign vector v, then alternate

$$u \leftarrow \mathrm{sign}(Rv), \qquad v \leftarrow \mathrm{sign}(R^\top u) \tag{4}$$

for T iterations. The optimal scalar for fixed binary u, v is $\alpha = (u^\top R v)/(mn)$, which minimizes $\|R - \alpha u v^\top\|_F^2$ subject to u, v being binary. We then deflate via $R \leftarrow R - \alpha u v^\top$ and proceed on to the next rank. In practice, $T = 5$ is sufficient for the alternation to settle. [‡]

Greedy peeling produces (u_i, v_i) at a coordinate-wise stationary point per term but uses the per-term closed form $\alpha_i = (u_i^\top R_i v_i)/(mn)$ that ignores cross-term correlations. Once all r signed terms are produced, we refit $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^r$ jointly in closed form:

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^r} \left\| W - \textstyle\sum_i \alpha_i u_i v_i^\top \right\|_F^2, \tag{5}$$

which solves the $r \times r$ system $A\boldsymbol{\alpha} = b$ with $A_{ij} = (u_i^\top u_j)(v_i^\top v_j)$ and $b_i = u_i^\top W v_i$. A small regularization ϵI (i.e., as in ridge regression) is added for numerical stability. The solve is $O(r^3)$ and runs in milliseconds per layer. Building b shares the $U^\top W V$ matmul that the peel step already amortizes. After the refit, the storage residual $R_{\mathrm{store}} = W - U\mathrm{diag}(\alpha)V^\top$ is quantized to NVFP4 once. Algorithm 1 summarizes the full peeling and refit procedure.

3.3 Theoretical Analysis

BiLoCo uses greedy sign-alternation peeling because each coordinate update maximizes the current bilinear objective over one set of signs, and therefore monotonically improves the single-term Frobenius decrease. After the signs are fixed, the final coefficient refit is the least-squares optimum over the selected rank-one components. Our theoretical analysis shows that the PTQ solver is a principled matching-pursuit procedure for the Frobenius norm as a proxy objective. (The proofs are in Appendix B.)

Theorem 3.1 (Monotone single-term sign alternation) *Let $R \in \mathbb{R}^{m\times n}$ be fixed. Starting from any $v^{(0)} \in \{\pm 1\}^n$, alternate $u^{(t)} = \mathrm{sign}(Rv^{(t-1)})$ and $v^{(t)} = \mathrm{sign}(R^\top u^{(t)})$, with deterministic tie-breaking. Then $u^\top R v$ is non-decreasing after each coordinate update. For the final signs u, v, the optimal scalar is $\alpha^\star = (u^\top R v)/(mn)$, and deflation yields*

$$\|R\|_F^2 - \|R - \alpha^\star u v^\top\|_F^2 \;=\; \frac{(u^\top R v)^2}{mn}.$$

[‡]The sign alternation and the joint refit follow Riasanovsky and Kazdadi [16]. Our contributions are using them to correct NVFP4 quantization error and the fused kernel of §3.4.

Proposition 3.2 (Global coefficient refit) *For fixed sign matrices $U \in \{\pm 1\}^{m\times r}$ and $V \in \{\pm 1\}^{n\times r}$, the least-squares coefficients*

$$\alpha^\star = \arg\min_\alpha \|W - \sum_i \alpha_i u_i v_i^\top\|_F^2$$

solve $A\alpha = b$, where $A_{ij} = (u_i^\top u_j)(v_i^\top v_j)$ and $b_i = u_i^\top W v_i$. Replacing the greedy coefficients by $\alpha^\star$ cannot increase the Frobenius residual.

Cut-norm motivation. The rank-one components can also be viewed through the cut norm. Let $\|R\|_\blacksquare = \max_{u\in\{\pm1\}^m,\, v\in\{\pm1\}^n} u^\top R v$ denote the signed cut norm of R. If, at each peeling step t, the selected signs satisfy

$$|u_t^\top R_t v_t| \geq \rho \|R_t\|_\blacksquare$$

for some $\rho \in (0, 1]$, then with $\alpha_t = (u_t^\top R_t v_t)/(mn)$ and $R_{t+1} = R_t - \alpha_t u_t v_t^\top$, the residuals satisfy

$$\min_{0\leq t<r} \|R_t\|_\blacksquare^2 \;\leq\; \frac{mn}{\rho^2 r}\|W\|_F^2.$$

This is a conditional Frieze–Kannan-style statement [1, 7]: it explains what a uniformly good signed-cut oracle would imply, but it is not a claim that our alternating heuristic attains a fixed approximation ratio to the optimal cut norm. Empirically, we measure the rate per step

$$\rho_F(t) = \frac{|u_t^\top R_t v_t|}{\sqrt{mn}\,\|R_t\|_F},$$

which is exactly the rate constant in

$$\|R_{t+1}\|_F^2 = \|R_t\|_F^2\left(1 - \rho_F(t)^2\right).$$

Across 63 Qwen3-4B matrices and 8064 measured rank-one updates, ρ_F remains bounded away from zero in the diagnostic sweep. The full ρ_F distribution is reported in Appendix A.1.

Scope of Analysis. The analysis establishes three local facts: the per-term sign search is monotone, the deployed coefficients are optimal over the chosen signs, and a conditional cut-norm oracle would yield a standard residual-decay statement. It does not bound output error, downstream accuracy, or PPL. The role of the analysis is to support the storage–accuracy argument from Section 3.1 and to make the greedy signed-peeling solver principled rather than ad hoc. The model-quality claims are validated empirically in Section 4.

3.4 Kernel Design

We realize the efficiency argument of BiLoCo with a *fused single kernel*, which runs the FP4 GEMM on tensor cores while computing the bit-packed binary correction on CUDA cores. For a layer with $W \in \mathbb{R}^{N\times K}$, input $X \in \mathbb{R}^{M\times K}$, and output $Y \in \mathbb{R}^{M\times N}$, we have

$$Y = \underbrace{X\,R^\top}_{\text{tensor-core path}} + \underbrace{\sum_{i=1}^{r} \alpha_i (X v_i) u_i^\top}_{\text{BiLoCo branch}}, \qquad u_i \in \{\pm1\}^N,\ v_i \in \{\pm1\}^K. \tag{6}$$

Fused execution. The kernel is built on a CUTLASS SM100 block-scaled FP4 GEMM and launches at most one thread block (cooperative thread array, CTA) per streaming multiprocessor (SM). The GEMM uses output tiles of 128×64, with the batch zero-padded to 128 rows, so the first $N/64$ CTAs each compute one output tile with the unmodified tensor-core GEMM. Because these CTAs wait for the correction inside the kernel, all CTAs must run at the same time, which requires $N/64$ to be smaller than the number of SMs (148 on B300). The remaining CTAs never touch the GEMM. They compute the branch on CUDA cores directly from the packed sign bits, where each inner product $v_i^\top x$ is a sequence of additions whose signs come from the bits of v_i. Each scaled result $z_i = \alpha_i(v_i^\top x)$ is written to a buffer in global memory together with the index of the current launch (an epoch tag). The buffer is reused across launches, and the tag lets a reader tell a value written by the current launch from a stale one, so the buffer never has to be cleared, which would cost an extra kernel launch. Every output tile needs all r values of z. In each GEMM CTA, the epilogue warps, which are idle

during the main loop, therefore wait until z has been written, expand it through the packed u_i into the tile's correction, and add it to the FP32 accumulators before the BF16 stores. The low-rank product is thus split in two: the side CTAs compute z on SMs the GEMM does not use, and the epilogue warps compute each tile's part of $\sum_i z_i u_i^\top$ while the tensor cores run the main loop. By scheduling the correction onto idle SMs and executing the addition inside the GEMM epilogue, BiLoCo avoids separate kernel launches, CUDA streams, or auxiliary memory passes. Implementation details are in Appendix A.6.

Memory and precision. Both sign factors stay bit-packed at runtime, with $V \in \{0,1\}^{r\times K/8}$ and $U \in \{0,1\}^{r\times N/8}$. At r=128 the branch reads about 0.16 MiB of signs plus the BF16 token, while the FP4 path reads ∼12.5 MiB of E2M1 weights and 1.6 MiB of scales, so the correction moves 80× less data. Its cost is the time of a chain of dependent steps (summing over K, publishing z, waiting for it in the GEMM CTA, and expanding it), not the data it reads.

Batch size and rank. The same single kernel runs at every batch size and rank we measure. The work of the branch grows with Mr, while the FP4 GEMM time, dominated by reading the weights, does not change for $M \leq 128$. The branch is hidden as long as it finishes during the GEMM main loop, so the batch range over which it stays hidden depends on the rank (§4.2). For M=16 with $r\geq$256 and M=32 with $r\geq$128, the side CTAs alone are too slow, so every CTA first computes the correction and the GEMM runs afterwards. This is still one launch, but the correction is no longer overlapped with the GEMM. Beyond that batch range, and at the wide MLP projections where $N/64$ exceeds the SM count, a tensor-core correction is the better choice.

4 Experiments

We evaluate BiLoCo in two parts. §4.1 asks whether BiLoCo improves model quality, measured by perplexity, KL divergence to the BF16 model, and downstream task performance over FP4 post-training quantization. §4.2 then measures the decode latency of the fused kernel of §3.4 against the FP4-only kernel across batch sizes.

4.1 Performance Evaluation

Setup. We evaluate on Qwen3-4B, Llama-3-8B, Qwen3-14B, Qwen3-32B, and Qwen2.5-72B. All quantization uses SmoothQuant [23] presmoothing with $\alpha = 0.5$, $T = 5$ sign-alternation steps per rank-one component, and the "four over six" NVFP4 per-block MSE scale search following Cook et al. [3] for both weights and activations. For r=512, the full BiLoCo PTQ pass takes approximately 1 minute for Qwen3-4B and 5 minutes for Qwen3-32B on a single B300. Calibration uses 128 sequences of length 2048 from the C4 training set. SVDQuant is deterministic given this calibration set. The WikiText-2 test set is concatenated and split into non-overlapping 2048-token windows, and C4 validation uses 256 fixed-seed random 2048-token windows.

We compare BiLoCo and SVDQuant at the same storage budget b=512 on perplexity (Table 2), KL divergence to the BF16 model (Table 3), and downstream accuracy (Table 4). We also report BiLoCo at r=128, which uses a quarter of that storage and has a γ at least as large as SVDQuant k=32 (Table 7). The appendix also reports perplexity across ranks (Appendix A.2), per-task downstream accuracy (Appendix A.4), and KL divergence at b=128 (Appendix A.5).

Perplexity. BiLoCo entries are means over results from six sign initializations, and the table reports their standard deviation. At b=512, BiLoCo (r=512) has lower perplexity than SVDQuant (k=32) on both datasets for Qwen3-4B/14B and Llama-3-8B. On Qwen3-32B, BiLoCo is slightly better on WikiText-2 and tied on C4. On Qwen2.5-72B, it is tied on WikiText-2 and better on C4. At r=128, which uses a quarter of the storage, BiLoCo is slightly behind SVDQuant, by at most 0.05 PPL, on every model and dataset.

KL divergence. We compare the quantized next-token distributions to the BF16 reference model on 200 held-out FineWeb documents of up to 2048 tokens (Table 3). We report the mean KL over all tokens and the 99th percentile of the per-document mean KL. At b=512, BiLoCo r=512 has lower 99th-percentile KL than SVDQuant on all five models and lower mean on all but Qwen2.5-72B.

Table 2: Perplexity of BiLoCo vs. baseline methods. BiLoCo with b=512 is memory-matched to SVDQuant writh $k = 32$. BiLoCo entries report mean $\pm$ std over sign initializations. Bold marks the best corrected FP4 method, with ties bolded.

Model	Dataset	BF16	NVFP4	SVDQuant k=32	BiLoCo r=128	BiLoCo r=512
Qwen3-4B	Wiki2	14.22	15.13	15.07	15.09 $\pm$0.58	**14.94** $\pm$0.37
Qwen3-4B	C4	15.25	15.88	15.85	15.88 $\pm$0.23	**15.81** $\pm$0.17
Llama-3-8B	Wiki2	6.27	7.00	6.92	6.93 $\pm$0.01	**6.84** $\pm$0.01
Llama-3-8B	C4	6.69	7.68	7.59	7.61 $\pm$0.01	**7.48** $\pm$0.01
Qwen3-14B	Wiki2	8.89	9.36	9.19	9.21 $\pm$0.03	**9.17** $\pm$0.04
Qwen3-14B	C4	11.19	11.55	11.46	11.49 $\pm$0.03	**11.44** $\pm$0.02
Qwen3-32B	Wiki2	7.85	8.23	8.18	8.23 $\pm$0.03	**8.16** $\pm$0.03
Qwen3-32B	C4	10.09	10.46	**10.40**	10.44 $\pm$0.01	**10.40** $\pm$0.02
Qwen2.5-72B	Wiki2	3.92	4.29	**4.23**	4.26 $\pm$0.01	**4.23** $\pm$0.01
Qwen2.5-72B	C4	7.83	8.10	8.05	8.07 $\pm$0.01	**8.00** $\pm$0.01

Table 3: KL divergence to the BF16 model at b=512 on FineWeb document. Lower is better.

	KL mean (↓)		KL p99 (↓)	
Model	BiLoCo r=512	SVDQuant k=32	BiLoCo r=512	SVDQuant k=32
Qwen3-4B	**0.0740**	0.0835	**0.1421**	0.1649
Qwen3-14B	**0.0480**	0.0517	**0.1303**	0.1671
Llama-3-8B	**0.0756**	0.0824	**0.3252**	0.3409
Qwen3-32B	**0.0589**	0.0605	**0.3175**	0.3363
Qwen2.5-72B	0.0515	**0.0497**	**0.1937**	0.2097

On four of the five models the relative gap is larger at the 99th percentile (on Qwen3-14B, 0.130 vs. 0.167, 22% lower), so the improvement is not limited to typical documents. Appendix A.5 also compares the two methods at b=128 (BiLoCo r=128 vs. SVDQuant k=8).

Downstream tasks. Table 4 reports the WinoGrande (5-shot) and HellaSwag (10-shot) mean accuracy, following the protocol of Egiazarian et al. [4]. At b=512, BiLoCo improves over SVDQuant on the four Qwen models by 0.1–0.7 points and is 0.3 lower on Llama-3-8B. With r=128, which uses a quarter of the storage, BiLoCo is 0.4 points above SVDQuant k=32 on Qwen3-14B and at most 0.3 points below it on every model. Appendix A.4 reports per-task results on five standard PTQ tasks (ARC-easy, ARC-challenge, PIQA, WinoGrande, OBQA). There, BiLoCo r=512 has a higher mean than SVDQuant on Qwen3-4B and Qwen3-14B and is 0.3 points lower on Llama-3-8B, where BiLoCo r=128 has the highest mean.

4.2 Decode Kernel Efficiency

Setup. Kernel measurements run on NVIDIA B300 (SM103, 148 SMs) at the decode shape K=N=5120 for batch sizes M=1–32. We capture 64 launches in one CUDA graph and report the median time per launch over 15 interleaved passes. Timing one launch at a time would add launch overhead comparable to the kernel itself. Every ratio is computed against the fastest FP4-only GEMM kernel with the same 128×64 tiles and the activation zero-padded to 128 rows, measured in the same run, because FP4 GEMM time is data-content sensitive at small M and cross-run ratios are not meaningful. The padded kernel is the fastest FP4-only kernel we found at this shape (5.8–5.9 μs). The fastest unpadded FP4-only GEMM is 23–36% slower, so ratios against it would make the BiLoCo kernel look faster than it is.

Fused kernel per batch size/rank. Table 5 reports decode latency across batch sizes and ranks. At batch size M=1, the correction adds 6–11% over the FP4-only kernel for every rank from 16 to 512 (0.34–0.64 μs) in single launch. Because the binary branch runs on CUDA cores, its cost grows with M and r while the FP4 GEMM stays flat, so the batch range over which BiLoCo stays within 20% of FP4-only declines with larger ranks (from M=32 at r=16 to M=2 at r=512 in Table 5). Beyond those points the binary branch is no longer hidden, and we expect a tensor-core correction to be the better choice. At equal rank, a dense BF16 correction placed in the same kernel costs the same as the

Table 4: Mean accuracy on WinoGrande (5-shot) and HellaSwag (10-shot). Higher is better.

Method	Qwen3-4B	Qwen3-14B	Llama-3-8B	Qwen3-32B	Qwen2.5-72B
BF16 (reference)	0.686	0.772	0.803	0.803	0.857
SVDQuant k=32 (b=512)	0.669	0.759	**0.780**	0.796	0.849
BiLoCo r=128 (b=128)	0.667	**0.763**	0.777	0.795	0.846
BiLoCo r=512 (b=512)	**0.670**	**0.763**	0.777	**0.799**	**0.856**

Table 5: Decode latency of the BiLoCo kernel on B300 (K=N=5120). Each entry gives the time per launch in μs and, below it, the ratio to the padded FP4-only kernel measured in the same run (5.8–5.9 μs). Gray entries are more than 20% slower than the FP4-only kernel.

	Rank r					
Batch M	16	32	64	128	256	512
1	6.17 1.06×	6.18 1.06×	6.23 1.06×	6.33 1.08×	6.39 1.09×	6.49 1.11×
2	6.36 1.08×	6.36 1.09×	6.36 1.09×	6.32 1.08×	6.36 1.09×	6.71 1.15×
4	6.36 1.09×	6.36 1.09×	6.46 1.10×	6.47 1.11×	6.90 1.18×	8.45 1.44×
8	6.42 1.10×	6.42 1.10×	6.82 1.17×	6.74 1.15×	8.02 1.37×	10.10 1.74×
16	6.55 1.11×	6.52 1.11×	7.48 1.28×	9.08 1.55×	14.70 2.51×	17.81 3.04×
32	6.90 1.18×	8.86 1.51×	11.46 1.96×	15.03 2.56×	16.94 2.90×	19.33 3.30×

binary one (Appendix A.6). The advantage of the binary format is the higher rank that a fixed byte budget buys.

Fused kernel per layer shape. Table 6 reports latency at M=1 for the three supported GEMM shapes in a Qwen3-14B decoder layer, at r=128, 256, and 512, against the faster of the padded and unpadded FP4-only GEMM of the same shape measured in the same run. The overhead is smallest at the down projection, where the FP4 GEMM itself is longest, and largest at the narrow k/v projection. The kernel does not support the gate and up projections, because their $N/64$=272 output tiles exceed the 148 SMs.

5 Related Work

Post-training quantization for LLMs. GPTQ [5] and AWQ [12] target W4A16 (4-bit weights, BF16 activations). SmoothQuant [23] introduced the per-channel scale migration we use as a preprocessing step. QuaRot [2] and SpinQuant [14] are rotation-based W4A4 methods that pre-rotate weights and activations to make them more quantization-friendly. Rotation is orthogonal to correction format: our binary branch can be applied on top of a rotated backbone, and Appendix A.3 evaluates it with QuaRot and SpinQuant on Llama-3-8B.

Low-rank corrections for FP4 quantization. SVDQuant [10] introduced the FP4-with-low-rank-correction recipe for diffusion models, with a BF16 rank-32 projector by default. We extend this pattern to LLMs and study an alternative format under the same byte budget, fitting $16 \times 32 = 512 \pm 1$ binary rank-one components in place of a BF16 rank-32 correction. As demonstrated in Section 3.1, LLM weight matrices exhibit flat top spectra, meaning that allocating a fixed memory budget toward higher rank coverage yields greater fidelity than allocating it toward the precision of each rank-one component. LQ-LoRA [9] explored low-rank + low-bit decomposition for fine-tuning, focused on training-time storage rather than inference latency.

1-bit and ternary networks. BitNet [20] and 1.58-bit LLMs [26] train models entirely in binary or ternary weights from scratch. They are not post-training corrections to a higher-precision model. We borrow the binary representation but apply it as an additive PTQ branch on top of FP4.

Table 6: BiLoCo kernel across Qwen3-14B projection shapes (M=1, B300). Each entry gives the time per launch in μs and, below it, the ratio to the fastest FP4-only GEMM of the same shape measured in the same run.

Shape ($K\times N$)	Role	padded	unpadded	r=128	r=256	r=512
5120×5120	attn q/o	5.85	7.54	6.33 1.08×	6.39 1.09×	6.49 1.11×
5120×1024	attn k/v	5.72	5.62	6.10 1.09×	6.55 1.17×	7.00 1.25×
17408×5120	MLP down	12.42	22.65	12.93 1.04×	12.97 1.04×	13.18 1.06×

Signed cut decompositions on LLM weights. The closest line of work [16] applies Frieze–Kannan-style signed cut decompositions to LLM weight matrices. Their Algorithm 5 (greedy sign-alternation peel) is the same per-component search used inside our Algorithm 1, and their Algorithm 6 is the global α refit. We use binary rank-one components as a low-rank *additive* correction to an FP4 main path, framed by the matched-storage argument of Section 3.1, and run the correction on CUDA cores inside the FP4 GEMM kernel.

Spectral structure of LLM weights. Prior work has used singular-value spectra to analyze or compress transformer weights: random-matrix analyses study where information is stored across the spectrum, and SVD-based compression methods quantify the loss from truncating singular components [19, 21, 22, 24]. Our Section 3.1 measurement is complementary. Rather than using SVD for compression directly, we measure the top-spectrum shape at the SVDQuant rank and find that it is flat enough that a fixed correction budget is better spent on many binary rank-one components than on fewer high-precision ones.

Heterogeneous execution on GPUs. Recent GPU kernels increasingly overlap matrix multiplication, memory movement, and other work in attention and serving systems [13, 18, 25, 27]. BiLoCo applies this heterogeneous design pattern within a single projection layer by assigning the residual GEMM to FP4 tensor cores and the binary correction to CUDA cores.

6 Limitations

Kernel scope. The fused single kernel covers every batch size and rank we measured. It stays within 20% of the FP4-only kernel through M=8 for ranks up to r=128, through M=4 at r=256, and through M=2 at r=512 (Table 5). Larger batches, and the wide MLP projections whose tile count exceeds the SM count, are better served by a tensor-core correction. The KV cache stays BF16 throughout our experiments, as BiLoCo specifically targets linear weight projections and their activations rather than attention operations or KV-cache memory traffic. Our latency claims are scoped to B300 and the projection shapes of Tables 5 and 6.

Composition with rotations. Our rotation experiments (Appendix A.3) are limited to a single model and seed, where both tested rotation techniques underperform presmoothing under NVFP4 block quantization.

7 Conclusion

We presented BiLoCo, a post-training quantization method that splits each LLM weight matrix into a low-rank component with one-bit sign factors and an FP4 residual. At a fixed storage budget, many one-bit rank-one components describe LLM weights better than a few high-precision ones, and BiLoCo surpasses or matches the performance of SVDQuant at the same storage across models from 4B to 72B parameters. The binary format simplifies the arithmetic of the low-rank branch into additions and subtractions, allowing it to execute on the idle CUDA cores concurrently with the tensor core GEMM. We turned this into a single fused kernel on Blackwell, in which idle SMs compute the low-rank product from the packed signs and the GEMM epilogue adds it to the output. Because the fused kernel closely matches baseline FP4 latency at small decode batch sizes, BiLoCo recovers significant quantization accuracy with minimal runtime overhead.

Acknowledgments

This study was supported by a grant from Korea Technology and Information Promotion Agency (TIPA) for SMEs (with LLOYKD) and MIT-IBM Computing Research Lab. David Jin is supported by an Amazon PhD Fellowship. We thank Modal for a compute grant that supported the experiments in this work.

References

[1] Noga Alon and Assaf Naor. Approximating the cut-norm via grothendieck's inequality. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 72–80, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138520. doi: 10.1145/1007352.1007371. URL https://doi.org/10.1145/1007352.1007371.

[2] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms, 2024. URL https://arxiv.org/abs/2404.00456.

[3] Jack Cook, Junxian Guo, Guangxuan Xiao, Yujun Lin, and Song Han. Four over six: More accurate nvfp4 quantization with adaptive block scaling, 2025. URL https://arxiv.org/abs/2512.02010.

[4] Vage Egiazarian, Roberto L. Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Bridging the gap between promise and performance for microscaling fp4 quantization, 2026. URL https://arxiv.org/abs/2509.23202.

[5] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

[6] Elias Frantar, Roberto L. Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language models. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '25, page 239–251, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714436. doi: 10.1145/3710848.3710871. URL https://doi.org/10.1145/3710848.3710871.

[7] Alan Frieze and Ravi Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19 (2):175–220, Feb 1999. ISSN 1439-6912. doi: 10.1007/s004930050052. URL https://doi.org/10.1007/s004930050052.

[8] Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric P. Xing, and Yoon Kim. Fast matrix multiplications for lookup table-quantized LLMs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12419–12433, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.724. URL https://aclanthology.org/2024.findings-emnlp.724/.

[9] Han Guo, Philip Greengard, Eric P. Xing, and Yoon Kim. Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning, 2024. URL https://arxiv.org/abs/2311.12023.

[10] Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Junxian Guo, Xiuyu Li, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. SVDQuant: Absorbing outliers by low-rank component for 4-bit diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vWR3KuiQur.

[11] Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

[12] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2026. URL https://arxiv.org/abs/2306.00978.

[13] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. QServe:w4a8KV4 quantization and system co-design for efficient LLM serving. In *Eighth Conference on Machine Learning and Systems*, 2025. URL https://openreview.net/forum?id=1FfmStySS1.

[14] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations, 2025. URL https://arxiv.org/abs/2405.16406.

[15] NVIDIA. Introducing NVFP4 for Efficient and Accurate Low-Precision Inference | NVIDIA Technical Blog — developer.nvidia.com. https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference, 2024. [Accessed 05-05-2026].

[16] Alex W. Neal Riasanovsky and Sarah El Kazdadi. Efficient 1-bit tensor approximations, 2024. URL https://arxiv.org/abs/2410.01799.

[17] Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea J. Goldsmith, and Mert Pilanci. Compressing large language models using low rank and low precision decomposition, 2024. URL https://arxiv.org/abs/2405.18886.

[18] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024. URL https://arxiv.org/abs/2407.08608.

[19] Max Staats, Matthias Thamm, and Bernd Rosenow. Small singular values matter: A random matrix analysis of transformer models, 2025. URL https://arxiv.org/abs/2410.17770.

[20] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. Bitnet: Scaling 1-bit transformers for large language models, 2023. URL https://arxiv.org/abs/2310.11453.

[21] Xin Wang, Samiul Alam, Zhongwei Wan, Hui Shen, and Mi Zhang. Svd-llm v2: Optimizing singular value truncation for large language model compression, 2025. URL https://arxiv.org/abs/2503.12340.

[22] Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression, 2025. URL https://arxiv.org/abs/2403.07378.

[23] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024. URL https://arxiv.org/abs/2211.10438.

[24] Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models, 2025. URL https://arxiv.org/abs/2312.05821.

[25] 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, 2026. URL https://arxiv.org/abs/2603.05451.

[26] Di Zhang, Xun Wu, Shaohan Huang, Yudong Wang, Hanyong Shao, Yingbo Hao, Zewen Chi, Li Dong, Ting Song, Yan Xia, Zhifang Sui, and Furu Wei. Sparse-bitnet: 1.58-bit llms are naturally friendly to semi-structured sparsity, 2026. URL https://arxiv.org/abs/2603.05168.

[27] Han Zhao, Weihao Cui, Quan Chen, and Minyi Guo. Ispa: Exploiting intra-sm parallelism in gpus via fine-grained resource management. *IEEE Transactions on Computers*, 72(5):1473–1487, 2023. doi: 10.1109/TC.2022.3214088.

Table 7: Frobenius-norm capture $\mathrm{capA}(r) = \|\widehat{W}_r\|_F / \|W\|_F$ per projection type, as the median over 9 evenly spaced decoder layers per model. SVDQuant uses rank k=32 (b=512), and BiLoCo uses r=128 (b=128, a quarter of that storage) and r=256 (b=256, half). Bold marks BiLoCo r=256, which is the highest in every row. Higher is better.

Model	Projection	SVDQuant k=32	BiLoCo r=128	BiLoCo r=256
Qwen3-4B	q_proj	28.1%	35.4%	**44.6%**
	k_proj	36.9%	43.5%	**55.8%**
	v_proj	24.7%	38.3%	**51.8%**
	o_proj	21.5%	30.8%	**41.7%**
	gate_proj	18.5%	26.5%	**34.9%**
	up_proj	15.0%	23.4%	**32.3%**
	down_proj	17.4%	24.1%	**33.0%**
Qwen3-14B	q_proj	32.1%	31.0%	**39.5%**
	k_proj	39.7%	40.9%	**53.1%**
	v_proj	29.0%	35.1%	**47.9%**
	o_proj	25.0%	25.5%	**34.8%**
	gate_proj	22.6%	22.0%	**28.6%**
	up_proj	16.5%	17.4%	**24.0%**
	down_proj	18.1%	19.3%	**26.0%**
Llama-3-8B	q_proj	35.1%	34.4%	**44.0%**
	k_proj	41.3%	43.2%	**55.5%**
	v_proj	30.7%	35.6%	**48.5%**
	o_proj	27.3%	28.3%	**38.5%**
	gate_proj	24.6%	24.5%	**31.1%**
	up_proj	17.4%	19.4%	**26.6%**
	down_proj	19.3%	20.5%	**27.6%**

A Additional Experimental Details

A.1 Structural and cut-premise diagnostics

Aggregating the per-matrix capture diagnostic of Section 3.1 by projection type shows that the binary advantage holds for every projection type on all three models. Table 7 reports medians across 9 evenly-spaced decoder layers per model. The metric is the Frobenius capture $\mathrm{capA}(r) = \|\widehat{W}_r\|_F / \|W\|_F$ from Table 1, where $\widehat{W}_r$ is the rank-r truncated SVD reconstruction for SVDQuant and the rank-r greedy signed-peel reconstruction for BiLoCo.

BiLoCo r=256 captures more than SVDQuant k=32 on all 21 projection–model pairs while using half the storage. At b=128, a quarter of the storage, BiLoCo captures about the same fraction as SVDQuant on Qwen3-14B and Llama-3-8B and substantially more on Qwen3-4B. This pattern follows the differing spectral flatness of the three families (Table 1).

We also check the premise of the conditional cut-decay theorem empirically. The premise of Theorem B.2 involves the cut norm $\|R_t\|_{\blacksquare}$, which is NP-hard to compute. We instead measure the Frobenius analog $\rho_F(t) = |u_t^\top R_t v_t| / (\sqrt{mn}\,\|R_t\|_F)$. The quantity is exactly the rate constant in $\|R_{t+1}\|_F^2 = \|R_t\|_F^2(1 - \rho_F(t)^2)$ and by Cauchy–Schwarz it lower-bounds the cut-norm condition. Figure 3 shows the distribution per iteration across 63 Qwen3-4B matrices. Over all 8064 rank-one updates $\rho_F \geq 0.019$, and the median decays only mildly from 0.036 at iteration 0 to 0.026 at iteration 127. So in practice, the premise holds with ρ_F bounded away from zero.

A.2 Solver ablation, construction cost, and perplexity across ranks

Table 8 compares four ways to construct the r=512 low-rank correction on Llama-3-8B, using the setup of §4.1 and three sign seeds each. Random ± 1 sign vectors with the optimal joint coefficient refit give no improvement (7.000 against the uncorrected 6.998 on WikiText-2), so all the gain comes from the learned signs. The other three constructions are greedy peeling, greedy peeling with the global refit, and greedy peeling followed by 10 alternating-minimization sweeps over the rank-one components (*pm-altmin*), where each step removes one rank-one component, re-solves its sign alternation against the joint residual, and re-fits its coefficient. At this rank the three are within seed noise of one another.

Offline construction cost. Table 9 breaks the Llama-3-8B PTQ pass into its stages on one node. Building 512 binary rank-one component greedily takes 86 s, half the 170 s of the rank-32 SVD construction it replaces.

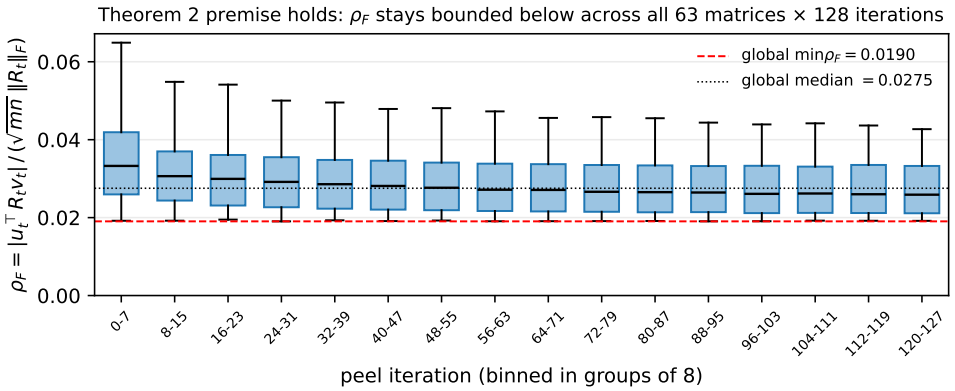


Figure 3: Frobenius-relative capture rate over 128 peel iterations on Qwen3-4B projection matrices.

Table 8: Solver ablation on Llama-3-8B at r=512: perplexity (mean ± std over three sign seeds) for four ways of constructing the low-rank component. Without a low-rank component (NVFP4 only), perplexity is 6.998 on WikiText-2 and 7.680 on C4.

construction	WikiText-2	C4
random signs + joint coefficient refit	7.000 ±0.001	7.684 ±0.008
greedy learned signs	6.842 ±0.007	7.475 ±0.018
greedy learned signs + global refit (Algorithm 1)	6.843 ±0.017	7.480 ±0.011
+ pm-altmin, 10 sweeps	6.838 ±0.013	7.459 ±0.014

The global coefficient refit adds ∼5 s, and residual NVFP4 quantization with bookkeeping takes under a second. Full alternating minimization is an order of magnitude more expensive and it is not part of the default construction.

Table 9: PTQ time on Llama-3-8B in seconds on one node. "Other" is the NVFP4 quantization of the residual and bookkeeping. The PTQ total also includes collecting the calibration activations.

method	solver	other	layer total	PTQ total
SVDQuant, BF16 rank 32 (SVD)	170.4	2.5	174.3	222
greedy BiLoCo, r=512	86.3	0.7	90.3	116
greedy + global refit, r=512	91.3	0.2	92.5	118
batched pm-altmin, r=512	213.9	0.7	215.9	263
full pm-altmin, r=512	1070.2	0.2	1071.3	1097

Perplexity across correction ranks. Table 10 sweeps the rank on Llama-3-8B with setup of §4.1. Perplexity improves monotonically with rank and keeps improving past the matched-storage point, so r=512 is set by the byte budget rather than accuracy. Ranks r=1024 and r=2048, at 2× and 4× the storage of SVDQuant k=32, improve further, but the kernel overhead also grows with ranks (Table 5).

A.3 Composition with Rotated Backbones

Rotation-based W4A4 methods rotate weights and activations so that outliers spread across channels. Table 11 replaces SmoothQuant presmoothing with two full-width backbones on Llama-3-8B. QuaRot [2] absorbs a 4096-dimensional Hadamard rotation R_1 into the residual stream and adds a block-2048 rotation before the down projection. SpinQuant [14] instead learns a global R_1, parametrized by the Cayley transform, initialized at the Hadamard matrix, and trained on the calibration cross-entropy of the quantized model with straight-through gradients through the FP4 fake-quant. Every other setting is unchanged, and the KV cache stays in BF16, so the KV rotation R_2 does not apply. BiLoCo improves both rotated backbones, and on each it is better than SVDQuant at the same storage. The FP4-only rows are worse than the presmoothed backbone, and the control without any preprocessing indicates that rotation itself hurts under NVFP4. We attribute this to rotation spreading localized outliers across the 16-element scaling blocks. The experiment therefore shows that BiLoCo is compatible with QuaRot and SpinQuant, not that rotation is the preferred NVFP4 preprocessing, and it uses one model and one seed.

Table 10: BiLoCo perplexity versus rank on Llama-3-8B (C4-train calibration, pm-altmin)

	WikiText-2	C4
BF16	6.266	6.690
NVFP4 only	6.998	7.680
SVDQuant, BF16 rank 32	6.919	7.592
BiLoCo r=32	6.983	7.656
BiLoCo r=128	6.926	7.605
BiLoCo r=256	6.892	7.549
BiLoCo r=512	6.835	7.466
BiLoCo r=1024	6.749	7.333
BiLoCo r=2048	6.627	7.133

Table 11: Perplexity of Llama-3-8B with full-width rotated backbones (seed 42, lower is better). SVDQuant uses BF16 rank 32, and BiLoCo uses r=512 with the pm-altmin solver (Appendix A.2).

backbone / correction	WikiText-2	C4
no preprocessing, NVFP4 only	7.056	7.751
presmoothing, NVFP4 only	6.998	7.680
QuaRot, NVFP4 only	7.206	7.981
QuaRot + SVDQuant	7.130	7.904
QuaRot + BiLoCo	**6.999**	**7.731**
SpinQuant, NVFP4 only	7.240	8.035
SpinQuant + SVDQuant	7.158	7.923
SpinQuant + BiLoCo	**7.014**	**7.745**

A.4 Full Downstream Task Table

Table 12 reports the per-task PTQ accuracy on five standard PTQ tasks for three of the models in Table 4. ARC-easy, ARC-challenge, PIQA, and OBQA report `acc_norm`, and WinoGrande reports `acc`. All entries use the setup of§4.1: C4-train calibration, SmoothQuant presmoothing α=0.5, and the `mse`-based NVFP4 scale search. BiLoCo r=512 has a higher five-task mean on Qwen3-4B and Qwen3-14B and is 0.3 points lower on Llama-3-8B, where BiLoCo r=128 has the highest mean.

Table 12: Standard PTQ downstream task breakdown. Higher is better.

Model	Method	b	ARC-e	ARC-c	PIQA	WinoG.	OBQA	mean
Qwen3-4B	SVDQuant k=32	512	0.775	0.467	**0.738**	0.639	0.296	0.583
	BiLoCo r=128	128	**0.789**	0.485	0.727	0.618	**0.298**	0.583
	BiLoCo r=512	512	0.782	**0.486**	0.724	**0.652**	0.286	**0.586**
Qwen3-14B	SVDQuant k=32	512	**0.830**	0.557	0.780	**0.717**	0.344	0.646
	BiLoCo r=128	128	**0.830**	0.560	**0.795**	0.688	0.344	0.643
	BiLoCo r=512	512	0.828	**0.583**	**0.795**	0.715	**0.348**	**0.654**
Llama-3-8B	SVDQuant k=32	512	0.787	0.472	0.775	0.716	0.328	0.616
	BiLoCo r=128	128	**0.790**	**0.485**	**0.777**	**0.728**	**0.338**	**0.624**
	BiLoCo r=512	512	0.767	0.469	0.774	**0.728**	0.326	0.613

A.5 Full KL Table

Beyond perplexity and downstream accuracy, we measure the mean and 99th-percentile per-token Kullback–Leibler divergence $\mathrm{KL}(p_{\mathrm{BF16}} \,\|\, p_{\mathrm{quant}})$ between the BF16 reference model and each quantized model on 200 held-out FineWeb samples (Table 13). Table 13 covers three models at two storage budgets, and Table 3 adds Qwen3-32B and Qwen2.5-72B at b=512. At b=512, BiLoCo has lower mean KL than SVDQuant on all three models (Qwen3-4B 0.074 vs. 0.084, Qwen3-14B 0.048 vs. 0.052, Llama-3-8B 0.076 vs. 0.082) and also lower KL_{p99} on all three. At b=128, the results are mixed. BiLoCo is better on both metrics for Llama-3-8B, better at the 99th percentile but slightly worse on the mean for Qwen3-4B, and worse on both for Qwen3-14B.

Table 13: KL divergence to the BF16 model at b=128 and b=512 on held-out FineWeb samples. Lower is better.

			KL mean (↓)		KL p99 (↓)	
Model	Method	b	BiLoCo	SVDQ	BiLoCo	SVDQ
Qwen3-4B	r=128 vs. k=8	128	0.0914	**0.0901**	**0.1849**	0.1965
	r=512 vs. k=32	512	**0.0740**	0.0835	**0.1421**	0.1649
Qwen3-14B	r=128 vs. k=8	128	0.0548	**0.0541**	0.2026	**0.1618**
	r=512 vs. k=32	512	**0.0480**	0.0517	**0.1303**	0.1671
Llama-3-8B	r=128 vs. k=8	128	**0.0856**	0.0881	**0.3611**	0.4789
	r=512 vs. k=32	512	**0.0756**	0.0824	**0.3252**	0.3409

A.6 Fused-Kernel Implementation

Implementation details. The kernel zero-pads the FP4 activation to the GEMM's full 128-row tile. At small M, the block-scaled FP4 GEMM without padding is 23–36% slower, because every CTA re-reads the same small A tile at every k-step. Padding is valid because the scale-factor layout is already padded to 128 rows and the padded output rows are discarded. The correction is still computed only for the M real rows, and the FP4-only baseline in §4.2 uses the same padding.

At K=N=5120 the grid has 80 GEMM CTAs, one per 64-column output tile, and up to 68 side CTAs on the remaining SMs. The side CTAs compute $z = \alpha \odot (Vx)$ on CUDA cores directly from the packed signs of V. Each value is written with an epoch tag, so a GEMM CTA can tell a value from the current launch from a stale one without clearing the buffer between launches. In each GEMM CTA, the four epilogue warps are idle while the tensor cores run the main loop. They wait for z and expand it through the packed signs of U for the tile's 64 columns, using 16-entry lookup tables in shared memory, one per group of four ranks and token row. Each lookup replaces four sign-selected additions with one table read. The result is added to the FP32 accumulators before the BF16 store, so the correction never makes a separate pass over the output. When Mr is large (M=16 with $r \geq 256$, and M=32 with $r \geq 128$), all CTAs instead compute the correction at kernel entry and the GEMM CTAs start the GEMM afterwards. This is still one launch, but the correction is no longer hidden under the GEMM.

Equal-rank control: binary versus BF16 factors. To separate the effect of the one-bit format from the effect of the fusion, we run a dense BF16 rank-32 correction, which has SVDQuant's factor format and the same storage as binary r=512, inside the same kernel as the binary branch. The side CTAs compute z from dense V and the epilogue expands it through dense U. At M=1 and K=N=5120, the dense rank-32 kernel takes 6.20 μs, the same as the binary r=32 kernel, against 5.84 μs for the padded FP4-only GEMM in the same run. The same dense correction run unfused, as two BF16 GEMMs followed by the FP4 GEMM with a residual add, takes 17.64 μs. At equal rank the two formats cost the same, so the one-bit format does not make the arithmetic faster. Its advantage is the rank that a fixed storage budget buys: binary r=512 runs in 6.49 μs (Table 5) and improves accuracy over BF16 rank 32 (Table 2).

B Proofs

We include the formal statements and proofs for the signed-cut claims referenced in Section 3.3.

B.1 Monotone Single-term Sign Alternation

Theorem B.1 (Monotone single-term sign alternation) *Run Algorithm 1's inner loop from any sign initialization $v^{(0)} \in \{\pm 1\}^n$, alternate $u^{(t)} = \mathrm{sign}(Rv^{(t-1)})$ and $v^{(t)} = \mathrm{sign}(R^\top u^{(t)})$ with deterministic tie-breaking, and set $\alpha^\star = (u^\top Rv)/(mn)$ where u, v are the final iterates. Then $u^\top Rv$ is non-decreasing after each u- or v-update. The deflated residual $R' = R - \alpha^\star uv^\top$ satisfies*

$$\|R\|_F^2 - \|R'\|_F^2 \;=\; \frac{(u^\top Rv)^2}{mn}.$$

Monotonicity. For fixed $u \in \{\pm 1\}^m$, the v-update $v \leftarrow \mathrm{sign}(R^\top u)$ maximizes $u^\top Rv$ over $v \in \{\pm 1\}^n$ (sign-of-each-coordinate is a separable maximization). Symmetrically, the u-update maximizes $u^\top Rv$ over $u \in \{\pm 1\}^m$. Thus each u- or v-update is non-decreasing in the single-term objective. With deterministic tie-breaking, the finite state space implies convergence to a fixed point or plateau.

Frobenius decrease. For any $u \in \{\pm 1\}^m$, $v \in \{\pm 1\}^n$ and $R' = R - \alpha uv^\top$, $\|R'\|_F^2 = \|R\|_F^2 - 2\alpha\, u^\top Rv + \alpha^2 mn$. Minimizing in α gives $\alpha^\star = u^\top Rv/(mn)$ and $\|R\|_F^2 - \|R'\|_F^2 = (u^\top Rv)^2/(mn)$. □

B.2 Conditional Signed-cut Decay

Theorem B.2 (Conditional signed-cut decay) *Let $W \in \mathbb{R}^{m\times n}$ and run a peeling procedure for r steps. At step t, let R_t be the residual and suppose the selected signs satisfy*

$$|u_t^\top R_t v_t| \;\geq\; \rho \|R_t\|_{\blacksquare} \qquad \textit{for some } \rho \in (0,1].$$

With $\alpha_t = (u_t^\top R_t v_t)/(mn)$ and $R_{t+1} = R_t - \alpha_t u_t v_t^\top$, the residuals satisfy

$$\min_{0\leq t<r} \|R_t\|_{\blacksquare}^2 \;\leq\; \frac{mn}{\rho^2 r}\, \|W\|_F^2.$$

Let $\Phi_t = \|R_t\|_F^2$. The oracle condition and Theorem 3.1's Frobenius identity give

$$\Phi_t - \Phi_{t+1} \;=\; \frac{(u_t^\top R_t v_t)^2}{mn} \;\geq\; \frac{\rho^2 \|R_t\|_{\blacksquare}^2}{mn}.$$

Since $\Phi_0 = \|W\|_F^2$ and $\Phi_r \geq 0$, telescoping yields

$$\sum_{t=0}^{r-1} \|R_t\|_{\blacksquare}^2 \;\leq\; \frac{mn}{\rho^2}\, \|W\|_F^2.$$

At least one of the r residuals therefore satisfies $\|R_t\|_{\blacksquare}^2 \leq mn\|W\|_F^2/(\rho^2 r)$. This proves the stated minimum-over-iterations bound. No monotonicity of the final cut norm is required or claimed. □

B.3 Optimal Coefficient Refit

Theorem B.3 (Optimal α refit) *Given fixed binary $\{u_i, v_i\}_{i=1}^r$, the closed-form solution $\boldsymbol{\alpha}^* = A^{-1}b$ with $A_{ij} = (u_i^\top u_j)(v_i^\top v_j)$ and $b_i = u_i^\top W v_i$ is the unique minimizer of $\Phi(\boldsymbol{\alpha}) = \|W - \sum_i \alpha_i u_i v_i^\top\|_F^2$, provided A is invertible. The matrix A is positive semi-definite and is strictly positive definite whenever the terms are linearly independent in $\mathbb{R}^{m\times n}$.*

Let $a_i = \mathrm{vec}(u_i v_i^\top)$ and let $G = [a_1, \ldots, a_r] \in \mathbb{R}^{mn\times r}$. Then

$$\Phi(\boldsymbol{\alpha}) = \|\mathrm{vec}(W) - G\boldsymbol{\alpha}\|_2^2.$$

The normal equations are

$$G^\top G\boldsymbol{\alpha} = G^\top \mathrm{vec}(W).$$

Their entries are

$$(G^\top G)_{ij} = \langle u_i v_i^\top, u_j v_j^\top\rangle_F = (u_i^\top u_j)(v_i^\top v_j)$$

and

$$(G^\top \mathrm{vec}(W))_i = u_i^\top W v_i.$$

Thus $A = G^\top G$ and $b = G^\top \mathrm{vec}(W)$, giving $A\boldsymbol{\alpha} = b$. The Gram form makes A positive semi-definite. If the terms are linearly independent, G has full column rank and A is positive definite, so the minimizer is unique. With a numerical ridge, the implemented solve is $(A + \epsilon I)^{-1}b$. □